人工智能预测

Forecasting with AI

李丰

北京大学光华管理学院

2026 年 5 月 21 日

目录

人工智能预测

本书整理自李丰老师在北京大学光华管理学院开设的《人工智能预测》"Forecasting with AI" 课程的视频、讲义、notebook 和课堂示例，目标是把人工智能预测（AI Forecasting）写成一本可阅读、可运行、可复现的中文在线课程书。这里的人工智能（Artificial Intelligence, AI）时代预测（forecasting），既包括模型方法，也包括数据、评估和决策流程。

课程从预测基础开始，逐步进入人工智能预测、大语言模型（Large Language Model, LLM）、API 调用（Application Programming Interface, API）、预测评估（forecast evaluation）、金融预测智能体（financial forecasting agent）、Transformer、时间序列基座模型（Time Series Foundation Model, TSFM）、混合专家模型（Mixture of Experts, MoE）、模型蒸馏（Knowledge Distillation, KD）和 TimeGPT 应用。附录提供运行课程代码所需的计算平台、Python 和 Linux 基础。

如需离线阅读，可以下载本书的 PDF 版。

如何使用本书

建议按章节顺序学习：

1. 先阅读课程概览和 C01-C03，建立 forecasting、AI 预测和商业决策的概念框架。
2. 再完成 C04-C08，掌握大语言模型、API 预测、预测评估和金融预测智能体。
3. 阅读 C09-C13，理解 Transformer、时间序列基座模型、混合专家模型、模型蒸馏和间歇性需求等进阶主题。
4. 如果对 Python、Linux 或 PKU 集群不熟悉，阅读第四部分实践附录并完成其中的命令练习。

每章末尾的练习用于课堂讨论或作业设计。涉及 API key 的示例都应使用环境变量，不要把真实 key 写进 notebook、截图或提交记录。

课程定位

本课程不把预测理解为单一模型技巧，而是把它看作数据、模型、评估和决策的闭环。一个预测结果只有在目标明确、数据可追溯、评价可复现、风险可解释时，才真正适合进入业务决策。

作者

本书作者为北京大学光华管理学院李丰博士。作者简介、研究方向和联系方式见作者简介。如果在论文、作业或报告中使用本书内容，请参考如何引用。

作者简介

李丰博士现任北京大学光华管理学院商务统计与经济计量系副教授、研究员、博士生导师。本科毕业于中国人民大学统计学院，硕士毕业于瑞典达拉那大学统计学系，博士毕业于瑞典斯德哥尔摩大学统计学系。

他的研究方向包括贝叶斯统计学、计量经济学、预测方法和大数据分布式学习。相关研究发表在统计、预测、管理运筹、金融会计和医学等领域期刊，包括 *Journal of Computational and Graphical Statistics*, *Journal of Business and Economic Statistics*, *International Journal of Forecasting*, *European Journal of Operational Research*, *Contemporary Accounting Research*, *BMJ Open* 和 *Journal of Affective Disorders* 等。

李丰博士长期关注大规模时间序列预测、预测组合、分布式统计计算和 AI 支持的预测方法。他开发并维护适用于大规模时间序列预测的开源算法和程序，相关代码见 GitHub。

研究兴趣

贝叶斯统计学、计量经济学、预测方法、大数据分布式学习、大规模时间序列预测

联系方式

- 个人主页：https://feng.li
- 实验室：https://kllab.org
- ORCiD：0000-0002-4248-9778
- 邮箱：feng.li@gsm.pku.edu.cn
- 办公地址：北京市海淀区颐和园路 5 号北京大学光华管理学院 2 号楼 420

如何引用

如果你在论文、课程作业、教学材料、研究报告或项目文档中使用本书内容，请引用本书。由于本书是在线课程书，建议在引用中保留访问日期。

推荐引用格式

李丰. 人工智能预测：Forecasting with AI [EB/OL]. 2026. 可访问：https://feng.li/files/forecasting-with-ai/book/. 访问日期：请填写你的实际访问日期.

English version

Li, F. (2026). *Forecasting with AI*. Online course book. https://feng.li/files/forecasting-with-ai/book/

BibTeX

```
@book{li2026forecastingai,
  author = {Li, Feng},
  title = {{人工智能预测: Forecasting with AI}},
  year = {2026},
  publisher = {Online course book},
  url = {https://feng.li/files/forecasting -with -ai/book/},
  note = {Accessed: YYYY -MM -DD}
}
```

引用具体章节

如果只引用某一章，建议在正文中说明章节标题，并在参考文献中引用整本书。例如：

李丰. 金融预测智能体. 见：人工智能预测：Forecasting with AI [EB/OL]. 2026. 可访问：https://feng.li/files/forecasting-with-ai/book/chapters/c08-ai-in-financial-forecasting/.

请把 `YYYY-MM-DD` 或 “访问日期” 替换为你实际访问本书的日期。

课程概览

人工智能正在改变预测的工作方式。传统预测（traditional forecasting）强调统计模型（statistical model）、专家经验和少量结构化数据（structured data）；现代预测系统则同时使用时间序列（time series）、文本、图像、事件、外生变量（exogenous variable）和大规模预训练模型（large-scale pretrained model），这也延续了预测研究中从理论、实践到 AI 工具扩展的主线 [Petropoulos et al., 2022, Makridakis et al., 2023]。本课程的目标，是帮助学生理解这种变化，并能把 AI 工具放入可复现、可评估的预测流程中。

面向读者

本书面向大学本科生及以上学员、MBA 或 EMBA 学员，以及在企业、政府、金融、运营、供应链、市场、医疗、能源和科技团队中有预测需求的工作人员和管理者。读者不需要先成为机器学习（Machine Learning, ML）专家，但需要愿意把实际问题拆成数据、模型、评估和决策几个环节，并通过 Python、notebook 或 API 完成基本实验。

对本科生和研究生来说，本书提供进入现代预测和 AI 工具链的系统路径；对 MBA 学员来说，本书强调预测如何服务商业判断、资源配置和风险管理；对一线工作人员来说，本书的重点是把日常遇到的销量、需求、客流、价格、风险、产能和服务量问题转化为可以运行、可以评估、可以复盘的预测任务。

学习目标

完成本课程后，你应该能够：

- 区分一般预测（prediction）、时间序列预测（forecasting）、点预测（point forecast）和概率预测（probabilistic forecast）。
- 识别一个业务问题中的目标变量（target variable）、预测步长（forecast horizon）、信息集（information set）和评估指标（evaluation metric）。
- 使用 Python、notebook 和 API 构造基础预测流程。
- 解释大语言模型、Transformer 和时间序列基座模型如何进入预测任务。
- 通过留后法（holdout validation）和滚动窗口（rolling window）评估预测质量。

- 将预测误差（forecast error）、不确定性（uncertainty）和业务成本（business cost）联系起来。

课程结构

课程概览放在四个部分之前，帮助读者理解课程目标、学习路径、材料来源和评价方式。全书正文分为四部分：预测基础与商业决策，大模型、API 与预测评估，时间序列基座模型，实践附录。

第一部分是预测基础与商业决策。C01 介绍 forecasting 与 prediction 的区别，讨论可预测性、时间序列基本特征、预测不确定性和现代预测系统的组成。C02 从时间序列特征、特征工程和自动化学习出发，说明 AI 如何扩展传统预测方法。C03 把预测放入企业管理流程中，强调预测、决策、成本、平台化和组织协作之间的关系。

第二部分是大模型、API 与预测评估。C04 介绍 LLM 的基本能力、token、embedding、prompt、API，以及为什么序列建模思想可以连接语言模型和时间序列预测。C05 展示如何把结构化时间序列写入 prompt，调用 OpenAI 兼容 API，解析 JSON 预测结果，并进行基本质量检查。C06 从点预测和概率预测出发，解释预测误差、评价指标、未来真实值不可见的问题，以及时间序列交叉验证。C07 使用航空乘客数据和 TimeGPT 示例，比较留后法、滚动窗口、不同预测步长和多类误差指标。C08 讨论金融市场中的 AI 交易、领域知识、反身性、企业文本预测、风险控制和监管问题。

第三部分是时间序列基座模型。C09 解释注意力机制、Query-Key-Value、Transformer 的计算逻辑，以及通用时间序列预测模型的共同范式。C10 介绍分解、多尺度建模、TimeMixer、Time-LLM、Prompt-as-Prefix 和时间序列基座模型的适用边界。C11 解释 No Free Lunch、组合预测、课堂手工 MoE 练习、gating、Top-K routing 和稀疏激活，说明为什么预测系统需要让不同专家处理不同序列。C12 讲解 Teacher Model、Student Model、Knowledge Distillation、模型压缩、领域迁移、Chronos 类模型蒸馏、嵌入式 AI 和个人化小模型的预测应用。C13 以 M5 子集和 TimeGPT notebook 为例，讲解间歇性需求、`log1p` 变换、经典基线、MAE 评估和外生变量。

第四部分是实践附录。C14 说明如何申请 PKU 集群账号、使用 Web 界面和 JupyterLab、通过 SSH/OTP 登录、传输文件，以及在集群中安装 Python 包。C15 覆盖课程需要的 Python 基础，包括表达式、变量、字符串、列表、内置函数、模块、函数定义和第三方包安装。C16 介绍终端、SSH、路径、文件操作、编辑器、重定向、管道、上传下载和在服务器上运行 Python 的基本命令。

学完本书后，学生应能够将业务问题转化为可评估的预测任务；使用统计模型、AI 模型和大语言模型构造预测流程；用留后法和滚动窗口评价模型，而不是只展示单次预测；理解预测误差、不确定性和业务决策之间的关系；并能在本地或集群环境中运行课程 notebook 与示例代码。

学习节奏

课程按十六次课组织。每次课通常包含概念讲解、课堂讨论和一个小型实践练习。练习的目标不是增加机械负担，而是让你在课堂或课后用一小时左右把当次内容跑通：打开 notebook、确认数据、调用模型或 API、检查输出，并把预测结果解释成业务语言。

课堂讨论也属于学习的一部分。预测不是只背模型名称，而是要把问题说清楚：哪个未来需要提前判断？哪些信息可用？误差会影响什么行动？你可以从天气、出行、库存、转化率、金融风险、云资源或家庭产业中找例子，练习把生活和行业问题翻译成预测任务。

如何学习

建议每章按四步学习。

1. 阅读章节正文，先抓住概念和业务问题。
2. 运行对应 notebook 或示例代码，确认输入、输出和环境设置。
3. 回到章节练习，把模型结果解释成业务语言。
4. 记录评估结果，包括数据切分、预测步长、指标和失败案例。

学习预测时不要只追求 “模型看起来高级”。任何模型都要回答三个问题：它看到了什么数据，它预测了什么未来，它如何证明自己比简单基准更好。后续章节还会反复强调，多个预测者的组合经常比盲目寻找单一 “最好模型” 更稳健 [Wang et al., 2023]。

代码与环境

课程代码主要在 notebook、Python 脚本和命令行中运行。推荐工作方式如下：

- 用 Git 管理课程材料和自己的修改。
- 用独立 Python 环境安装依赖。
- 把 API key 放在环境变量中，不要写入 notebook。
- 对每次实验记录数据版本、模型参数和评估指标。
- 在本地资源不足时使用 PKU 集群或其他计算平台。

如果你还不熟悉命令行，先读 C16；如果你不熟悉 Python，先读 C15；如果你需要使用学校集群，先读 C14。

评价方式

预测任务的评价应包括模型表现和报告质量。一个合格的预测报告至少要说明：

- 数据来源和清洗方式。
- 预测目标、频率和步长。
- 使用的模型或 API。
- 训练集（training set）、测试集（test set）或滚动窗口设计。
- 至少一个简单基准模型（simple baseline model）。
- 误差指标和业务解释。
- 模型局限与下一步改进。

预测结果不是孤立数字，而是一个决策输入。报告中应明确说明误差可能造成什么后果，以及哪些场景下不应直接使用该预测。

学术诚信与安全

使用 AI 工具时，应保留自己的判断和说明。不要把模型输出直接当作事实，也不要提交无法复现的结果。

涉及账号、密码、API key 和企业数据时，应遵守以下原则：

- 不在仓库、notebook、截图或作业正文中暴露密钥。
- 不上传敏感企业数据到未经批准的外部模型服务。
- 对模型生成的引用、数据和结论进行核验。
- 在报告中说明 AI 工具参与了哪些步骤。

本书材料来源

本书由李丰老师在北京大学讲授的《人工智能预测》课程视频、PPT、notebook、课堂示例和人工整理内容合并而成。当前版本是第一轮课程书整理稿，后续仍可继续补充图表、代码输出、案例数据和作业说明。L00-L11 的段落化课堂转写已经用于补充课程节奏、预测概念、企业平台案例、评估边界、自动化特征工程、TimeGPT 课堂演示、大语言模型概念、API 工作流、概率预测、点预测误差、滚动评估、复苏预测、金融 AI 交易案例、现代预测平台、Transformer 机制、PyTorch 练习、Chronos-2 本地时间序列基座模型、长表多序列预测、协变量需求预测、个人 AI 算力、Autoformer、TimeMixer、多尺度分解、时间序列 token、No Free Lunch、组合预测、MoE 路由、模型蒸馏、Teacher-Student 学习、嵌入式 AI 和课堂环境排错说明。

第 1 章

预测基础

预测不是占卜，也不是对未来随口给出一个判断。本课程中的预测指的是：在已有数据、明确目标和可检验方法（testable method）的基础上，对未来状态做出可量化、可评估的估计。一个预测任务至少要回答三个问题：预测什么未来，使用什么信息，以及如何检验它是否可靠；这也是现代预测教材和综述反复强调的基本框架 [Hyndman and Athanasopoulos, 2021, Petropoulos et al., 2022]。

预测的难点在于未来尚未发生，但商业决策必须提前做出。因此，预测的价值不只在于“猜中一个数字”，而在于帮助我们更早、更稳健地配置资源、控制风险和比较方案。

1.1 学习目标

完成本章后，你应该能够：

- 区分 forecasting 和 prediction 的含义。
- 说明一个场景是否可预测，取决于哪些数据和机制条件。
- 识别时间序列预测中的趋势、季节性、自相关和不确定性。
- 解释为什么好的预测必须在大量样本和多期未来中接受检验。
- 用业务语言说明预测误差如何影响后续决策。

1.2 从占卜到预测

历史上，人们一直试图知道未来。从巴比伦人的肝卜，到二十四节气对季节和气候的周期性总结，人类很早就意识到：未来并不是完全无规律的。有些现象几乎无法提前判断，例如明天是否中彩票；有些现象则存在稳定结构，例如零售销量的节日高峰、用电量的季节变化、旅游需求的恢复节奏。

巴比伦肝卜可以看作早期人类试图寻找关联的例子：天气、牲畜、疾病和虫害之间也许存在某种经验联系，但这种联系没有清晰机制、统一数据和可复现评价。它更接近占卜或解释性叙事，而不是现代意义上的预测。

二十四节气则更接近现代预测的雏形。它来自长期天象和气候观察，能够给出季节变化的点估计（point estimate），也承认“最多相差一两天”这样的不确定性。它不是完全精确的未来知识，而是基于重复规律形成的可用预测。

案例：二十四节气作为早期时间序列预测

中国二十四节气可以看作人类一个非常早期的时间序列预测系统。它要预测的不是某个抽象概念，而是季节和农事活动何时转换：什么时候适合播种，什么时候需要浇水，什么时候进入秋收，什么时候大寒小寒到来。

这个系统的输入不是现代传感器数据，而是长期积累的天象、日照、气候和农事经验。古人并不知道现代天文学和气象学的完整机制，但他们能够反复观察“每天和每天有什么变化”“气候如何随时间移动”，并把这些规律整理成可传承的节令体系。

从预测形式看，二十四节气同时包含两个关键要素。第一，它给出点预测，例如“上半年六二十一，下半年八二十三”，让人们知道节令大致落在哪些日期。第二，它给出不确定性说明，也就是“每月最多相差一两天”。这已经不是一句模糊判断，而是一个带误差范围的时间预测。

这个案例的启发在于：可用预测不一定等于完全精确。只要规律稳定、误差范围可理解，并且能够支持播种、灌溉、收获和防寒等行动，它就有实际价值。同时，它也提醒我们传统预测的代价很高：这套规律需要几千年观察、校正和传承。现代商业预测不能等待这么久，所以才需要统计模型、自动化特征学习和时间序列基座模型来加速规律发现。

这个例子也说明了传统预测的成本：规律需要在很长时间里被观察、校正和传承。农业社会可以用上千年的经验稳定节气规律；现代商业没有这种奢侈。一个电商部门可能每天要预测数以万计的商品、地区和仓库组合，很多商品刚上市还没有足够历史数据。如果仍然依赖少数专家慢慢观察，每个品类都手工总结规律，预测系统很快会被规模压垮。

现代预测和占卜的根本区别在于可检验性（testability）。预测必须说明使用了什么信息、如何生成结果、如何评价误差，以及下一次是否可以用同样流程复现。没有这些条件，预测就只是判断或叙事。彩票这类几乎独立随机的事件，即使有人画出复杂图形，也不等于具备可预测性。

1.3　Forecasting 与 Prediction

`Prediction` 是一个更宽泛的概念，指对尚未发生或尚未观察到的事件做判断。`Forecasting` 是其中更严格的一类：它基于历史数据和已知模式，对未来事件做出量化估计。可以这样理解：

- 所有 forecasting 都是 prediction。
- 不是所有 prediction 都是 forecasting。
- Forecasting 通常要求明确的时间范围、目标变量、数据来源和误差评价方式。

例如，“这个产品未来会火”是一个 prediction；“未来 12 个月每月销量分别是多少，并给出预测区间”才更接近 forecasting。

机器学习中的 prediction 常常是横截面问题（cross-sectional problem）。例如，我们

记录一届学生的课堂表现、作业完成情况和其他特征，再预测他们期末成绩是 A、B、C 还是 D。这里的重点是从特征 x 到标签（label）y 的关系。

Forecasting 更强调时间顺序。例如，我们每天记录一个学生的测验成绩，想知道今天的成绩和昨天、前天、过去五周的学习状态是否有关。此时，过去值本身会变成特征：y_{t-1}、y_{t-2} 等历史信息被用来解释 y_t 或预测 y_{t+h}。

因此，forecasting 可以看作一类特殊的 prediction：它仍然是在用信息预测结果，只是信息来自同一对象的历史状态、相邻时间点和外部事件。一般 prediction 中，样本之间常被近似看作独立；forecasting 中，行与行之间有时间依赖（temporal dependence）。今天的状态可能来自昨天的状态、上周的状态，甚至过去几个月的累积影响。

随着数据采集越来越自动化，很多原本看似横截面的数据也会带上时间戳（timestamp）。订单、点击、车辆轨迹、课堂表现、设备日志和 API 调用记录都天然包含时间。是否把它们作为时间序列处理，取决于我们是否关心“过去如何影响现在和未来”。

问题类型	典型输入	典型输出	关键区别
Prediction	学生特征、用户画像、产品属性	成绩类别、是否购买、是否违约	样本之间通常可近似独立
Forecasting	历史观测、滞后值（lagged values）、时间特征、外生变量	未来一期或多期数值	样本按时间排列，前后观测通常相关

1.4　什么决定可预测性

一个问题是否容易预测，通常取决于四个条件。

第一，目标是否存在稳定机制。彩票号码之间近似独立，过去号码对下一期开奖帮助很小；用电量、零售销量和交通流量往往存在季节性、惯性或制度安排，历史信息更可能有用。

彩票例子特别适合提醒我们区分“看起来有图形”和“真的可预测”。研究历史开奖分布、画折线或寻找所谓冷热号，并不会改变下一期开奖的概率。如果只是为了娱乐，可以避开很多人偏好的数字，减少中奖后和别人平分奖金的可能；但这不会提高中奖概率本身。预测必须来自可利用的关联，而不是来自复杂图形带来的心理安慰。

第二，我们是否能观察到关键影响因素。新能源电力需求可能受天气、政策、价格和产业活动影响；保研结果可能受课程成绩、科研经历、排名和综合表现影响。如果这些变量完全缺失，模型只能从历史序列中猜测。

第三，是否有足够的数据支持预测。数据不只是样本数量，还包括频率、质量、覆盖周期和外部变量。一个小学生未来能否考上某所大学，很难在第一天上学时判断；但到了高三，长期成绩、考试排名和学习状态已经形成大量历史信息，预测依据就更充分。

第四，未来与过去是否仍然相关。新冠后出境游恢复这类问题，历史数据的参考价值会下降，因为未来结构已经改变。但这并不意味着完全不能预测：航班运力、酒店预订、手机位置、签证政策和旅游搜索等相关信号，仍可能为恢复节奏提供线索。

这四个条件可以压缩成一句话：预测要求未来和过去之间存在可以学习的联系，并且我们能观察到足够多的相关信息。如果关键机制不可见、样本太少、未来制度发生突变，模型即使在历史上拟合得很好，也可能只是把过去画得很漂亮。

1.5　时间序列的基本特征

时间序列预测关心的是按时间排列的数据。与普通横截面数据相比，它最重要的特点是相邻观测往往不独立。今天的销量、需求或价格，通常与昨天、上周、去年同月有关。

常见特征包括：

- **趋势（trend）**：序列长期上升或下降，例如用户数持续增长。
- **季节性（seasonality）**：固定周期内重复出现的模式，例如节假日销售高峰。
- **自相关（autocorrelation）**：当前值与过去值之间的相关性，例如本月需求受上月需求影响。
- **异常（anomaly）与结构变化（structural change）**：促销、政策、疫情或供应链中断造成的突然变化。

很多特征可以先从图形直观看到，再转化为可计算指标。自动售货机或快递柜中的饮水需求可能有夏季高、冬季低的季节性，也可能有上午、下午和晚上不同的小时周期。用电量可能同时包含季度、月度、周度和日内周期。趋势可以通过移动平均（moving average）等简单方法提取：固定一个窗口，逐步向前滑动并计算局部平均，就能看到比原始波动更平滑的长期方向。移动平均不同于对所有历史值求一个总平均；总平均只给出一条水平线，移动平均则保留了序列随时间变化的方向。

季节性也不只是“每年重复一次”。企业用电可能有季度周期、月末冲业绩带来的月度周期、周五提交任务带来的周周期，以及白天和夜晚的日内周期。英国足球比赛结束后，大量家庭同时烧水喝茶，会给电网带来短时负荷冲击；类似地，如果大量用户同时关灯再开灯，看起来是节能活动，实际上可能对电网造成突发压力。预测这些场景时，模型不仅要知道有周期，还要知道周期发生在哪个时间尺度、会不会被群体行为放大。

“昨天和今天的学习状态”可以用来解释滞后特征：如果今天的表现和昨天、前天、过去五周都有关系，那么过去值就可以被重新组织成预测特征。这也是时间序列预测与一般回归问题的一个重要连接点。

用符号写就是：模型可以把 y_{t-1}、y_{t-2} 乃至过去多期状态重新组织成特征，用历史依赖解释当前和未来。时间序列预测并不是凭空预测未来，而是在已有信息集中寻找可重复的动态关系。

股票收益率也是这个思想的例子。现实市场中很难找到一个单独变量解释价格波动，新闻、交易行为、情绪和制度约束都会混在一起。与其假设已经找到了全部原因，不如先把价格或收益看成时间相依的数据，研究波动、自相关、异常和外部冲击如何沿时间传播。时间序列方法提供的是一种从时间维度理解复杂系统的方式。

1.6　什么是好的预测

好的预测不是偶尔一次猜中，而是在可重复的评价中稳定表现良好。只说“我们昨天预测今天股票会涨，而且真的涨了”，证据很弱；如果一个策略在许多时间段、许多资产和许多市场环境中都表现稳定，才更接近可用预测。

我们在这里强调几个朴素但重要的原则：

- 预测要在大量数据上表现优异。
- 多条时间序列之间可以交互学习，提升整体预测精度。
- 历史数据不足时，需要主动创造或补充可用数据。
- 速度是现代商业预测的重要竞争力。
- 长期预测通常比短期预测更难，因为误差会逐步累积。

多序列交互学习（multi-series learning）在商业中尤其重要。一个全新产品没有自己的历史销量，但可以参考相似产品、竞品、配套产品和共同购买关系。一个区域需求不稳定，也可以借助其他地区、其他品类和外部变量。预测不一定只盯着一条序列，而是要在更大的数据集合中寻找相似结构。

例如尚未上市的 iPhone 18 没有自身历史销量，但并不等于完全没有预测依据。系统可以参考上一代 iPhone、同生态产品、竞品发布节奏、老用户换机周期、配件购买关系和相似地区的消费结构。预测系统要学习的不是“这个产品过去卖了多少”，而是“与它相似、竞争或共生的对象过去如何变化”。这也是现代多序列预测与传统单序列外推的重要区别。

速度同样是预测质量的一部分。电商平台可能需要每天为成千上万种商品和地区生成补货预测；如果模型太慢，即使误差较低，也可能错过补货、排产或调度窗口。阿里巴巴 3C 预测就是典型场景：系统可能要在夜间为几万类核心商品预测第二天各地需求，例如海淀区某类联想笔记本明天需要多少库存。低估会造成缺货和用户体验下降，高估会占用现金流和仓储空间。因此，评价预测时不要只问“这一次准不准”，还要问：是否在多个时间段、多个序列、多个预测步长上稳定？是否比简单基准模型更好？是否能及时生成，供业务系统使用？

1.7　预测与不确定性

未来永远无法完全知道，所以预测不应只给一个点估计。更完整的预测应当讨论不确定性，例如预测区间（prediction interval）、情景分析（scenario analysis）或概率分布（probability distribution）；概率预测的评价也需要使用鼓励诚实报告分布的评分规则（scoring rule）[Gneiting and Raftery, 2007]。二十四节气给出日期规律，同时承认前后可能相差一两天；现代预测也应明确误差范围，而不是把一个数字包装成确定事实。

五个专家可能给出五种旅游恢复预测，问题不是简单选择谁“看起来更专业”，而是比较他们的假设、数据、误差记录和决策后果。

在商业应用中，不确定性常常比点预测本身更重要。库存不足和库存过剩的成本不同；产能闲置和产能短缺的风险不同；金融预测中高估和低估可能导致完全不同的交易行为。

预测步长越远，不确定性通常越大。一个孩子今天考了 90 分，预测明天或下周的学习状态还有一定依据；要在一年级时预测十二年后的高考结果，就几乎没有足够信息。冰川变化、气候风险和长期公共健康预测也是类似问题：远期结果不只由历史趋势决定，还会被排放路径、政策选择、技术变化和社会行为改变。青少年近视项目也说明了这一点：如果把 1983-2023 年的近视率趋势直接外推到 2050 年，可能得到“几乎所有高中生都会近视”的荒谬结论。历史趋势本身可能是真的，但社会会调整教育、户外活动和用眼习惯，远期预测必须把机制变化和情景不确定性纳入考虑。

1.8　现代时间序列预测

现代时间序列预测是统计思维、计算机工程和人工智能的结合。统计思维帮助我们定义目标、误差和不确定性；工程能力帮助我们处理大规模数据、自动化建模和部署；人工智能方法让模型能够从复杂、多源、高维数据中学习结构。

亚马逊和 Google 的预测实践说明，预测已经从单个模型问题变成平台问题。企业需要同时处理大量产品、地区、资源和时间粒度。模型不只是算法，还包括数据管道、特征工程、自动评估、人工干预和业务系统集成。

这也是本课程从传统预测转向 AI 预测的原因。当数据规模和模式复杂度上升时，单靠人眼观察趋势或专家经验已经不够。AI 的价值不是替代所有统计思想，而是帮助我们在更大规模、更复杂的数据中自动发现特征、比较模型、补充外部信息，并把预测更快地接入业务流程。

所以，预测既是 science，也是 art。Science 体现在可检验的数据、模型、误差和流程；art 体现在对假设、场景、外部信息和决策后果的判断。好的预测不是声音更大、模型更复杂或专家头衔更响，而是能用证据说明它为什么可信、什么时候会失效、怎样帮助行动。

1.9　小结

预测基础可以归纳为一句话：用过去和现在的信息，构造一个可以被检验的未来估计。学习预测时，不要急着选择最复杂的模型。先明确目标变量、预测步长、可用数据、可预测性来源和误差评价方式，再讨论模型。

1.10　练习

1. 找一个你熟悉的业务场景，说明它更像 prediction 还是 forecasting。
2. 用学生期末成绩和每日测验成绩两个例子，比较横截面 prediction 与时间序列 forecasting。

3. 对自动售货机饮水需求或用电量序列，指出可能存在的趋势、季节性、自相关和异常因素。

4. 设计一个评价方案，说明如何判断一个 12 期预测是否比简单基准更好。

5. 解释为什么彩票号码预测和库存需求预测的可预测性不同。

6. 讨论一个预测很准但决策仍然失败的例子，解释失败可能来自哪里。

第 2 章

人工智能预测

人工智能预测的核心不是把传统预测问题简单交给一个更大的模型，而是改变预测系统获取信息、学习特征、比较模型和服务决策的方式。传统方法往往依赖人工设定趋势、季节性、滞后项、节假日和外生变量；AI 方法则希望从大规模、多来源、高维度的数据中自动发现可预测结构，并把这些结构转化为稳定可用的预测表示。大语言模型对预测领域的影响，也主要体现在这种信息组织、自动化和流程重构上 [Makridakis et al., 2023]。

在本课程中，AI 预测仍然首先是时间序列 forecasting，而不是任意机器学习 prediction。用学生画像预测期末成绩属于一般 prediction；用历史销量、价格、节假日、促销和外部事件预测未来销量，则是更严格的 forecasting。这个区别能帮助我们避免把任何模型输出都称为预测系统。

AI 进入预测以后，真正要解决的是四类瓶颈：序列数量太多，人工无法逐条建模；关键特征隐藏在业务系统和外部环境中，肉眼不容易发现；疫情、政策和供应链冲击会打断历史规律；预测结果还必须进入库存、产能、价格和政策等后续决策流程。

我们把这个问题概括为一个现实压力：现代社会的数据已经多到不能靠手工消化。订单、点击、图片、文本、定位、航班、天气、设备日志和财务记录都在持续生成。传统预测不是没有用，而是在数据规模、变量种类和更新频率上遇到了边界。AI 预测要回答的不是“能不能把一个模型叫作智能”，而是能不能在这种数据规模下更快地发现结构、形成预测并服务决策。

我们还要补充一个重要提醒：AI 不保证永远超过人类专家或传统统计模型。它更像一种会逐渐基础设施化的预测起点，可以快速给出候选判断、候选代码和候选曲线，但这些输出最终仍要被现实检验。一个严肃的 AI 预测系统，不能只问“模型能不能给答案”，还要问“这个答案是否在统一评价规则下超过了专家、朴素基准或传统模型”。

从企业角度看，AI 预测会逐渐像数据一样成为常规输入。管理者不一定要把 AI 输出当成最终答案，但可以把它当作 reference：当新人、专家、统计模型和 AI 给出不同预测时，系统要记录这些候选判断，并用统一评价规则决定谁在什么场景下更可信。这样 AI 不是替代所有判断，而是进入预测平台，成为可比较、可追踪、可复用的一类信息资产。

在这个基础上，我们进一步强调：时间序列基座模型让预测更像基础设施，而不只是一次性建模项目。预训练阶段已经付出了主要学习成本，推理阶段则把新的时间序列送入模型，快速得到候选预测；TimeGPT 这类模型正是这种思路的代表 [Garza et al., 2024]。健康设备是一个直观例子：手表持续记录血氧和心率，如果模型能预测未来几分钟可能出现异常，就可以提前提醒老人坐下或求助。这里的价值不一定来自

benchmark 上多提高 1% 的精度，而是来自预测能否及时进入真实行动。

如果继续把基础设施问题推进到本地 AI 算力，就会看到另一层约束。云端大模型使用方便，但医院、企业、教学和个人健康数据常常不能随意上传。个人或小团队如果拥有本地 AI 计算设备，就可以在自己的环境里运行开源模型、做领域微调、保存敏感数据，并把预测能力嵌入团队或家庭流程。AI 预测的基础设施因此不只包括算法和 API，也包括数据是否留在本地、算力是否可持续、模型是否能在小团队中被反复使用。

只要业务记录带有时间戳，它就可能被重新组织成时间序列或时间序列特征。一次点击、一次预约、一次车辆等待、一张商品图片的上传、一次 API 调用，本身看起来都不是传统时间序列；但当它们持续发生并与目标变量相连时，就会变成预测系统的输入。AI 预测的一个核心能力，是把这些分散痕迹转化为可学习信号。

2.1 学习目标

完成本章后，你应该能够：

- 解释时间序列特征（time-series features）在 AI 预测中的作用。
- 区分人工特征工程（manual feature engineering）和自动化特征学习（automated feature learning）。
- 说明趋势、季节性、自相关、外生变量和代理变量如何影响预测结果。
- 解释时间序列特征矩阵如何帮助模型匹配。
- 解释泛化能力不足、过拟合、欠拟合和结构突变在预测中的表现。
- 为一个业务预测任务设计基本的 AI 预测流程。

2.2 时间序列特征

时间序列特征是从历史数据和相关信息中提取出来、能够反映规律性和动态变化的属性。模型能学到什么，往往取决于输入中是否包含有用特征。

常见特征包括：

- **趋势特征（trend feature）**：序列是否长期上升或下降，例如某个行业的销售额持续增长，或某类商品需求逐步下降。
- **季节性特征（seasonal feature）**：固定周期内重复出现的模式，例如夏季饮料需求高于冬季、春节期间电商消费结构变化、用电量存在季度和日内周期。
- **自相关特征（autocorrelation feature）**：当前时刻与过去时刻之间的相关性，例如本月需求受到上月需求影响，或者金融收益率与前几期收益存在弱相关。
- **外生特征（exogenous feature）**：价格、天气、促销、政策、宏观经济、搜索指数、航班运力和地缘政治等外部信息。

- **代理特征（proxy feature）**：不能直接观察目标机制时，用夜间灯光、搜索行为、客流、定位、图片、声音或视频等信号近似刻画真实活动。

这些特征有些很容易看见。自动售货机中的饮料销量可能有夏季高、冬季低的季节性，也可能有上午、下午和晚上不同的日内周期。用电量可能同时包含季度、月度、周度和日内周期。趋势可以通过移动平均等简单方法提取：固定一个窗口，逐步向前滑动并计算局部平均，就能看到比原始波动更平滑的长期方向。

也有些特征不容易直接从图上看出。阿里巴巴某类 3C 产品的日销量如果只看黑白曲线，很多尖峰和下跌都像噪声；一旦把 618、双十一、春节、预售、优惠券和平台活动标出来，许多异常波动就变成了可解释的业务事件。预测像破案：历史序列是基本证据，特征是关键线索。没有线索，模型只能猜测；线索越接近真实机制，预测越可能稳定。

还有一些特征来自行为轨迹，而不是传统表格。地图软件里的拥堵和红灯等待时间，可能来自手机导航轨迹、速度变化和周边车流，而不是来自每个路口的官方传感器。企业 demo 预约和成单率预测，也会使用客户预约、跟进、地区、渠道和转化时间分布。这类数据说明，AI 预测经常要把业务过程中的 “痕迹” 转化为可学习信号。

2.3　特征为什么重要

特征决定模型能否区分 “噪声” 和 “信号”。如果一个商品在 双十一前后出现销量尖峰，但模型不知道那一天是促销期，它可能把尖峰当作随机异常；如果模型知道促销、折扣、预售和购物车行为，尖峰就可以成为未来促销预测的有效依据。

代理变量也体现了特征的重要性。夜间灯光强度可以近似反映地区经济活动，旅游搜索指数可以反映潜在出行意愿，航班运力可以反映出境游恢复条件，声音和视频数据可以补充文本中没有表达出来的沟通信号。这些信息不一定来自目标变量本身，却可能提前反映需求、风险或情绪变化。

但是代理特征不能盲目信任。一个代理变量在某个时期有效，不代表永远有效。夜间灯光与经济活动的关系可能受到产业结构、节能政策或生产方式变化影响；搜索指数与真实消费之间也可能被舆论事件、平台推荐或短期恐慌放大。AI 预测需要自动发现特征，也需要持续检查特征是否仍然代表同一件事。

航班延误是另一个例子。传统模型容易想到天气、航空公司、航线距离和季节性；但在大型枢纽机场，一个小的前序延误可能沿着航班网络传递，造成后续机场持续拥堵。这种 “传染效应” 不是单条时间序列内部很容易看出的结构，而是跨航班、跨机场网络中的动态模式。AI 方法的价值之一，就是帮助我们在复杂系统中发现这类人工难以穷尽的交互特征。

2.4　人工特征工程与自动化特征学习

人工特征工程依赖专家经验和大量试错。差分（differencing）、移动平均、傅里叶项（Fourier terms）、节假日哑变量（holiday dummy variables）、自相关函数（Autocorrelation

Function, ACF）和统计特征（statistical features）都很有用，也具有较强解释性。很多经典工具会把一条时间序列转换成一组可计算特征，再根据这些特征判断更适合使用哪类模型。

问题在于，商业预测很少只有一条序列。电商平台可能要同时预测上百万个商品、地区、仓库和时间粒度。少量专家可以解释几个重点品类，却无法持续手工维护所有产品的促销规则、生命周期、替代关系和区域差异。人工特征工程在规模上会遇到成本边界。

案例：阿里巴巴电商预测为什么需要 AI 特征学习

阿里巴巴这类电商平台的预测任务，不能只理解成“预测某个商品明天卖多少”。更真实的问题是：某个品类、某个地区、某个仓库、某个时间粒度下，明天甚至未来几周需要准备多少库存。例如 3C 部门可能关心海淀区明天会有多少用户购买某类联想电脑；这个预测会直接影响补货、仓储、配送时效、现金流和用户体验。如果只看某个 3C 品类的日销量曲线，很多波动会像噪声：某一天突然上升，某一天突然下降，肉眼很难判断原因。把 618、双十一、春节、预售、优惠券、平台补贴、购物车加购和价格变化标出来以后，许多“异常”就会变成可解释的业务事件。也就是说，销量曲线本身只是结果，真正有预测价值的信息常常藏在平台运营、用户行为和促销机制里。

这个例子也说明人工特征工程的边界。一个懂 3C 的专家可以解释几个重点商品，七八个人的团队也许能维护少量核心品类；但当平台面对数万甚至更多商品、多个地区、多个仓库和频繁变化的活动节奏时，人工逐条解释“为什么这里卖得多、那里卖得少”就不可持续。更合理的方式，是把促销、价格、节假日、地区、点击、加购、库存、物流和历史销售等信号沉淀为自动化特征，让模型在大量序列中学习相似结构。

因此，阿里巴巴电商预测的关键不是把一个专家经验复制成几千份，而是把专家能够识别的业务逻辑转化为平台能力：数据自动接入，特征自动生成，模型批量比较，预测结果进入补货和调度系统。这样，AI 预测才真正从单个模型变成企业基础设施。

我们用创业公司作一个简单成本估算：如果一家预测公司拿到一笔启动资金，却必须为每个行业请高水平专家、再配一个十人团队手工做特征，资金很快会被人力成本吃掉。Amazon、京东、阿里这类平台也不能把一个十人预测团队简单扩成几百人、几千人来追赶业务规模。自动化特征学习的管理价值，正是在于把专家经验沉淀成可复用流程，让系统可以在海量序列上稳定运行。

维度	人工特征工程	自动化特征学习
主要来源	专家经验、统计定义、业务规则	大规模历史数据、多源数据、模型表示
优点	可解释、容易检查、适合明确机制	可扩展、能捕捉非线性和高维交互
风险	成本高、覆盖有限、依赖少数专家	可能难解释、可能学习伪相关
适用场景	关键业务、数据较少、机制清楚	多序列、大规模、模式复杂、外部信号丰富

自动化特征学习并不等于取消领域知识。更准确的说法是：领域知识帮助我们定义问题、筛选数据、设定约束和解释结果；AI 模型帮助我们在高维数据中学习人工难以逐一设计的结构。把时间序列转换成图像或其他可学习表示，再用于模型选择和组合，就是自动化特征学习进入预测的一个例子 [Li et al., 2020]。

在 AI 预测流程中，特征可以来自几个层次。第一层是人工可解释特征，如趋势、季节性、促销和滞后项。第二层是自动提取的统计特征，如峰值、波动性、自相关强度和间歇性。第三层是模型学到的表示，例如 embedding、深度网络隐藏状态或时间序列基座模型中的序列表示。好的系统通常不会只依赖其中一层，而是把可解释特征、自动化特征和业务约束结合起来。

图像、文本和自然语言处理（Natural Language Processing, NLP）也会进入预测流程。二手书交易中，图片可以帮助系统识别书的新旧程度，文字识别可以确认书名和版本，历史销售记录可以估计流行度和合理价格。这类信息本来很难被传统时间序列模型直接使用，但 AI 可以把它们转化为结构化特征，再进入销量、定价或库存预测。

2.5　特征矩阵与模型匹配

当平台有几十万条甚至上百万条时间序列时，不能靠直觉逐条选择模型。一个可操作的做法是先为每条序列计算一组特征，形成特征矩阵（feature matrix）：每一行是一条时间序列，每一列是一个特征，例如 `Linearity`、`Curvature`、`ACF`、`Trend`、峰值强度、波动性和季节性强度。

有了特征矩阵，就可以比较不同模型在不同特征区域中的表现。Auto ARIMA（Automatic ARIMA）更擅长某些自回归、差分和季节结构；ETS（Error-Trend-Seasonal）更适合一些平滑趋势和季节性问题；神经网络可能在非线性和复杂交互中更有优势；Naive（朴素法）方法虽然简单，在很多短期、稳定场景中仍是重要基线。模型表现不是抽象地“谁最好”，而是和序列特征有关。

这就是模型匹配（model matching）的思想：先问这条序列是什么形状，再问哪个模型适合这种形状。线性很强的数据不一定需要复杂神经网络；高峰明显、促销强烈或外部冲击频繁的数据，可能需要更多事件特征和非线性模型。AI 预测系统要学习的，不只是预测值本身，还包括“什么样的序列交给什么样的模型或专家”。

MoE 会把这个问题进一步推进。模型匹配不是人工列一张固定规则表就结束，而

是可以变成可学习的路由机制：系统先识别序列形态和业务上下文，再决定调用哪些专家模型，以及各自占多少权重。这样，AI 预测平台学习的不只是 forecast，还包括 forecast 的分工方式。

这一思想也解释了为什么简单基线不能省略。`Naive` 方法用最后一个观测值预测未来，看起来粗糙，却能检验复杂模型是否真的学到了额外信息。如果复杂模型不能稳定超过 Naive、季节性 Naive 或移动平均，就很难说它提供了可用的智能。

2.6　可解释特征与隐藏表示

`tsfeatures` 这类工具代表了一种重要路线：先由专家定义一组可解释时间序列特征，再把它们写成函数自动提取。早期研究曾整理出几十个核心特征，例如线性、曲率、自相关、趋势和差分相关特征。它们计算成本低，解释性强，适合作为自动化预测的起点。

另一类工具如 `catch22` 会从大量候选特征中筛选更紧凑的代表性特征。特征数量并不是越多越好。上千、上万维特征可能改善模型输入，却会让管理者难以理解为什么系统相信某个预测。企业系统通常需要在可解释性（interpretability）和预测性能之间折中：可解释统计特征帮助人理解，深度学习 embedding 或隐藏状态帮助模型捕捉传统特征之外的结构。

因此，AI 预测不是 “人工特征” 与 “深度学习特征” 二选一。更稳健的做法是把可解释特征、自动化统计特征、深度表示和业务约束组合起来，让系统既能发现隐藏规律，也能向人说明主要依据。

2.7　泛化能力

泛化能力（generalization）指模型在未见过的新数据上仍能保持稳定预测性能。预测模型不能只拟合历史，它必须面对未来。

泛化能力不足通常表现为：

- **过拟合（overfitting）**：历史数据拟合很好，未来误差很大。
- **欠拟合（underfitting）**：模型太简单，连趋势和季节性都没有学到。
- **对新情境脆弱**：外部条件变化后，预测效果骤降。
- **特征失效**：过去有效的外生变量或代理变量，在新的制度、技术或行为环境下不再有效。

提升泛化能力的方法包括滚动预测验证、合理的特征选择、简单基准模型、组合预测、概率预测，以及在统计模型和 AI 模型之间做稳健集成。不要只比较训练集误差；更应该用 “过去预测未来” 的方式模拟真实使用场景。

新冠后出境游恢复说明，AI 预测并不只是寻找更复杂的函数。疫情前的旅游需求有明显趋势和季节性，常规模型可以较好外推；疫情冲击后，历史规律被打断，问

题变成了“系统何时恢复、恢复到什么水平、以什么路径恢复”。这时需要先构造没有冲击时的基准预测，再观察恢复窗口中的航班、搜索、政策和客流信号，用恢复系数（recovery coefficient）、分层预测（hierarchical forecasting）、预测调和（forecast reconciliation）和组合预测（forecast combination）把不同信息整合起来。

这个例子说明，泛化能力不是模型名带来的，而是由问题设定、数据窗口、外部信号、验证方式和业务假设共同决定的。越是依赖历史数据的模型，越需要对“未来是否仍像过去”保持警惕。

在恢复预测中，模型之外的假设同样重要。我们通常需要先假设系统会逐渐恢复，再估计恢复的时间、水平和曲线形状。海外活动、交通流量、住宿、航班运力、签证政策、地缘政治和搜索行为不会直接给出答案，但会给出恢复速度的暗示。AI 的作用之一，就是帮助整合这些弱信号，而不是机械地把疫情前趋势向后延长。

泛化也意味着不要把预测理解成单一方法竞赛。我们把预测系统类比为一辆车：稳定长距离运行不只靠发动机，也不只靠轮胎，而是动力、电池、能耗、稳定性和抗疲劳系统共同工作。Forecasting 同样不是纯粹 methodology-driven 的事情，而是统计工具、工程系统和决策支持共同构成的能力。

2.8　组合预测与稳定性

现代时间序列预测很少追求“一个模型打天下”。ARIMA、ETS、树模型、神经网络、时间序列基座模型和人工规则都可能在某些序列上有效，但几乎没有一个模型能在所有特征、所有频率和所有步长上胜出。组合预测或集成方法的直觉是：让多个模型分别捕捉不同维度的信息，再用验证表现、序列特征或业务规则分配权重。

稳定性是企业预测的硬要求。一个模型如果今天预测海淀区某类电脑需求 100 台，明天跳到 500 台，后天又变成 10 台，即使某些单点误差不大，也会给库存、现金流、仓储和调货计划带来冲击。自动化预测平台要追求的不是孤立一次的“看起来很准”，而是在大量序列、多个预测步长和不同业务周期中稳定输出可用结果。

2.9　AI 预测流程

一个完整的 AI 预测流程至少包括七步：

1. 定义预测目标：预测什么变量、预测多远、服务什么决策，明确点预测、区间预测或概率预测需求。
2. 整理目标数据：统一时间粒度，处理缺失值、异常值、多序列层级和数据口径变化。
3. 接入外部信息：加入价格、天气、节假日、促销、搜索、文本、图片、声音、视频和业务事件等可能影响未来的信号。
4. 构造或学习特征：结合人工特征、自动统计特征、深度表示和多序列相似性。

5. 训练和比较模型：同时保留简单基准、统计模型、机器学习模型、深度学习模型和时间序列基座模型候选。
6. 评价和集成结果：使用滚动验证、留出测试、业务指标、组合预测、分层预测和预测调和检查稳定性。
7. 部署与监控：持续检查数据漂移（data drift）、误差变化、特征失效、计算成本和决策效果。

AI 预测最容易被误解的地方，是把模型调用当成完整流程。事实上，模型只是流程中的一个环节。数据质量、特征设计、验证方式、业务反馈和系统监控同样决定预测系统的价值。

亚马逊预测平台的案例强调了规模问题。少量产品可以靠专家团队手工预测；当产品、地区、仓库和时间粒度同时扩张时，预测必须变成平台能力，包括数据接入、特征生成、批量建模、自动评估和业务系统集成。Google Cloud 的产能管理也类似：如果客户未来需要多少机器、在哪个区域部署、需求如何增长都不确定，平台就必须用预测来平衡服务水平、能源成本和闲置资源。

因此，AI 预测系统不应是 “这个场景调一个模型、那个场景换一套脚本” 的碎片化工具。它更像统一平台：能够在大量序列上接入数据、学习特征、比较模型、生成预测、记录误差并把结果交给业务系统。只有这样，预测才可能从研究示例变成企业能力。

2.10　预测与决策的连接

预测本身不产生价值，只有进入决策才产生价值。销量预测可以影响库存；需求预测可以影响排班；价格预测可以影响采购；风险预测可以影响授信；旅游恢复预测可以影响航线、签证、营销和外交政策。

因此，在设计预测任务时要同时回答两个问题：预测误差如何衡量？误差会怎样影响决策？如果高估和低估的成本不同，模型选择和评价指标也应当不同。对电商来说，高估需求可能造成库存积压，低估需求可能造成缺货和现金流损失；对云计算平台来说，高估算力需求会造成机器闲置，低估需求会影响服务可用性。

AI 预测系统还需要向决策者解释 “为什么可以相信”。可信不是因为模型复杂，也不是因为专家声音大，而是因为系统能展示数据来源、验证记录、误差范围、关键假设和业务后果。预测是科学，也是实践中的判断艺术：它既要有可检验的方法，也要能在不确定条件下支持具体行动。

2.11　常见错误

- 把 “模型更复杂” 误认为 “预测一定更好”。
- 把 AI 给出的快速起点当成已经验证过的最终预测。
- 忽略简单基准模型，没有证明 AI 模型带来增量价值。

- 只看点预测，不看预测区间、误差分布或决策损失。
- 用未来信息构造训练特征，造成数据泄露。
- 把某个代理变量过去的相关性当作稳定机制。
- 只依赖少数专家手工特征，无法覆盖大规模多序列场景。
- 忽略序列特征与模型表现之间的匹配关系，默认一个模型适合所有序列。
- 遇到疫情、政策、供应链中断等结构突变时，仍然机械外推历史规律。
- 在一个场景上调参成功后，直接假设能泛化到所有场景。
- 做出预测后没有部署、监控和业务反馈，导致模型无法持续改进。

2.12 练习

1. 为一个零售销量预测任务列出至少五个可能有用的外生特征，并区分哪些是可直接观测特征，哪些是代理特征。
2. 解释为什么 618、双十一和春节变量可能改善电商销量预测，也可能造成过拟合。
3. 为三条不同形状的时间序列列出可能的 Linearity、ACF 和季节性特征，并说明你会优先比较哪些模型。
4. 选择一个你熟悉的业务场景，比较人工特征工程和自动化特征学习各自适合解决的问题。
5. 设计一个新冠后旅游恢复预测方案，说明如何构造无冲击基准、恢复窗口信号和最终预测路径。
6. 找一个你认为有用的代理变量，说明它为什么可能有效，以及在什么情况下会失效。
7. 设计一个滚动验证方案，比较简单基准模型、统计模型和 AI 模型的泛化能力。

第 3 章

商业预测与决策

企业做预测，不是为了得到一个漂亮的数字，而是为了做更好的决策。库存、产能、定价、风险控制、营销投入和供应链安排，都需要在未来尚未发生时提前行动。因此，商业预测（business forecasting）的核心问题是：如何把预测结果嵌入决策流程，从“预测准确”走向“决策有效”。

3.1 学习目标

完成本章后，你应该能够：

- 说明预测准确性和决策有效性之间的区别。
- 描述企业级预测系统的基本组成：数据、模型、算力、平台和业务流程。
- 解释为什么企业通常需要多类预测模型并存。
- 用库存、风险或产能场景说明预测误差如何转化为业务成本。
- 设计一个从预测到行动的闭环评价框架。

3.2 从描述到决策

传统预测常常停留在“描述与拟合”：给定历史数据，选择模型，生成未来值，计算误差。现代商业预测则更强调“决策驱动（decision-driven）与智能优化（intelligent optimization）”。预测结果必须进入后续行动，例如补货、排产、调价、投放或风控。

这意味着预测任务不能只由数据科学团队单独定义。业务部门需要说明决策目标、约束条件和错误成本；技术团队需要提供数据管道、模型服务和自动化评估；管理层需要决定预测结果在流程中拥有多大权重。判断调整和供应链预测研究也说明，预测流程如果脱离业务约束和执行机制，误差改善很难自然转化为决策改善 [Fildes et al., 2009]。

很多学生学习预测时容易把注意力放在模型本身：选择哪个算法、调什么参数、误差指标是否更低。真正进入企业场景后，模型只是解决方案中的一个组件。预测要发挥作用，必须被放到一个能够持续接入数据、生成特征、调用模型、展示结果并触发行动的平台中。

因此，本章讨论的商业预测不是“训练一个模型”，而是“设计一个可运行的预

测-决策系统”。一个管理者如果只看模型层，就很难判断上游数据是否可靠、下游动作是否可执行，也很难在模型失效时及时干预。

3.3　预测-决策一体化

预测-决策一体化（forecast-decision integration）的基本逻辑是：

1. 根据业务问题定义预测目标。
2. 用历史数据和外部信息生成预测。
3. 将预测输入到决策规则或优化模型（optimization model）中。
4. 执行动作并记录结果。
5. 同时评价预测误差和决策收益。

例如，库存管理中，预测销量只是第一步。真正的决策是订多少货、放在哪个仓、何时补货。如果预测低估，可能缺货；如果预测高估，可能积压。一个模型即使 MAE 较低，也可能因为系统性低估高峰需求而造成严重业务损失。

决策也会反过来驱动预测。假设管理目标是今年粮食增产 5%，预测系统就不能只在年底报告实际产量，而要在播种后、汛期、病虫害发生时持续监控不同地区和作物的产量风险。预测一旦显示某地可能减产，决策者就要考虑补种、调拨、灭虫或调整作物结构。此时预测不是旁观的报表，而是干预行动的看板。

我们把这点概括为“预测驱动的决策”。管理者的经验也像一种模型，只不过存在人的脑子里，难迁移、难验证，也难稳定复用。企业如果能把历史时间序列、专家判断、行业知识和模型评估结构化沉淀下来，就能让预测从个人经验转化为组织能力。预测值本身也应被看成企业每天生产的数据之一，反过来支持库存、现金流、产能、人力和战略安排。

更完整的企业流程可以写成：数据输入（data input）-> 特征层（feature layer）-> 模型层（model layer）-> 服务层（serving layer）-> 决策层（decision layer）。

这个链条中任何一环薄弱，都会削弱预测价值。数据输入不稳定，模型会学到错误信号；特征层缺失，模型只能看到原始序列；模型层缺少基准和比较，复杂模型未必带来增量；服务层没有看板、API 或监控，预测无法进入业务系统；决策层没有审批和覆盖规则，预测错误可能直接放大为业务损失。

3.4　企业预测平台的五层架构

企业级预测平台通常可以拆成五层。

层级	核心问题	典型内容
数据层	预测系统看到了什么？	业务订单、价格、库存、天气、宏观变量、实时 API、外部事件
特征层	原始数据如何变成模型可用信息？	滞后项、移动平均、节假日、促销、embedding、特征库（feature store）
模型层	用什么方法生成预测？	统计模型、机器学习模型、深度学习模型、时间序列基座模型、人工规则
服务层	预测如何交付给业务？	批量任务、API、看板（dashboard）、报告、告警、模型监控（model monitoring）
决策层	谁根据预测采取行动？	自动补货、人工审批、copilot 建议、风控限额、异常覆盖

这五层说明，现代预测已经从单个算法问题变成工程问题。单机、单模型仍然可以处理小任务，但面对上千条时间序列、实时市场数据或跨区域需求预测时，企业需要自动化和平台化能力。预测平台不仅要算得准，还要算得快、接得上数据、解释得清楚，并能把结果送到正确的人或系统面前。

在 AI 场景中，服务层和决策层会变得更重要。系统可能不只是给出一个数字，而是像 copilot 一样建议“买多少”“补多少”“是否停止交易”“是否需要人工复核”。人的角色也随之变化：不是手工计算每个预测，而是监督系统、批准关键动作、设置边界，并在异常情况下接管。

这五层也说明，模型不是平台的全部。数据层要持续接入可靠数据源；特征层要把原始数据转成可复用的特征或 embedding，并保存到 feature store；模型层要同时容纳统计模型、机器学习模型、深度学习模型和时间序列基座模型；服务层要把预测做成 dashboard、API 或小程序；决策层才把预测转成买卖、补货、排产或审批。只学一个模型算法，通常只是在模型层工作；真正的平台设计者要能看懂整条链路。

3.5　企业为什么需要多类模型

几乎没有企业只依赖一个预测模型。企业级预测通常包含多层模型：

- 简单统计模型：用于稳定、低成本、可解释的场景。
- 机器学习模型：用于多特征、多序列、非线性问题。
- 深度学习模型：用于大规模序列和复杂依赖结构。
- 时间序列基座模型：用于跨领域迁移、快速适配和少样本场景。
- 人工判断与业务规则：用于特殊事件、政策变化和异常干预。

统计模型并没有过时。对于简单、稳定、数据量有限的任务，统计模型可能仍是最可靠的主力。AI 模型的价值在于处理更大规模、更复杂结构和更自动化的预测需求。

模型选择取决于数据频率、序列规模、业务稳定性和决策成本。一个成熟门店的周度补货，可能用季节性朴素模型或指数平滑就足够；一个高频金融看板，可能更依赖低延迟特征和树模型；一个新产品或长尾 SKU，可能需要时间序列基座模型从相似序列中迁移信息。平台的价值正在于允许多类模型共存，并用统一评估和监控机制比较它们。

3.6　平台化预测

当企业面对成千上万条时间序列时，预测就不再是单个 notebook 或单个模型的问题，而是平台问题。平台化预测通常需要：

- 自动接入数据；
- 处理缺失值、异常值和时间粒度；
- 自动生成和管理特征；
- 批量训练或调用模型；
- 分布式计算和 GPU/云资源；
- 自动生成评估报告；
- 将结果写回业务系统或展示在看板中；
- 监控误差、漂移和服务成本。

即时零售可以展示平台化预测的数量级：100 个小区、每天 32 个十五分钟时段、200 个商品，就已经是 64 万条需求序列；如果每条序列比较 10 个候选模型，就是 640 万次模型计算。更重要的是，这类任务可能每 15 分钟刷新一次。这个例子说明，企业预测不能依赖“一个专家手工维护一条序列”，而要依赖批量建模、任务调度、共享特征、统一评估和自动监控。

亚马逊和 Google 的预测实践说明，大型企业会把预测作为云服务或内部平台能力。用户上传业务数据，平台补充外部信息、自动训练模型、生成预测并按计算、存储或调用资源收费。预测服务的竞争力不仅来自算法，还来自数据、算力、工程和业务集成。

这里要区分两类“云”。一种只是把机器搬到云端，本质上仍是远程主机；另一种是真正的平台服务，能够把数据接入、特征生成、模型训练、评估、部署、监控和看板连接起来。后者才是现代企业预测需要的能力。

现代预测平台还需要 MLOps（Machine Learning Operations）思维。生产系统不是把数据倒进 ARIMA 或某个深度模型后一次性输出结果，而是不断检查特征是否仍然有效、模型是否稳定、预测结果是否漂移、指标是否可靠。新数据进入后，系统可能

需要自动重新训练、重新评估、重新发布，并把每一步记录下来。否则，平台看起来有很多模型，实际仍然依赖人工临时维护。

案例：Amazon 怎样把预测从专家经验变成平台能力

Amazon 的平台化预测可以从一个很小的商品讲起。一个用户可能不是在买畅销手机，而是在找某种草坪工具、浴室管道配件、汽车维修零件，或者一本二手书的特定版本。这类长尾需求分散在不同国家、不同地区和不同使用场景里。单个商品看起来很小，但当平台覆盖大量国家和数以亿计的商品时，“哪一个地方、哪一种商品、未来几周会卖多少” 就变成一个巨大的预测系统问题。

早期的做法可以依赖少数专家。2007 年前后，Amazon 的基础预测团队还可以是十人量级，里面有统计学家、算法工程师和平台工程师，先集中解决最重要的预测问题。随着商品规模、地区规模和服务承诺继续扩张，团队当然可以扩大；但这条路不能无限走下去。一个专家可能擅长 3C，另一个专家可能擅长汽车配件，如果每个品类都靠少数人手工维护规则，那么专家离职、品类扩张或市场变化都会让系统变脆。

真正的转折，是把专家经验沉淀成平台流程。平台不应要求某个人逐个解释每个商品为什么会卖得好，而应自动读取商品文本、图片、历史销量、价格、流行度、地区差异和相似商品信息。二手书是一个直观例子：卖家上传图片后，系统可以识别书的新旧程度，结合文字识别确认书名和版本，再根据历史销售记录估计流行度、价格和未来需求。自然语言处理、图像识别和后来的 Transformer 方法，都是在把这些分散信息转化为预测特征。

服务形态上，Amazon 的预测能力也不只是内部工具。Amazon Forecast 这类预测服务可以理解为平台化预测能力的外部化：企业上传自己的时间序列数据，平台补充缺失处理、天气或时空等外部信息，自动训练和生成未来预测，再按存储、计算和调用资源计费。这里出售的不是某一个公式，而是一条生产线：数据接入、特征生成、模型训练、评估、部署和看板交付。

这个案例说明，商业预测的组织形态会发生变化。少数专家的价值不是消失，而是从 “手工预测每个对象” 转向 “设计、监控和校正整套预测平台”。当预测结果直接影响定价、补货、仓储和配送承诺时，企业需要的不是一次漂亮预测，而是能在数亿级商品和持续变化市场中稳定运行的平台能力。

MoE 思想给平台化预测补上了组织层面的解释。企业不应要求一个专家或一个模型理解所有商品，而应把不同专家的能力沉淀为可调用的预测者：有人或模型擅长 3C，有人擅长汽车配件，有人擅长季节性需求，有人擅长促销冲击。平台的任务是记录这些预测者在相似场景中的表现，并在新任务到来时决定谁参与、谁权重更高；这和组合预测长期强调的稳健性逻辑是一致的 [Wang et al., 2023]。

例如，一个零售企业不是只预测一个商品，而是要同时预测许多门店、地区、品类和时间粒度。一个金融或算力调度平台也不是只预测一个指标，而是要接入实时数据流，对大量对象持续更新预测。此时分布式计算（distributed computing）、GPU 加速、任务调度（job scheduling）和自动监控不再是技术细节，而是预测系统能否运行的前提。

Google Cloud 的产能管理则展示了另一类商业预测。云平台必须预测客户未来对

机器、区域、带宽和计算时长的需求。低估需求会影响服务水平；高估需求会造成机器、电力、制冷和维护成本浪费。以新业务出海为例：客户今天可能说需要五千台机器，明天需求可能变成一万台；如果云平台没有准备，服务就会出问题，但如果长期预留过多资源，闲置成本又会由平台承担。

更困难的是，客户自己也未必知道未来需要多少资源，尤其是新产品出海、模型上线或业务快速增长时。平台必须根据客户计划、历史用量、行业相似案例、地区选择和实时信号估计资源需求，并把预测结果连接到服务保障、资源预留和动态定价。未来这类系统还可能通过对话式入口收集需求：客户说明要出海、要部署在哪个区域、预计有多少用户，系统再把自然语言描述转化为资源规划和价格建议。

因此，企业预测可以理解为连接算力、云平台和多类模型的系统工程。平台要把数据、模型、评估、权限、费用和业务动作连接起来。模型效果只是其中一部分；在跨地区、跨品类和多层级业务中，还要处理上下层预测是否一致的问题，预测调和研究正是为这类平台约束提供方法基础 [Zhang et al., 2023]。如果没有稳定的数据入口、可追踪的评估记录和明确的责任边界，预测结果很难进入真实决策。

算力也是平台能力的一部分。传统单序列预测可以在单机上完成；但如果要同时预测几千种加密货币、上万只股票、数十万商品或每个社区的实时需求，就需要分布式计算、GPU 加速和任务调度。学校高性能计算平台提供的是算力入口，而完整商业云平台还要把数据、特征、模型、服务和前端连接成一条生产线。

3.7　决策成本与评价指标

预测误差只有放进业务场景中才有意义。不同场景对误差的容忍度不同：

- 库存预测：低估可能造成缺货，高估可能造成积压。
- 风险控制：低估风险可能带来坏账，高估风险可能损失客户。
- 产能规划：低估需求会影响服务质量，高估需求会造成闲置。
- 营销预算：预测过高可能浪费投放，预测过低可能错失增长机会。

因此，评价指标不应只有 RMSE 或 MAE。还应考虑服务水平、缺货率、库存周转、利润、现金流、客户满意度和风险暴露。预测团队要把技术指标翻译成业务指标。

库存例子可以把这一点说得更具体。点预测如果告诉我们某个关键备件平均需要 4 个，并不代表只准备 4 个就合理；如果需求波动较大，为了达到 90%、95% 或 99% 的服务水平（service level），安全库存（safety stock）可能要显著高于均值。服务水平越高，缺货风险越低，但持有成本、仓储占用和资金压力也越高。关键基础设施备件、普通鞋类库存和电商长尾商品，对同一个预测误差的容忍度完全不同，所以预测系统必须把误差分布、服务水平和业务损失连在一起。

在平台化场景中，评价还要覆盖服务质量。一个模型误差较低，但延迟太高、成本太大、无法解释、无法接入业务系统，也不一定适合上线。相反，一个简单模型如果能稳定运行、及时告警、容易被业务人员理解，可能更适合关键流程。

预测进入决策后，还要设计人工介入边界。自动系统可以给出建议，例如建议补

货数量、交易方向或产能配置；但高风险动作通常需要审批、限额或熔断机制（circuit breaker）。评价预测系统时，不仅要问“模型准不准”，还要问“谁会看到结果”“谁能批准动作”“系统出错时谁来接管”。

错误预测还会沿供应链放大。如果一个平台通过补贴快速进入外卖或即时零售业务，前端需求可能突然被拉高；如果骑手、门店备货、食品加工和后端调度没有同步预测，结果就可能是订单没人送、食品堆积、资源浪费和用户体验下降。此时问题不是单个误差指标不好看，而是预测、产能和执行系统没有同步。

京东物流和外卖补贴的例子说明，促销决策、库存预测和履约预测必须同时工作。如果一毛钱奶茶突然带来大量订单，但门店产能、骑手运力和库存没有提前准备，用户看到的是“下单成功”，系统承受的却是履约失败。配送到达时间预测（Estimated Time of Arrival, ETA）也一样：平台承诺晚上六点送到，用户就会据此安排工作和等待；如果九点半还没送到，预测误差就直接变成客户体验损失。

天气预测这类行业例子可以帮助理解“误差如何变成决策”。天气预报如果只告诉你今天可能下雨，已经不足以满足很多场景；农业灌溉、出行安排和大型活动更关心未来几十分钟、某个地点上方那片云是否会降雨。现代天气预测的业务问题已经从“明天下不下雨”推进到更细的时间和空间粒度。

地图导航中的拥堵和红灯等待预测，会影响路线选择。很多人以为红灯倒计时来自官方传感器，实际系统经常是根据手机导航轨迹、车辆速度和路口等待时间推断红灯周期。国内许多红绿灯有固定周期，因此下一轮等待时间可以预测；但如果数据采集被异常行为干扰，例如大量手机同时移动，系统也可能把不存在的拥堵判断为真实拥堵。

汽车、保险和智能驾驶场景还提醒我们，预测系统必须考虑法律和治理边界。车辆刹车频率、急加速、路线和方向盘动作可以用于预测风险，但保险公司能否据此动态定价，取决于当地法律和公平性要求。相同数据在智能驾驶中可能用于判断驾驶员是否突发疾病、车辆是否需要接管；在保险定价中却可能涉及歧视或合规问题。预测结果不能只停留在分数上，它必须服务于是否提醒、限速、接管、报警或人工复核等具体动作。

同样，销售 demo 预约和成单率预测不是为了得到一个转化概率，而是为了决定何时联系客户、由谁跟进、预算投向哪个地区，以及海外扩张时如何配置团队。预测的商业价值来自后续动作：如果系统不能改变资源分配和行动顺序，再漂亮的概率也只是报表。

3.8 组织协作

商业预测与决策需要多类角色协同：

- 数据团队负责数据源、口径、质量和权限。
- 特征与平台团队负责自动预处理、feature store、任务调度和服务稳定性。
- 数据科学家负责模型、误差分析、实验设计和基准比较。
- 领域专家负责解释变量、业务规则、异常事件和可操作约束。

- 决策者负责权衡成本、风险、行动方案和覆盖规则。

如果只有模型，没有业务约束，预测可能无法落地；如果只有经验，没有数据验证，决策可能难以扩展；如果只有平台，没有责任边界，系统很难持续改进。

这种协作要求团队从 “预测师” 转向 “预测系统设计者”。未来进入企业实习或工作时，不应只问某个模型怎么调参，还要问数据如何收集、特征如何生成、模型如何部署、结果如何展示、业务如何使用，以及失败时如何回滚。

这也是为什么预测既是科学也是实践判断。科学部分要求数据、模型、误差和验证可复现；实践部分要求团队说清关键假设、可接受风险和业务边界。真正能说服决策者的，不是 “我们是专家”，而是系统能证明预测在过去相似场景中有效、在当前场景中有合理证据，并且在出错时有可控的接管方案。

3.9　实施清单

在启动一个商业预测项目之前，至少回答以下问题：

1. 预测结果服务哪个具体决策，动作是什么？
2. 决策需要什么时间粒度、提前量和刷新频率？
3. 哪些数据流进入系统，数据口径和权限是否清楚？
4. 哪些特征可以自动生成，是否需要 feature store？
5. 允许哪些模型族进入比较，简单基准是什么？
6. 预测通过 API、看板、报告还是自动规则交付？
7. 高估和低估的业务成本是否对称？
8. 模型上线后如何监控误差、漂移、延迟和调用成本？
9. 谁有权审批、覆盖或停止模型建议？
10. 当数据异常、模型失效或业务环境突变时，系统如何回退？

3.10　练习

1. 选择一个企业场景，画出从数据层、特征层、模型层、服务层到决策层的流程图。
2. 对库存场景分别讨论高估和低估需求的成本。
3. 设计一个指标表，同时包含预测误差、业务结果、延迟和成本指标。
4. 为一个预测 dashboard 设计三个告警规则，说明何时需要人工接管。
5. 讨论一个你熟悉的平台化预测服务，它的价值来自算法、数据、工程还是业务集成？

第 4 章

大语言模型与预测

大语言模型最初是为自然语言任务（natural language tasks）训练的，但预测任务与语言建模（language modeling）共享一个重要结构：都在已有序列的基础上推断下一个元素。语言模型预测下一个 token，时间序列模型预测下一个观测值。这个相似性让大语言模型可以参与预测，但它只是机会，不是质量保证。关于 LLM 成功原因及其对预测领域影响的讨论，也提醒我们要把生成能力放回可评价的预测流程中 [Makridakis et al., 2023]。

传统统计模型通常追求用较少参数（parameters）凝练规律，例如趋势、季节性、自相关和误差结构。大语言模型走的是另一条路：用大规模数据和大量参数学习丰富的上下文表示（contextual representation），再根据任务提示生成结果。未来的预测系统很可能同时使用这两类能力：统计模型提供可检验结构，LLM 提供上下文理解、非结构化信息（unstructured information）处理和自然语言交互。

因此，本章讨论的重点不是“大模型是否会替代所有预测模型”，而是三个更实际的问题：数值时间序列如何进入 LLM？LLM 在预测流程中可以扮演什么角色？怎样避免把一次看似合理的回答误当成可靠预测？

4.1 学习目标

完成本章后，你应该能够：

- 解释大语言模型如何把文本、图像、视频和时间序列切分为 token。
- 说明 embedding 为什么不是简单编号，而是对上下文含义的向量表示。
- 区分 LLM 作为预测器、特征提取器、助手和模型接口的不同用法。
- 解释 prompt 与 API 在预测流程中的角色差异。
- 识别大语言模型用于预测时的幻觉、隐私、成本、格式和验证风险。

4.2 大语言模型的基本能力

大语言模型是在大规模数据上训练的深度学习模型，能够理解和生成自然语言，也能处理代码、表格说明、图像、视频和多轮对话。它可以总结、翻译、问答、写代码、生成报告，并在一定程度上进行推理。与传统任务专用模型不同，大语言模型更像一个通用接口：同一个模型可以通过不同提示词适配不同任务。

这种能力对预测很有吸引力，因为真实预测任务很少只有一列数字。销量预测可能伴随促销方案、新闻舆情、客户反馈和渠道调整；金融预测可能伴随公告、会议纪要、研报和监管政策；旅游预测可能伴随航班、签证、搜索和地缘政治信号。LLM 可以把这些非结构化信息整理成事件、假设、解释或结构化特征，再交给预测流程使用。

我们把大语言模型比作“带有联想能力的超级输入法”。早期输入法可以根据前几个字补全词语，或者按规则纠正拼写；大模型则在海量文本、代码和多模态数据中学习上下文关系，能够根据一段输入继续生成解释、步骤、表格或代码。这个比喻有助于理解它的底层任务：给定上下文，预测后续最可能出现的 token。

大模型也可以帮助做分类、数据标注、文献梳理、代码生成和初步咨询。它可能给出传统专家没有想到的联想，也可能生成不存在的论文、错误引用或貌似专业的解释。把它用于预测时，必须把“生成了合理文字”和“给出了可靠预测”分开看。

也要注意，LLM 给出的是基于已学知识和当前上下文的概率性判断，而不是确定事实。它可能提供传统专家没有想到的视角，也可能生成听起来合理但没有数据支撑的解释。学习 LLM 预测的第一原则，就是把它放在可检验流程里，而不是把它当成自动正确的专家。

4.3　统计模型与大模型的两种哲学

传统统计模型通常强调浓缩和凝练。一个回归模型（regression model）、ARIMA 模型（Autoregressive Integrated Moving Average, ARIMA）或状态空间模型（State Space Model, SSM），用有限参数刻画趋势、季节性、自相关和误差结构。模型越简洁、假设越清楚，解释和检验通常越容易。

大模型走的是另一条路线。它不追求用五个或十个参数解释一个小问题，而是用大规模数据和大量参数吸收语言、图像、代码、知识和上下文模式。传统统计训练常问“这个参数是否显著”；大模型训练更关心“这么多参数能否学到可迁移的表示”。两条路线逻辑不同，并不互相替代。

当领域机制清楚、数据较少、解释要求高时，统计模型和专家判断仍然重要。当场景陌生、文本和图像信息丰富、专家成本高，或者需要把复杂资料整理成结构化输入时，LLM 可以提供新的工具。预测系统的现实选择通常不是“统计模型或大模型”，而是把两者放进同一个可评估流程。

4.4　Token 与 Embedding

LLM 不直接处理人类看到的完整句子，而是先把输入切分成 token。Token 可以是一个字、一个词、词的一部分，也可以是图像中的图像块（patch）、视频中的帧或片段。对于时间序列，token 可以是一个数值、一个变化方向、一个分桶标签，或者一段时间窗口。

Token 本身只是最小输入单元。模型真正使用的是 embedding，也就是把 token 映射成向量（vector）。Embedding 的作用不是记录这个词在字典里的编号，而是用一串数值表示它在上下文中的含义。比如 `powerful` 和 `strong` 的含义接近，它们的向

量距离可能较小；但二者仍有细微差别，模型可以在大量语料中学习这种差别。

图像和视频也能用类似方式进入模型。一张图像可以切成许多 patch，每个 patch 成为一个 token；视频可以先拆成帧或片段，再映射成 token。无论输入是文字、图片、视频还是数值序列，模型真正处理的都是 token 及其向量表示。

时间序列进入 LLM 时也需要类似处理。假设一个销售序列是 `[100, 120, 90]`，至少有几种表示方式：

表示方式	例子	适合表达什么
原始数值	`100, 120, 90`	保留精确数值
分桶标签	`中, 高, 低`	强调相对水平
差分符号	`+20, -30`	强调变化方向和幅度
局部窗口	`[100, 120, 90]` 作为一个片段	捕捉短期形状
学习型 embedding	神经网络生成向量	捕捉难以手工命名的模式

这些表示没有绝对优劣。原始数值保留精度，但模型未必理解业务含义；分桶和差分损失部分信息，却能突出趋势和变化；学习型 embedding 更灵活，但解释性较弱。选择哪一种，取决于数据频率、数值范围、预测目标和后续模型。

茅台价格例子可以帮助理解 embedding 的价值。`1940` 单独出现时只是一个数字；如果告诉模型这是某一天京东上茅台酒的价格，再给出过去价格、节日位置、去年同期、上周价格和市场背景，这个数字就有了上下文。它可能意味着“比去年同期低很多”“临近节日出现小幅波动”“本周处在相对低位”。Embedding 的价值就在这里：它把数字放回场景，把单点数值转化成带有历史和业务含义的表示。

另一个直观例子是收入数字。出租车司机说“今天挣了 300 元”，这个数字本身不足以判断好坏；如果同时知道今天堵车、换电、空驶、平台派单不顺，300 元就带有完全不同的含义。数值 embedding 的目标也是如此：不仅表示“是多少”，还表示“在当前上下文中意味着什么”。

但 embedding 不是魔法。一个向量如果捕捉的是与预测目标无关的信息，或者把节日、价格、竞争对手、时间位置和空间关系混在一起却没有学到真正有效的结构，就不一定改善预测。早期的 TF-IDF（Term Frequency-Inverse Document Frequency, TF-IDF）也可以看作一种文本向量化；现代 embedding 的进步在于能从更丰富的数据和上下文中学习表示，但它仍然需要与目标变量、预测步长和评价指标匹配。

4.5　从词频到 GPT

自然语言处理的发展说明了为什么 Transformer 会成为预测课程的关键工具。早期处理文本时，可以先列出关键词，再统计每个词在文档中出现多少次，形成词频矩阵。TF-IDF 在词频基础上进一步惩罚过于常见的词，让更有区分度的词获得更高权重。这些方法直观、便宜、可解释，但它们主要停留在词面层面，很难理解词序、上下文和跨语言语义。

Word2Vec 代表了下一步：把词映射到稠密向量空间，使语义相近的词在向量空间中更接近。这里需要特别说明，Word2Vec 的经典结构主要是 CBOW（Continuous Bag-of-Words, CBOW）和 Skip-gram，而不是通常意义上的 CNN（Convolutional Neural Network, CNN）。它的贡献不是完成了全部语言理解，而是把文本从稀疏词频表转成了可计算、可比较的 embedding。

BERT（Bidirectional Encoder Representations from Transformers, BERT）则把 Transformer 编码器（Transformer encoder）引入文本理解。它不是给一个词固定向量，而是根据上下文生成带语境的表示。同一个词出现在不同句子中，embedding 可以不同。GPT（Generative Pre-trained Transformer, GPT）系列进一步把 Transformer 用于生成任务：给定前文，预测后续 token，并在大规模数据和参数上学习更通用的文本生成能力。Transformer 的自注意力结构（self-attention mechanism）是这一路线的关键技术基础 [Vaswani et al., 2023]，入门时也可以把 LLM 看作由 token、embedding、注意力和生成头组合起来的系统 [Alammar and Grootendorst, 2024]。

这条路径和时间序列预测直接相关。时间序列也可以从原始数值、差分、分桶标签或局部窗口转成 embedding；一旦变成模型可读的序列表示，就可以用注意力机制学习哪些历史片段和模式对未来更重要。因此，LLM 与时间序列基座模型并不是两个完全分离的话题，它们共享 “把输入 token 化、embedding 化，再用 Transformer 学上下文” 的基本路线。

4.6　为什么 LLM 可以做时间序列预测

时间序列预测和语言建模都依赖上下文。语言中的上下文是前面的词和句子；时间序列中的上下文是过去观测值、外生变量和事件。只要能把时间序列转化为模型可读的序列形式，LLM 就有机会学习其中的模式。

这种 “可读” 可以来自零样本提示（zero-shot prompting），也可以来自少量示例。零样本时，我们只告诉模型任务和历史数据；少样本提示（few-shot prompting）时，我们再给它几个输入-输出样例，让它模仿预测格式或推理步骤。可以用代码助手类比这一点：模型不一定真正证明了牛顿法的数学性质，但它可以根据示例和上下文生成可运行的近似流程。时间序列预测也类似，prompt 中的数据表、历史窗口、差分和业务说明都会被转成 token 和 embedding，成为模型生成下一段输出的上下文。

LLM 与时间序列预测结合通常有四条路径：

1. **直接预测**：把历史数据写入 prompt，让模型输出未来值。
2. **特征提取**：用 LLM 从新闻、报告、会议纪要或评论中提取事件变量，再交给时间序列模型。
3. **表示学习**：把数值片段映射为 embedding，让模型学习趋势、季节、异常和局部形状。
4. **混合框架**：让 LLM 连接统计模型、专用时间序列模型和业务规则，形成可解释的预测工作流。

直接预测最容易上手，也最容易误用。把 CSV 上传到聊天框，要求模型预测未来五期，模型通常能给出一个看起来合理的结果。但这个结果可能只是语言模型根据上下文生成的文本，并没有经过滚动验证、基准比较或业务损失评估。更稳健的做法是把 LLM 放在完整流程中，让它辅助表示、提取、解释和自动化，而不是单独承担所有预测责任。

还要区分两种 “AI 预测”。一种是模型真正使用数值 embedding、上下文表示或时间序列基座模型能力来生成预测；另一种只是识别了数据格式，然后在背后调用 ARIMA、ETS 或其他传统包。后者仍然有工程价值，但它更像自动化编排或工具调用，不应被误认为大模型自己理解了时间序列机制。判断一个系统是否可靠，仍然要看它的输入、模型、输出和评估证据。

4.7 Prompt 与 API

Prompt 是给模型的任务说明和上下文，决定模型应该看到什么、完成什么、怎样输出。API 是调用模型的接口，使这个过程可以被程序重复执行、记录和评估。

在预测任务中，prompt 至少应说明：

- 目标变量是什么；
- 历史数据是什么格式；
- 时间频率是什么；
- 预测步长是多少；
- 是否需要只输出 JSON；
- 是否需要给出预测区间或解释；
- 不允许编造不存在的数据。

聊天框上传文件适合探索，但不适合稳定生产。它依赖人工操作，输入格式和输出格式很难完全固定，也不容易批量评估。API 的价值在于把模型能力嵌入工作流：程序可以构造同样格式的请求，接收同样结构的输出，解析成表格，再和真实未来数据比较。

一个概念性的 LLM 预测流程可以写成：

1. 整理历史序列，明确时间频率和预测步长。
2. 把历史数据、任务说明和输出约束写入 prompt。
3. 要求模型返回严格结构化结果，例如 JSON。
4. 解析输出，检查行数、日期、数值类型和量级。
5. 用留出集或滚动窗口评价误差。
6. 与简单基准、统计模型或专用时间序列模型比较。

Prompt 不是装饰，它定义了模型能看到的信息和必须遵守的输出约束。API 也不是预测质量的保证，它只是让模型能力进入可复现流程。真正决定质量的，仍然是数据、表示、验证、基准和决策后果。

4.8 LLM 的角色

LLM 用于预测时不只有一种用法。更准确地说，它可以在预测系统中扮演不同角色：

- **直接预测器**：根据历史数值和上下文输出未来值，适合教学演示和低成本原型。
- **特征提取器**：从文本、公告、新闻、政策或客服记录中抽取事件、情绪、风险和约束。
- **解释助手**：把异常、预测假设和业务背景翻译成可沟通的语言。
- **代码与流程助手**：生成清洗、画图、调用 API、解析 JSON 和评估误差的辅助代码。
- **模型接口**：帮助用户用自然语言调用统计模型、专用时间序列模型或时间序列基座模型。

在多源数据和复杂业务语境中，LLM 的价值往往不是 “直接预测一个数”，而是把预测任务组织得更清楚，把文本信息转化为模型或决策可以使用的结构化信息。对于关键业务，LLM 直接给出的数值应被视为候选预测，而不是最终答案。

组合预测也给了一个有用类比。没有一个模型在所有场景都最好，不同模型可能在不同数据、不同步长和不同损失函数下各有优势。LLM 可以作为其中一个专家，也可以帮助组织多个专家的输出；但最终仍要根据验证表现分配信任，而不是因为某个模型更大就自动相信它。

4.9 局限与风险

LLM 预测的风险可以分成几类：

风险	典型表现	控制方法
幻觉（hallucination）	编造不存在的数据、事件或解释	限定输入来源，要求引用字段，人工抽查
数值不稳定	同一问题多次输出不同数值	固定参数，重复调用，做稳定性检查
格式失败	返回说明文字而不是可解析表格	要求严格 JSON（JavaScript Object Notation, JSON），解析后校验字段
隐私泄露	把内部销售、客户或财务数据发给外部服务	脱敏、本地部署、权限控制
成本和延迟	大规模 API 调用费用高、响应慢	估算 token 成本，缓存（caching）结果，分层调用
解释不可靠	解释听起来合理，但不是模型真实依据	用验证结果和业务证据支撑解释
评估不足	一次回答看起来合理就进入决策	留出集、滚动验证、基准模型比较

这些风险并不意味着不能使用 LLM，而是说明 LLM 预测必须有约束。语言模型预测下一个 token，时间序列模型预测未来观测；二者都依赖上下文，但数值预测必须回到时间序列验证、输出解析和基准比较，才能判断是否可靠。

成本和部署方式也需要进入决策。云端 API 使用方便，但涉及费用、速率限制和数据外发；本地部署能降低隐私风险，但需要硬件、运维和模型压缩能力。预测系统不是只比较精度，还要比较投入、产出、隐私、延迟和可持续维护成本。

4.10　本地大模型与隐私

个人 AI 算力盒子的意义在于提醒我们：大模型不一定只能放在云端。随着小型 AI 计算设备、共享内存架构和开源模型发展，小团队、实验室、医院或家庭可能在本地运行专门模型。这样做的核心价值不是炫耀硬件，而是让敏感数据留在本地：病人数据、学生数据、企业销售数据、家庭健康数据和个人行为记录，都不必默认上传到外部平台。

案例：算力盒子把云端大模型拉回本地

算力盒子的故事可以从一个现实问题开始：我们使用 ChatGPT、DeepSeek 或其他云端模型时，表面上只是输入一条指令，实际上通常是把数据传到远端，由云端的大算力完成推理，再把结果返回。这个方式方便，也降低了普通用户使用大模型的门槛，但它默认数据可以离开本地。对医院、实验室、企业销售、学生记录、家庭健康数据或个人行为记录来说，这个默认并不总是可以接受。

个人 AI 算力盒子提供的是另一种部署想象：把一台小型 AI 计算设备接入本地网络，像一台专用 Linux 机器一样运行模型、管理文件和提供远程访问。本地电脑没有 GPU 并不必然阻止我们使用本地模型，因为模型可以在盒子上运行，用户只是

把任务提交过去。盒子连入网络后，也可以通过固定地址远程访问，于是它既不是纯粹的云服务，也不是只能放在桌面旁边的单机设备，而是一个可由个人或团队控制的小型 AI 基础设施。

这个盒子的关键不只是“有 GPU”，而是 CPU、GPU 和共享内存更紧密地结合在一起。传统独立显卡的瓶颈常常不是算力本身，而是显存大小：一张常见 RTX 4090 的显存只有 24GB，直接承载许多 70B 级模型会很困难。共享内存架构可以缓解这个限制，让较小设备也有机会运行更大的本地模型，或者运行经过量化、压缩、蒸馏后的模型。当然，粗糙量化可能带来误差，蒸馏也需要验证任务表现，所以真正可用的本地模型不是把大模型简单塞进盒子，而是要把硬件、模型大小、推理速度、精度和维护成本一起设计。

截至 2026 年 5 月，商业市场已经把这种形态产品化。官方名称不一定都叫“算力盒子”，但它们共同指向同一个方向：在桌面、实验室或部门内网中放置一台专门的本地 AI 计算节点。NVIDIA DGX Spark（早期称 Project DIGITS）是基准形态，使用 GB10 Grace Blackwell Superchip、128 GB 统一内存和预装的 NVIDIA AI 软件栈，面向桌面上的原型开发、微调和推理。Lenovo ThinkStation PGX 把这个平台做成 1.13L 小型工作站，强调 240W 供电、DGX OS、Ubuntu Pro、AI Workbench，以及两台设备互联后处理更大模型的能力。Dell Pro Max with GB10、HP ZGX Nano G1n AI Station、ASUS Ascent GX10 和 Acer Veriton GN100 也都围绕 GB10、128 GB 统一内存、1-4 TB 本地存储、NVIDIA AI 软件栈和 ConnectX 扩展网络来组织产品。这些商业产品说明，算力盒子不是“买一个更强显卡”的问题，而是一种部署边界的变化。我们可以把它放在办公室网络里，作为销售预测、文本特征抽取、私有知识库问答和模型蒸馏实验的本地节点；也可以先在本地验证数据流程，再把成熟模型迁移到云端或集群。选型时不要只看 PFLOPS、参数上限或宣传里的“本地大模型”，还要看实际模型是否适配、统一内存是否够、推理速度是否满足业务节奏、多人并发如何处理、散热功耗是否可接受、IT 支持是否跟得上，以及数据安全策略能否真正落地。

对预测任务来说，算力盒子的价值在隐私和持续使用。一个实验室可以把内部文档、实验记录和时间序列数据留在本地；一个家庭可以把健康设备数据留在自己的设备中；一个小团队可以在自己的网络里运行领域模型，服务销售预测、库存预测或设备监控。这样，AI 不再只是“把问题发到云端”的服务，而可以变成嵌入本地流程的预测基础设施。代价也同样清楚：用户要承担硬件、模型版本、依赖环境、能耗、远程访问和安全管理。

本地模型尤其适合细分领域。很多场景并不需要一个知道全世界所有知识的通用模型，而是需要一个熟悉本团队、本行业或本人历史数据的小模型。一个二十人的团队可以把内部文档、实验记录和领域数据放在本地微调；一个家庭也可以把长期健康、生活和行为数据留在自己的设备中，让模型提供个性化提醒。对预测来说，这意味着“模型能力”要和“数据归属”一起讨论。

本地部署（local deployment）也有代价。用户要管理硬件、模型文件、内存、能耗、依赖版本和远程访问。传统 GPU 的显存（Video RAM, VRAM）可能限制模型大小；共享内存（unified memory）设备可以缓解这一点，但仍要考虑量化（quantization）、精度、推理（inference）速度和并发使用。云端 API 与本地模型不是谁彻底替代谁，而是

两种不同的部署选择：前者轻便，后者更可控。

模型蒸馏会进一步把这个问题连接到部署形态。未来很多 AI 能力可能不再以“云端大模型聊天框”的形式出现，而是嵌入手机、手表、企业内网或专用设备中。要做到这一点，通常需要把大模型压缩、迁移或蒸馏成小模型，让它只保留当前用户或当前业务真正需要的能力。这样，本地部署不只是硬件问题，也是一套从 Teacher Model 到 Student Model 的模型工程问题。

4.11　一个实用判断

当你考虑用 LLM 做预测时，先问六个问题：

1. 它是在直接预测数值，还是在辅助提取信息？
2. 输入数据是否清楚说明了目标、频率、历史窗口和预测步长？
3. 输出能否被程序稳定解析和校验？
4. 是否与简单基准或专用时间序列模型比较过？
5. 是否经过留出集或滚动窗口验证？
6. 成本、隐私和延迟是否符合业务要求？

如果这些问题答不上来，就不要把 LLM 输出直接用于决策。它可以作为探索工具、解释工具或辅助工具，但还不是可靠的预测系统。

4.12　练习

1. 将一个月度销量序列分别表示为原始数值、分桶标签和差分符号，比较三种表示各自保留和损失了什么信息。
2. 解释为什么 `1940` 单独作为数字意义有限，而“某日期的茅台价格 1940 元”可以成为预测上下文。
3. 写一个不调用 API 的 prompt，要求 LLM 对月度销量序列只输出 JSON，并说明必须包含哪些字段。
4. 比较聊天框上传 CSV 和 API 批量调用在可复现性、成本、隐私和评估上的差异。
5. 设计一个实验，比较 LLM 直接预测、季节性 naive 和一个统计模型在 12 期留出集上的表现。
6. 找一个业务场景，列出使用云端 LLM 做预测时可能出现的幻觉、隐私和成本风险。

第 5 章

使用 DeepSeek API 进行预测

本章把大语言模型从 “聊天工具” 放回预测流程中：我们给模型一段结构化的时间序列历史，让它按指定格式返回未来若干期的点预测，再把结果解析成数据表并画图检查。这里的重点不是证明 DeepSeek 一定优于统计模型，而是训练一个可复现的 API 工作流：数据准备、提示词约束、请求发送、JSON 解析、结果检查和误差评估。

把 CSV 上传到网页对话框，可以快速探索一个序列；但这种方式依赖人工操作，不容易复现，也不方便批量评价。API 的价值在于把同一个任务变成程序：同样的数据格式、同样的 prompt、同样的输出 schema、同样的解析和检查。只有这样，LLM 输出才有机会进入正式预测实验。

5.1 学习目标

完成本章后，你应该能够：

- 说明为什么调用大模型做预测时必须约束输入数据、预测步长和输出格式。
- 使用 OpenAI 兼容客户端（OpenAI-compatible client）调用 DeepSeek API，并避免把 API key 写进 notebook。
- 将模型返回的 JSON 解析为 `pandas.DataFrame`，并与历史数据放在一起检查。
- 区分 “模型返回了一个数字” 和 “预测结果可以用于决策” 这两件事。
- 设计一个简单实验，比较不同预测步长、提示词和历史窗口对结果的影响。
- 记录模型、参数、prompt、数据切分和调用成本，使实验可以复现。

5.2 核心思路

DeepSeek API 的角色可以理解为一个远程模型服务（remote model service）。Notebook 不直接训练模型，而是把历史序列、任务说明和输出格式发送给 API。API 返回文本，文本里包含预测结果。课程里的 notebook 采用三个约束来降低不可复现风险：

1. 历史数据整理成 `unique_id`, `ds`, `y` 三列。
2. 提示词明确预测步长 `h` 和频率 `freq`。

3. 返回结果要求为严格 JSON，字段包含 `forecast`, `ds`, `yhat`。

这三个约束都很重要。时间序列预测不是只问“下个月是多少”；它还需要知道历史数据的时间间隔、预测几期、是否只预测一个序列、返回值能否被程序继续处理。

一个可复现的 API 预测实验可以看成六个环节：

环节	作用	失败时的后果
结构化输入	明确序列、日期和目标变量	模型猜错列含义或时间顺序
约束 prompt	明确任务、频率、步长和输出格式	返回内容不可控
严格 schema（output schema）	让输出能被程序解析	后续画图和评估中断
解析器（parser）	把文本转成数据表	把说明文字误当预测
校验规则（validation rules）	检查日期、行数、数值和量级	格式错误进入评价
留出评价	判断预测是否有增量价值	只凭直觉判断好坏

因此，API 不是网页聊天框，而是把模型服务嵌入 notebook、脚本和业务系统的接口。学习重点不是手动问一次模型，而是把数据整理、请求构造、响应解析和结果评估串成可重复运行的流程。

我们强调这个差别：把 CSV 上传到对话框可以探索，但每次都依赖人工操作；API 调用则把历史数据、任务说明、换行符、输出约束和模型参数都编码成一个可记录的请求。程序把 prompt 作为字符串发送给模型，模型返回文本，Python 再把文本解析成 JSON 或 DataFrame。只有这样，LLM 预测才有机会进入批量实验和正式评估。

同样的工作流也可以扩展到本地时间序列基座模型。DeepSeek API、TimeGPT API 和离线 Chronos 模型的部署方式不同，但对使用者来说都应被包装成清楚的输入输出接口：历史数据是什么表，预测步长是多少，输出是点预测还是分位数，结果如何合并真实值并评价。TimeGPT 把预训练时间序列模型包装成 API，是理解这类工作流的一个直接参照 [Garza et al., 2024]。区别在于，云端 API 更方便但有成本、速率和数据外发问题；本地模型下载后可以离线运行，更适合隐私敏感或网络不稳定的课程练习，但需要管理模型文件、依赖版本和硬件资源。

5.3　API Key 与环境

不要在课程仓库或 notebook 中写真实 API key（API key）。推荐把 key 放在环境变量（environment variable）中：

```
export DEEPSEEK_API_KEY=" 你的 key"
```

Python 代码中再读取：

```
import os
from openai import OpenAI

client = OpenAI(
    api_key=os.environ["DEEPSEEK_API_KEY"],
    base_url="https://api.deepseek.com/v1",
)
```

OpenAI 兼容客户端不只用于 OpenAI 模型，也可以连接 DeepSeek 等兼容接口。关键是同时记录 `base_url`、`model`、调用日期、prompt 模板和主要参数。否则即使代码能跑，后来也很难复现同一次预测到底问了哪个模型、用的是什么上下文。

如果你是在 Jupyter 中运行，先确认当前 kernel 能导入依赖：

```
import json
import os
import re

import pandas as pd
from openai import OpenAI
```

示例 notebook 中为了演示会出现 `YOUR_DEEPSEEK_API_KEY` 这样的占位符。正式作业和公开材料中不要保留真实 key，也不要把包含 key 的输出截图放进作业。

API key 相当于可计费的访问凭证。小型课程实验可能只花很少费用，但重复调用、长历史窗口、多预测步长和多序列实验会很快放大成本。正式报告应说明模型、参数、调用次数和大致成本，以便复现实验。

项目	推荐做法
Key 保存位置	环境变量或本地密钥管理，不写进 notebook
Git 提交	不提交 key，不提交包含 key 的日志
截图和报告	不展示 key，不展示完整敏感请求
模型记录	记录 `model`、`base_url`、调用日期和关键参数
成本记录	记录调用次数、历史窗口长度和预测步长
数据敏感性	上传前判断是否包含个人、客户、收入或内部经营数据

5.4　数据格式

本章示例使用航空乘客数据，输入表至少包含三列：

- `unique_id`: 序列编号。只有一条序列时也保留这一列，便于以后扩展到多序列预测。
- `ds`: 时间戳，例如月度数据的每个月第一天。
- `y`: 观测值，也就是要预测的目标变量。

这套格式的好处是清楚、可扩展，并且和许多预测工具兼容。即使现在只有一条航空乘客序列，保留 `unique_id` 也能让同一套代码扩展到多条产品、门店、地区或资产序列。

读取后先统一列名、解析时间、排序：

```
data_path = "../data/air_passengers_with_id.csv"
df = pd.read_csv(data_path)
df.columns = [c.lower() for c in df.columns]
df["ds"] = pd.to_datetime(df["ds"], errors="raise")
df = df.sort_values(["unique_id", "ds"]).reset_index(drop=True)
uid = df["unique_id"].iloc[0]
```

在真正调用 API 之前，先画历史序列。这个步骤不是装饰，而是为了发现明显的数据错误：时间顺序是否反了，是否有缺失月份，是否存在异常尖峰，训练样本长度是否足够。

```
hist = df[df["unique_id"] == uid].sort_values("ds").set_index("ds")
hist["y"].plot(title=f"{uid} - history", figsize=(10, 4),
↪  legend=True)
```

还要判断数据能不能上传到云端。航空乘客公开数据可以用于教学演示；如果数据是家庭收入、企业销售、客户订单、财务流水或内部产能，就不能随意发送给外部 API。API 工作流越方便，越要先建立数据边界。

案例：从航空乘客序列到航空预测工作流

本章的 AirPassengers 数据看起来很小：一条月度序列，记录 1949 年到 1960 年的国际航空乘客数量。它适合教学，因为趋势和季节性都很清楚：长期看，航空旅行需求不断增长；一年之内，暑期和假期附近的客流又会反复抬升。我们用它演示 DeepSeek API，不是因为真实航空公司只需要预测一条总量曲线，而是因为这条序列足够干净，能够把 API 预测流程讲清楚。

这个流程的关键，是把“上传文件让模型看一眼”改成“用程序提交一个可复现请求”。我们写 `build_prompt`，就像过去写 `pd.read_csv` 一样，把历史数据、频率、预测步长和输出格式固定下来；再通过 OpenAI 兼容客户端把请求发给 DeepSeek；最后把模型返回的 JSON 解析成 `DataFrame`。模型在内部会把日期、数值、换行和字段名转成 token，再进入自己的表示空间。它也许能生成一条看起来合理的未来曲线，但我们仍然要追问：为什么是这个数，而不是相邻的另一个数？它是否真的抓住了季节性？是否只是生成了一个平滑故事？

真实航空预测会很快超出这条单序列。IATA 长期航空客运需求预测 使用 RPK、国家对、收入、人口、票价、航班供给和运营瓶颈等变量来组织问题；这说明航空需求预测往往同时受宏观经济、航线网络、价格和供给约束影响。美国 BTS 航空准点与延误原因数据 也说明，运营预测不是只看旅客数量，还要看航班是否准点、延误原因、前序飞机是否晚到、机场拥堵和天气等因素。一个枢纽机场的十分钟拥堵，可能让后续多个航班、多个机场和多个机组排班都受到影响，这就是航空预测里的传播性。

因此，如果我们把这个案例从教学数据扩展到业务系统，输入表就不应只有 ds 和 y。客流预测可能要增加航线、机场、舱位、票价、节假日、促销、竞争航班和天气；延误预测可能要增加计划起降时间、历史准点率、机场流量、前序航班到达状态、地面保障和空管限制。DeepSeek API 在这里的价值不是替代所有专业模型，而是帮助我们把多源信息组织成一个受约束的预测请求，并把输出变成可检查、可比较、可复现的数据表。

这个案例也提醒我们不要把 “能调用 API” 误认为 “可以直接决策”。公开航空乘客数据可以上传到云端练习；真实航司数据可能包含航线策略、价格、客座率、客户行为和内部运营信息，必须先判断能否外发。即使数据可以使用，预测结果仍要和 naive、seasonal naive、ARIMA、ETS 或专业航空模型比较，并用留出集检验。航空预测的错误成本通常不对称：低估客流可能导致座位、机组或地面资源不足；高估客流可能造成空座、成本和排班浪费。API 输出只有经过这样的业务解释和误差评估，才可能进入决策。

5.5　构造提示词

提示词的目标是把预测任务说清楚，并让返回内容稳定到可以被程序解析。一个实用模板如下：

```
def build_prompt(df, h=12, freq="MS", tail=120):
    sub = df[df["unique_id"] == uid].sort_values("ds").tail(tail)
    hist = [
        {"ds": row.ds.strftime("%Y -%m -%d"), "y": float(row.y)}
        for row in sub.itertuples(index=False)
    ]
    return (
        f"你是一名时间序列预测助手。下面给出乘客序列的历史数据（频率 {freq}）。
        ↪  \n"
        f" 请预测未来 {h} 期，并只输出严格 JSON。\n\n"
        f"历史数据：\n{json.dumps(hist, ensure_ascii=False,
        ↪  indent=2)}\n\n"
        "{\n"
        f'  "h": {h},\n'
        f'  "freq": "{freq}",\n'
        '  "forecast": [ { "ds": "YYYY -MM -DD", "yhat": <float> }
        ↪  ]\n'
```

```
        "}"
    )
```

一个预测 prompt 至少包含这些部件：

部件	作用
角色	告诉模型它是一名时间序列预测助手
目标变量	说明要预测 `y`，而不是其他列
时间频率	说明月度、日度或其他频率
预测步长	明确未来几期
历史数据	给出机器可读的日期和值
输出 schema	固定字段名、日期格式和数值类型
限制条件	要求只输出 JSON，不编造不存在的数据

这里的 `tail` 控制给模型看的历史长度。历史太短，模型看不到季节性；历史太长，提示词会变长、成本会上升，也可能超过上下文限制。示例中使用 `tail=120`，实际任务中应根据业务周期和数据频率调整。

从模型角度看，`build_prompt` 不是简单地把表格贴进字符串，而是在定义时间序列如何被模型读取：日期、数值、频率、预测步长、输出字段和限制条件都会转成 token，并进一步进入模型的表示空间。这个过程类似为模型搭建一个受控输入接口。数据越结构化，模型越不容易把列含义、时间顺序或未来日期猜错。

如果下一步要解析结果，就不要要求模型写漂亮说明。说明文字可以另开一个字段或另做一次请求；核心预测请求应优先保证结构稳定。

5.6 调用模型并解析结果

调用时建议把 `temperature`（温度参数）设为 `0`，降低输出波动。即便如此，模型返回的内容仍然是文本，所以需要解析和校验。

```
def parse_json(text):
    try:
        return json.loads(text)
    except json.JSONDecodeError:
        pass

    if "```" in text:
        for part in text.split("```"):
            candidate = part.strip()
            if candidate.lower().startswith("json"):
                candidate = candidate[4:].strip()
            try:
                return json.loads(candidate)
```

```
            except json.JSONDecodeError:
                pass

    match = re.search(r"\{[\s\S]*\}", text)
    if match:
        return json.loads(match.group(0))
    raise ValueError(f" 无法解析 JSON: {text[:200]}")
```

一个 6 期预测请求如下：

```
prompt = build_prompt(df, h=6, freq="MS")
resp = client.chat.completions.create(
    model="deepseek -chat",
    messages=[{"role": "user", "content": prompt}],
    temperature=0,
)

raw = resp.choices[0].message.content
obj = parse_json(raw)
fc6 = pd.DataFrame(obj["forecast"]).assign(unique_id=uid)
fc6["ds"] = pd.to_datetime(fc6["ds"])
fc6["yhat"] = pd.to_numeric(fc6["yhat"], errors="coerce")
fc6 = fc6.dropna().sort_values("ds").reset_index(drop=True)
```

模型响应只是文本；解析后的表格才是候选预测。解析后至少检查：

- 行数是否等于预测步长 `h`。
- `ds` 是否接在历史数据之后。
- 日期频率是否符合 `freq`。
- `yhat` 是否是数值。
- 是否有缺失行、重复日期或重复预测。
- 预测量级是否明显偏离历史数据。

```
assert len(fc6) == 6
assert fc6["yhat"].notna().all()
assert fc6["ds"].is_unique
assert fc6["ds"].min() > df["ds"].max()
assert fc6["yhat"].between(0, df["y"].max() * 3).all()
```

这些检查不能保证预测正确，但能避免把格式错误当成预测结果。严格 JSON 输出是这个流程的关键。LLM 生成的是文本；如果文本不能稳定解析，后面的画图、合并、误差计算都会中断。

5.7　可视化预测

把预测和历史放在同一张图中，是最便宜的质量控制方法：

```
hist = df[df["unique_id"] == uid].sort_values("ds").set_index("ds")
fc_plot = fc6.set_index("ds")

ax = hist["y"].plot(label="history", figsize=(10, 4))
fc_plot["yhat"].plot(ax=ax, label="forecast")
ax.set_title(f"{uid} - history and 6 -step forecast")
ax.legend()
```

观察图形时不要只看曲线是否平滑。平滑曲线可能只是模型生成了一个“看起来合理”的故事。对航空乘客这种月度数据，至少要问三个问题：预测是否延续长期上升趋势？是否保留月度季节性？未来值的量级是否和历史末端衔接？

如果同时做 6 期、12 期和 36 期预测，建议分开画图，再比较它们的形状。不要只展示最长预测步长，因为长步长图形更容易显得平滑，却未必更可靠。

5.8　预测步长的影响

示例 notebook 分别演示了 6 期、12 期和 36 期预测。预测步长越长，模型越容易回到“看起来合理”的平滑曲线，但这不一定代表更可靠。长步长预测通常需要额外解释：

- 模型是否知道业务周期，比如月度季节性。
- 历史窗口是否覆盖多个完整周期。
- 是否需要给出预测区间，而不只是点预测。
- 预测结果用于什么决策，决策是否能承受误差。

预测步长还会影响 API 成本和输出稳定性。一次 36 期预测通常比 6 期预测需要更长输出；如果你还要重复调用、比较提示词或处理多条序列，费用和等待时间都会增加。在正式实验中，应先确定评价目标，再决定需要哪些预测步长。

5.9　评估与决策

API 返回的 `yhat` 只是一个候选预测。要判断它能不能用于决策，需要留出一段真实数据做验证。例如把最后 12 个月作为测试集，用更早的数据构造提示词，再比较预测和真实值。

```
test_h = 12
train = df.iloc[: -test_h].copy()
```

```
test = df.iloc[ -test_h:].copy()

prompt = build_prompt(train, h=test_h, freq="MS")
resp = client.chat.completions.create(
    model="deepseek -chat",
    messages=[{"role": "user", "content": prompt}],
    temperature=0,
)
obj = parse_json(resp.choices[0].message.content)
fc = pd.DataFrame(obj["forecast"]).assign(unique_id=uid)
fc["ds"] = pd.to_datetime(fc["ds"])
fc["yhat"] = pd.to_numeric(fc["yhat"], errors="coerce")

eval_df = test.merge(fc[["ds", "yhat"]], on="ds", how="left")
mae = (eval_df["y"] - eval_df["yhat"]).abs().mean()
rmse = ((eval_df["y"] - eval_df["yhat"]) ** 2).mean() ** 0.5
```

还要和简单基准比较。一个很朴素的基准是 naive：未来每一期都等于训练集最后一个观测值。月度季节性数据还可以用 seasonal naive（季节性朴素法）：未来某个月等于上一年同月。

```
eval_df["yhat_naive"] = train["y"].iloc[ -1]

seasonal = train.tail(12)[["y"]].reset_index(drop=True)
if len(seasonal) == test_h:
    eval_df["yhat_snaive"] = seasonal["y"].to_numpy()

mae_naive = (eval_df["y"] - eval_df["yhat_naive"]).abs().mean()
mae_snaive = (eval_df["y"] - eval_df["yhat_snaive"]).abs().mean()
```

如果 DeepSeek 预测不能超过简单基准，就不能说这个 API 工作流带来了预测价值。它仍然可能有教学价值、解释价值或原型价值，但不应直接进入业务决策。预测实践中的共同原则是，任何新工具都要和简单、透明、可复现的基准放在同一评价框架下比较 [Petropoulos et al., 2022]。

这里还要区分 “API 生成了预测” 和 “模型真正理解了时间序列”。有些系统可能把表格识别出来后，在背后调用 ARIMA、ETS 或其他工具；有些系统可能主要依赖自身的上下文表示直接生成数值。二者都可以有工程价值，但都必须接受同样的留出测试、滚动验证和基准比较。对使用者来说，关键不是先判断它到底属于哪一类，而是记录输入、输出、模型和评价证据，避免把一条曲线误认为可靠结论。

评价指标要和业务问题对应。库存和产能问题通常关心低估或高估的非对称成本；预算或收入预测可能更关心总量误差；运营监控可能更关心短期 $T+1$ 到 $T+3$ 的表现。更深入的评价指标会在后续章节展开，本章只要求把 API 输出放进可检验实验。

5.10　成本、隐私与可复现

使用 API 做预测时，技术流程之外还要管理三件事。

第一是成本。API 调用按 token、请求量或模型规格计费。历史窗口越长、预测步长越长、重复实验越多，成本越高。课程实验可以从一条序列、小步长和短窗口开始，再逐步扩展。

第二是隐私。模型服务如果在云端 API（cloud API）中运行，输入数据就会离开本地环境。公开数据和脱敏数据适合教学演示；客户、收入、交易、医疗、财务和企业内部经营数据需要严格审批或本地部署。

第三是可复现。一次预测结果如果没有记录 prompt、模型名、参数、数据切分和调用时间，就很难复查。正式报告至少应记录：

- 数据来源和时间范围；
- 训练集和测试集切分；
- prompt 模板和历史窗口 `tail`；
- 模型名、`temperature` 和调用日期；
- 解析和校验规则；
- 评价指标和基准模型；
- 是否发生重试或手工修正。

这个记录不是形式主义。LLM 预测的输出可能随模型版本、参数、上下文和提示词变化。没有实验记录，就无法判断结果来自模型能力、提示词调整，还是偶然生成；这也是 LLM 进入预测流程后必须强调复现、验证和边界控制的原因 [Makridakis et al., 2023]。

5.11　常见错误

- **把 API key 写进 notebook**：这会造成泄露风险。使用环境变量。
- **把 key 发到公开群或截图里**：API key 是可计费凭证，泄露后可能产生费用。
- **上传敏感业务数据**：云端 API 方便，但不等于所有数据都可以上传。
- **提示词没有要求 JSON**：模型可能返回解释性文字，后续程序无法稳定解析。
- **只看一次预测结果**：同一任务需要比较不同步长、不同提示词，最好再与简单基准模型比较。
- **没有检查日期**：模型可能返回错误的未来日期或重复日期。
- **把大模型输出当成统计保证**：DeepSeek 给出的是文本生成结果，不自动提供置信区间或误差分布。

- **忽略成本和延迟**：API 调用有价格、速率限制和网络延迟，不适合所有高频场景。
- **看见测试集后反复改 prompt**：这会把测试集变成调参集，导致评价过于乐观。
- **没有记录模型和参数**：模型版本或参数变化后，结果可能无法复现。

5.12 练习

1. 将一个自己的月度业务序列整理成 `unique_id`, `ds`, `y` 三列，运行 6 期预测，并画出历史与预测曲线。
2. 固定数据不变，分别使用 `h=6`, `h=12`, `h=36`，比较预测形状、输出长度和可解释性。
3. 固定 `h=12`，分别使用不同的 `tail`，观察预测结果和 prompt 长度如何变化。
4. 修改提示词，要求模型说明使用了哪些时间序列特征，但仍然必须返回 JSON。检查解释是否与图形一致。
5. 留出最后 12 期作为测试集，计算 MAE 和 RMSE，并写一段话说明这个结果是否足以支持业务决策。
6. 与 naive 和 seasonal naive 比较。如果 DeepSeek 不能超过基准，解释可能原因。
7. 对同一 prompt 重复调用三次，比较预测是否稳定，并说明是否适合进入自动化流程。
8. 将航空乘客预测扩展成一个真实业务问题，说明还需要哪些外部变量，例如航线、机场、票价、节假日、天气、前序航班状态或机场拥堵。
9. 写一段不超过三页的实验报告，包含数据、切分、prompt、模型、参数、指标、基准和决策结论。

第 6 章

预测评估

预测评估不是在模型训练结束后补做的一张表，而是预测系统的设计原则。只有先说明如何评价，我们才知道应该优化什么、比较什么，以及一个预测结果是否足以支持决策。

预测评估最容易出错的地方，是在设计模型时已经知道了未来。如果模型或研究者看到了测试期真实值，再回头调整特征、参数或提示词，评价结果就被污染了。时间序列预测尤其如此，因为时间顺序本身就是信息边界：站在今天预测明天，不能使用明天之后才会知道的事实。

本章从现代预测观点出发，强调从点预测转向概率预测，并说明两类评价方式：一类是在历史数据中构造“过去的未来”来回测模型；另一类是真正提前提交预测，等未来发生后再评价。前者更高效，后者更可信，但也更耗时。

6.1 学习目标

完成本章后，你应该能够：

- 解释为什么未来观测值可以被看作给定信息条件下的随机变量。
- 区分点预测和概率预测，并说明二者适用的业务场景。
- 选择合适的误差指标评价点预测结果。
- 说明为什么时间序列评估不能随机打乱样本。
- 设计滚动窗口或扩展窗口的时间序列交叉验证。
- 识别时间序列评价中的数据泄露。
- 说明历史回测在哪些结构突变场景下可能失效。
- 为多个模型或多个团队设计公平的预测比较规则。

6.2 预测值是随机变量

在预测发生时，目标变量 y_t 尚未被观察到。下个月销售额、明日用电负荷、未来一周客流量，都可能有一系列取值。我们掌握的信息记为 I，真正要描述的是 $y_t \mid I$：给定当前所有已知信息条件（information condition）时，未来值可能落在哪里。

这组可能取值及其概率构成预测分布（forecast distribution）。点预测只是从预测分布中取出一个代表值，通常是均值或中位数。概率预测则保留更多信息，例如区间、分位数（quantile）或完整分布。

换句话说，未来值在被观察之前是相对于当前信息集的随机变量（random variable）；一旦发生并被记录，它才变成固定事实。信息集变化时，预测分布也会变化。今天只知道历史销量，明天又知道促销政策、天气预报或竞争对手价格，$y_t \mid I$ 的分布就应该更新。预测不是脱离信息条件地报一个数字，而是说明“在我们目前知道这些信息时，未来可能怎样”。

这个观点很重要。业务决策并不只关心“最可能是多少”，还关心“最坏可能有多坏”“超过某个阈值的概率多大”“需要准备多少安全库存”。如果只输出一个数字，很多风险信息会被压扁。

因此，预测评价不是只评价模型是否猜中一个数，而是评价它是否能支持行动。库存、产能、航班、免签政策、医疗资源和金融风险，都需要把未来的不确定性转化成当前可以执行的选择。

6.3　点预测

点预测给出一个确定值，记作 $\hat{y}_t$。它直观、易沟通，也便于用误差指标比较模型。短期、稳定、风险较低的场景中，点预测往往足够。例如一个成熟产品明天的常规销量，或者稳定门店下周的补货需求，都可以先用点预测作为基准。

点预测的局限同样明显。它不告诉我们预测可能错多少，也不告诉我们不同错误方向的概率。如果库存系统只知道“明天预计卖 1200 件”，却不知道 1000 到 1500 件的概率范围，就很难设定安全库存。

我们把点预测比作一顶“帽子”：$\hat{y}_t$ 是帽子上露出来的一个数字，帽子下面其实藏着完整的不确定性分布。只看帽子上的数字，沟通很方便，却容易忘记未来可能落在中心值两侧，也可能出现小概率但高损失的尾部情形。

点预测适合建立共同语言，也适合做第一轮模型筛选。但如果决策本身依赖风险范围，点预测只能作为入口，不能作为完整答案。

6.4　概率预测

概率预测输出未来值的可能范围和概率。它回答的不是“明天一定卖多少”，而是“明天销量落在不同区间的概率分别是多少”。对于库存、能源、金融、医疗资源配置等场景，这种信息比单个点更接近决策需求。评价这类分布预测时，严格适当评分规则提供了重要原则：模型应因诚实表达自己的预测分布而得到激励 [Gneiting and Raftery, 2007]。

概率预测可以有不同表达层次。最完整的是给出整条预测分布，说明 y_{t+1} 可能取哪些值以及概率如何分布；更简洁的做法是给出若干分位数或预测区间，例如 95% 区间只需要中位数、2.5% 分位点和 97.5% 分位点。还有一些场景只关心分布的一部分，例如左尾损失、超过库存阈值的概率或低于收入目标的概率。

Chronos 练习给了一个很实用的分位数例子。如果设置 `quantile_levels = [0.1, 0.9]`，模型输出的是 10% 分位数和 90% 分位数，中间覆盖的是 80% 预测区间，而不是 90% 区间。要得到更宽的区间，应选择更低和更高的分位数；要得到更窄的区间，则选择更接近中位数的分位数。报告预测区间时，必须明确分位数水平和覆盖率，不能只画阴影区域。

在库存管理中，概率预测可以支持服务水平、期望缺货和持有成本计算。在电力系统中，概率负荷预测可以帮助配置备用容量并降低调度风险。在宏观和政策场景中，概率分布能呈现不确定性，避免把一个目标值误读成确定承诺。

从点预测转向概率预测，既是技术问题，也是组织认知问题。很多管理者习惯听一个确定答案，但高质量预测往往应该明确表达不确定性。

食品成本、库存和服务水平等例子说明，点预测给出一个代表值，概率预测给出可能范围。当错误方向的成本不同，或者极端情形会造成较大损失时，预测分布比单个数字更有用。

例如，校医院准备退烧药时，点预测可能告诉我们下周平均需求是 6000 盒；概率预测进一步告诉我们，为了达到 95% 的服务水平，可能需要准备 7500 盒。前者回答“中心位置”，后者才直接连接到备货决策。家庭买退烧药、学校食堂备菜、ATM 放现金和仓库补库存都是同一类问题：只看一个平均值容易低估极端需求，按预测区间和服务水平决策才更接近真实成本。

能源预测进一步说明概率预测为什么重要。风电和太阳能的输出受天气影响很大，平均发电量不足以保证电网稳定。一个风电企业如果承诺未来几天供电，却遇到无风天气，就可能不得不用高价火电补足缺口；高耗电制造企业如果遭遇突然断电，生产损失可能极高。此时企业关心的不是“平均能发多少电”，而是不同时间段电力缺口的概率、缺口规模和备用容量安排。

案例：罕见药需求预测，为什么不能只看平均值

校医院准备退烧药的例子可以继续往前推一步：如果准备的不是常用退烧药，而是低频、救急、有效期有限、供应链又很特殊的药品，点预测就更不够用了。很多医疗物资的需求不是每天稳定发生，而是长时间为零，突然出现一两个病例；一旦没有药，后果又非常严重。这类问题的核心不只是“下周平均需要多少”，而是“在多大概率下不能断供”“断供一次的代价是多少”“多备一支药的过期和占用成本是否可接受”。

蛇毒血清就是典型案例。世界卫生组织把蛇咬伤中毒列为被忽视的公共卫生问题，并指出许多地区蛇咬伤报告不足，导致真实需求被低估；需求被低估又会削弱企业生产动力，进一步造成安全、有效、可负担蛇毒血清的短缺。WHO 也强调建立推荐血清储备，以稳定供给和需求。这个链条非常像一个预测评价问题：如果医院或地区卫生系统只根据过去几年登记病例的平均数备货，就会系统性低估那些没有进入医院统计的真实需求；如果只看全国总量，又会忽视毒蛇分布、农忙季节、雨季、山区交通、县域急诊能力和不同蛇种血清适配性的差异。

对蛇毒血清来说，好的预测不是一个全国平均点预测，而应是分地区、分季节、分品类的概率预测。我们关心的指标也不应只有 MAE 或 MAPE，还应包括服务水平、缺货概率、到院后可用时间、过期报废率和预期短缺损失。一个县医院一年可能

只遇到少数几例毒蛇咬伤，但这并不意味着可以不备；反过来，每个点都大量囤货也会造成过期浪费。更合理的做法，是用历史病例、基层报告、蛇种地理分布、气候、农事活动和转运时间来估计风险分布，再决定中心医院、县医院和区域储备点分别承担多少库存。

甲状腺癌需求展示了另一种医疗预测问题。它不是典型罕见病，但治疗资源带有明显的专科性和派生需求。国家卫健委《甲状腺癌诊疗指南（2022 年版）》指出，甲状腺癌是头颈部常见恶性肿瘤，分化型甲状腺癌占主要部分；美国国家癌症研究所也列出手术、放射性碘治疗、甲状腺激素治疗、靶向治疗等治疗方式。也就是说，甲状腺癌需求不是只预测“新增患者数”，还要沿着诊疗路径预测手术量、病理分型、是否需要放射性碘、后续复查、长期甲状腺激素用药和少数晚期患者的靶向治疗需求。

放射性碘治疗尤其能说明需求预测的复杂性。CDC 资料显示，碘-131 可用于诊断和治疗甲状腺癌，半衰期约 8 天。半衰期意味着它不是普通仓库里可以随意长期摆放的商品：采购、配送、核医学科排班、隔离病房、患者准备和治疗窗口都要对齐。如果预测过高，药品活度衰减、病房和人员安排会浪费；如果预测过低，患者治疗可能延期。此时评价指标就要从“预测病例数准不准”扩展到“是否减少等待时间、是否减少过期损耗、是否保证高风险患者优先治疗、是否在资源约束下维持公平可及”。

这两个例子说明，医疗需求预测的目标不是追求表格上一个最小误差，而是把不确定性翻译成可执行的资源配置。蛇毒血清强调低频高损失、地理异质性和应急储备；甲状腺癌强调诊疗路径、专科资源和长期随访。它们都要求我们在评价模型时加入业务损失：低估会造成延误甚至生命风险，高估会造成药品过期、资金占用和医疗资源浪费。概率预测、分位数预测和服务水平，比单一的点预测更接近这类决策的真实需要。

6.5　概率预测为什么难以落地

概率预测的困难不只在技术，也在组织认知。很多管理者更习惯听确定指令：“明天到底卖多少件？”如果销售员说“明天有 70% 的概率卖 1000 到 1500 件”，听起来像是不敢承担责任；另一个人拍胸脯说“保证卖 2000 件”，反而可能更容易被相信。

这种偏好会让组织把计划、目标和预测混在一起。计划可以写“增长 5% 左右”，目标可以要求“尽力达到 2000 件”，但预测应该说明不确定性：中心值是多少，区间有多宽，低于关键阈值的概率多大。一个成熟的管理者在听到点预测时，应继续追问：方差是多少？下行风险在哪里？如果预测错了，成本由谁承担？

6.6　预测步长与可评价性

预测步长不是技术细节，而是评价设计的一部分。短期预测通常拥有更多可用信息，长期预测则更容易遇到机制变化。预测一个学生明天或下周的学习状态，和在一年级时预测十二年后的高考结果，是完全不同的信息问题。前者可以使用近期成绩、

作业和课堂表现；后者几乎没有足够稳定信号。

远期预测还会触发反馈效应。青少年近视率如果从 1983-2023 年的历史趋势直接外推到 2050 年，可能得到过度悲观甚至荒谬的结果；但正因为这种风险被看见，学校、家庭和政策也会改变户外活动、照明、作业和用眼习惯。预测本身可能影响未来行为，所以评价长期预测时不能只看历史曲线是否拟合得漂亮。

因此，一个评估报告应按步长拆开误差。$T+1$ 准确不代表 $T+12$ 准确，短期补货模型也不一定适合年度预算。真正严肃的比较要说明模型在哪些步长上有效，在哪些步长上只是延长了过去趋势。

6.7　点预测误差

点预测评估的基本对象是误差：

$$e_t = y_t - \hat{y}_t \tag{6.1}$$

其中 y_t 是真实值，$\hat{y}_t$ 是预测值。误差指标的作用，是把一组误差汇总成可以比较、解释和优化的数字。

常用指标包括：

- **ME（Mean Error）**：平均误差，用来检查系统性偏差。正负方向有解释意义。
- **MAE（Mean Absolute Error）**：平均绝对误差，表示平均偏离多少，解释直接，对极端值不太敏感。
- **MSE / RMSE（Mean Squared Error / Root Mean Squared Error）**：平方误差及其平方根，会更重地惩罚大误差。
- **MAPE（Mean Absolute Percentage Error）**：平均绝对百分比误差，便于跨规模解释，但真实值接近 0 时不稳定。
- **sMAPE（symmetric Mean Absolute Percentage Error）**：对称百分比误差，在预测值和真实值都可能较小时比 MAPE 稳定。
- **MASE（Mean Absolute Scaled Error）**：尺度化误差，用模型误差和简单基准误差比较，适合跨序列评估。

没有一个指标适合所有问题。库存、能源和金融场景中，大误差、方向性偏差和相对误差的重要性不同，指标选择必须回到业务损失。

本章只保留这些指标的功能性解释。下一章会在 notebook 层面计算 ME、MAE、RMSE、MAPE、sMAPE 等点预测指标，并展示如何把误差表转化为模型比较。

6.8　真实未来无法提前评价

预测评估的根本困难在于：真正关心的未来还没有发生。如果今天要预测下一季度销售额，我们暂时没有 y_t，也就不能直接计算误差。

机器学习中的随机训练集和测试集划分，在时间序列中通常不适用。时间顺序承载了因果和信息结构。如果把未来样本打乱放进训练集，再评价过去的样本，模型就使用了本不该知道的信息。

因此，时间序列评估依赖“过去的未来”。我们在历史数据中选择某个切分点，用切分点以前的数据训练模型，再预测切分点之后的一段历史。那段历史对模型来说是未来，但对评估者来说真实值已经可见。

这也是评估不能直接针对“今天刚做出的未来预测”的原因。真正的未来在预测发生时还没有真实值，所以只能在历史中构造一个当时不可见、现在已知的未来。

数据泄露是评价失真的核心风险。如果一个预测任务声称站在 2023 年春天预测未来两年旅游恢复，却在建模时使用了 2024 年或 2025 年已经发生的数据，那么误差再低也不能说明模型真的有预测能力。它只是把已经知道的未来重新包装成了模型结果。

最严格的评价方式是真正的实时评价：现在提交预测，等未来发生后再比较。旅游恢复预测竞赛就是这种设计。多个团队在同一时间提交中国到主要目的地国家的出境游恢复预测，之后等真实客流数据可得再统一评价。这类设计耗时，但信息边界最干净，也能检验模型、专家假设和外部信号整合是否真的经得起未来检验。恢复预测需要综合航班、住宿、海外活动、签证、交通流量和地缘政治等信号，而不是只把疫情前趋势向后外推。

6.9　时间序列交叉验证

时间序列交叉验证（time series cross-validation）常见两种窗口设计。

滚动窗口保持训练窗口长度相对固定，每次向前移动。它适合数据生成机制可能变化、旧数据参考价值有限的场景。

扩展窗口（expanding window）从较早时间开始训练，每一折增加更多历史数据。它适合历史信息持续有用、样本越多越稳定的场景。

每一折都应明确三件事：

- 训练窗口：模型允许看到的过去。
- 预测步长：模型要预测未来多少期。
- 测试窗口：用来计算误差的已发生未来。

如果业务真正关心未来 60 天，就不要只评价 $T+1$。多步预测误差通常随步长增加而累积，应分步长汇总，也可以报告整体平均表现。

旅游预测竞赛的例子说明，公平比较需要共同的数据、共同的预测步长和共同的误差指标。多个国家、多个时间段或多个模型放在一起比较时，必须保证每个模型看到的信息边界一致，否则误差排名没有意义。

可以把时间序列交叉验证理解为三层时间：

名称	含义	评价中的角色
历史的历史	切分点之前的数据	训练模型允许看到的信息
历史的未来	切分点之后、现在已经发生的数据	测试模型预测能力
真正的未来	当前时点之后尚未发生的数据	最终业务目标

回测的逻辑是假设历史中的规律会延续到真正未来。这个假设在很多稳定业务中有效，但不能自动成立。

衬衫销量例子可以帮助理解这个过程。假设我们手里有过去五六十天每天卖出的衬衫数量，今天想预测未来六天。评估时可以先把历史末端的六天暂时“遮住”，只用更早的蓝色历史训练模型，再预测这六天已经发生但对模型隐藏的浅色区间。接着把切分点向前或向后滚动，重复多次。这样得到的不是一次漂亮曲线，而是一组不同预测起点下的误差证据。

6.10　当历史评估会失效

历史回测的弱点在于，它默认未来与过去仍然相关。当结构突变发生时，这个假设会变得脆弱。

新冠后的出境游恢复就是典型例子。疫情前，中国到加拿大、日本、美国等目的地的客流有趋势和季节性；在疫情冲击后，序列断崖式下跌，历史规律被打断。一个模型如果只在疫情前数据上回测，可能表现很好，但这并不保证它能预测疫情后的恢复路径。

这种场景下，评价要多问几个问题：

- 回测窗口是否覆盖了类似冲击？
- 训练数据中的历史机制是否仍然存在？
- 是否需要构造“无冲击基准”作为参照？
- 是否有新的恢复信号，例如航班运力、搜索指数、签证政策和住宿数据？
- 最终评价是否需要等真实未来发生后再做？

复苏预测的难点不只是误差计算，而是评价对象本身变了。我们不再只是外推一条平稳的历史曲线，而是在判断系统什么时候恢复、恢复到什么水平、以什么路径恢复。此时，评价必须同时检查基准预测、当前信号、专家假设和最终真实结果。

案例：中国出境旅游复苏预测

2023 年春天，中国出境游重新启动。这个时间点很适合理解“真正预测”和“事后解释”的区别。UNWTO 当时把中国重新开放称为全球旅游复苏的最后一块拼图，因为疫情前中国已经是全球最大的出境游市场。2019 年，中国游客的国际旅行和境外消费对目的地国家、航空公司、酒店、景区、零售和签证服务都有直接影响。

预测中国游客何时回到加拿大、日本、美国、韩国、泰国等目的地，不只是旅游网站关心的问题，也会影响航线恢复、酒店招聘、目的地营销、签证政策和外交安排。

如果我们站在 2023 年 7 月做预测，困难并不在于“历史数据不够长”。疫情前的数据很长，而且趋势和季节性很清楚。困难在于 2020 年之后的断崖式下跌切断了历史机制：疫情前模型可以很好地解释正常时期，却不能自动回答复苏路径。中国到加拿大的例子很直观：疫情前客流有季节性和长期增长，疫情期间几乎停摆，2023 年初开始抬头，但这几个月的斜率不能机械外推。很多人仍然不敢或不能出国，航班也没有完全恢复，机票价格、签证政策和地缘政治都会改变不同目的地的恢复速度。

一个可用的复苏预测应先构造“无疫情基准”：如果没有疫情，2020 年以后中国到各目的地的客流大概应该走到哪里。这个基准不是事实，而是正常世界里的参照线。接着观察 2023 年早期恢复窗口，看现实客流离这条参照线还有多远，以及恢复动能有多强。然后设定一个终点：到预测期末，某个目的地是接近无疫情基准，还是因为航班、签证、价格、地缘政治或旅游偏好变化而系统性低于基准。

这就引出恢复系数。我们可以把不同目的地的恢复系数放在 0 到 1 之间：接近 1 表示接近无疫情基准，较低则表示仍有结构性阻碍。长线目的地如美国、加拿大，可能受到航班供给、票价和签证因素影响；东亚、东南亚的一些目的地可能恢复更快；体量很小的目的地即使误差大，对总体决策的影响也可能有限。恢复系数不是随意拍脑袋，而是把航班运力、签证政策、搜索指数、目的地偏好和地缘政治信息转成可进入预测流程的先验。

最后要把当前状态和终点连接起来。连接方式可以是线性曲线、二次曲线、logistic 曲线，也可以由多条曲线组合。一个更完整的系统还会使用分层预测和预测调和：先预测中国到美洲、亚洲、欧洲等区域的总量，再和美国、加拿大、日本、韩国等目的地的单独预测协调起来，避免各国误差简单累积。公开研究中，RISE 把这种思路概括为“初始预测、终端预测、恢复曲线预测”三部分，并强调在结构突变和高不确定性下，要把模型预测与专家判断结合起来。

后来公布的旅游数据也说明，复苏并不是一个统一的全球开关。UN Tourism Barometer 2025 年 11 月摘要 显示，国际旅游在 2024 年基本恢复到疫情前水平，但到 2025 年 1-9 月，亚太地区国际到访仍约为 2019 年同期的 90%。也就是说，全球平均恢复不等于每个区域、每个目的地、每条航线都恢复。评价这种预测时，不能只问总体误差多小，还要问主要目的地是否预测准、恢复拐点是否判断对、区间是否覆盖真实路径，以及预测是否真的帮助了航班、签证、营销和酒店容量安排。

6.10.1　复苏预测的 RISE 拆解

中国出境旅游复苏预测可以作为结构突变场景的完整案例。2023 年春天，如果要预测未来一到两年中国到 20 个主要目的地国家的出境游客量，疫情前的季节性和趋势仍然有参考价值，但疫情期间的断崖式下跌已经切断了普通历史外推。直接用疫情前模型往后延长，或者只把 2023 年初几个月的恢复斜率线性外推，都可能误导决策。

我们把一种实用思路概括为 RISE。它不是一个单一黑箱模型，而是一套复苏预

测拆解方式：

- 先构造“无疫情基准”：如果疫情没有发生，2020 年之后各目的地客流应该怎样发展。
- 再观察恢复窗口：例如 2023 年 2 月到 7 月的早期恢复动能，但不把这几个月机械外推。
- 设定恢复终点：判断两年后系统会接近无疫情基准、低于基准，还是因政策和偏好变化出现目的地差异。
- 加入有信息量的主观先验：航班运力、签证政策、地缘政治、搜索指数和游客偏好都可能影响不同国家的恢复系数。
- 选择恢复曲线：直线、二次曲线和 logistic 曲线都可能描述恢复路径，不同目的地可以组合使用。

这个案例还用到了分层预测、预测调和和组合预测。中国到美国、加拿大、日本、韩国等目的地既可以逐国预测，也可以先预测区域或总体，再向下分配。预测调和保证各层级之间一致，组合预测则把不同基准模型、恢复曲线、外部信号和恢复系数合成一个更稳健的结果。RISE 方法把这种复苏预测拆解为初始预测、终端预测和恢复曲线预测三部分，用来处理疫情后旅游需求的结构突变 [Li and Ruan, 2026]。

评价方式也要改变。真正严格的做法是在 2023 年提交预测，等到 2025 年 3 月左右真实数据积累后再统一比较点预测和区间预测。这个设计耗时，但它守住了信息边界：参赛者提交预测时不知道未来两年会发生什么，因此评价更接近真实预测能力。

6.11　评价结果如何用于决策

评价结果至少要回答四个问题。

第一，模型是否优于简单基准。例如季节性朴素预测、上一期预测或移动平均。不能超过简单基准的复杂模型，很难获得业务信任。

第二，模型在哪些预测步长上表现好。一个模型可能 $T+1$ 很好，但 $T+12$ 很差。

第三，模型误差是否有系统性方向。长期高估会造成库存积压，长期低估会造成缺货或产能不足。

第四，模型最坏情况是否可接受。平均误差低不代表极端错误少，尤其在能源、金融和公共服务场景中。

还要问第五个问题：错误是否发生在真正重要的地方。旅游恢复预测中，如果一个小目的地误差较大，但总体客流和主要目的地预测准确，政策影响可能有限；如果加拿大、美国、日本、韩国等主要目的地严重偏误，航线、签证、营销和外交政策都会受到影响。

评价结果最终要转化成决策语言。模型 A 的 MAE 比模型 B 低 5%，是否足以支付更高 API 成本？某模型短期准确但长期偏差大，是否适合库存补货而不适合年度预算？这些问题比单纯排名更接近业务。

6.12　公平比较

无论是课程实验、企业模型选型，还是预测竞赛，公平比较都需要共同规则。预测准确性比较本身也有统计检验传统，例如 Diebold-Mariano 思路提醒我们，模型差异不能只靠一次误差表的视觉印象判断 [Diebold and Mariano, 1995]。至少应保持：

- 相同训练截止时间。
- 相同预测目标和预测步长。
- 相同测试集和误差指标。
- 相同是否允许使用外部数据。
- 相同是否允许联网搜索或调用工具。
- 相同是否允许人工修正。
- 不允许看见测试期真实值后修改模型、prompt 或参数。

大模型预测尤其需要明确这些规则。一个模型如果可以联网搜索，而另一个模型不能；一个团队可以在测试集公布后改 prompt，而另一个团队不能，那么比较结果就不再反映预测能力。评价设计必须先定义信息边界，再计算误差。

6.13　小结

预测评估的核心不是找一个最好看的指标，而是把预测问题放回时间和决策之中。点预测适合清晰沟通，概率预测适合风险决策；历史回测提供必要证据，但不能完全替代实时评价和未来监控。

一个可信的预测评价至少要守住三条底线：不泄露未来信息；比较规则公平一致；评价指标服务于真实决策。一个可用的预测系统，应当同时报告误差大小、偏差方向、步长表现、不确定性和适用边界。

6.14　练习

1. 为一个零售补货问题选择两个误差指标，并说明为什么。
2. 设计一个 10 折时间序列交叉验证方案，明确训练窗口、测试窗口和预测步长。
3. 找一个 MAPE 不适用的业务场景，并提出替代指标。
4. 解释为什么随机划分训练集和测试集会破坏时间序列评估。
5. 将同一模型的 $T+1$、$T+3$、$T+7$ 误差分开报告，并讨论差异。
6. 找一个存在数据泄露的预测评价设计，说明未来信息是如何进入模型的。

7. 设计一个旅游恢复预测竞赛，明确提交时间、数据边界、预测步长和评价指标。
8. 解释为什么新冠后旅游恢复不能只依赖疫情前回测结果判断模型好坏。
9. 为两个 AI 预测模型设计公平比较规则，说明是否允许联网、外部数据和人工修正。
10. 选择蛇毒血清、放射性碘治疗或另一种低频高损失医疗物资，设计一个评价方案，至少包含服务水平、缺货损失和过期浪费三个指标。

第 7 章

点预测评估

本章把上一章的评估思想落到可运行流程中：用航空乘客数据比较 TimeGPT 模型，在留后测试和滚动窗口中计算 ME、MAE、MSE、RMSE、MPE（Mean Percentage Error）、MAPE 和 sMAPE。重点不是记住每个指标公式，而是理解一套可复现的评估流程如何防止我们被一次好看的预测曲线误导。

7.1 学习目标

完成本章后，你应该能够：

- 使用留后法评价一个固定预测步长的点预测。
- 使用滚动窗口评价 $T+1$ 到 $T+H$ 的多步预测表现。
- 解释 ME、MPE、MAE、RMSE、MAPE 和 sMAPE 分别回答什么问题。
- 比较不同 TimeGPT 模型在同一数据上的误差。
- 说明“历史中的未来”为什么是滚动评估的核心思想。
- 将评估表转化为业务解释，而不是只报告数字。

7.2 AI 输出也需要评价

本章练习的一个重要背景是：预测结果不一定由我们亲手训练的模型产生，也可能来自 TimeGPT、DeepSeek、Darts、StatsForecast 或其他 AI 平台。模型越像一个自动化平台，越容易让使用者误以为“输出了曲线”就等于“可以使用”。点预测评估的任务正好相反：无论预测来自专家、统计模型、深度学习模型还是 API，都必须放到同一套评价流程里接受检验。

我们把这个原则说得更直接：预测不管是人工做出来的、智能做出来的，还是人工加智能做出来的，都要回头看真实值。点预测最基本的对象是 $e_t = y_t - \hat{y}_t$。如果误差平均为正，按这个符号约定说明模型倾向低估；如果误差平均为负，说明模型倾向高估。误差的方向、大小、波动和极端值，才是评价曲线是否可信的起点。

因此，本章的重点不是证明某个 AI 模型一定更好，而是训练一种工作习惯：先规定数据切分、预测步长、频率、指标和基准，再比较模型。只有这样，`timegpt-1`、`timegpt-1-long-horizon` 或其他模型之间的差异才有可解释性。

7.3　数据与任务

示例 notebook 使用 `air_passengers_with_id.csv`。数据至少包含三列：

- `unique_id`：序列编号。
- `ds`：日期，航空乘客数据是月度频率。
- `y`：目标变量，也就是乘客数量。

读取后先统一排序：

```
df = pd.read_csv("data/air_passengers_with_id.csv")
df["ds"] = pd.to_datetime(df["ds"])
df = df.sort_values(["unique_id", "ds"]).reset_index(drop=True)
```

评估之前应先画图检查序列。航空乘客数据有明显趋势和季节性，如果模型输出完全平滑的曲线，或者完全忽略季节波动，误差表之外也能看出问题。

近视度数、用电量和销量趋势等例子都说明：评价点预测之前，先要理解序列本身。一个模型如果连明显趋势、周期或异常点都没有反映出来，即使某个汇总指标还可以，也需要回到图形和分步长误差中检查。

用 TimeGPT 对航空乘客数据做快速预测时，可以先整理 `unique_id`、日期列和 `y`，设置月度频率和 $H = 12$，再调用模型生成未来 12 期曲线。TimeGPT 的意义在于把大规模预训练时间序列模型封装成可调用服务 [Garza et al., 2024]。这个示例的目的不是当场证明 TimeGPT 一定最好，而是先看到 AI 平台如何把一个标准格式的数据表转成可视化预测。真正判断它好不好，仍然要回到本章的留后法、滚动窗口和误差指标。

7.4　留后法

留后法把最后一段历史数据当作测试集。例如保留最后 12 个月：

```
H_TEST = 12
train = df.iloc[: -H_TEST]
test = df.iloc[ -H_TEST:]
```

模型只能使用 `train`，然后预测未来 12 期。预测结果与 `test` 按 `unique_id` 和 `ds` 合并后计算误差。这个方法简单、直观，适合演示和快速比较。

留后法的局限是只评价一个切分点。如果最后 12 个月恰好是异常时期，结果可能过于悲观或乐观。因此正式评估通常还需要滚动窗口。留后法适合快速检查，但不应成为唯一证据。

7.5　滚动窗口评估

滚动窗口把评估重复多次。每一次选择一个历史切分点，用切分点之前的数据预测接下来 H 期，然后向前滚动到下一个切分点。示例 notebook 中使用 `cross_validation` 生成多次窗口，并给每个预测样本标注步长：

```
cv["h"] = cv.groupby(["unique_id", "cutoff"]).cumcount() + 1
```

这样就可以分别计算 $T+1$、$T+2$、一直到 $T+H$ 的平均误差。这个拆分非常重要。模型短期预测好，不代表长期预测好；长期误差也不应掩盖短期可用性。

滚动窗口的直觉可以称为“历史中的未来”（past futures）。站在某个历史时点，我们把那一刻之后已经发生、但在当时还未知的数据当作模拟未来。这样既不会偷看真正未来，又能在已有数据里反复检验模型。这个方法隐含一个前提：历史规律在未来仍有一定延续性。如果疫情、政策冲击或结构性变化让历史和未来脱节，滚动评估仍然有用，但它只能说明模型在相似环境下的表现，不能保证它能处理全新制度或全新行为。

一个具体做法是：先保留历史末端一小段，例如未来 6 天或 12 个月，模拟站在更早时点做预测；计算一次误差后，再移动预测起点，重复多组窗口。最后汇总这些窗口的 MAE、sMAPE、RMSE 或偏差方向。这样做比只看最后一个测试窗口更稳健，也能发现模型是否只在某个偶然时期表现好。

7.6　指标解释

ME 和 MPE 用于检查偏差方向。ME 为正，表示真实值平均高于预测值，模型整体低估；ME 为负，表示模型整体高估。MPE 是百分比版本，便于解释偏差比例。

MAE 表示平均绝对偏离，单位和目标变量一致。它适合回答“平均错多少人、多少件、多少元”。

MSE 和 RMSE 对大误差更敏感。RMSE 回到原单位，便于解释。如果业务特别害怕极端错误，例如电力负荷调度或现金流短缺，RMSE 往往比 MAE 更有警示性。

MAPE 用百分比衡量相对误差，适合向管理层沟通。但真实值为 0 或很小时，MAPE 会失真，因此在零销量、低销量和跨序列比较中要谨慎使用 [Hyndman and Koehler, 2006]。

sMAPE 对高估和低估更对称，常用于预测竞赛和跨模型基准。它也不是万能指标，低销量、零销量和间歇性需求仍需谨慎处理。

点预测指标适合回答“平均错多少”“是否系统性高估或低估”。如果业务真正关心风险范围、服务水平或安全库存，就需要进一步进入概率预测和区间覆盖评估。

7.7　指标冲突

不同指标经常给出不同排序，这是正常现象。一个模型可能 MAE 较低，说明大多数时期平均偏离较小；同时 RMSE 较高，说明少数时期出现了很大的错误。一个模型也可能 MAPE 较低，但在高峰期绝对误差很大。我们让你比较多个指标，目的就是看到：指标不是装饰，指标代表业务偏好。

指标选择也会影响“谁看起来赢”。较宽松的指标可能让某个模型显得提升明显，较严苛的指标则会暴露少数严重失误。正式报告不能先看哪个模型结果好，再反过来挑支持它的指标；应先根据业务损失说明为什么选择 MAE、RMSE、MAPE、sMAPE 或基准尺度化误差。

如果业务关心日常稳定性，MAE 可能更自然；如果业务害怕极端错误，RMSE 更重要；如果需要跨产品、跨地区汇报，百分比误差更便于沟通；如果真实值接近 0，百分比误差就需要谨慎使用。不同选择背后对应的是不同的业务风险偏好。

7.8　比较模型

notebook 比较了 `timegpt-1` 和 `timegpt-1-long-horizon`。比较时必须保证数据切分、预测步长、频率和指标一致。否则误差差异可能来自评估设计，而不是模型能力。若要进一步判断两个模型误差差异是否稳定，还可以使用预测准确性比较的统计检验思想 [Diebold and Mariano, 1995]。

一个合理的比较表至少包括：

- 留后测试的整体误差。
- 滚动窗口下每个预测步长的误差。
- 各步长平均后的总体表现。
- 偏差方向，也就是 ME 或 MPE。

排序时可以先看业务最关心的指标。例如运营排班可能优先看短期 MAE；年度预算可能更关心 12 期总量偏差；风险控制场景可能更关心 RMSE 或预测区间。

本章练习的设计可以推广成一套通用评估模板。先把 AirPassengers 整理成 `unique_id`, `ds`, `y` 三列，再让 TimeGPT 和 long-horizon 版本分别预测同一组 horizon，例如未来 5 步。随后把两个模型的预测值和测试集真实值合并，分别计算 MAE、RMSE、MAPE、sMAPE、ME 和 MPE。输出表中的每一行不是“模型身份”的装饰，而是一个待比较的预测者：它也可以换成专家预测、DeepSeek 预测、统计模型预测或组合预测。

这个练习最重要的观察是：不同指标可能给出不同排序。某个模型在 MAE 上更好，另一个模型可能在 RMSE 或 sMAPE 上更好；短期 horizon 表现好的模型，长期 horizon 未必更好。因此评估表应同时按模型和 horizon 展开，而不是只给一个总分。

简单基准是比较模型时的最低门槛。例如“今年同月等于去年同月”或“下一期等于上一期”看起来朴素，但它们能检验复杂模型是否真的带来增量价值。不能超过基

准的高级模型，通常不应直接进入业务流程。

模型比较还要注意信息边界。所有模型必须站在同一个历史时点，只能使用当时可获得的信息。如果一个模型在评估时隐含使用了未来新闻、未来政策或后来才知道的修正数据，它的误差表会很好看，但这种结果不能代表真实预测能力。

7.9　从误差表到业务解释

评估表不能只停留在“模型 A 的 MAE 更低”。需要继续追问：

- 改善幅度是否足够大，值得引入更复杂的模型或 API 成本？
- 模型是否在高峰期和低谷期都稳定？
- 长期低估或高估会造成什么业务后果？
- 是否存在某些窗口误差特别大，暗示结构变化或异常事件？
- 模型输出是否能按时进入业务系统？

例如航空乘客预测中，季节高峰低估可能比淡季高估更严重，因为它会影响机队、人员和票价策略。指标解释要服务于这种具体判断。

旅游复苏预测是更困难的评价场景。预测者站在 2022 年或 2023 年时，并不知道未来一年国际旅行会以什么路径恢复。此时评价不只是计算 $y_t - \hat{y}_t$，还要考虑预测竞赛如何设定提交时间、何时揭晓结果、是否允许更新信息、是否按目的地或区域分层评价。这个例子提醒我们，点预测评估看起来是公式问题，实际是实验设计问题。

7.10　实操注意事项

不要把 API key 写入 notebook。使用环境变量更安全：

```
from nixtla import NixtlaClient

nixtla = NixtlaClient(api_key=os.environ["NIXTLA_API_KEY"])
```

评估前确认频率参数正确。月度起始频率常写作 `MS`。频率错误会导致未来日期错位，进而让合并和误差计算失效。

评估时保留原始预测表、合并后的评估表和汇总表。这样在结果异常时可以回溯到具体窗口、具体日期和具体预测步长。

使用 TimeGPT、DeepSeek 或其他云端 API 时，还要先判断数据是否可以上传。航空乘客公开数据适合教学演示；家庭收入、企业订单、客户交易、内部产能和未公开经营数据不应随意发送到外部服务。API 预测越方便，越要把数据边界、调用成本和 key 管理写进实验流程。

7.11　小结

点预测评估是一套流程：定义切分、调用模型、合并真实值、计算指标、按步长和业务场景解释结果。留后法提供快速检查，滚动窗口提供更稳健的证据。一个预测结果只有经过这种流程，才从“看起来合理”变成“可以被比较和讨论”。

7.12　练习

1. 将最后 6、12、24 期分别作为测试集，比较同一模型的 MAE。
2. 用滚动窗口分别报告 $T+1$ 到 $T+5$ 的误差，并解释误差是否随步长增加。
3. 找出 ME 和 MAE 同时较大的模型，判断它是系统性偏差还是波动误差。
4. 解释为什么一个模型 MAPE 低但 RMSE 高时，业务上仍可能有风险。
5. 为一个 AI 平台预测结果设计评估流程，说明数据切分、基准模型和主要指标。
6. 把 TimeGPT、TimeGPT long-horizon 和一个第三方预测结果放进同一张评估表，比较不同指标是否给出一致排序。
7. 将评估结果写成一段管理层可读的结论，避免只罗列指标。

第 8 章

金融预测智能体

金融市场（financial market）是预测模型最难也最有吸引力的应用场景之一。它有大量数据、即时反馈和明确收益指标，但也有噪声（noise）、反身性（reflexivity）、非平稳性（nonstationarity）和强竞争。本章以 AI 交易竞赛（AI trading competition）、企业文本预测和金融智能体为线索，讨论为什么金融预测不能只依赖通用智能，还需要领域数据、风险控制和制度约束。

8.1 学习目标

完成本章后，你应该能够：

- 说明金融市场为什么是 AI 预测能力的高压测试。
- 区分“会聊天的模型”和“能在市场中稳定决策的模型”。
- 解释领域数据、交易经验和风险控制在金融预测中的作用。
- 说明 K 线、文本、声音、视频和另类数据各自提供什么信息。
- 理解市场反身性如何改变预测和决策的关系。
- 识别实盘 AI 交易中的伦理、透明性和监管风险。

8.2 金融预测为什么特殊

金融预测和普通需求预测不同。需求预测的目标通常是降低误差，金融预测则直接面对收益、风险和竞争对手。市场是动态系统：如果某个信号被越来越多人使用，它的收益可能会下降；如果模型行为影响了其他参与者，预测对象本身也会改变。

金融数据也高度多源。价格、成交量、订单簿（order book）、新闻、财报电话会（earnings call）、社交媒体、政策事件和宏观指标都会影响资产价格。只看 K 线（candlestick chart），等于只看市场已经压缩后的结果；很多更早、更细的信号已经在压缩过程中丢失。机器学习资产定价研究也说明，非线性交互和高维预测信号可以显著改变传统收益预测问题的建模方式 [Gu et al., 2020]。

因此，AI 金融预测的核心不是让模型“猜涨跌”，而是构建信息、行为和收益之间的闭环。

8.3　AI 交易竞赛的启发

我们用 Nof1.ai 的 Alpha Arena 说明金融预测评估：多个大模型在相同数据、相同提示词和真实资金约束下交易，最终以账户余额和风险表现作为结果。这个设定把模型评测从静态问答推向动态市场。

这个案例更接近一个真实实验：实验者给多个主流大模型分别分配账户，每个账户有初始资金，让模型读取当前时间、账户状态、过去一段市场数据和可交易资产信息，然后输出买什么、买多少、持有还是卖出。模型没有为这个任务重新训练，而是通过相对统一的 prompt 进入交易流程。实验者再按模型指令执行真实加密货币交易。

案例：Nof1.ai 的 Alpha Arena

Nof1.ai 的 Alpha Arena 把 “AI 会不会做金融预测” 从一道问答题变成了一个真实交易问题。Nof1 团队 在第一季中给六个领先大模型各分配 1 万美元，让它们在 Hyperliquid 上交易加密货币永续合约（perpetual futures contract）；模型只能使用数值型市场数据和统一的 prompt/harness，不做任务专门微调，也不由人类在交易过程中代为判断。模型每隔几分钟读取市场价格、成交量、技术指标（technical indicators）、账户余额、持仓、收益和 Sharpe ratio（夏普比率），然后输出结构化行动：做多还是做空、买哪种币、数量、杠杆、止损（stop-loss）、止盈（take-profit）、信心分数和退出计划。

这个实验的叙事价值在于，它把金融预测的几个关键层次同时摆在台面上。过去量化交易的门槛很高：机构需要低延迟服务器、靠近交易所的机房、快速算法、C/C++ 工程能力和专业交易团队。普通投资者面对机构，往往缺少数据、模型、风控和执行系统。大模型介入以后，门槛似乎变低了：我们可以把市场状态整理成 prompt，让模型像交易员一样给出下一步行动。可是，门槛降低并不等于风险消失，它只是把风险从 “不会写算法” 转移到 “模型是否真的理解交易”。

在 nof1.ai 的页面上，排行榜不只显示谁赚了多少钱，还展示风险调整后的指标、交易次数、账户状态和模型输出。`Model Chat` 尤其值得看：它暴露了模型每一次如何理解市场数据、如何解释自己的仓位、如何设置止损和止盈。这样我们能看到一个很重要的事实：同样的提示词，不会产生同样的交易员。有的模型更频繁交易，有的模型更耐心；有的模型仓位集中，有的模型更愿意分散；有的模型文字解释很漂亮，但交易判断并不一定更好；有的模型看起来很积极，却可能被手续费和滑点侵蚀收益。

这也和散户、机构和量化交易的关系连在一起。一个公开看板会让普通投资者看到原本只有机构内部才有的策略曲线、交易频率和风险表现，它像量化投资里的 “自动驾驶”：系统建议买什么、卖什么、买多少、何时退出，人仍然可以接管。但这个比喻必须加上刹车。看到某个模型短期领先，不等于可以机械跟买；如果很多人都跟随同一个公开模型，模型交易会影响市场，市场反过来影响模型表现，原来的预测问题就会变成反馈系统问题。

Alpha Arena 也说明，金融 AI 的评价不能只看最终收益。我们至少要同时看 PnL（Profit and Loss）、最大回撤（maximum drawdown）、Sharpe ratio、交易次数、手续费、杠杆、持仓集中度、止损执行和策略一致性。更进一步，还要检查信息边界：

模型是否联网，是否读到外界对自己表现的评论，是否有历史行动记忆，prompt 是否让不同模型承担了同样约束。Nof1 团队也明确提醒，第一季不是为了用一次短期实验宣布永久赢家，而是为了观察模型在真实、动态、有风险的环境中暴露出的行为差异和失效模式。

因此，这个案例最适合被当作金融智能体的原型，而不是投资建议。它告诉我们，大模型可以从 “解释市场” 走向 “参与决策”，但真正可用的金融 AI 必须把数据输入、特征工程、模型推理、交易执行、风险控制、审计日志和人工接管放在同一个系统里。金融预测的难点从来不是让模型说一句 “看涨” 或 “看跌”，而是让它在成本、风险、约束和反馈中持续做出可审计、可中止、可评价的行动。

这种实验有几个启发。

第一，同样的提示词不代表同样的决策。模型的训练数据、架构和安全偏好会影响风险反应。有的模型可能频繁交易，有的模型更像量化策略。

第二，收益不是唯一指标。最大回撤、单笔极端亏损、交易频率和风险暴露同样重要。一个模型短期收益高，但承担了不可接受的尾部风险，并不能说明它更适合真实资产管理。

这也是概率预测在金融中尤其重要的原因。金融决策通常不只关心收益分布的中心位置，而是关心左尾：亏损超过某个阈值的概率有多大，最坏一段尾部的平均损失是多少。Value-at-Risk 和 Expected Shortfall 就是这类风险摘要。完整预测分布包含这些信息，但实际风控报告往往会把尾部风险单独列出，因为它直接决定仓位、保证金和止损边界。

第三，市场反馈会改变模型表现。如果很多人开始跟随某个模型的交易，原有策略可能被拥挤交易削弱，甚至产生反向机会。

第四，交易频次本身就是风险。一个模型可能看起来很勤奋，频繁买卖，但加密货币交易也有手续费和滑点（slippage）。过度交易会消耗收益，甚至把原本略有优势的预测变成亏损策略。另一个模型交易次数少，却通过较稳健的分散持仓保持更好结果，这说明预测能力必须和执行成本一起评价。

这类竞赛还提醒我们，评估大模型交易能力时必须控制信息边界。提示词是否完全一致，模型是否可以联网，模型是否能读到网上关于自己策略表现的评论，都会影响结果。如果一个模型在交易过程中不断看到 “某策略表现很好” 的外部反馈，它可能强化已有行为；这时我们评估到的就不只是模型的原始预测能力，而是模型、搜索、舆论和市场共同构成的反馈系统。

我们还要提醒你，这类实验不构成投资建议。看到某个模型短期盈利，不能直接得出 “跟着它买就能赚钱” 的结论。真实交易中，人会观察模型，模型可能观察外界反馈，市场又会对跟随行为作出反应。越是实时公开的策略，越容易从预测问题变成反馈系统问题。

8.4　领域知识的重要性

通用大模型擅长语言理解和生成，但金融预测需要更多领域先验。交易任务中，模型必须理解风险预算（risk budget）、仓位管理（position management）、止损、流动性、交易成本和市场冲击（market impact）。只会解释新闻，不等于能生成稳健交易策略。

领域知识至少体现在三层：

- 数据层：是否接入高质量行情、财报、新闻和另类数据。
- 表征层（representation layer）：是否把价格行为、波动、趋势、成交和事件转化为有效特征。
- 决策层：是否把预测转化为仓位和风险控制，而不是直接把方向判断当作交易。

课程中强调，金融预测不只需要智能，还需要算法和经验融合。一个有交易背景的数据团队，往往比只使用通用模型更清楚哪些信号可交易、哪些信号只是事后解释。

DeepSeek 等模型的金融表现可以说明：大模型的领域能力很大程度上来自训练数据和训练任务。如果一个模型在预训练或后续训练中接触过大量金融文本、交易数据和风控逻辑，它在金融任务中可能比通用聊天能力更强的模型更稳健。反过来，如果训练数据没有反映分散投资、风险预算和市场结构，模型给出的交易建议就可能很危险。

个人投资者和机构投资者的差异也说明了这一点。个人投资者往往依赖有限信息和短期判断，机构则更强调数据、模型、风控和执行系统。AI 工具可以降低信息处理门槛，但不能消除风险预算和交易纪律。

不同模型在交易实验中的行为差异，可以看作领域知识和训练数据差异的外显。有的模型会给出很长、很规范的文字解释，却缺少明确有效的交易判断；有的模型可能只押注最高自信资产，短期收益高但集中风险也大；有的模型偏好某类资产，可能与训练语料中的社交媒体叙事有关；还有的模型更重视分散持仓、止损和交易成本。模型会不会“像交易员一样思考”，取决于它是否在训练和对齐过程中接触过足够多的金融数据、仓位管理和风控逻辑。

8.5　从 K 线到高维信息空间

K 线图浓缩了市场交易后的结果，但它不是全部信息。价格变化背后可能有公司基本面、行业景气、政策预期、投资者情绪和流动性变化。我们把 K 线理解为市场信息的低维 embedding：它把无数信号压缩成开盘价、最高价、最低价、收盘价和成交量等少数数字。这个压缩有价值，也会丢失大量细节。AI 方法的优势在于可以把更多信号拉入更高维的信息空间。

这种扩展并不意味着“数据越多越好”。维度越高，噪声越多，伪相关也越多。关键是学习有效表征，例如从财报文本中提取语气变化，从新闻中识别事件类型，从市场微结构中识别流动性压力。

现代金融预测常见流程是：

1. 收集多源数据。
2. 将结构化和非结构化信息转化为统一特征或 embedding。
3. 用模型预测收益、波动、风险或事件概率。
4. 通过组合优化和风控规则生成决策。
5. 用真实或模拟交易结果持续回测和监控。

LLM 和其他 AI 方法的价值，是尝试从新闻、财报、政策、公司事件和投资者情绪中恢复更高维的信息空间。但这一步必须接受回测和风控检验，不能把“信息更多”直接等同于“交易更好”。

另类数据（alternative data）就是这种思路的具体体现。卫星图像、用电量、物流运输、招聘信息、地方论坛、社交媒体、IP 地址和行业调研，都可能提供价格序列中尚未充分显现的信号。关键不是盲目收集更多数据，而是判断这些数据是否能形成独特、可验证、可持续的预测特征。

8.6　企业文本与未来表现

财报电话会、管理层讨论、风险提示和新闻文本，都包含对企业未来表现有用的信息。LLM 可以从这些文本中提取语气、主题、风险暴露和战略变化，再与财务指标和市场数据结合。FinBERT 等金融领域语言模型表明，领域化文本表示能够改善金融文本情绪和信息抽取 [Huang et al., 2023]。

这种任务的价值不在于让模型替代分析师，而是扩大分析师可处理的信息范围。模型可以快速阅读大量文本，生成结构化特征；人类分析师则负责判断这些特征是否有经济意义，是否会被市场提前定价。

文本之外，声音和视频也可以成为金融预测数据。财报电话会中，管理层如何回答尖锐问题、是否需要翻译、语速和停顿如何变化、分析师是否反复追问，都可能影响市场对企业可信度和风险的判断。这些信息不一定完整写在公告里，却会通过投资者沟通影响价格；音频-文本金融大模型正是在尝试把这类多模态信号（multimodal signals）纳入风险预测 [Liu et al., 2025]。

处理这类非书面数据通常需要三步。第一，做说话人切分（speaker diarization），识别 CEO、CFO、分析师或主持人分别在什么时候发言。第二，把声音转成文本，并保留语速、停顿、音调等声学特征。第三，把文本、声学特征和市场数据合并，检验它们是否能解释未来收益、波动或风险事件。

8.7　反身性

金融市场存在反身性：参与者的认知会影响市场，市场变化又会反过来改变参与者认知。叙事经济学也提醒我们，故事、情绪和社会传播会影响经济行为和市场结果

[Shiller, 2019]。AI 交易让这种反馈更复杂。模型预测市场，模型交易影响市场，其他参与者观察模型行为后又调整策略。

这意味着金融预测不能假设环境静止。一个在历史回测中有效的信号，公开后可能迅速失效。一个模型在无人关注时表现好，被大规模跟随后可能出现拥挤风险。

因此，金融 AI 系统需要持续监控策略容量、市场冲击和行为反馈，而不是只保存一次回测结果。

A 股市场的讨论提醒我们，投资者结构会影响波动和行为反馈。非机构投资者占比较高时，情绪、跟风和政策预期可能更快反映到价格中。模型不仅要读价格，还要理解市场参与者结构和制度环境。

本地信息和社交传播也会放大这种反身性。某些地方企业的经营问题，可能先在员工、供应商或本地论坛中口口相传，之后才反映到价格或新闻中。模型如果只读公开价格，就会晚一步；但如果模型使用社交和地理信息，又必须面对隐私、噪声和操纵风险。

8.8　伦理与监管

实盘 AI 交易涉及透明性、问责性和公平性。黑盒模型可能做出难以解释的交易；自动化系统可能放大市场波动；模型也可能被用于操纵、跟单诱导或信息不对称套利。

大模型交易还带来新的公平性问题。如果某些模型拥有更多私有交易数据、新闻数据或社交数据，它们的 “知识” 会变成普通投资者难以复制的优势。对于个人投资者、监管者和新兴市场来说，这可能形成新的力量不对称。预测能力越强，越需要明确谁能使用、使用什么数据、结果如何审计，以及错误导致损失时由谁承担责任。

可行的治理措施包括：

- 对模型输入、输出和交易决策保留审计日志（audit log）。
- 设置风险限额、人工熔断和异常交易监控。
- 在监管沙盒（regulatory sandbox）中测试高风险策略。
- 对关键模型建立可解释性报告。
- 明确模型是否允许联网、是否使用私有数据、是否可能读取其他参与者的反馈。
- 明确模型错误导致损失时的责任边界。

金融预测的最终目标不是让 AI 自由下注，而是在风险可控的边界内提高信息处理和决策质量。

8.9　小结

金融预测是 AI 能力、数据质量、市场机制和风险治理的综合测试。通用模型提供新的信息处理能力，但真正可用的金融 AI 还需要领域知识、另类数据、可交易信

号、风险控制和监管框架。市场中的预测不是旁观者游戏，预测行为本身会成为市场的一部分。

8.10 练习

1. 设计一个 AI 交易模型评估表，至少包含收益、回撤、交易频率和风险指标。
2. 说明为什么只看 K 线可能会丢失重要信息。
3. 为财报电话会文本预测未来收益设计一个特征提取流程。
4. 选择一种另类数据，说明它可能提供什么独特信号，以及如何验证其预测价值。
5. 举例说明反身性如何让一个有效策略失效。
6. 写出一个实盘 AI 交易系统必须具备的三条风控规则。
7. 以 Nof1.ai Alpha Arena 为例，设计一张模型交易评价表，要求同时包含收益、风险、交易成本、信息边界和人工接管机制。

第 9 章

面向时间序列的 Transformer

Transformer 改变了自然语言处理，也正在改变时间序列预测。本章讨论的是面向时间序列的 Transformer（Transformer for Time Series）：它的核心优势是自注意力机制，模型可以在整个输入序列中自动学习哪些历史时刻最重要，而不是只按照固定滞后或短期记忆工作 [Vaswani et al., 2023]。本章介绍 Transformer 为什么适合通用预测模型，以及它在时间序列中的基本计算逻辑、练习方式和局限。

课程进入 Transformer，是因为前面几章已经说明了一个关键问题：数值时间序列必须被重新编码，才能进入大模型式的预测系统。一个孤立的价格 `1940` 只是数字；经过上下文、位置、趋势、节日和相似序列重新表示后，它才可能成为模型可以比较和迁移的 token 或 embedding。Transformer 正是处理这类序列表示的核心工具。

9.1 学习目标

完成本章后，你应该能够：

- 说明传统深度预测范式为什么难以扩展到多领域任务。
- 解释通用预测模型的预训练和微调思想。
- 用 Query、Key、Value 描述注意力机制。
- 说明原始数值序列如何通过可学习投影进入注意力计算。
- 理解 Transformer 在长序列预测中的优势和计算成本。
- 设计一个最小 PyTorch 练习，并用评估指标比较 Transformer 与简单基准。
- 说明 PatchTST、TimeGPT、Chronos、Moirai 和 TimesFM 的共同范式。

9.2 传统预测范式的局限

许多传统深度预测模型遵循“一模型对应一数据集、一上下文长度、一预测长度”的模式。每个数据集单独训练，每个任务设置单独调参。一旦数据频率、输入窗口或预测步长变化，模型往往需要重新训练。

这种范式有四个问题：

- 可扩展性差：大量产品、地区和频率会产生大量模型。

- 泛化能力弱：模型很难跨行业、跨时间尺度迁移。
- 灵活性不足：固定窗口和固定步长限制了实际应用。
- 资源浪费：参数和训练经验不能在任务之间充分共享。

我们用即时零售说明这个规模问题。假设一个城市有 100 个小区，每天按 15 分钟切成 32 个重点时段，每个时段要预测 200 种商品需求，就会产生：

$$100 \times 32 \times 200 = 640000 \tag{9.1}$$

如果每条序列还要比较 10 个候选模型，就会变成 640 万次建模或调用。更重要的是，这些预测可能每 15 分钟刷新一次。此时 “给每条序列手工选模型” 的方式已经不现实，预测系统必须能够批量处理、共享信息并自动更新。

单序列建模还浪费了跨序列知识。海淀和朝阳的消费需求不完全相同，但都受工作日、节假日、天气、通勤和促销影响；燃气、蔬菜、电力、外卖和交通数据也都和人的活动节奏有关。如果每条序列独立训练，模型看不到这些共同规律，也无法把一个场景中学到的季节、活动和异常经验迁移到另一个场景。

通用预测模型（Universal Forecaster）试图解决这些问题。它先在大规模、多领域时间序列上预训练（pretraining），再根据具体任务零样本使用或少量微调（fine-tuning）。我们把这种模型称为类似 “预测核心能力” 的 Universal Forecaster：它不是只会预测某一条销量曲线，而是在许多序列中学习 “历史如何影响未来” 的一般规律。

这条路线并不要求抛弃传统统计特征。趋势、季节、滞后和移动平均仍然有价值；深度模型的作用，是在这些显式结构之外学习更高维的 embedding 和非线性交互。

真正的变化是：预测工作从 “为每个对象训练一个模型” 转向 “训练一个可迁移的预测能力，再把它适配到具体业务”。

9.3　为什么是 Transformer

时间序列和语言有共同点：二者都是序列。语言模型根据前文预测下一个 token，时间序列模型根据历史观测预测未来值。Transformer 的自注意力机制可以同时查看整个上下文，因此天然适合捕捉长程依赖。

相对 RNN 或 LSTM，Transformer 更容易并行训练；相对固定卷积，它能在任意位置之间建立联系。对于企业数据，这意味着模型可以同时关注近期点击、历史购买、节假日、促销、天气和长期趋势。

我们反复强调的一点是：Transformer 不要求我们先人工指定 “最近 7 天”“去年同周” 或 “某个固定滞后” 一定重要。模型会从数据中学习哪些历史片段对当前预测更有用。人工经验仍然重要，但它更多体现在数据组织、特征选择和评估设计上，而不是手工写死每一个依赖关系。

供暖或天然气需求是一个直观例子。预测今年冬天的用量时，只看昨天、前天或最近一周往往不够；去年同期、上一轮寒潮、供暖季开始前后的变化都可能影响当前判断。如果用 ARIMA 手工加入 365 天滞后，模型结构会变得很重；如果完全依赖短

期记忆，又会漏掉长期周期。Transformer 的价值在于让远处的历史位置也有机会直接参与当前预测。

Transformer 还适合处理不同长度的历史。一个城市可能有 5 年数据，一个新区域可能只有 1 年数据，一个新品可能只有几周数据。实际系统不能要求所有序列都有完全相同的历史长度。更灵活的序列表示和注意力机制，使模型可以在不同上下文长度下提取可用信息。

9.4 注意力机制

注意力机制可以理解为“让模型学会关注重点”。传统自回归模型其实也隐含一种注意力：AR(1) 默认昨天最重要，季节模型默认某些固定周期位置重要。但这种关注方式是人预先指定的。Transformer 的自注意力让模型自己在整个历史窗口中寻找相关位置。

它使用三个对象：

- **Query（查询向量, Q）**：当前要回答的问题，例如“哪些过去事件会影响这次预测？”
- **Key（键向量, K）**：历史信息的标签，例如某天是否节假日、促销期或淡季。
- **Value（值向量, V）**：历史信息的内容，例如当天销量、价格、库存或流量。

模型先计算 Query 与所有 Key 的相似度，再通过 softmax（归一化指数函数）转成权重，最后对 Value 加权求和。简化写法是：

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{9.2}$$

在时间序列预测中，这相当于自动判断“哪些历史时刻最应该被听见”。如果当前预测目标是春节前销量，模型可能更关注往年春节前的时间点，而不是机械地只看最近几天。

我们还会用一个教学比喻：判断全班是否理解时，不应平均看每个学生一眼，而可能会特别观察后排、皱眉或没有跟上的同学。注意力机制也是如此，它把有限关注度分配给对当前判断更关键的位置。

直观地说，Query 是当前预测要问的问题，Key 是历史片段可被匹配的标签，Value 是真正被带入计算的信息。这个视角能把抽象矩阵运算和业务问题连接起来：模型不是平均看所有历史，而是在不同预测目标下重新分配关注。

这些 Q、K、V 通常不是原始输入本身，而是模型从输入中学出来的表示。假设输入窗口已经整理成矩阵 X，模型会学习三组权重，把它们投影成：

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V \tag{9.3}$$

这里的 W_Q、W_K 和 W_V 都是训练得到的参数。也就是说，神经网络不只是计算相似度，还在学习“怎样把原始时间序列变成适合比较的特征”。这就是为什么同样一

条销售序列，在不同任务中可以被编码成不同的内部表示。

以销售预测为例，Query 可以表示 “明天销量会怎样”，Key 可以包含历史日期的节日、促销、季节和价格标签，Value 则包含真实销量、库存、折扣和流量。注意力权重决定哪些历史片段被更多带入预测。编码器把这些输入和注意力结果转换成中间表示，预测头或解码器（decoder）再输出未来数值。

9.5 Softmax 的作用

注意力分数本身可以很大也可以很小。softmax 把这些分数转换成一组非负权重，并让它们加总为 1。这样每个历史时刻都可以被赋予不同关注度。

除以 $\sqrt{d_k}$ 是为了数值稳定。如果向量维度很高，点积可能过大，softmax 会变得过于极端，训练梯度不稳定。

这个机制的直觉很简单：模型不是平均看所有历史，而是学习在不同预测问题下如何重新分配注意力。

可以把 softmax 理解成一种 “注意力预算”。模型面对很多历史片段时，不能让每个片段都无限重要，而是要把关注度分配到 0 到 1 之间，并让总量可比较。某些历史点权重高，某些历史点权重低，最终预测就由这些加权后的信息共同决定。

可以用 `50, 70, 60` 这组三个分数解释 softmax。原始分数不能直接当权重，因为它们没有加总为 1，也可能包含负值。softmax 先对每个分数取指数，再除以全部指数值之和，使输出全部为正并且总和等于 1。这样模型才能把 “相似度分数” 转成 “可比较的注意力比例”。

9.6 时间序列输入如何进入 Transformer

原始时间序列不能直接被 Transformer 使用。需要先映射到内部表示空间。常见做法包括：

- 用线性层（linear layer）把数值窗口映射为 embedding。
- 把序列切成 patch，把每个时间块当作 token。
- 对数值进行离散化或分桶，使其类似语言 token。
- 加入时间特征、频率特征、节假日或外生变量。

在实现中，用户输入的通常只是原始序列 x。模型先通过可学习投影和偏置把它变成特征矩阵 X，再从 X 分别得到 $Q = XW_Q$、$K = XW_K$ 和 $V = XW_V$。也就是说，Q、K、V 不是人工填入的三张表，而是模型为了完成当前任务从同一组输入特征中学习出的三种视角。

映射后，每个时间步或时间块都有一个向量表示。Transformer 对这些向量做多头注意力（multi-head attention），得到新的序列表示，再由预测头（prediction head）输出未来值或分布。

这种输入映射解释了为什么时间序列 Transformer 经常和特征工程一起出现。原始数值可以进入模型，但频率、日历、促销、价格、天气、异常标记等信息会帮助模型形成更有预测意义的 embedding。可以用“把一个数值重新编码成更长的特征向量”来说明这个过程：数值本身只是一个点，编码后的向量则可以包含历史位置、波动、周期和上下文。

多头注意力可以理解为从多个角度看同一段历史。一个头可能更关注短期波动，一个头可能更关注节假日，一个头可能更关注长期趋势。实际训练中这些角色不是人工规定的，而是由数据和损失函数推动模型自己形成。

茅台价格可以继续说明 embedding 的含义。某天价格 `1940` 对普通人只是一个数字，但如果放进酒类市场、政策变化、渠道库存、节日前后和历史低点的上下文里，它就携带了丰富含义。Transformer 的编码器可以把这个点和周围历史一起编码成高维向量，例如 512 维 embedding。原来长度为 T 的序列不再只是 T 个数字，而可以被表示为 $T \times 512$ 的特征矩阵。

时间序列里的 token 也不一定只是单个数值。它可以是一个时间点、一个窗口 patch、一个降采样片段、一个分桶后的状态，或者一个已经包含上下文的 embedding。模型设计的关键，是让这些 token 足以表达趋势、周期、异常、冲击和局部形状，而不是机械模仿文本 token。

9.7　计算复杂度与改进

标准 Transformer 的注意力需要比较所有时间步两两关系，复杂度大约随序列长度平方增长。这让它在超长时间序列上成本较高。

如果输入长度扩大一倍，注意力比较的数量会增长得更快。我们把这点概括为：注意力很强，但它需要算力。长序列、细粒度和多变量都会增加成本，因此工业预测很少只靠最原始的 Transformer 结构直接处理所有时间点。

因此，时间序列模型发展出多种改进，代表性工作包括面向长序列预测的 Informer [Zhou et al., 2021] 和把时间序列切成 patch 的 PatchTST [Nie et al., 2023]：

- 稀疏注意力（sparse attention）只关注关键时间点。
- Patch 方法把多个时间步压缩为一个时间块。
- 分解方法先提取趋势和季节项，再建模残差。
- 多尺度方法（multi-scale methods）同时处理小时、日、周等不同粒度。

这些改进的目标不是放弃 Transformer，而是让注意力机制更适合长序列和大规模工业数据。

分解方法也可以和 Transformer 组合使用。先拆出趋势和季节，再让注意力机制处理残差和跨时间依赖，说明现代模型不是抛弃统计结构，而是把统计结构放入可学习框架。

Patch 和多尺度方法可以理解为先改变模型“看历史”的单位。秒级数据可以聚合到分钟、小时或天；一条很长的序列也可以切成多个时间块。这样做会牺牲一部分细

节，但能让模型在更高层次上看到周期、趋势和突变。真正的设计问题是：哪些细节必须保留，哪些细节可以通过聚合换取更低成本。

9.8 一个最小 PyTorch 练习

本章练习的目的不是马上训练出最强模型，而是看清 Transformer 如何被放进一个预测流程。可以按下面的顺序实现：

1. 生成或读取一条长度约为 100 的时间序列。
2. 按 80% 和 20% 切分训练集与测试集。
3. 用滑动窗口构造监督学习样本，例如 `window_size=20`，用前 20 个点预测第 21 个点。
4. 用线性层把 `input_size=1` 的数值映射到 `d_model=64` 的内部表示。
5. 使用 `TransformerEncoderLayer` 堆叠一到数层编码器。
6. 用线性预测头把最后的表示映射回一个未来数值。
7. 使用 MSE 作为损失函数（loss function），使用 Adam 优化器（Adam optimizer）训练模型。
8. 在测试集上计算 RMSE、MAE 或 MAPE，并与简单基准比较。

关键超参数（hyperparameters）可以从小范围开始：

参数	含义	初始建议
`window_size`	输入历史长度	20
`d_model`	内部表示维度	32 或 64
`nhead`	注意力头数	2 或 4
`num_layers`	编码器层数	1 或 2
`lr`	学习率	0.001
`epochs`	训练轮数	先用较小轮数观察损失

训练时同时观察训练误差和测试误差。如果训练误差持续下降但测试误差上升，可能已经过拟合。此时可以减少层数、降低表示维度、缩短训练轮数或增加正则化（regularization）。小数据集上 Transformer 不一定超过 ARIMA、指数平滑或季节性朴素模型，这不是失败，而是提醒我们：复杂模型需要足够数据和合适任务。

练习的重点是形成实验习惯。改动 `nhead`、`d_model`、`num_layers`、`window_size` 和学习率后，要记录结果；与 ARIMA 或朴素基准比较时，要使用同样的训练/测试切分。C06-C07 的点预测和滚动评估方法仍然适用，不能因为模型换成 Transformer 就跳过评估。

如果在本地电脑运行较慢，可以使用学校集群或 GPU 资源。小型示例在 CPU 上也能完成；更大模型、更多序列或更长窗口才真正需要 GPU。我们的建议是先让最小 notebook 跑通，再逐步增加模型复杂度。

9.9 通用时间序列预测模型

本章列举了多个代表模型：

- **PatchTST**：借鉴 Vision Transformer，把时间序列切成 patch。
- **TimeGPT**：在大规模多领域时间序列上预训练的 decoder-only Transformer（仅解码器 Transformer）。
- **Chronos**：把时间序列 token 化，用类似 GPT 的架构学习 “时间的语言”。
- **Moirai**：使用统一训练和稀疏 Transformer，支持多频率和概率预测。
- **TimesFM**：在大规模序列上预训练，强调零样本泛化（zero-shot generalization）。

这些模型的共同点是把预测从 “为每个任务单独建模” 推向 “训练一个可迁移的预测器”。

TimeGPT 的教学意义在于它把时间序列预测包装成 API：用户整理好 `unique_id`、`ds` 和 `y`，指定频率和预测步长，就可以调用预训练模型。Chronos 则展示了另一条路线：把连续数值离散化或 token 化，让模型像学习语言一样学习时间序列模式。Amazon 需要这类方法，是因为电商平台面对全球市场、长尾商品和大量冷启动对象，不能为每个商品单独维护一套专家模型。

这也要求把不同来源、不同频率的时间序列映射到统一特征空间。STL 分解、patch 化和 Transformer 是互补工具：一个提供可解释结构，一个降低长序列成本，一个学习跨时间依赖。

这条路线也解释了为什么后续会继续讨论时间序列基座模型。Transformer 本身只是结构；要让它成为可复用的预测能力，还需要大规模多领域数据、预训练目标、评估流程、部署成本控制和业务治理。

9.10 管理含义

Transformer 让企业预测从模型选择问题扩展为数据资产问题。模型能否泛化，取决于企业是否有多领域、多频率、高质量的时间序列数据，以及是否能把这些数据转化为一致的训练和评估格式。

管理者需要理解两个边界。第一，Transformer 能学习复杂模式，但不自动保证可解释性。第二，通用模型能降低建模门槛，但关键业务仍需回测、监控和人工判断。

更实际地说，企业采用 Transformer 或时间序列基座模型时，不应只问 “模型是否先进”，还要问 “是否有足够多序列可供学习”“评估是否覆盖高风险场景”“算力和延迟

是否可接受”“模型失败时是否有回退规则”。这些问题决定模型能否从教学示例走向生产系统。

9.11　小结

Transformer 的核心贡献是让模型在整个序列中学习相关性。对于时间序列预测，它打开了通用预测模型的道路：预训练、迁移、零样本、多频率和概率预测。与此同时，长序列成本、数据质量、可解释性和业务落地仍是必须面对的问题。

本章主线可以概括为三点。第一，企业预测规模使 “一条序列一个模型” 难以持续。第二，注意力机制通过 Query、Key、Value 和 softmax 自动分配历史信息的重要性。第三，Transformer 练习必须回到可复现评估：小数据上不一定赢，真正价值来自多序列、大规模和可迁移能力。

9.12　练习

1. 用 Query、Key、Value 解释一个节假日销量预测任务。
2. 比较 ARIMA、LSTM 和 Transformer 对长期依赖的处理方式。
3. 说明为什么 patch 化可以降低长序列建模成本。
4. 写出 $Q = XW_Q$、$K = XW_K$、$V = XW_V$ 在一个销售预测任务中的直观含义。
5. 找一个业务场景，判断它更适合单独训练模型还是使用通用预测模型。
6. 讨论 Transformer 预测结果如何向业务人员解释。
7. 修改一个 PyTorch Transformer 练习中的 `nhead`、`d_model`、`num_layers` 或 `window_size`，记录测试误差变化，并与一个简单基准比较。
8. 比较 “单个数值点”“时间窗口 patch” 和 “高维 embedding” 三种时间序列 token 表示各自保留了什么信息。

第 10 章

时间序列基座模型

时间序列基座模型（Time Series Foundation Models, TSFMs）把预测从单任务建模推进到跨领域、跨频率和跨任务的统一建模。它们通常结合分解、多尺度表示、Transformer、重编程（reprogramming）和大规模预训练，目标是在新数据集上也能快速给出可用预测。本章围绕分解、TimeMixer、Time-LLM 和 Chronos 等思路，说明时间序列基座模型的技术路线和应用边界。

10.1 学习目标

完成本章后，你应该能够：

- 说明时间序列模型从统计模型到基座模型的发展脉络。
- 解释趋势、季节和残差分解在现代模型中的作用。
- 理解多尺度建模为什么适合复杂时间序列。
- 说明时间序列重编程如何让 LLM 处理数值序列。
- 说明为什么基座模型需要路由、蒸馏和迁移学习（transfer learning）等部署机制。
- 判断基座模型在业务预测中的优势和风险。

10.2 从统计模型到基座模型

时间序列预测经历了多个阶段。早期方法包括 Holt-Winters、ARIMA 和 ETS，强调可解释结构和统计假设。之后，DeepAR、N-BEATS、TCN、Informer 等深度模型开始从大量序列中学习模式。近几年，PatchTST、TimeMixer、TimesNet、Time-LLM、Chronos、Moirai 和 TimesFM 等模型进一步推动了通用化。

我们把这条脉络讲得更完整。1970 年前后，Box-Jenkins 系统化了 ARIMA，Holt-Winters 和指数平滑也长期服务于趋势和季节性预测。2000 年以后，神经网络、卷积、集成和深度学习进入时间序列领域，DeepAR、N-BEATS 等模型开始从多条序列中学习。2020 年前后，Informer、Autoformer、ETSformer 等 Transformer 模型把注意力机制引入长序列预测。到 2024 年前后，基座模型思想进入时间序列领域，问题从“为某个数据集训练模型”变成“能否学习足够丰富的时间序列形态空间”。

基座模型的核心问题是：能否训练一个模型，让它在许多领域、许多频率、许多预测步长上都能工作？如果可以，企业就不必为每个产品、门店、地区和时间粒度单独维护模型。

因此，时间序列基座模型不同于普通监督学习（supervised learning）表格上的 prediction。它的目标是在大量不同时间序列中学习可迁移的时序表示，并在新序列上快速适配。

我们进一步强调，基座模型还改变了上下文处理方式。传统模型往往要求固定输入窗口和固定预测步长；真实业务中，一个地区可能有 5 年历史，一个新市场可能只有 1 年历史，一个新品可能只有几周历史。基座模型要学习的不是某个固定窗口，而是在不同上下文长度中提取有用历史，并把跨序列经验迁移到新任务。

选择基座模型时，领域数据非常关键。一个模型在金融、零售、能源或交通任务中的表现，不只取决于模型名字，也取决于预训练和后续训练中是否吸收了足够多的相关数据、风险逻辑和业务结构。金融交易案例说明，如果训练数据和任务理解更接近真实交易，模型可能更像交易员；如果训练数据主要强化文本表达，它可能更像报告撰写助手。

企业规模越大，这个问题越现实。电商平台、供应链系统和金融机构可能同时维护成千上万条序列。早期做法是每条序列或每类序列单独建模，随着业务扩大，预测团队会被模型维护、特征更新和异常处理拖住。基座模型的管理意义在于把大量重复建模工作转化为一个可复用的预测能力，再让业务人员把精力放在数据质量、评估和决策上。

可以用“预训练 checkpoint（检查点）”解释这种变化。一个训练好的模型文件，本质上保存了大量参数；这些参数来自大规模数据训练后的经验。使用者拿到 checkpoint 后，整理好时间列、目标列、序列编号和预测步长，就可以直接做零样本预测或少量微调。它不再要求每次都从头训练一个 ARIMA、ETS 或深度模型。

TimeGPT 可以作为这种变化的教学入口。它和 ChatGPT、DeepSeek 这类通用聊天模型不同，目标是处理时间序列预测任务 [Garza et al., 2024]。在示例中，用户把航空乘客数据整理成 `unique_id`、`ds` 和 `y` 三列，指定月度频率和未来 12 期，模型就能返回一条未来曲线。这个过程展示了基座模型的便利性，也提醒我们：输出曲线只是开始，是否可用仍要用 C06-C07 的评估流程证明。

Chronos 提供了另一种直觉：把连续时间序列转换成 token，让模型像学习“时间的语言”一样学习上升、下降、波动、季节和异常模式 [Ansari et al., 2024]。Amazon 需要这类统一模型，是因为全球电商平台同时面对大量地区、品类、长尾商品和冷启动对象；如果每个商品都单独维护模型，专家、算力和工程流程都会被规模压垮。

我们把这个问题进一步落到 zero-shot forecasting（零样本预测）。所谓零样本预测，是指新的时间序列直接输入预训练模型，不再为这条序列重新训练，模型就给出预测。对企业来说，这很有吸引力：大量新品、长尾 SKU、新区域和临时指标没有足够历史数据，也没有时间逐条调参，但仍然需要一个稳健的候选预测。

零样本能力来自预训练阶段的大规模投入。基座模型已经在大量长短不一、频率不同的时间序列上学习过模式；推理时只是把当前序列映射到这个已经学好的表示空间中，再输出未来分布。我们特别提醒，这不等于模型永远最好，而是说它可以成为一个强基准。使用者仍然要检查它是否超过 Naive、季节性 Naive、ARIMA、LightGBM、

DeepAR、NHITS 或其他合适基线。

我们进一步强调，预训练模型在数据很少时尤其有价值。一个新企业可能只有几百个经营观测值，但历史上有大量类似企业经历过扩张、收缩、季节波动和促销冲击。基座模型就像把这些“过来人经验”压缩进参数中，让新企业在早期数据不足时也能得到参考预测。换句话说，Foundation Model 不只是大企业海量数据的工具，也是小样本场景借用外部经验的方式。

模型表现还和数据频率有关。月度和周度商业数据通常更适合 TimeGPT 这类基座模型发挥跨序列泛化能力；小时级、分钟级或 15 分钟级高频数据中，LightGBM、XGBoost 等树模型仍可能非常有竞争力，尤其在量化和低延迟场景中。DeepAR 这类神经网络自回归模型也常被用作稳定基线。真正的企业系统不只追求某个模型平均第一，还要避免某些场景下特别差的预测误导决策。

10.3　Chronos-2 与离线推理

本章练习使用一个用于离线推理（Offline Inference）的 Chronos-2 模型。模型文件提前下载后放在本地目录中，目录里通常包含 `config.json` 和 `model.safetensors` 等文件；示例版本的权重文件约为数百 MB，可以直接加载到内存中推理。这个练习说明，时间序列基座模型不一定只能通过云端 API 使用，也可以作为本地模型进入 notebook。

Chronos-2 相比早期版本的一个重要扩展，是对单变量（univariate）、多变量（multivariate）和外生变量的支持。单变量预测只使用目标序列本身；多变量预测可以同时处理多条相关序列；加入协变量（covariates）时，节假日、促销、开店状态、顾客数或未来已知日历信息都可以成为输入。它还支持在预训练模型基础上进一步微调，从通用能力转向更贴近某个领域的数据。

上下文长度（context length）也是基座模型的重要约束。早期版本上下文长度约为 512，而新的版本可以支持到 8192。对日度数据来说，8192 个点大约覆盖二十多年历史；对小时级数据来说，则覆盖更短的真实时间跨度。选择模型时，不能只看模型名字，还要看它能接收多长历史、是否支持协变量、是否能输出分位数、是否能在本地硬件上以可接受速度运行。

本地模型的管理问题和 API 不同。API 主要管理 key、成本和数据外发；本地模型要管理模型路径、依赖版本、CPU/GPU 选择和内存占用。几百 MB 的模型可以在很多笔记本电脑 CPU 上运行，只是速度较慢；如果使用 GPU，推理速度会明显提升。对教学来说，这种本地练习很有价值，因为学生可以看到“预训练模型文件”如何变成实际预测，而不是只看到一个远程服务返回结果。

10.4　分解思想

复杂时间序列通常包含多个层次：

- 趋势项（trend component）：长期上升或下降。

- 季节项（seasonal component）：固定周期内重复出现的波动。
- 残差项（residual component）：随机扰动、异常和未解释部分。

STL 分解（Seasonal-Trend decomposition using LOESS, STL）可以写成：

$$y_t = S_t + T_t + R_t \tag{10.1}$$

有些业务场景也可以用乘法形式理解：

$$y_t = S_t \times T_t \times R_t \tag{10.2}$$

分解的价值在于把复杂问题拆开。趋势帮助理解长期方向，季节项帮助捕捉周期结构，残差则提醒我们还有异常和随机性。现代模型并没有抛弃分解，反而经常把它嵌入深度结构。

这也是本章的一个主线：新模型并不是把旧统计思想全部扔掉。很多有效的模型仍然在使用上世纪形成的基本判断，例如趋势、季节、残差、聚合、组合和调和。差别在于，现代模型可以把这些结构放入更大的可学习系统中，让它们服务于跨领域预测。

我们特别强调分解的诊断价值。如果趋势预测不好，应改进长期方向建模；如果季节性预测不好，应检查周期结构；如果残差很大，可能说明有异常、冲击或外部变量没有进入模型。相比把所有波动直接丢进黑箱，分解让模型结构更容易解释，也更容易和业务人员讨论。

10.5　分解与 Transformer

Autoformer 将分解思想和 Transformer 结合。它用滑动平均提取趋势，用注意力机制捕捉季节和残差结构，并在多层结构中递归分解。这种设计让模型在长序列预测中更稳定，也更容易解释。

这个思路说明，基座模型不一定是纯黑箱。统计结构仍然有价值，尤其是在业务场景中，趋势和季节性本身就是管理者能理解的语言。

Autoformer 的教学意义在于，它把“先理解结构，再让深度模型学习”的思想写进了网络。原始序列先被拆成趋势和季节等部分，再分别进入可学习模块。这样既保留了 Transformer 处理复杂依赖的能力，也保留了传统时间序列分解的可解释性。这个例子提醒我们：当我们对数据有明确理解时，应把这种理解嵌入模型，而不是盲目追求完全端到端（end-to-end）。

10.6　多尺度建模

时间序列往往同时包含粗尺度和细尺度信息。电商销量可能有小时级波动、日内周期、周末效应和年度季节性；能源负荷可能受分钟级扰动、日周期和季节气温共同影响。

多尺度建模试图同时捕捉这些层次。TimeMixer 的核心思想是把不同时间尺度的特征放入统一结构中混合，既建模跨时间依赖，也建模跨变量关系。它通过 temporal mixing、channel mixing 和多尺度残差连接来处理趋势、周期和局部波动。

对业务来说，多尺度模型的价值在于减少单一频率视角的盲区。只看日度数据会丢失小时级异常，只看短期数据又可能看不到长期趋势。

多尺度也可以降低计算压力。秒级或分钟级序列太长，直接做全局注意力成本很高；把序列聚合到小时、日、周或月，可以让模型从粗到细地理解信号。车流量数据在秒级看起来很细碎，聚合到天或周后才容易看到通勤、周末和节假日规律。基座模型要学习的正是这种跨尺度的表示。

我们用 “离得太近反而看不清” 的例子说明尺度问题。秒级车流量包含细节，但噪声也大；聚合到小时、天、周或月后，通勤、周末、节假日和长期变化反而更清楚。经济因子模型常常看月度或季度结构，高频交易则看分钟级甚至秒级微结构。对同一条序列，不同尺度不是重复信息，而是不同层次的信息。

多尺度还直接帮助注意力计算。标准 Transformer 的注意力复杂度随序列长度平方增长；如果把长度为 T 的序列降采样（downsampling）为 $T/7$ 或 $T/52$ 等粗尺度序列，粗尺度上的 attention 成本会显著降低，同时还能捕捉更稳定的宏观结构。微观尺度负责局部细节，宏观尺度负责长期趋势和周期，二者结合比单独盯着高频原始序列更稳健。

TimeMixer 可以作为多尺度思想的具体例子。它先对原始序列做不同尺度的降采样，得到多个分辨率下的序列；再在每个尺度上做 decomposition，把趋势项和季节项拆开；随后分别对趋势和季节做 mixing，让不同尺度之间交换信息；最后用前馈网络生成各尺度预测并合并 [Wang et al., 2024]。它表面上是端到端模型，内部却遵循清楚的时间序列逻辑：先看多个尺度，再拆趋势和季节，再组合预测。

10.7　从基座模型到专家路由

基座模型越想服务多种场景，就越需要路由机制。稳定农产品、空调销售、金融高频价格、冷启动新品和间歇性备件需求并不是同一种预测问题；一个通用模型如果要同时处理它们，就必须学习哪些内部能力适合哪些输入。

这个问题在 C11 中作为独立主题讨论。Mixture of Experts, 简称 MoE, 把不同专家模型按权重组合起来，让门控网络根据序列特征决定谁参与预测、谁占更高权重。它既继承了统计预测中的组合预测思想，也解释了现代大模型为什么可以通过稀疏激活降低推理成本。

在本章中，只需要先抓住一个连接点：基座模型不是 “一个模型解决一切”，而是把表示学习、分解、多尺度、路由、蒸馏和迁移组合成一套预测系统。MoE 负责回答 “把当前序列交给谁” 的问题，蒸馏和迁移则负责回答 “如何把大模型能力变成可部署、可适配的业务模型”。

10.8　时间序列重编程

另一条路线是让大语言模型处理时间序列。LLM 本来处理文本 token，不直接理解连续数值。时间序列重编程（time-series reprogramming）通过外部适配器（adapter）或重编程器，把数值片段映射到 LLM 可以处理的表示空间。

基本流程是：

1. 输入时间序列片段。
2. 用重编程器生成 prompt、embedding 或语义表示。
3. 冻结（freeze）LLM 主体，只训练外部适配器。
4. 输出预测、分类或异常检测结果。

这种方法的吸引力在于利用 LLM 已有的表示能力和推理能力，而不需要从头训练大模型。风险是数值信号和语言语义之间的映射并不天然，必须通过合理设计和严格评估证明有效。

音频信号提供了一个有用类比：原始波形需要先变成可学习特征，时间序列也需要合适的表示。每条序列、每个时间窗口和每个时间点都可以成为训练样本的一部分；样本组织方式决定了模型能否从大规模数据中学到通用规律。

数值 token 可以解释重编程直觉。一个序列 `[100, 120, 90]` 可以直接作为数值片段，也可以转成“中、高、低”这样的分桶标签，或者转成“上涨 20、下跌 30”这样的变化 token。基座模型真正要学的，不只是这些数字本身，而是它们在上下文中的含义：当前点相对过去处在什么位置、是否接近节日前后、是否延续了历史趋势，以及和相似序列相比是否异常。

我们还会从另一个角度追问：时间序列里的 token 在哪里？答案并不唯一。它可以是单个原始数值点，也可以是一个时间片段、一个 patch、一个经过编码的 embedding，或者一个经过降采样后的尺度片段。文本模型处理词和子词，时间序列模型处理数字及其上下文形态。关键不是 token 长什么样，而是它能否承载趋势、周期、异常、冲击和局部形状等预测相关信息。

10.9　Prompt-as-Prefix

Prompt-as-Prefix 是一种更直观的重编程方式：把结构化时间序列信息编码为提示词前缀。例如把最小值、最大值、中位数、趋势、滞后值和任务说明写入 prompt，让 LLM 在语义空间中理解时间动态。

这种方法适合教学和快速原型，但不能替代正式评估。提示词改变可能带来结果波动，模型也可能生成看似合理但无法验证的解释。因此，Prompt-as-Prefix 更适合作为特征生成、解释辅助或低成本基线。

10.10　文本原型与跨注意力

Time-LLM 等方法使用文本原型（text prototypes）和跨注意力机制（cross-attention mechanism），把时间序列片段对齐到语言空间 [Jin et al., 2023]。文本原型可以表示 “短暂上升”“平稳下降” 等典型模式。模型让时间序列片段作为 Query，与这些文本原型的 Key 和 Value 匹配，从而生成 LLM 可理解的表示。

直觉上，这是把数值信号翻译成语言模型熟悉的 “模式词汇”。如果翻译得好，LLM 可以利用预训练知识处理预测任务；如果翻译不好，就会引入噪声和不稳定性。

10.11　从基座模型到蒸馏

基座模型训练成本高，推理成本也可能高。可以用 Teacher Model 和 Student Model 解释知识蒸馏：Teacher 是大而强的预训练模型，Student 是更小、更便宜、更适合部署的模型。训练 Student 时，不只让它拟合真实值，也让它模仿 Teacher 的输出、分布或中间表示。

蒸馏不是简单删掉参数。好的 Student Model 应该保留当前任务真正需要的能力。例如一个大模型可能同时擅长周期、趋势、短期和长期预测，而某个业务只需要稳定的一周销量预测。此时 Student 不必在所有任务上都追平 Teacher，只要在目标场景中足够接近，甚至经过领域数据微调后更稳定，就有实际价值。

这一主题在 C12 中单独展开。对本章来说，蒸馏的作用是把基座模型从 “通用能力” 推向 “可部署能力”：先用大规模预训练获得广泛知识，再把目标任务真正需要的能力迁移到更小、更便宜、更容易部署的模型中。迁移是否成功，仍然要回到 C07 的评估流程。

10.12　性能与适用边界

基座模型的优势在于泛化和部署效率。它们可以在新数据集上零样本预测，也可以通过少量微调适配业务数据。对于长尾产品、新市场、冷启动序列和多频率场景，这种能力很有价值。

但基座模型不是万能工具。需要注意：

- 预训练数据是否覆盖当前业务场景。
- 模型是否支持所需频率、外生变量和概率预测。
- 零样本结果是否超过简单基准。
- 成本、延迟和数据隐私是否可接受。
- 模型失效时是否有监控和回退方案。
- 蒸馏或微调后的小模型是否保留了关键预测能力。
- 本地推理时模型文件、依赖版本、CPU/GPU 和上下文长度是否满足任务要求。

在企业中，基座模型通常应与传统统计模型、业务规则和人工审查共同使用。

冷启动案例能说明基座模型的价值。比如尚未上市的下一代手机没有自己的历史销量，但相关产品、竞品、配件、价格带和共同购买关系可以提供信息。基座模型要学习的正是这种跨序列、跨对象的相似性。

10.13　未来方向

时间序列基座模型会继续向跨模态融合、自适应优化、可解释性和实时决策发展。Moirai 这类统一训练框架也说明，未来模型会更强调跨频率、跨领域和概率预测能力[Woo et al., 2024]。未来模型可能同时处理文本、图像、传感器、交易和运营数据，并在业务流程中自动更新。

但越接近自动化决策，治理越重要。模型需要说明输入来源、评估记录、适用边界和责任机制。预测能力只是智能决策的一部分。

10.14　小结

时间序列基座模型的核心是用大规模预训练和统一结构提升泛化能力。分解、多尺度、Transformer、专家路由、重编程和蒸馏分别解决结构、尺度、依赖、模型选择、跨模态表示和部署成本问题。学习这些模型时，应同时关注技术原理、数据覆盖、评估证据和业务治理。

10.15　练习

1. 对一个月度销售序列，说明趋势、季节和残差分别可能来自什么业务因素。
2. 设计一个多尺度预测任务，说明需要哪些频率的数据。
3. 写一个 Prompt-as-Prefix 模板，包含统计特征和预测任务说明。
4. 比较基座模型零样本预测和传统 ARIMA 模型的评估设计。
5. 说明为什么一个基座模型仍然需要路由机制，并举出三类应该交给不同专家处理的时间序列。
6. 解释为什么蒸馏后的小模型可能比直接部署大模型更适合企业。
7. 列出一个企业引入时间序列基座模型前必须检查的五项条件。
8. 比较 TimeGPT API 和本地 Chronos-2 模型在隐私、成本、部署、协变量支持和推理速度上的差异。
9. 画一张表，比较 ARIMA、LSTM 和 Transformer/基座模型在输入表示、长期依赖、趋势季节处理、多尺度能力、数据需求和可解释性上的差异。

第 11 章

混合专家模型与预测组合

混合专家模型（Mixture of Experts, MoE）是理解现代 AI 预测系统的一把钥匙。它回答的不是“哪个模型永远最好”，而是“面对这条序列、这个预测步长和这个业务场景，应该相信哪些专家，以及各相信多少”。本章从没有免费午餐（No Free Lunch, NFL）、预测组合（forecast combination）、手工 MoE 练习和现代稀疏激活模型出发，说明 MoE 为什么适合时间序列预测，以及它怎样把专家差异转化为可学习、可评估的系统能力。

11.1 学习目标

完成本章后，你应该能够：

- 解释为什么不存在一个在所有预测场景下都最优的模型。
- 说明组合预测和 MoE 之间的关系。
- 用手工 MoE 练习理解专家（expert）、门控（gating）和最终预测（final forecast）。
- 区分等权组合、加权组合、路由（routing）和 Top-K 门控（Top-K gating）。
- 说明 MoE 如何降低极端预测、过拟合和单一专家失误带来的风险。
- 判断 MoE 在时间序列基座模型、业务预测平台和管理决策中的适用边界。

11.2 为什么要讨论 MoE

前面几章已经看到，AI 预测不是把一个模型拿来就结束。真实应用中，手里通常有很多数据，也有很多候选模型。关键问题是匹配：什么样的数据适合什么样的模型，什么样的场景应该调用什么样的工具。

一个自然想法是训练一个唯一的通用模型，让它解决所有问题。但这在理论和实践上都不可靠。即便把大量数据合在一起，也很难得到一个在所有场景下都最优的模型。模型和人一样会有专长：有人擅长创意，有人擅长工程，有人擅长数学；一个模型可能擅长稳定趋势，另一个模型可能擅长强季节性，第三个模型可能更适合促销冲击或高频金融数据。

过去如果只有一千条时间序列和十个候选模型，可以把模型全部跑一遍，再比较误差。进入 API 和大模型时代后，成本变得更重要。把全部数据交给一个外部 API 跑

一晚上，第二天发现效果不好，可能已经花掉几千元。再换另一个 API 重跑，又是一笔成本。每一次调用是否能带来有效反馈，必须提前考虑。

自己训练大模型的成本更高。我们用时间序列大模型的训练作例子：一个有效模型可能需要多张 A100 连续训练数周。如果训练很久后才发现效果不佳，时间、算力和资金都会被消耗。因此，我们需要一种低成本、高效率的集成方式，把应用、数据和 AI 工具快速连接起来。MoE 正是这种思想的代表。

MoE 的核心不是“用一个更大的模型压倒所有模型”，而是让多个专家各自处理擅长的问题，再通过一个路由或门控机制把结果组合起来。DeepSeek 等模型受到关注，也与这种思路有关：系统规模可以很大，但每次任务不必激活全部能力，而是激活最相关的一部分专家。

11.3　没有免费午餐

MoE 的理论背景是没有免费午餐。它的直观含义是：不存在一个工具或模型能在所有场景下都比其他工具更好 [Wolpert and Macready, 1997]。即使没有学过这个理论，实践中也很容易理解：没有一个老师能把所有课都教得最好，没有一个学生能把所有课都学得最好，没有一个厨师能把所有菜都做得最好，也没有一个司机能把所有车都开得最好。

预测场景也一样。有些产品需求很稳定，比如某些农产品或长期消费品，如果没有极端天气或供应冲击，产出和需求变化较平缓。有些产品季节性很强，比如空调在夏天需求明显更高。有些序列受到促销、节假日、舆情或政策影响；有些序列则像高频金融价格一样噪声大、反馈快、机制复杂。

不同序列需要不同能力：

序列形态	需要的模型能力
稳定趋势	长期方向、平滑变化、低噪声外推
强季节性	周期结构、日历效应、节假日规律
冷启动新品	跨序列迁移、相似产品经验、少样本适配
促销需求	外生变量、事件冲击、非线性响应
高频金融	局部模式、低延迟、风险约束和微结构信号
间歇性需求	需求是否发生、发生后的规模、服务水平成本

因此，模型选择不应只问“哪个模型最高级”，而要问“这个问题像什么，哪些模型在这类问题上有证据”。如果所有问题都交给同一个模型，就会把很多不同形态强行压进同一个判断逻辑里。

大模型也不能完全逃避 No Free Lunch。大模型的综合能力很强，说明它在许多任务上的平均水平高；但当多个强模型都达到较高水平后，真正的差异往往出现在细分场景。一个模型可能在通用聊天中表现很好，但不一定最懂金融交易；一个模型可能

语言能力强，但不一定最适合时间序列预测。学习 AI 预测时，要把大模型看成强专家之一，而不是万能答案。

11.4　大模型为什么仍然有价值

既然大模型也不万能，为什么还要学习它？从传统统计模型进入大模型支持的预测，至少有两个实际价值。

第一，大模型可以把预测能力规模化。传统统计模型在单条序列上可能非常好，但当企业面对上万、上百万条序列时，逐条建模、逐条调参、逐条解释会遇到规模边界。Chronos、TimeGPT 等时间序列基座模型把大量跨领域经验压缩到预训练参数中，使用者整理好 `unique_id`、`ds` 和 `y`，就可以快速生成候选预测。

第二，大模型可以帮助缺少专家经验的场景。一个新产品、新区域或新企业可能只有几百个观测值，内部并没有足够历史判断未来走势。预训练模型在大量相似序列上见过扩张、收缩、季节波动和异常冲击，可以把这些外部经验迁移到当前任务中。这不保证最优，但能把完全没有经验的起点提高到一个较稳定的候选基准。

可以用一个比喻理解：传统统计模型像分散的业余选手，如果恰好选中了适合当前场景的模型，它可能给出很高精度；如果选错，表现也会很差。大模型像经过集中训练的选手，平均水平更稳定，见过的题型更多。但即便如此，它仍然需要面对具体场景的选择问题。MoE 要解决的就是这个选择和组合问题。

11.5　从组合预测到 MoE

MoE 并不是大模型时代才出现的思想。在时间序列预测中，它和组合预测有很深的关系。Bates 和 Granger 在 1969 年提出 forecast combination, 组合预测的思想：没有一个模型在所有地方都最好，不同模型在某些场景下各有可取之处，因此把多个预测结果组合起来，往往比只相信一个模型更稳健。

最简单的组合是平均。假设有三个模型分别给出预测，最终预测可以取三者均值。复杂一些的组合会给不同模型分配不同权重。如果某个模型在相似场景中过去表现更好，就给它更高权重；如果某个模型经常失误，就降低它的权重。

神经网络预测模型中也有类似思想。例如可以用不同损失函数训练多个模型：一个更重视 MAE，一个更重视 RMSE，一个更重视分位数损失。也可以针对不同预测步长训练模型：有的模型擅长 `T + 1`，有的模型擅长 `T + 7`，有的模型擅长 `T + 14`。短期预测和长期预测关心的模式不同，得到的参数和结果也会不同。

这样会形成一个预测矩阵：

维度	例子
模型结构	ARIMA、ETS、LSTM、Transformer、TimeGPT
损失函数	MAE、RMSE、分位数损失、业务成本损失
预测步长	1 步、7 步、14 步、长期预测
输入特征	只用历史值、加入日历、加入促销、加入价格
数据频率	小时、日、周、月

最终 forecast 可以来自这个矩阵中的多个结果。组合预测解决的是 “多个预测者如何合成一个预测”；MoE 则进一步把 “谁参与预测、各占多少权重” 做成可以学习的机制。

11.6　手工练习：MoE

我们用一个手工练习解释 MoE。每位同学拿到一张印有时间序列的纸。纸上左边是历史数据，右边留出未来区间。每个同学根据历史走势，在右边画出自己认为未来可能出现的曲线。

在这个练习中，每个同学就是一个 expert，每条手画曲线就是一个 expert forecast (专家预测)。四个同学组成一组，每组有四条独立预测。每个人先独立判断，不抄别人的线。这个设置对应真实业务中的专家预测：管理者面对历史销量、客流或库存数据时，也可能先凭经验形成未来判断。

我们随后给出真实未来走势，并让同学给自己的预测打分。规则很简单：如果预测线和真实线落在同一个格子里，得 1 分；如果落在相邻格子里，得 0.5 分；其他情况得 0 分。这样每个专家都有一个个人预测得分。

接下来，每组把四条预测线组合成一条 MoE 预测。最朴素的方法是在每个时间点取四条线的中间位置或平均位置，再连成最终预测线。注意，MoE 的得分不是四个个人得分的平均，而是组合后那条线本身与真实线比较得到的分数。

这个练习虽然粗糙，但已经包含 MoE 的三个核心部件：

练习对象	MoE 中的含义
每个同学	expert, 专家
每条手画预测线	expert forecast
选择哪些同学参与	routing, 路由
给某些线更高信任	gating, 门控
组合后的曲线	final forecast
真实曲线后的打分	样本外评价（out-of-sample evaluation）

练习的第一个结论是：预测非常难。即便只是一条看起来不复杂的时间序列，很多同学的预测仍然偏离真实走势，也有少数同学表现很好。这说明 “看一眼历史曲线

就能画出未来”是错觉。真实业务中，如果每个专家都要花很久才能给出预测，而且结果还不稳定，纯人工预测的成本会很高。

第二个结论是：专家之间有显著差异。有人画得保守，有人画得激进；有人认为未来会上升，有人认为会下降。对同一条序列，即使大家看到完全相同的历史数据，共识也不一定高。这种差异不是噪声本身，而是 MoE 可以利用的信息来源。

第三个结论是：组合预测通常“不太差”。MoE 未必总能超过最好专家，但它常常能避免最差专家把结果带偏。如果一个人的预测极端偏高或偏低，平均或加权组合会把这个极端判断拉回中间。如果有一个专家非常接近真实结果，组合也可能把整体预测向正确方向拉动。

11.7　为什么 MoE 能降低风险

MoE 的价值不在于神奇地找到未来，而在于降低单一判断的脆弱性。

第一，它可以减少极端预测的影响。某个专家在一条序列上预测得很激进，可能刚好猜中，也可能在另一条序列上造成大错。组合机制会让其他专家的判断参与制衡，降低单个极端输出对最终结果的影响。

第二，它可以利用局部专长。一个专家不需要在所有任务上都好，只要在某一类任务上有明显优势，就有价值。例如一个模型擅长季节性需求，另一个模型擅长趋势外推，第三个模型擅长促销冲击。只要 gating 能识别当前序列更像哪一类，专家差异就能转化为预测优势。

第三，它可以降低管理决策风险。管理中常说重大决策要广泛听取意见。MoE 把这件事模型化：多个专家独立判断，再由系统根据证据组合。对企业来说，这比只相信一个人、一个模型或一次 API 输出更稳健。

第四，它可以避免把所有能力塞进一个巨无霸模型。单一大模型训练成本高，推理成本高，也不一定适应所有场景。几个中等复杂度、专长清楚的专家模型，通过路由机制协作，可能更便宜、更可控。

但 MoE 也有前提。如果所有专家高度同质化，给出的预测几乎一样，那么组合没有意义。MoE 真正需要的是差异化专家：不同模型、不同损失、不同输入、不同频率、不同领域经验。差异越有结构，路由越可能学到“谁更适合什么”。

11.8　Gating：谁参与，谁更重要

手工练习中，最简单的组合是四个同学等权平均。真实 MoE 系统会加入 gating，也就是门控或路由机制。它要回答两个问题：

1. 当前任务应该交给哪些专家？
2. 每个专家的预测应该占多少权重？

假设当前时间序列被编码成一个状态向量（state vector）x_t。这个向量可以包含历史数值、趋势强度、季节性、波动率、促销、节假日、价格、天气、产品类别和过去

误差等信息。gating 网络根据 x_t 输出每个专家的权重：

$$g_1(x_t), g_2(x_t), \ldots, g_K(x_t) \tag{11.1}$$

每个专家给出自己的预测：

$$\hat{y}_{t+h}^{(1)}, \hat{y}_{t+h}^{(2)}, \ldots, \hat{y}_{t+h}^{(K)} \tag{11.2}$$

最终预测就是加权组合：

$$\hat{y}_{t+h} = \sum_k g_k(x_t)\hat{y}_{t+h}^{(k)} \tag{11.3}$$

如果所有 g_k 都相等，就是等权平均。如果某些专家权重更大，就是加权组合。softmax 常用于把 gating 输出归一化为权重，使它们可以解释为“相对信任程度”。

真实预测中，gating 看不到未来真实值。它只能根据历史表现、当前特征和相似序列经验来判断。例如，某个专家过去在强季节性序列上表现好，当当前序列也呈现强季节结构时，gating 就可以给它更高权重。某个专家在促销场景下经常低估峰值，遇到促销日就应降低权重或交给更合适的专家。

这说明 MoE 的关键不只是专家本身，还包括“判断专家”的机制。一个预测平台真正要沉淀的，不仅是模型库，还有模型选择经验。

11.9　Top-K gating 与稀疏激活

如果系统里只有三五个专家，可以让所有专家都参与预测。但现代大模型或大规模预测平台中，专家数量可能很多。每次都激活所有专家，成本会很高。Top-K gating 的思路是：系统有很多专家，但每次只激活权重最高的前 K 个专家。

例如系统中有 64 个专家，当前任务只选择最相关的 2 个或 4 个参与计算。稀疏门控专家层最早就在大规模神经网络中展示了“总容量很大、单次激活较少”的效率思路 [Shazeer et al., 2017]，Switch Transformer 则进一步把这一路线推向大规模语言模型训练 [Fedus et al., 2022]。这样可以同时获得两类好处：

- 模型总体容量很大，因为专家库很丰富。
- 单次推理成本较低，因为只激活少数专家。

这就是稀疏激活（sparse activation）的价值。系统可以“看起来很大”，但每个输入只使用其中一部分。DeepSeek 等模型引发市场关注，一个重要原因就是这种思路让人重新思考算力成本：AI 发展不只是堆更多 GPU，也包括更聪明的架构、更有效的路由和更低的单次调用成本。

稀疏激活并不是说算力不重要。算力仍然重要，尤其在预训练阶段。但 MoE 提醒我们：同样的预算，可以通过专家分工和路由机制获得更高效率。一个系统不必每次都让所有参数一起工作，就像一个企业不必让所有员工参加每一次会议。

现代 MoE 还可能包含共享专家（shared experts）和稀疏专家（sparse experts）。共享专家处理所有任务都需要的通用能力，稀疏专家处理特定场景。对时间序列预测来说，共享专家可以学习通用趋势和尺度变换，稀疏专家可以分别处理季节性、间歇性、促销、金融高频或冷启动等任务。

11.10　MoE 在时间序列预测中的设计

把 MoE 放回时间序列，可以从专家设计开始。专家不一定都是同一种模型，也不一定都是神经网络。一个实用预测系统可以同时包含统计模型、机器学习模型、深度模型和人工规则。

专家类型	可能负责的任务
Naive / Seasonal Naive	简单基准、稳定季节性序列
ETS / ARIMA	趋势、季节、可解释单序列预测
Croston / TSB	间歇性需求和备件需求
LightGBM / XGBoost	外生变量丰富、表格特征强的任务
LSTM / DeepAR	多序列共享模式和自回归结构
Transformer / TimeGPT	跨领域预训练、长上下文和多序列迁移
Chronos / TimesFM	零样本或少样本时间序列基座模型预测
业务规则	库存约束、政策限制、人工红线

gating 的输入也可以分层设计。第一层看序列形态，例如趋势、季节性、波动率、零值比例和历史长度。第二层看业务上下文，例如品类、地区、价格、促销和节假日。第三层看模型历史表现，例如过去滚动窗口误差、预测区间覆盖率、尖峰召回能力和业务成本。

一个简化流程如下：

1. 对每条序列提取统计特征和业务特征。
2. 根据特征选择候选专家，例如 Top-3。
3. 每个专家生成点预测或分位数预测。
4. gating 根据历史误差和当前特征分配权重。
5. 组合预测输出给评估系统和业务系统。
6. 真实值出现后，把误差反馈给 gating 和专家库。

这个流程把 MoE 和 C06-C07 的评估章节连接起来。没有评估，gating 就没有学习信号；没有滚动窗口，就很难知道专家在相似未来中是否可靠。MoE 不是替代评估，而是更依赖评估。

11.11　等权、多数、去极值与业务权重

MoE 不一定一开始就要很复杂。很多业务系统可以从简单组合开始。

等权平均是最低成本方案。所有专家权重相同，适合模型能力相近、没有足够历史误差可学习权重的情况。

按验证误差加权更进一步。过去在滚动测试中表现好的专家获得更高权重，表现差的专家权重降低。这里要注意，权重应来自过去可见的数据，不能使用未来真实值。

去极值平均类似体育比赛评分，去掉最高和最低预测，再对中间结果平均。它适合担心极端预测误导决策的场景。

业务成本加权把库存、缺货、现金流或服务水平写入权重。比如关键备件宁可高估一些也不能缺货，促销销量宁可人工复核也不能盲目低估。

学习式 gating 用模型自动学习权重。它需要更多数据和更严格验证，但能处理高维特征、复杂相似性和多专家系统。

这些方法没有绝对优劣。一个成熟预测平台往往从简单组合开始，再逐步引入学习式 gating。过早使用复杂 MoE 可能会让系统难以解释，也可能在数据不足时过拟合。

11.12　回归组合与概率组合

我们还要补充两类更一般的组合方法。第一类是基于回归的组合（regression-based combination）。假设多个模型分别给出对 Y_{t+h} 的预测，可以把这些预测值作为解释变量，再用回归模型学习组合系数。它和 MoE 相似，但允许更自由的系数：某个模型的权重可以大于 1，也可以小于 0。

负权重并不一定荒谬。如果某个模型在某类场景中经常反向偏误，它的预测仍然可能提供信息。回归组合可以利用这种反向信号；再配合 LASSO（Least Absolute Shrinkage and Selection Operator）等正则化，还可以把没有贡献的模型系数收缩到零。这样，组合不仅是平均，也可以是带约束的统计学习问题。

第二类是概率组合（probabilistic combination）。点预测可以平均，但预测区间和预测分布不能随便平均。两个模型都声称给出 90% 预测区间，直接把上界平均、下界平均，并不能保证组合后的区间仍然有 90% 覆盖率（coverage）。概率组合要处理的是分布层面的一致性（coherence）：均值、分位数、尾部风险和覆盖率都要一起检查。

这说明 MoE 只是预测组合工具谱系中的一部分。它的优势是可以自然嵌入深度学习和大模型架构；但在正式预测系统中，仍应理解回归组合、概率组合和机器学习集成（ensemble）的更广背景。

11.13　Many simple experts 与 few complex experts

MoE 有两种常见思路。

第一种是 many simple experts: 每个专家很简单，但专家数量很多。它接近 “群众

智慧”，优点是分散、稳健、容易并行；缺点是如果专家质量参差不齐，平均可能稀释真正有价值的判断。

第二种是 few complex experts: 专家数量不多，但每个专家很强，分别在某些领域有明显优势。现代大模型中的 MoE 更接近这种模式。一个系统可以有文字专家、图像专家、推理专家、代码专家；时间序列内部也可以有趋势专家、季节专家、短期专家、长期专家、促销专家和间歇性需求专家。

选择哪种思路，取决于问题和数据。如果你无法明确构造强专家，可以先用许多简单模型和基准组合，利用稳定性。若你有足够领域知识和历史评估证据，few complex experts 可能更高效。

这也对应学习和职业发展。今天追求 “什么都会” 越来越难，因为通用能力会被大模型快速覆盖；更有价值的是形成一个明确垂直领域的专长，并学会与其他专家协作。MoE 是模型架构，也是组织协作的隐喻。

11.14　MoE 与时间序列基座模型

时间序列基座模型需要 MoE 思想，是因为它们面对的数据形态非常多。一个统一模型要处理月度宏观指标、小时级电力负荷、零售销量、交通流量、金融价格和长尾商品需求，就必须有内部差异化能力。Time-MoE 等工作正是在把稀疏专家路由引入大规模时间序列基座模型 [Shi et al., 2024]。

TimeGPT、Chronos、TimesFM、Moirai 等模型的具体实现不同，但都要解决类似问题：

- 如何把不同频率、不同长度、不同尺度的序列放入统一表示空间？
- 如何让模型在短历史、长历史、冷启动和多变量场景中都能工作？
- 如何决定当前序列更依赖趋势、季节、外生变量还是相似序列经验？
- 如何在推理成本可控的情况下提供跨领域能力？

MoE 是一种回答。它可以在模型内部路由，也可以在模型外部作为预测平台。内部 MoE 把不同 token、patch 或 embedding 送给不同专家模块；外部 MoE 则把同一业务任务交给多个模型，再在平台层组合结果。

对教学来说，外部 MoE 更容易理解和实现。你可以先把 ARIMA、ETS、LightGBM、TimeGPT 和 Chronos 都当作专家，在同一留后集或滚动窗口上评估，再学习一个简单权重。等理解清楚后，再进入神经网络内部的稀疏 MoE。

11.15　与管理决策的关系

MoE 也能帮助理解管理决策。企业做预测时，通常有多类信息来源：销售团队、供应链专家、统计模型、AI 模型、市场新闻、促销计划和管理者经验。把这些信息来源变成可比较、可记录、可追踪的预测者，本质上就是把组织经验平台化。

这个练习中，同学们先独立画线，再把预测组合起来。这个过程类似集中决策：先保留独立判断，再进行汇总，而不是一开始就互相影响。真实企业中也应避免所有人先听最高级别人员的判断再表态，否则组合只是在重复同一个声音。

好的预测平台应记录：

- 每个专家或模型看到了哪些信息。
- 每个专家在相似历史任务中的表现。
- 最终采用了哪些专家。
- 每个专家的权重是多少。
- 预测错误后应如何复盘。

这样，预测从个人经验转化为组织能力。专家离职、模型升级、业务变化时，系统仍然保留可追溯的证据。

11.16　误区和边界

MoE 不是万能方法。使用时至少要注意以下边界。

第一，专家必须有差异。如果所有专家都使用相同数据、相同模型、相同损失和相同假设，组合只是重复。

第二，gating 不能偷看未来。权重必须基于预测时可用的信息。用测试集真实误差直接调权重，会造成数据泄漏。

第三，组合可能掩盖关键风险。对关键备件、金融风控或医疗资源等场景，平均预测不一定足够。少数尖峰或尾部风险可能比平均误差更重要。

第四，复杂 MoE 需要足够评估数据。专家越多，gating 越复杂，越容易在历史上过拟合。没有滚动评估和稳定监控，就不能把复杂组合直接用于自动决策。

第五，成本仍然存在。MoE 可以降低单次推理成本，但如果专家库过大、特征工程过重、评估流程复杂，整体系统仍可能变得昂贵。设计 MoE 时要同时考虑预测精度、延迟、可解释性、维护成本和数据隐私。

11.17　一个可实施的小项目

可以用以下小项目把本章内容落到代码。

选择一个多序列数据集，例如 M4 子集、M5 子集、门店销量或公开电力负荷。对每条序列建立至少四个专家：

- Seasonal Naive。
- ETS 或 ARIMA。
- LightGBM 或 XGBoost。

- TimeGPT、Chronos 或另一个时间序列基座模型。

然后做三种组合：

1. 等权平均。
2. 按滚动窗口 MAE 的倒数加权。
3. 按序列特征分组后分别加权，例如强季节性组、间歇性组、短历史组。

最后比较每种方法在测试集上的 MAE、RMSE、sMAPE、预测区间覆盖率和关键业务指标。报告中要说明：哪些专家在哪些序列上更好，组合是否超过最好单模型，哪些场景组合反而变差。

这个项目的重点不是证明 MoE 一定赢，而是训练一种思维：模型库、专家差异、路由规则和评估证据必须一起出现。

11.18　小结

MoE 把 No Free Lunch 转化成可操作的预测系统。它承认没有单一模型适合所有场景，因此让不同专家分别发挥专长，再用 gating 决定谁参与、谁更重要。组合预测是 MoE 的历史基础，Top-K gating 和稀疏激活是大模型时代的效率扩展。

对时间序列预测来说，MoE 的价值在于处理形态差异：趋势、季节、冷启动、促销、外生变量、高频噪声和间歇性需求都需要不同能力。学习 MoE 后，不应再问“哪个模型最好”，而应问“这个任务需要哪些专家，权重如何学习，评估证据是否支持这个组合”。

11.19　练习

1. 解释 No Free Lunch 为什么会出现在预测问题中，并举两个业务例子。
2. 用一条销售序列设计四个专家模型，说明每个专家擅长什么。
3. 比较等权平均、验证误差加权和去极值平均的优缺点。
4. 说明 Top-K gating 如何降低大模型推理成本。
5. 设计一个 gating 特征表，至少包含五个序列特征和三个业务特征。
6. 解释为什么 MoE 的得分不是专家得分的简单平均。
7. 找一个场景说明组合预测可能比最好单模型差，并解释原因。
8. 用滚动窗口评估一个简单组合预测，报告单模型和组合模型的误差。
9. 讨论在库存预测中，MoE 的权重是否应该只由 MAE 决定。
10. 写一页短文：MoE 作为模型架构和 MoE 作为管理决策机制有什么共同点。

第 12 章

模型蒸馏与时间序列小模型

模型蒸馏（Knowledge Distillation, KD）是把大模型能力转化为小模型能力的方法。它的目标不是机械地删掉参数，而是在尽量保留关键预测能力的前提下，降低模型规模、推理成本和部署门槛。本章接在 MoE 之后，说明当我们已经能够组合多个专家时，如何进一步把时间序列基座模型、专家模型或闭源 API 的能力迁移到更小、更便宜、更适合特定业务场景的模型中。

12.1 学习目标

完成本章后，你应该能够：

- 解释为什么大模型不能直接部署到所有预测场景中。
- 区分模型剪枝（pruning）、量化、微调、迁移学习和知识蒸馏。
- 说明教师模型（Teacher Model）和学生模型（Student Model）的角色。
- 理解为什么先训练大模型、再蒸馏小模型不同于直接训练小模型。
- 设计时间序列蒸馏中的数据、损失函数和评估流程。
- 判断蒸馏模型在成本、精度、隐私和边缘部署中的价值与边界。

12.2 从 MoE 到蒸馏

上一章讨论了 MoE。MoE 的基本思想是把多个专家模型的预测按权重组合起来。每个专家不必在所有场景都最强，只要在某一类问题上有专长，就能为最终预测提供贡献。权重可以等分，也可以由历史误差、序列特征或 gating 网络学习出来。

这个思想在大模型预测中很重要，因为训练和调用巨型模型的成本很高。如果只做财务预测、电商预测、库存预测或某个具体行业的时间序列预测，大模型中大量能力可能不会被用到。一个通用模型可能懂语言、代码、图像和推理，但当前任务只需要处理季节性销量和促销冲击。此时，直接调用完整大模型就会浪费算力、内存和调用费用。

MoE 解决的是“多个专家怎样分工和组合”。蒸馏解决的是另一个问题：如果某个大模型已经学到了很多知识，但我们只需要其中一部分能力，能否把这些能力压缩到一个更小的模型中？这就是从大模型走向可部署小模型的关键。

蒸馏这个词很形象。化学中，蒸馏是从混合物中提取更关键、更纯净的部分；模型蒸馏则是从庞大模型中提取当前任务真正需要的知识。一个 70B 参数模型很强，但一张常见 RTX 4090 只有 24GB 显存，无法完整加载这类模型。即使硬件能勉强运行，推理速度、能耗和成本也可能无法接受。

更重要的是，70B 参数并不意味着每个参数都对你的任务有用。如果你只做电商月销量预测，模型中关于诗歌、代码、图像描述和通用聊天的大量能力可能都不是关键。蒸馏要做的，就是让一个小模型尽量保留当前任务需要的能力，而不是保留大模型的全部能力。

12.3　为什么不直接训练小模型

既然最后想要小模型，为什么不一开始就训练一个小模型？这个问题很自然，也是理解蒸馏的关键。

直接训练小模型的难点在于，小模型参数空间更小，表达能力受限。如果训练数据有限、任务复杂或优化空间非凸（non-convex），小模型很容易停在一个不好的局部最优（local optimum）。大模型参数多、容量大，可以先在更大的空间中学习复杂结构，再把这种结构通过输出、概率分布或中间表示传给小模型。

可以把 Teacher Model 看成一个强先验（strong prior）。它在大规模数据和复杂训练过程中，已经探索过很多可能的规律。Student Model 不再从零开始摸索，而是在老师模型的引导下学习：哪些方向更有希望，哪些模式更重要，哪些输出分布更合理。

可以用一个 “抓小偷” 的例子理解蒸馏。假设要在北京发现所有偷盗行为，最极端的办法是给每个人身边都放一个监测器。这相当于一个过参数化大模型，覆盖范围极广，但成本极高。之后我们根据数据发现：某些时间段风险更低，某些区域风险更低，某些楼宇有监控和门禁，于是可以减少这些地方的部署，把资源集中在更有风险的区域。这样系统变小了，但仍然保留了捕捉关键信号的能力。

蒸馏也是这样。先让大模型覆盖尽可能多的场景，再根据目标任务把不必要的部分压缩掉。最终的小模型不必拥有老师的全部能力，只要在目标任务上接近甚至超过老师，就有部署价值。

12.4　Teacher Model 与 Student Model

蒸馏通常有两个角色。

角色	含义	典型特点
Teacher Model	老师模型	大、强、训练成本高、覆盖场景广
Student Model	学生模型	小、快、部署便宜、面向特定任务

Teacher Model 可以是一个开源预训练模型、时间序列基座模型、企业内部大模型，

也可以是只能通过 API 调用的闭源模型。对本书来说，Chronos 和 TimeGPT 分别代表本地开源时间序列基座模型与远程 API 时间序列基座模型两种 teacher 来源 [Ansari et al., 2024, Garza et al., 2024]。Student Model 通常结构更小，参数更少，部署环境更轻量。

蒸馏不是简单裁剪。直接删掉一半层数或一半参数，并不保证剩下的模型仍然知道怎么预测。真正的蒸馏要让 Student Model 学习 Teacher Model 的行为，例如输出值、概率分布、隐藏表示或决策边界。

在时间序列预测中，Teacher Model 可能是 Chronos、TimeGPT、TimesFM、Moirai 或企业内部模型；Student Model 可能是一个小型 Transformer、轻量 LSTM、TCN、树模型，甚至是一个面向特定业务的回归模型。关键不是学生模型形式必须与老师完全相同，而是它能否在目标任务上继承老师的有用能力。

12.5　蒸馏的三个目标

蒸馏至少有三个常见目标。

第一是 **压缩模型规模**。大模型参数多，推理慢，占用内存大。蒸馏后的小模型可以降低延迟、减少 GPU/CPU 占用，并让模型进入手机、手表、边缘设备（edge devices）、本地服务器或普通业务系统。

第二是 **领域迁移**。开源时间序列基座模型的训练数据可能和你的业务不完全一致。它可能在公开时间序列、电商数据或通用文本上训练，而你关心的是电力负荷、旅游需求、学生成绩、医院物资或量化交易。蒸馏可以把老师模型的通用能力迁移到你的领域数据中。

第三是 **任务纠正和增强**。Student Model 不一定永远弱于 Teacher Model。如果你只关心多步预测、关键尖峰、非零需求日或某个特殊业务指标，可以让 Student Model 在这些目标上继续训练。它可能在整体能力上不如老师，但在目标任务上更稳定、更便宜、更可控。

这也是蒸馏和普通压缩的区别。压缩只关心变小；蒸馏关心“小模型是否学到了目标任务需要的知识”。一个好的 Student Model 应该是面向使用场景的模型，而不是老师模型的简单低配版。

12.6　蒸馏、量化、剪枝和微调

蒸馏经常和几种相近方法一起出现，需要区分。

方法	核心动作	主要目的	风险
量化	降低参数数值精度，例如 16-bit 到 8-bit 或 4-bit	减少内存和加速推理	精度损失、数值不稳定
剪枝	删除权重、神经元、attention head 或网络层	缩小模型结构	删掉关键能力
微调	在领域数据上继续训练原模型或适配层	让模型适应新场景	成本高、过拟合、遗忘
蒸馏	训练小模型模仿大模型行为	迁移知识并降低部署成本	学错老师、评估不足

量化可以让更大模型塞进有限硬件。例如把参数压缩到 4-bit，可以显著减少显存需求，但会引入数值误差。如果直接把一个很大的模型用很粗糙精度装进设备，可能得到一个 “大而不准” 的模型。另一条路线是先把模型蒸馏到更小规模，再用较高精度运行，得到 “小而准” 的模型。

剪枝看起来也像蒸馏，但它主要是删结构。蒸馏则强调训练一个学生，让它学会老师输出背后的规律。剪枝可以和蒸馏结合：先设计更小结构，再用老师模型引导训练。

微调与蒸馏也不同。微调通常保留原模型结构，只在新数据上调整参数或适配层。蒸馏则可以改变模型结构，让学生模型更小。实际项目中，常常会把微调、量化、剪枝和蒸馏组合使用。

12.7　蒸馏的基本损失函数

蒸馏的核心是定义 “学生和老师有多像”。一个常见训练目标可以写成：

$$\mathcal{L} = \alpha\, \mathcal{L}_{\text{true}}(y, y_{\text{student}}) + (1-\alpha)\, \mathcal{L}_{\text{teacher}}(y_{\text{teacher}}, y_{\text{student}}) \tag{12.1}$$

第一项让 Student Model 拟合真实值。第二项让 Student Model 模仿 Teacher Model。α 控制真实标签和老师输出之间的权衡。

如果老师输出的是概率分布，可以使用 KL distance（Kullback-Leibler divergence, KL）：

$$\mathrm{KL}(p_{\text{teacher}} \,\|\, p_{\text{student}}) \tag{12.2}$$

它衡量老师分布和学生分布之间的差距。对分类任务，例如 “明天是否会发生需求”“未来是否上涨”，可以让学生模仿老师的类别概率。对概率预测任务，可以让学生模仿老师给出的分位数、分布参数或采样分布。

如果老师输出的是点预测，可以使用 MAE、MSE 或 Huber loss 衡量二者差距。对

时间序列预测，损失还可以按 horizon 加权：短期预测和长期预测的重要性不同，多步预测中越往后误差越容易累积。

蒸馏损失不应只看数学方便。它要服务业务目标。如果库存任务更怕缺货，损失函数就应对低估需求赋予更高代价；如果金融任务更怕极端亏损，损失就应关注尾部风险。

12.8　几种蒸馏层次

蒸馏可以发生在不同层次。

基于响应的蒸馏（response-based distillation）只关心最终输出。Teacher Model 给出预测值、类别或文本答案，Student Model 学习产生相似输出。这种方法适合闭源 API，因为我们拿不到老师模型内部参数，只能观察输入和输出。

概率或 logits 蒸馏（probability or logits distillation）关心输出分布。分类或 token 模型会给出 logits（逻辑值）或概率分布，Student Model 学习老师的“软标签”（soft labels）。软标签比硬标签包含更多信息：老师不仅告诉学生正确答案，还告诉学生其他候选答案有多接近。

Token-level distillation 适用于 Transformer 类模型。Chronos 类时间序列模型会把数值序列 token 化；学生可以在每个 token 的概率分布上模仿老师。这样学习的是序列局部结构，而不只是最终预测值。

特征或表示蒸馏（feature or representation distillation）让学生模仿老师的中间表示，例如 embedding、hidden state（隐藏状态）或 attention pattern（注意力模式）。它适合老师和学生结构相近的情况，但实现更复杂。

Prediction-level distillation 在时间序列中很常见。老师对未来多个 horizon 给出预测，学生学习这些预测曲线、分位数或分布。它可以忽略模型内部结构，直接服务最终 forecast。

这些层次可以组合使用。比如一个学生模型同时拟合真实值、老师点预测、老师分位数和某个中间 embedding。但组合越复杂，越需要严格验证，避免只是在训练集上贴近老师，未来却不稳定。

12.9　时间序列蒸馏流程

一个面向时间序列预测的蒸馏流程可以分为七步。

1. 选择 Teacher Model，例如 Chronos、TimeGPT、TimesFM、Moirai 或企业内部模型。
2. 准备领域数据，包括目标序列、时间列、序列 ID、外生变量和预测步长。
3. 让 Teacher Model 对训练窗口和验证窗口生成预测、分位数或概率分布。
4. 设计 Student Model，例如减少 Transformer 层数、hidden size 和 attention head。

5. 定义蒸馏损失，同时包含真实值误差和老师输出误差。

6. 在滚动窗口或留后集上验证 Student Model。

7. 比较老师、学生、简单基线和业务规则的精度、延迟、成本和稳定性。

需要注意，蒸馏不是先看测试集再调学生模型。所有权重、损失和超参数都应来自训练集或验证集。最终测试集要模拟真实未来，只用于最后评估。

在多序列任务中，数据格式通常仍然是长表：

列	含义
`unique_id`	序列编号
`ds`	时间戳
`y`	目标变量
外生变量列	促销、节假日、价格、天气、事件等

Student Model 应在和 Teacher Model 相同的信息边界下训练。老师看不到未来促销之外的信息，学生也不能看；否则学生可能看似超过老师，实际是因为数据泄漏。

12.10　以 Chronos 类模型为例

Chronos 类模型提供了一个直观例子。它把原始时间序列转成 token，再用类似 Transformer 的结构做编码、解码和预测。用户下载预训练 checkpoint 后，可以在本地加载模型，输入历史序列并得到未来预测。

如果要把 Chronos 类模型蒸馏成更小版本，可以先拿到 Teacher Model 的 checkpoint。开源模型通常来自 Hugging Face 或其他模型仓库。checkpoint 记录了模型在某个训练阶段的参数状态。你不一定知道它完整训练工艺，但至少能加载它的结构和权重。

然后设计 Student Model。假设 Teacher Model 有 12 层 Transformer，可以把学生设为 6 层；如果 hidden size 很大，可以减半；如果有 12 个 attention head，可以减到 6 个。参数量通常不会只线性减少，因为 Transformer 中很多矩阵维度相乘，hidden size 减半可能带来更大幅度压缩。

接下来，用领域数据生成老师预测。比如用 M4、M5、电力负荷或企业销量数据，把历史窗口交给 Teacher Model，记录未来 horizon 的点预测和分位数预测。Student Model 训练时，一方面拟合真实未来，一方面模仿老师输出。

一个简化训练目标可以写成：

$$\mathcal{L} = 0.5\,\mathrm{MAE}(y_{\text{true}}, y_{\text{student}}) + 0.5\,\mathrm{MAE}(y_{\text{teacher}}, y_{\text{student}}) \tag{12.3}$$

如果老师能输出分位数，还可以让学生同时学习中位数、10% 分位数和 90% 分位数。这样学生不只学一条曲线，也学老师对不确定性的表达。

12.11　为什么 Student 可能超过 Teacher

Student Model 不一定只能模仿 Teacher Model。它也可以在目标场景中超过老师。

原因很简单：Teacher Model 是通用模型，要兼顾很多场景；Student Model 可以只服务一个场景。比如老师在公开时间序列上训练，覆盖零售、能源、交通、金融和传感器数据；学生只做某家企业的周度销量预测。只要领域数据足够、损失函数合适，学生就可能在这个窄场景中比老师更稳定。

这也是蒸馏与领域迁移的结合。老师提供通用先验，学生吸收领域数据。最终学生模型不是老师的缩小复制品，而是 “老师知识 + 领域校正” 的结果。

不过，这种超过必须通过严格评估证明。不能只看训练集，也不能只看一个漂亮案例。应在多个滚动窗口、多个 horizon、多个序列组上比较：

- Student 是否超过 Teacher。
- Student 是否超过简单基线。
- Student 是否降低延迟和成本。
- Student 是否保留预测区间覆盖率。
- Student 是否在关键业务样本上稳定，例如高需求日或非零需求日。

如果学生只是平均误差略低，却错过关键尖峰，业务上未必更好。

12.12　工程细节

蒸馏看起来是模型压缩，实际很大程度上是工程工艺。以下细节会显著影响结果。

窗口切分：训练窗口、验证窗口和预测 horizon 要模拟真实使用场景。不能把未来信息泄漏进历史窗口。

尺度处理：不同序列量级差异大时，需要标准化、log 变换或按序列 scaling（尺度变换）。否则大销量序列会主导损失。

数据增强：适度噪声、重采样或窗口扰动可以提高学生模型稳定性，但过度增强会改变真实数据分布。

temperature：在概率或 logits 蒸馏中，temperature 可以让老师输出分布更平滑或更尖锐。太尖锐的分布像硬标签，信息少；太平滑则可能模糊关键差异。

学习率：蒸馏时学习率通常比从头训练更低。目标是精细迁移，而不是让学生模型大幅震荡。

batch size 和调度：batch size（批大小）、warmup（预热）、weight decay（权重衰减）、early stopping（早停）和学习率调度都影响收敛。很多成功蒸馏依赖长期经验，而不是一次运行就能稳定得到。

老师预测缓存：如果 Teacher Model 很大，每次训练都实时调用会很贵。可以提前把老师输出缓存下来，但要记录模型版本、输入窗口和生成参数，保证可复现。

评估维度：蒸馏完成后，不能只报告精度。还要报告模型大小、推理延迟、内存占用、每千次预测成本和失败案例。

我们特别提醒：蒸馏本身不一定省算力。训练期间可能需要运行 Teacher Model、Student Model 和大量验证流程，成本仍然高。蒸馏的收益主要体现在部署之后：推理更快、更便宜、更适合本地或边缘设备。

12.13　闭源模型的输出蒸馏

如果 Teacher Model 不开源，例如只能通过 API 调用，就无法拿到 checkpoint 和中间层。这时仍然可以做响应蒸馏（response distillation）。若 teacher 是大语言模型，还要先理解其 token、embedding、上下文窗口和生成机制，避免把一次文本输出误当作稳定知识来源 [Alammar and Grootendorst, 2024]。

基本做法是准备一批输入样本，把它们发送给 Teacher API，记录输出结果，然后训练自己的 Student Model 模仿这些输入输出。对预测任务来说，输入可以是历史序列、外生变量和任务说明；输出可以是点预测、分位数或结构化 JSON。

这种方法有实际价值，但必须谨慎。

第一，API 输出可能随模型版本变化。必须记录调用日期、模型名、参数、prompt 模板和返回格式。

第二，闭源模型输出不一定是事实标签。学生如果只学习老师错误，就会复制错误。最好同时使用真实标签约束。

第三，要注意服务条款、数据权限和模型许可。不是所有 API 输出都允许用于训练另一个模型。

第四，闭源模型的行为可能依赖 prompt 和上下文。如果 prompt 变化，老师输出会变，学生学到的能力也会变化。

因此，输出蒸馏适合做原型和内部工具，但进入生产前必须经过合规、评估和数据治理检查。

12.14　嵌入式 AI 与个人化模型

我们还要讨论一个未来趋势：AI 可能从云端大模型走向嵌入式 AI（embedded AI）。现在很多 AI 工具仍然把指令发送到云端，由云端大算力返回结果。未来，更多模型会嵌入手机、手表、车载设备、企业本地服务器或专用硬件中，在不联网时也能完成有效工作。

这些设备不可能长期运行巨大模型，因此蒸馏会变得更重要。一个健康手表可能不需要通用聊天模型全部能力，只需要理解个人心率、睡眠、运动和生活习惯；一个库存设备可能只需要预测本仓库的补货风险；一个量化工具可能只需要某些市场结构和风险信号。

这意味着未来可能出现“蒸馏服务”。客户提供目标数据、业务场景和性能要求，服务商帮助把时间序列基座模型蒸馏成一个专属小模型，再部署到本地或边缘设备。

这个小模型可能只有几十 MB 或几百 MB，但对特定用户、特定企业或特定领域很有价值。

对预测人员来说，这改变了角色。未来不一定是亲自从零训练每个模型，而是要判断：

- 哪个 Teacher Model 可信？
- 领域数据是否足够支持蒸馏？
- Student Model 应该保留哪些能力？
- 精度、成本、隐私和延迟如何权衡？
- 蒸馏后的模型什么时候需要重新训练？

模型变小以后，人的责任并没有消失。相反，模型选择、数据边界、评估设计和业务解释会变得更重要。

12.15　与 MoE 的关系

MoE 和蒸馏可以结合使用。

一种方式是 **先 MoE 后蒸馏**。先用多个专家模型和 gating 得到一个强预测系统，再训练一个 Student Model 模仿这个系统的输出。这样可以把复杂专家团队压缩成一个小模型，便于部署。

另一种方式是 **先蒸馏后 MoE**。先把几个大专家分别蒸馏成小专家，再用 MoE 组合它们。这样专家库仍然保留差异，但每个专家都更轻。

还可以 **蒸馏 gating**。如果一个复杂平台已经能根据序列特征选择专家，可以训练一个小 gating 模型模仿它的选择逻辑。这样，模型选择经验也能被压缩。

时间序列预测很适合这种组合。趋势、季节、节假日、外生变量、高频模式和间歇性需求可以由不同专家处理；每个专家又可以通过蒸馏降低成本。最终系统既有分工，又能部署。

12.16　实施清单

做一个时间序列蒸馏项目前，至少检查以下问题。

1. Teacher Model 是否可用，是否能稳定复现输出？
2. 是否有足够领域数据支持 Student Model 学习？
3. 目标是压缩、迁移、纠正，还是三者都有？
4. Student Model 的结构为什么适合部署环境？
5. 损失函数是否同时包含真实值和老师输出？

6. 是否有滚动窗口或留后测试模拟真实未来？
7. 是否比较了简单基线、Teacher 和 Student？
8. 是否报告模型大小、延迟、内存和成本？
9. 是否检查关键业务样本，而不只看平均误差？
10. 是否记录模型版本、数据版本和蒸馏配置？

如果这些问题没有回答清楚，蒸馏模型即使能跑，也不应直接进入业务决策。

12.17　小结

模型蒸馏把大模型能力转化为小模型能力。它不是简单删参数，而是通过 Teacher Model 引导 Student Model 学习输出、概率分布、中间表示或预测行为。蒸馏可以压缩模型、迁移领域能力，也可以在特定任务上纠正和增强老师模型。

对时间序列预测来说，蒸馏的价值在于把时间序列基座模型变成可部署工具。Chronos、TimeGPT、TimesFM 或企业内部模型可以作为老师，小型 Transformer、LSTM、树模型或业务模型可以作为学生。最终是否成功，不取决于概念是否先进，而取决于评估证据：小模型是否在目标 horizon、目标领域和目标业务成本下足够好。

12.18　练习

1. 用自己的话解释为什么蒸馏不同于直接训练小模型。
2. 为一个电商销量预测任务设计 Teacher Model 和 Student Model。
3. 写出一个同时拟合真实值和老师预测的蒸馏损失函数。
4. 比较量化、剪枝、微调和蒸馏的区别。
5. 说明为什么 Student Model 可能在某个细分任务上超过 Teacher Model。
6. 设计一个滚动窗口评估方案，比较 Teacher、Student 和 Naive 基线。
7. 如果只能调用闭源 API，如何构造 response distillation 数据集？
8. 讨论一个适合 embedded AI 的预测场景，并说明为什么需要蒸馏。
9. 为蒸馏后的模型设计一张部署报告表，包含精度、延迟、模型大小和成本。
10. 说明 MoE 和蒸馏如何组合成一个低成本预测平台。

第 13 章

间歇性需求预测与 TimeGPT

间歇性需求（intermittent demand）是商业预测中很常见、也很棘手的一类问题。某些商品大多数日期销量为 0，偶尔出现订单；某些备件、慢销品和长尾 SKU（Stock Keeping Unit, SKU）也有类似特征。普通模型容易被大量零值误导，或者给出负需求。本章以 TimeGPT 和经典间歇性需求模型为例，说明间歇性需求预测（Intermittent Demand Forecasting）如何处理这类数据、如何评估结果，以及外生变量为什么重要。

13.1 学习目标

完成本章后，你应该能够：

- 识别间歇性需求和普通平稳需求（stationary demand）的区别。
- 区分“需求是否发生”和“发生后需求有多大”两个问题。
- 使用 `log1p` 变换（log-one-plus transform）降低尺度差异并避免负值预测。
- 用 TimeGPT 对多条需求序列进行预测。
- 理解本地 Chronos-2 模型如何做零样本预测。
- 将多条长短不一的时间序列整理成长表格式。
- 将 TimeGPT 与 Croston、IMAPA、TSB 等经典基线比较。
- 判断外生变量是否能改善间歇性需求预测。

13.2 什么是间歇性需求

间歇性需求的典型特征是“很多 0，少数非 0”。例如备件、冷门商品、长尾电商 SKU、特殊医疗物资和低频企业采购。它和普通日销商品不同：没有需求的日子很多，但一旦出现需求，数量可能并不小。

还可以看一些更极端的例子：航空发动机、昂贵设备、关键零部件、光刻机相关工具等。这类产品可能很长时间只有 0 或 1 次需求，但一次需求就关系到高成本、高服务水平或关键生产能力。对这类序列，预测平均销量不是唯一目标，判断需求是否会发生同样重要。

这种数据会给模型带来三个困难。

第一，均值很低但方差可能很高。简单平均会低估峰值。

第二，百分比误差不稳定。真实值为 0 时，MAPE 无法定义。

第三，业务成本常常不对称。备件缺货可能造成停机损失，库存过多又会占用资金。

因此，间歇性需求可以拆成两个问题：第一，未来某天或某周是否会发生需求；第二，如果发生需求，需求量有多大。Croston 类方法正是沿着这个思路，把需求大小和需求间隔分开处理 [Croston, 1972]。时间序列基座模型也应该在评估中接受这个拆分，而不是只看一条平滑曲线是否贴近平均值。

13.3　数据准备

示例 notebook 使用 M5 子集数据，包含 `unique_id`、`ds`、`y` 以及价格、事件类型等外生变量。基本格式和前面章节一致：

```
df["ds"] = pd.to_datetime(df["ds"])
df = df.sort_values(["unique_id", "ds"]).reset_index(drop=True)
```

建模前先可视化单条序列。间歇性需求的图形通常会出现长时间贴近 0、偶尔跳起的尖峰。这个检查能帮助我们判断是否需要特殊模型，而不是盲目套用常规预测流程。

M5 这类多序列数据还提醒我们：间歇性需求不是一个单序列问题。不同商品、门店和品类之间可能共享节假日、价格、促销和地区消费习惯。TimeGPT 这类时间序列基座模型的优势之一，是能够在多条序列中学习共同结构；但前提是 `unique_id`、时间列、目标列和外生变量都整理正确。

Chronos-2 练习使用了类似的数据组织思想。时间序列数据和普通表格数据的一个重要差别，是不同序列长度可能不一致。如果用宽表把 H1、H2、H3 等序列横向摆放，缺失和切分都会很麻烦；长表则把所有序列纵向拼接，每一行是一个观测点，用 ID 区分它属于哪条序列。

长表至少需要三列：

列	含义
`item_id` 或 `unique_id`	时间序列编号，例如 `H1`, `H2`, `H414`
`timestamp` 或 `ds`	时间顺序，可以是真实日期，也可以是顺序编号
`target` 或 `y`	要预测的目标数值，通常称为目标列（target column）

M4 小时级示例数据有 414 条序列，拼接后约几十万个观测点。长表格式可以自然容纳长短不一的序列，也方便批量预测、分布式存储和后续评估。无论使用 TimeGPT API、Chronos-2 本地模型，还是经典统计基线，先把这三列整理正确，都是多序列预测的共同入口。

13.4　变换与切分

notebook 使用 `log1p` 变换：

```
df_t = df.copy()
df_t["y"] = np.log(df_t["y"] + 1)
```

`log1p` 有两个作用。第一，它压缩大需求值，降低尖峰对模型的影响。第二，它能处理 0，因为 $\log(0+1)=0$。预测后再用 $\exp(\text{value})-1$ 还原到原尺度。

评估时按每个 `unique_id` 留出最后 28 天：

```
H = 28
test_df = df_t.groupby("unique_id", group_keys=False).tail(H)
train_df = df_t.drop(test_df.index).reset_index(drop=True)
```

多序列任务必须按序列分组切分，不能简单取整个表最后 28 行。

13.5　使用 TimeGPT

TimeGPT 可以直接接收包含 `unique_id`、`ds` 和 `y` 的多序列表。示例中使用长预测模型、80% 区间和少量微调步数：

```
fcst_df = client.forecast(
    df=train_df,
    h=H,
    level=[80],
    finetune_steps=10,
    finetune_loss="mae",
    model="timegpt -1 -long -horizon",
    time_col="ds",
    target_col="y",
    id_col="unique_id",
)
```

正式 notebook 不应写真实 API key。使用环境变量：

```
from nixtla import NixtlaClient

client = NixtlaClient(api_key=os.environ["NIXTLA_API_KEY"])
```

预测后要把所有预测列从 log 空间还原到原尺度，再与测试集真实值合并计算误差。

使用 TimeGPT 时还要记住，它不是免评估的答案。时间序列基座模型能利用跨序列预训练经验，适合冷启动、长尾和多序列场景 [Garza et al., 2024]；但在一个具体库存问题中，它仍然必须和 Croston、IMAPA、TSB、LightGBM、DeepAR、NHITS 等基线放在同一测试集上比较。

13.6　本地 Chronos-2 练习

本地时间序列基座模型预测不必依赖远程 API。流程是先下载并解压预训练模型文件，然后在 notebook 中指定模型目录，加载 pipeline，再把长表格式的历史数据传入模型。Chronos-2 延续了把时间序列 token 化并进行零样本预测的路线 [Ansari et al., 2025]。模型目录中通常包含 `config.json` 和 `model.safetensors`；示例模型权重约为数百 MB，因此不少笔记本电脑可以用 CPU 跑通，只是速度慢于 GPU。

调用这类 pipeline 时，核心参数和 API 预测很相似：

- `context_df`：包含历史观测的长表。
- `id_column`：序列 ID 列。
- `timestamp_column`：时间列。
- `target_column`：目标变量列。
- `prediction_length`：向未来预测多少期。
- `quantile_levels`：需要输出的分位数水平。

示例对 414 条 M4 小时级序列同时预测未来 24 期。因为模型推理阶段不是重新训练，而是使用已经训练好的权重，所以几百条序列的预测可以在几十秒到一分钟量级完成。这个速度说明，时间序列基座模型可以成为实时或准实时预测系统的一部分，但前提是模型文件、依赖版本和硬件资源都稳定。

Chronos-2 也能输出预测区间。设置 `quantile_levels = [0.1, 0.9]` 时，输出的是 10% 和 90% 分位数，中间是 80% 区间。画图时应同时展示历史值、真实未来值、点预测和上下分位数，这样才能看到模型是否只是给出一条平滑曲线，还是对不确定性有合理表达。

13.7　经典基线

间歇性需求有专门的统计模型。示例 notebook 使用 `StatsForecast` 比较：

- **CrostonClassic**：分别估计需求大小和需求间隔。
- **CrostonOptimized**：优化平滑参数的 Croston 版本。

- **IMAPA（Intermittent Multiple Aggregation Prediction Algorithm）**：在多种聚合尺度上处理间歇性需求。
- **TSB（Teunter-Syntetos-Babai）**：用需求发生概率和需求大小共同建模。

这些基线很重要。TimeGPT 是强模型，但不能只和空模型比较。针对间歇性需求的特征化和模型选择研究也说明，长尾和零值序列需要专门诊断，而不能只套普通销量预测流程 [Li et al., 2023]。只有当它在相同训练集、测试集和指标下超过合理基线，才说明引入时间序列基座模型有实际价值。

13.8　评估指标

notebook 使用 MAE 汇总 TimeGPT 和基线模型。对于间歇性需求，MAE 比 MAPE 更稳健，因为真实值为 0 的样本很多。评估时应按序列计算，再汇总到整体平均，避免销量大的序列完全主导结果。

除了 MAE，业务中还可以关注：

- 缺货次数或缺货概率。
- 库存持有成本。
- 服务水平。
- 高需求日的召回能力。
- 预测区间覆盖率（prediction interval coverage）。

如果模型平均误差低，但总是错过少数关键需求尖峰，业务上仍可能不可接受。

这类任务尤其要避免让“最差的一次预测”伤害关键决策。少数尖峰日可能比平均误差更重要，因此评估时除了 MAE，还应检查高需求日、缺货风险和预测区间覆盖。

还可以把评估拆成两张表。第一张表评价所有日期的整体误差，回答模型日常表现如何。第二张表只看非 0 日期或高需求日期，回答模型是否抓住了真正需要备货的时刻。对于关键备件，第二张表往往比第一张表更接近业务损失。

13.9　外生变量

间歇性需求往往受事件影响，例如促销、节假日、价格变化和特殊活动。notebook 中把事件类型列作为未来已知外生变量传入 TimeGPT：

```
event_cols = [
    "event_type_Cultural",
    "event_type_National",
    "event_type_Religious",
```

```
    "event_type_Sporting",
]
```

外生变量的关键要求是：预测未来时这些变量必须已知或可可靠预测。这类输入通常称为未来已知外生变量（future known exogenous variables）。节假日通常已知，未来价格和促销计划也可能已知；但未来新闻热度或突发事件不能随便当作已知输入。

外生变量要区分 “历史可见” 和 “未来可用”。历史价格、过去促销和过去事件可以帮助模型学习关系，但预测未来时，只有已经排定的节假日、促销计划、价格策略或已知活动才能作为未来输入。把未来才会知道的信息放入模型，会造成数据泄漏，让评估结果虚高。

销售需求练习更直观地展示了协变量的作用。第一次预测只给模型 ID、时间和历史销量，模型能够捕捉部分下降趋势，但越往后越容易滞后或错位。第二次加入 `Open`、`Promo`、`SchoolHoliday`、`StateHoliday`、`Customers` 等变量后，预测曲线更贴近真实值，区间也明显收窄。这说明很多销量变化不是由历史销量本身决定的，而是由门店是否营业、促销、假期和客流共同驱动。

这些协变量可以理解为进入特征矩阵 `X` 的额外列。经过投影后，它们会影响注意力权重和最终预测。不过，模型究竟利用了变量名的语义，还是利用了列值与目标之间的统计关系，需要实验验证。一个可做的练习是打乱协变量列名或改变列顺序，观察预测是否变化；如果变化很大，就要进一步检查模型对字段名、字段含义和数据结构的依赖。

13.10　不同频率下的模型选择

我们提醒你，TimeGPT 在月度和周度等较低频数据上表现通常更稳健，而在高频数据上，树模型如 LightGBM、XGBoost 在量化和高频交易中仍非常重要。这提醒我们，时间序列基座模型不是所有频率的唯一答案。

低频经济和商业预测重视跨序列泛化和稳健性；高频金融或交易数据更强调微结构特征、低延迟和局部模式。模型选择必须服从数据频率和业务决策周期。

TimeGPT 应放在一组基线模型中比较，包括 LightGBM、LSTM、DeepAR、DFT 和 NHITS。重点不是宣布某个模型永远最好，而是说明模型表现会随数据频率、预测步长和业务场景改变。月度和周度商业数据中，TimeGPT 这类预训练模型常能利用跨序列信息取得较稳健表现；但在高频交易或非常局部的短期预测中，LightGBM、XGBoost 等树模型仍可能更有优势。

对间歇性需求来说，频率选择本身就是建模选择。日度数据可能太稀疏，周度或月度聚合后规律更清楚，但会牺牲补货时点的精度。实践中可以同时比较日度、周度和月度版本：低频版本用于战略备货，高频版本用于短期补货提醒。

13.11　从预测到库存决策

间歇性需求预测最终要服务库存和服务水平。一个预测值可以转化为订货量、安全库存、补货触发点或人工复核清单。模型输出越接近自动化决策，越需要把业务成本写进评估。

例如，某个模型把大多数 0 预测得很准，但错过一次关键备件需求，可能导致设备停机；另一个模型平均误差略高，但能提前识别关键需求日，业务上反而更有价值。因此，本章的评估不应停在 MAE 排名，而应继续问：这个模型能不能降低缺货，能不能控制库存，能不能让人工复核集中在最有风险的 SKU 上？

13.12　小结

间歇性需求预测的关键不是把 0 当作普通小数值，而是理解需求发生和需求大小的双重不确定性。TimeGPT 提供了强大的通用预测能力，但仍需要 log 变换、合理基线、稳健指标、外生变量检查和业务成本解释。不要让平均误差掩盖关键尖峰，也不要让最坏的一次预测伤害关键决策，是这类问题的底线。

13.13　练习

1. 找一条长尾商品需求序列，统计 0 值占比和非 0 需求的分布。
2. 比较原始尺度和 `log1p` 尺度下的预测图。
3. 用 Croston、IMAPA 和 TimeGPT 计算同一测试集的 MAE。
4. 单独统计非 0 日期或高需求日期的误差，并和整体 MAE 比较。
5. 设计一个比 MAE 更贴近库存决策的评价指标。
6. 判断一个外生变量是否能在预测未来时提前知道，并说明原因。
7. 比较日度、周度和月度聚合下同一商品的预测难度。
8. 将一组长短不一的时间序列整理成长表，并说明 `id_column`、`timestamp_column` 和 `target_column` 分别对应什么。
9. 比较只用历史销量和加入 `Promo`、`Holiday`、`Customers` 等协变量后的预测图与区间宽度。

第 A 章

使用 PKU 集群

本附录说明如何使用 PKU 集群（PKU Cluster）和北京大学高性能计算平台（High Performance Computing, HPC）运行课程 notebook 和预测实验。对于 TimeGPT 这类普通 API 调用和小规模示例，本地电脑通常足够 [Garza et al., 2024]；对于深度学习模型、长时间运行任务或 GPU 需求，集群会更合适。

我们还会强调另一类选择：本地个人 AI 算力设备。它们不替代集群，但补充了集群和云服务之间的空白。公开数据、大规模训练和多人共享任务仍适合放到集群；涉及病人、学生、企业经营或个人健康的敏感数据，或者需要反复快速试验的小模型，则可以优先考虑本地设备。选择运行环境时，不只看 “有没有 GPU”，还要看数据能否外发、任务是否需要多人共享、模型文件是否能装入内存，以及结果是否需要长期复现。

A.1 学习目标

完成本附录后，你应该能够：

- 申请并登录 PKU 高性能计算平台账号。
- 使用 Web 界面创建 JupyterLab 交互式应用（interactive application）。
- 通过 SSH（Secure Shell）和 OTP（One-Time Password）登录集群。
- 在本地和集群之间传输文件。
- 在 Jupyter 所用 Python 环境中安装课程依赖。

A.2 账号申请

进入北京大学高性能计算平台页面：

```
https://hpc.pku.edu.cn
```

按照 “使用指南 - 账户申请” 进入在线申请页面。我们建议申请教学系列的未名二号资源。申请完成后，记录你的用户名、登录入口和平台通知，不要把密码或动态验证码发给他人。

A.3　Web 登录与 JupyterLab

初学者优先使用 Web 界面。进入平台后，主要用两个功能：

- **Shell**：打开服务器上的命令行。
- **交互式应用**：创建 JupyterLab 等图形化应用。

创建 JupyterLab 应用时，可以从较小配置开始：

- 节点数：`1`
- 核心数：`2`
- 运行时间：`120` 分钟
- 课程预留资源：按课程通知填写 `reservation` 或其他调度参数

资源申请越大，等待时间可能越长。课程练习通常不需要一开始就申请大量 CPU、GPU 或长时间任务。先确认 notebook 能运行，再根据任务规模增加资源。

使用集群时不要把 “资源越大越好” 当成原则。核心数、节点数和运行时间都会影响排队和资源占用；课程练习应从小配置开始，确认 notebook、数据和 Python 环境都正确后再扩大。

我们建议的入门配置是单节点、2 个 CPU 核、120 分钟左右。这样足够完成多数入门 notebook，也不会过度占用教学节点。如果课程通知给出了专门的 reservation 字段，务必按原样填写；否则作业可能进入普通队列，等待时间会明显变长。

如果只是课程最后几十分钟做练习，可以把运行时间设成 30 或 60 分钟，减少排队和资源占用。进入 Jupyter 后如果发现没有网络、依赖装不上或 API 无法访问，先检查是否填写了课程 reservation；没有使用预留资源时，作业可能被分配到普通或无网络节点。平台临时不可用时，可以先在本地电脑跑同一个 notebook，但要记录使用的是哪个环境。

如果使用本地 AI 设备或实验室工作站，也要像使用集群一样记录环境：设备型号、可用内存、是否有 GPU、Python 路径、模型目录和关键包版本。很多新设备采用 CPU、GPU 和共享内存更紧密结合的架构，能缓解传统独立显卡显存不足的问题；但这不代表可以忽略环境记录。模型能否加载、推理速度多快、是否能并发使用，都和硬件与软件环境直接相关。

A.4　SSH 与 OTP

如果需要在终端中登录集群，需要先绑定手机动态令牌。可以使用 Microsoft Authenticator、Google Authenticator、FreeOTP 等 OTP 应用。绑定入口见平台文档：

```
https://hpc.pku.edu.cn/authotp/otp
https://hpc.pku.edu.cn/ug/guide/access/
```

Windows 用户建议使用 Windows Terminal。macOS 和 Linux 用户可以直接使用系统 Terminal。SSH 命令形式如下：

```
ssh 你的学号 @wmjx2 -login.pku.edu.cn
```

登录时终端输入密码通常不会显示字符，这是正常现象。输入完成后按回车，再根据提示输入 OTP 动态验证码。

第一次用 SSH 连接服务器时，终端可能会询问是否信任服务器指纹。确认地址来自学校平台文档后，可以输入 `yes`。注意，高性能计算平台的 OTP 或 verification code 可能和学校统一身份认证里的手机令牌不是同一套机制；如果密码、OTP 或账号状态不一致，需要按平台说明或联系助教处理。

还要区分登录节点和计算节点。SSH 直接连入时，通常先进入登录节点；登录节点适合查看文件、编辑脚本、提交作业，不适合直接运行重计算任务。通过 Web 界面创建 JupyterLab 交互应用时，平台会为你分配某个计算节点，在申请的时间内使用相应资源。理解这个区别，可以避免把大任务误跑在登录节点上，也能解释为什么 Web Jupyter 和 SSH 看到的环境有时不完全一样。

A.5　联网与安全

部分登录节点默认不联网，需要按学校平台说明开通网络。官方示例会包含账号、密码和网络命令。实际使用时要注意：

- 不要把真实密码写入 notebook、脚本、截图或 Git 提交。
- 不要在共享文档中保留包含密码的完整命令。
- 如果必须在终端输入密码，完成后不要复制到课程报告中。
- 优先使用平台 Web 功能或官方推荐方式完成联网。

联网成功后再安装依赖或下载公开数据。计算节点是否能联网取决于平台设置，不要假设所有节点都能直接访问外网。

还要区分密码、OTP 和 API key。集群登录需要账号密码和动态验证码，模型调用需要 API key，这些都不应该出现在 notebook 输出、截图、Git 提交或共享文档中。安全问题不是附加项，而是可复现实验的一部分。

A.6　文件上传与下载

图形化传输可以使用 FileZilla Client。配置时选择 SFTP（Secure File Transfer Protocol）：

- 协议：`SFTP`
- 主机：按平台说明填写，例如数据节点地址

- 端口：22
- 登录类型：交互式
- 用户名：你的学号或平台账号

第一次通常输入门户密码，第二次输入 OTP 动态验证码。传输课程 notebook、数据文件和结果图时，注意不要上传含有密钥的 `.env` 文件或 notebook 输出。

如果熟悉命令行，也可以使用 `scp` 或 `rsync`。例如：

```
scp local_file.csv 你的学号 @wmjx2 -login.pku.edu.cn:~/forecasting/
```

A.7　安装 Python 包

在 Jupyter 中先确认当前 notebook 使用的 Python：

```
import sys
sys.executable
```

它会输出类似路径：

```
/nfs -share/software/anaconda/2020.02/bin/python
```

在 Shell 中使用同一个 Python 安装包：

```
/nfs -share/software/anaconda/2020.02/bin/python -m pip install
↪  pandas
```

这样安装的包才能被当前 Jupyter kernel（内核）找到。不要只输入 `pip install ...` 就假设安装到了正确环境。

记录运行环境也很重要。集群上的 Jupyter、Shell 和登录节点可能使用不同 Python；安装依赖前先看 `sys.executable`，运行实验后保留日志、输出图和评估表，才能让后续复查知道代码究竟在哪个环境中运行。

A.8　运行课程 notebook 的建议流程

1. 登录平台并创建 JupyterLab 应用。
2. 上传课程 notebook 和所需数据。
3. 打开 notebook，确认 Python kernel。

4. 运行环境检查单元，例如 `import pandas`、`import numpy`。
5. 安装缺失依赖。
6. 从小样本开始运行预测代码。
7. 保存输出图表和评估结果。
8. 关闭不用的交互式应用，释放资源。

第一次使用时，可以先运行一个最小检查：

```
3 + 4
import sys
sys.executable
```

如果这两步成功，再安装课程包或运行预测示例。不要一开始就直接跑完整深度学习任务；先确认服务器环境、路径和包安装位置都正确。

运行 Transformer 练习时，也应从最小 PyTorch 检查开始。先确认当前 kernel 能执行 `import torch`，再查看 `torch.cuda.is_available()` 和可见 GPU 数量。小型时间序列示例可以先在 CPU 上跑通；如果要增加序列数量、窗口长度、`d_model`、注意力头数或训练轮数，再申请 GPU 资源。这样可以避免把环境错误、路径错误和模型调参问题混在一起。

运行本地 Chronos-2 练习时，还要先确认模型文件位置。Chronos 类模型依赖本地 checkpoint、配置文件和推理环境，因此路径和依赖管理会直接影响能否复现实验 [Ansari et al., 2024]。示例模型通常以压缩包形式提供，解压后目录中应能看到 `config.json` 和 `model.safetensors` 等文件。notebook 里填写的是模型目录，而不是压缩包路径。Windows 用户可以把模型解压到当前用户目录或课程目录下；macOS 和 Linux 用户也应放在自己有读权限、路径不含奇怪空格的位置。

本地模型练习的排错顺序可以更具体：

1. 先确认 notebook 使用的 Python 路径。
2. 再确认 `import pandas`, `import matplotlib`, `import torch` 能通过。
3. 检查模型目录是否存在 `config.json` 和权重文件。
4. 先用一小段数据或少量序列测试 pipeline（推理流水线）。
5. 再扩大到 M4 小时级数据或销售需求数据。

如果 Parquet 文件读取失败，可以切换到课程提供的 CSV 文件；如果 NumPy、PyTorch 或 Chronos 相关包版本冲突，先记录错误信息，再在当前 kernel 对应的 Python 中卸载并安装兼容版本。不要在多个终端和多个 kernel 之间反复安装，否则很容易出现 “终端显示安装成功，但 notebook 仍然导入失败” 的情况。

提交 notebook 时，如果直接导出 PDF 失败，可以先保存为 HTML，再用浏览器打开 HTML 并打印成 PDF。提交前检查输出图是否包含历史值、真实值、预测值和预测区间；如果作业要求比较协变量版本，也要保留两组图和简短解释。

A.9　常见问题

如果 Jupyter 能打开但无法导入包，通常是安装到了错误 Python 环境。回到 `sys.executable` 检查路径。

课程练习中最常见的错误，是在 notebook 里执行安装命令时没有装到当前 kernel。处理方式是先在 notebook 里查看 `sys.executable`，再用这个解释器执行 `-m pip install`；如果使用 notebook 魔法命令，也要确认它对应的是当前 Python 3.12 环境。不要只看到终端显示 “安装成功” 就认为 notebook 一定能导入。

如果 SSH 登录失败，先确认用户名、密码、OTP、登录地址和网络环境。OTP 和北京大学统一认证令牌不是同一个概念，不要混淆。

如果任务长时间排队，先减少核心数、节点数或运行时间。教学练习应从小资源开始。

如果下载数据失败，确认当前节点是否联网，以及是否需要先按平台说明开通网络。

如果 Web 界面已经打开但 notebook 运行很慢，先检查是否申请了过小或过大的资源。过小会导致运行慢，过大可能排队久。课程练习通常从 2 核短时任务开始最稳妥。

A.10　小结

集群不是预测方法本身，而是运行环境。使用集群时最重要的是三件事：使用正确账号和 OTP 登录，使用正确 Python 环境安装依赖，使用合适资源运行任务。先让一个小 notebook 跑通，再扩大实验规模。

A.11　练习

1. 创建一个 2 核、120 分钟的 JupyterLab 应用，并记录使用的 Python 路径。
2. 在 notebook 中运行 `import sys; sys.executable`，再用同一路径安装 `pandas`。
3. 上传一个 CSV 文件到集群，在 notebook 中读取并显示前 5 行。
4. 用 SSH 登录集群，运行 `pwd`、`ls` 和 `whoami`。
5. 写一段说明，解释为什么不能把密码或 API key 写入 notebook。

第 B 章

Python 入门

本附录提供课程所需的 Python 最小基础。你不需要先成为软件工程师，但需要能够读懂 notebook（交互式笔记本）、修改参数、运行函数、安装包，并把模型输出整理成表格和图形。

B.1 学习目标

完成本附录后，你应该能够：

- 使用 Python 做基础计算。
- 理解变量、字符串、列表和索引。
- 使用内置函数和第三方模块。
- 定义简单函数并阅读 docstring。
- 使用 `pip` 安装课程依赖。

B.2 本地 Jupyter 环境

如果你还没有 Python 环境，我们建议先安装 Anaconda。Windows 用户从 Anaconda 官网选择 64 位图形安装包；macOS 用户注意区分 Intel 芯片和 Apple Silicon 芯片。安装过程可以使用图形安装器，一路按提示完成即可。

安装后打开 Anaconda Navigator 或开始菜单中的 Anaconda 文件夹，启动 Jupyter Notebook。Jupyter 会在浏览器中打开一个文件列表，这个列表对应你电脑上的某个本地目录。新建 notebook 时选择 Python kernel（Python 内核），然后先运行一个最简单的单元：

```
3 + 4
```

如果输出 `7`，说明 notebook 基本可用。给 notebook 重命名并保存后，会得到一个 `.ipynb` 文件。Windows 上启动 Jupyter 时通常还会打开一个黑色终端窗口；不要关闭它，否则浏览器里的 notebook 也会断开。

B.3　Python 作为计算器

Python 可以直接做算术：

```
3**2
17 - 4
4 / 2
(5 + 4) * 3
```

注意 `**` 表示乘方，`^` 不是乘方。在 Python 中，`^` 是异或运算。取余使用 `%`：

```
x = 2004
print(x % 100 == 0)
print(x % 4 == 0)
```

这类表达式常用于日期、周期和分组判断。

B.4　变量

变量（variable）用等号 `=` 赋值：

```
a = 3
b = 5
c = a + b
c
```

变量是给值起名字。预测代码中常见变量包括 `df`、`train_df`、`test_df`、`h`、`model_name` 和 `freq`。阅读 notebook 时，先弄清每个变量保存了什么对象。

B.5　字符串

字符串（string）可以用单引号或双引号：

```
last_name = "Li"
first_name = "Feng"
```

反斜杠会触发转义。普通字符串中 `\n` 表示换行，原始字符串前加 `r`：

```
print("Hello\nWorld")
print(r"Hello\nWorld")
```

字符串可以拼接和重复：

```
"time" + "series"
"ha" * 3
```

在 API 预测中，prompt 本质上就是字符串。构造 prompt 时要特别注意换行、引号和变量插入。

后续章节中的 JSON 输出和模型提示词都可以从这里理解：prompt 是字符串，模型返回是待解析文本，预测结果最终要进入 DataFrame 才能画图、合并和评估。

B.6　索引与切片

Python 索引（indexing）从 0 开始：

```
name = "Feng Li"
name[0]
name[ -1]
name[0:4]
```

切片（slicing）`[a:b]` 取从 `a` 到 `b` 之前的位置，不包含 `b`。时间序列中也常用类似思想，例如取最后 12 期：

```
tail = values[ -12:]
```

B.7　列表

列表（list）用于存放一组值：

```
values = [1, 5, 7, 9, 12]
len(values)
values[0]
values[ -2:]
```

列表可以修改：

```
values[2] = 1000
values.append(9999)
```

预测任务中，列表常用于保存多个模型名、多个指标、多个预测步长或多条记录。

B.8　内置函数

Python 有很多内置函数（built-in functions），例如 `len`、`sum`、`min`、`max`、`print`、`range`、`type` 和 `help`。遇到不熟悉的函数，可以先查看帮助：

```
help(len)
```

调试 notebook 时，`type(obj)` 很有用。它能告诉你对象是字符串、列表、字典、DataFrame 还是其他类型。

B.9　导入模块

很多功能需要导入模块（module）：

```
import math
math.exp(0)
```

也可以只导入某个函数：

```
from math import exp
exp(0)
```

本课程常用模块包括：

- `pandas`：数据表（DataFrame）处理。
- `numpy`：数值计算。
- `matplotlib`：画图。
- `openai`：调用兼容 OpenAI 风格的 API。
- `nixtla`：调用 TimeGPT，这类 API 把时间序列基座模型封装成 Python 工作流 [Garza et al., 2024]。
- `utilsforecast`：评估预测结果。
- `tsfeatures` 或类似工具：把时间序列转换成趋势、自相关、线性度等特征表。

B.10　定义函数

函数把重复逻辑封装起来。一个简单例子：

```
def fib(n):
    """Print a Fibonacci series up to n."""
    a, b = 0, 1
    while a < n:
        print(a, end=" ")
        a, b = b, a + b
```

预测 notebook 中也应使用函数减少重复，例如：

```
def split_train_test(df, h):
    train = df.iloc[: -h].copy()
    test = df.iloc[ -h:].copy()
    return train, test
```

函数的好处是让实验更可复现。你只需要改变参数，而不是复制粘贴整段代码。

后续如果计算时间序列特征，代码通常也是这个思路：输入一个包含 `unique_id`、`ds` 和 `y` 的 DataFrame，函数返回一张特征表。每一行对应一条序列，每一列对应一个特征，例如 `trend`、`linearity`、`curvature` 或 `acf`。理解函数输入和输出，比背某个包的名字更重要。

B.11　安装第三方包

在 notebook 中可以安装第三方包（third-party packages）：

```
!python -m pip install pandas
```

课程中会用到 `nixtla` 等预测相关包。你可以直接在 notebook 或终端运行：

```
python -m pip install nixtla
```

安装成功后，再在 notebook 中测试：

```
import nixtla
```

在终端中也可以安装：

```
python -m pip install pandas
python -m pip install pandas -U
python -m pip uninstall pandas
```

推荐使用 `python -m pip`，因为它更明确地使用当前 Python 环境。不要在不了解环境的情况下随意加 `--break-system-packages`，除非你清楚它会修改系统 Python 管理规则。

Windows、Anaconda、Jupyter 和服务器环境可能各自使用不同 Python。安装包时先确认这个 `python` 就是当前 notebook 的 kernel；很多初学错误不是代码逻辑错，而是包安装到了另一个环境。

这个原则在本地 AI 设备上同样成立。无论是在普通笔记本、集群、云服务器还是本地 AI 盒子上运行 Chronos、PyTorch 或其他深度学习包，先确认当前 Python，再安装依赖。Chronos 这类本地时间序列基座模型尤其依赖正确的模型文件、运行库和解释器环境 [Ansari et al., 2024]。硬件变强不能自动解决环境问题；如果 notebook 使用的 kernel 和终端里安装包的 Python 不一致，仍然会出现 `ModuleNotFoundError`。

涉及 GPU 或本地模型时，可以在 notebook 中先做最小检查：

```
import sys
print(sys.executable)

import torch
print(torch.cuda.is_available())
```

如果模型只能在 CPU 上运行，也不代表不能做练习，只是推理速度可能更慢。本书中的小型时间序列预测模型通常可以先在 CPU 上跑通，再根据需要迁移到 GPU 或共享内存设备。

B.12　课程中的最小 Python 工作流

一个常见预测实验会按下面流程写：

```
import pandas as pd

df = pd.read_csv("data.csv")
df["ds"] = pd.to_datetime(df["ds"])
df = df.sort_values(["unique_id", "ds"])

h = 12
train = df.iloc[: -h].copy()
test = df.iloc[ -h:].copy()
```

之后再调用模型、生成预测、合并真实值并计算误差。只要理解变量、DataFrame、函数和模块，就能读懂大部分课程代码；更系统的时间序列实践可以继续参考 Forecasting: Principles and Practice 的 Python/R 思路与评价框架 [Hyndman and Athanasopoulos, 2021]。

因此，本附录不是孤立语法课。变量、列表、函数和模块的最终目的，是让你能读懂 notebook 中的数据流：历史数据如何变成训练集，预测步长如何传给模型，模型输出如何进入评估表。

B.13　常见错误

`NameError` 通常表示变量还没有定义，可能是前面的 notebook 单元没有运行。

`ModuleNotFoundError` 表示当前 Python 环境没有安装某个包，或者安装到了另一个环境。

`KeyError` 常见于 DataFrame 列名写错，例如使用了 `date` 但实际列名是 `ds`。

`TypeError` 常见于把字符串当数字、把列表当函数、或者参数类型不匹配。

遇到错误时不要只看最后一行。错误信息会告诉你出错的文件、行号和对象名称。

B.14　小结

学习本课程需要的 Python 并不复杂。重点是能读懂 notebook 中的数据流：读入数据、清洗列、切分训练测试、调用模型、保存预测、计算误差。先把这些基本动作练熟，再学习更复杂的模型。

B.15　练习

1. 用 Python 计算 `12` 个月后日期对应的预测步长变量 `h`。
2. 创建一个包含 5 个销量数字的列表，取最后 2 个值。
3. 写一个函数，输入真实值和预测值，返回绝对误差。
4. 导入 `math`，计算 exp(1)。
5. 用 `python -m pip` 检查当前环境是否安装了 `pandas`。

第 C 章

Linux 基础

本附录介绍运行课程代码所需的 Linux 和命令行（command line）基础。很多 AI 工具、服务器、集群和开源包都默认在 Linux 环境中运行；现代 LLM 和预测模型的实践也经常需要在命令行中管理环境、模型文件和实验脚本 [Alammar and Grootendorst, 2024]。你不需要掌握系统管理，但需要会登录服务器、移动文件、查看目录、运行 Python 和理解基本管道命令。

C.1 学习目标

完成本附录后，你应该能够：

- 使用终端（terminal）登录远程服务器。
- 用 `pwd`、`ls`、`cd` 浏览文件系统（file system）。
- 创建、移动、删除和查看文件。
- 使用简单编辑器修改服务器上的文件。
- 理解标准输入（standard input, stdin）、标准输出（standard output, stdout）、标准错误（standard error, stderr）、重定向（redirection）和管道（pipe）。

C.2 为什么要学 Linux

Linux 是大量服务器、集群和 AI 工具的默认运行环境。课程中的 notebook 可以在本地运行，但当你需要 GPU、大内存或长时间任务时，通常会进入 Linux 服务器或集群。

命令行看起来不如图形界面直观，但它有三个优势：

- 可复现：命令可以保存到脚本中重复运行。
- 高效：批量处理文件和日志比手工点击更快。
- 远程友好：服务器不一定有图形界面，但一定有终端。

预测项目里常见错误往往不是模型公式错，而是当前目录不对、数据文件不在预期位置、Python 环境不一致，或输出被写到了另一个文件夹。Linux 基础的主线就是

路径、文件和环境；可复现的时间序列分析同样依赖这种清楚的数据、代码和输出组织 [Hyndman and Athanasopoulos, 2021]。

我们用计算机历史解释为什么 AI 时代又需要命令行。早期计算机是大型机，后来个人电脑和图形界面让很多操作变成鼠标点击；但当数据、模型和算力重新回到集群和服务器上时，批量操作又变得重要。你不可能用鼠标点击一万次来处理一万个文件，也不可能靠本地笔记本完成所有深度学习任务。终端是你和服务器、集群、脚本和 AI 工具对话的接口。

个人 AI 设备也说明了这一点。设备连上显示器后看起来像一台普通 Linux 电脑，但真正运行模型、安装包、查看 GPU 状态、复制课程命令和远程访问，仍然主要通过 Terminal 完成。未来无论使用学校集群、云服务器、实验室工作站还是本地 AI 盒子，Linux 命令行都会是最稳定的操作入口。

C.3　如何获得 Linux 环境

Windows 10+ 用户可以使用 WSL（Windows Subsystem for Linux）安装 Ubuntu 或 Debian，也可以使用 Windows Terminal 登录远程服务器。

macOS 用户的 Terminal 与 Linux 命令非常接近，可以直接练习大部分命令。

课程中需要较大计算资源时，可以使用 PKU 集群或其他 Linux 服务器。

AI 预测任务可能从本地 notebook 开始，但训练深度模型、调用批量任务或处理大量文件时，命令行会变成主要工作界面。会用 `pwd`、`ls`、`cd`、`python -m pip` 和日志重定向，能显著减少环境排查时间。

我们把命令行称为你和服务器、AI 工具对话的界面。Windows 用户如果没有 Terminal，可以打开 Microsoft Store，搜索 `Terminal`，安装 Windows Terminal。安装后先输入：

```
ssh
```

如果终端打印出用法说明，说明 SSH 命令可用；如果提示找不到命令，先解决终端工具问题，再尝试登录集群。

C.4　登录服务器

SSH 命令的一般形式是：

```
ssh 用户名 @ 服务器地址
```

例如：

```
ssh 2512345678@wm2 -login.pku.edu.cn
```

其中 2512345678 是你的远程用户名，后面是服务器地址。输入密码时终端通常不会显示字符，这是正常的。集群环境还可能要求 OTP 动态验证码。

C.5　查看当前位置

登录后先确认自己在哪里：

```
whoami
pwd
ls
ls -htla
```

含义分别是：

- `whoami`：显示当前用户名。
- `pwd`：显示当前目录路径。
- `ls`：列出当前目录文件。
- `ls -htla`：按时间显示更详细文件信息，包括隐藏文件。

`$HOME` 是你的用户主目录：

```
echo $HOME
```

C.6　目录切换

使用 `cd` 切换目录：

```
cd project
cd ..
cd
```

`cd ..` 回到上一级目录，单独输入 `cd` 回到主目录。路径理解是 Linux 文件系统的核心基础。运行 notebook、安装包和读取数据时，很多错误都来自当前目录不对。

C.7　文件操作

创建空文件：

```
touch hello.txt
```

创建目录：

```
mkdir hello_linux
```

移动或重命名：

```
mv hello.txt notes.txt
```

删除文件：

```
rm notes.txt
```

删除目录要格外小心：

```
rm -r hello_linux
```

不要随便使用 `rm -rf`，尤其不要在不确认路径时删除目录。课程作业中通常不需要强制删除。

C.8　查看文件

常见查看命令：

```
cat file.txt
less file.txt
head file.txt
tail file.txt
wc file.txt
```

大文件不要直接用 `cat` 全部打印。先用 `head` 或 `tail` 查看开头和结尾；如果想像滚动页面一样查看，可以用 `less file.txt`，进入后用方向键或翻页键移动，按 `q` 退出。

`ls -htla` 是一个常用组合。`h` 让文件大小更容易读，`t` 按时间排序，`l` 使用长列表格式，`a` 显示隐藏文件。长列表最左侧会显示权限：第一个字符 `d` 表示目录；`r`、`w`、`x` 分别表示可读、可写、可执行。权限通常按“所有者、同组用户、其他用户”三组展示。

C.9　编辑文件

服务器上常见编辑器包括：

- `nano`：简单，适合初学者。
- `vim`：强大，但需要学习。
- `emacs`：功能完整，学习曲线较长。

初学者可以先用：

```
nano notes.txt
```

保存和退出按屏幕底部提示操作。`nano` 底部的 `^X` 表示按 `Ctrl+X`；退出时如果询问是否保存，输入 `Y` 表示保存，确认文件名后回车即可。它可以理解为服务器上的极简记事本，适合改小脚本、配置和笔记。

C.10　运行 Python

在终端中查看 Python 版本：

```
python - -version
python -c "import sys; print(sys.executable)"
```

在本地、Anaconda、Jupyter 和远程服务器之间切换时，最常见的问题是“以为自己在一个 Python 里安装包，实际 notebook 用的是另一个 Python”。所以每次排查环境问题时，都先运行上面的 `sys.executable` 检查路径，再用同一个解释器执行安装命令：

```
python -m pip install nixtla
```

运行脚本：

```
python script.py
```

注意区分系统终端和 Python 解释器。普通终端提示符常见形式是 `$` 或包含用户名、主机名的字符串；进入 Python 后通常会看到 `>>>`。如果你在 `>>>` 后输入 `ls`，Python 会把它当成 Python 代码而不是 Linux 命令。退出 Python 可以输入 `exit()`，回到终端后再运行文件操作命令。

如果 `.py` 文件里只写了一个表达式，运行脚本时不一定会显示结果。想看到输出，应使用 `print()`：

```
print(3 + 4)
```

安装包建议使用：

```
python -m pip install pandas
```

这样可以减少 `pip` 和 `python` 不属于同一环境的问题。

查看 GPU 或本地 AI 设备状态时，不同平台命令可能不同。英伟达环境常见命令是 `nvidia-smi`，新型本地设备还可能提供自己的 dashboard 或系统监控工具。不要只看“有 GPU”三个字；还要看当前进程、温度、功耗、可用内存和是否已有别人在运行任务。运行深度学习 notebook 前，先确认资源状态，可以减少中途失败和互相抢资源。

C.11　标准输入、输出和错误

Linux 中很多程序把结果写到标准输出 `stdout`，把错误或状态信息写到标准错误 `stderr`。默认情况下，它们都会显示在屏幕上。

可以把输出保存到文件：

```
ls -l /usr/bin > ls -output.txt
```

追加输出：

```
ls -l /usr/bin >> ls -output.txt
```

保存错误信息：

```
ls -l /bin/usr 2> ls -error.txt
```

这些机制对保存实验日志很有用。

C.12　管道

管道把一个命令的输出交给下一个命令：

```
ls -l /usr/bin | wc
ls /bin /usr/bin | sort | head
```

常见管道工具包括：

- `sort`：排序。
- `uniq`：去重或统计重复。
- `grep`：按模式筛选文本。
- `wc`：统计行数、词数和字节数。
- `head` / `tail`：查看开头或结尾。
- `tee`：同时显示并写入文件。

预测项目中，管道常用于查看日志、统计文件数量、筛选错误信息。

例如用 `wc` 统计输出行数，用 `grep` 筛选错误，用 `tail` 查看最新日志，用 `>` 或 `>>` 保存结果。这些命令帮助你把一次手工运行变成可复查的实验记录。

C.13　上传和下载

图形化工具可以使用 FileZilla。熟悉命令行后，可以使用：

```
scp local.csv 用户名 @ 服务器地址:~/project/
rsync -av data/ 用户名 @ 服务器地址:~/project/data/
```

传输前确认文件不含密码、API key 或敏感数据。

C.14　获取帮助

查看命令手册：

```
man mkdir
mkdir - -help
```

如果不知道命令是否会修改文件，先读帮助或在测试目录中练习。

C.15　小结

Linux 基础的目标不是背命令，而是理解文件、路径、环境和输入输出。预测项目中，绝大多数服务器问题都可以归结为：你在哪个目录，用哪个 Python，读哪个文件，把输出写到了哪里。

C.16　练习

1. 登录服务器后运行 `whoami`、`pwd`、`ls -htla`。
2. 创建目录 `forecasting_practice`，进入目录并创建 `hello.py`。
3. 在 `hello.py` 中写一行 `print("hello forecasting")`，并用 Python 运行。
4. 把 `ls -htla` 的输出保存到 `files.txt`。
5. 使用管道统计当前目录中文件列表的行数。

参考文献

J. Alammar and M. Grootendorst. *Hands-On Large Language Models: Language Understanding and Generation*. O'Reilly Media, 2024.

A. F. Ansari, L. Stella, C. Turkmen, X. Zhang, P. Mercado, H. Shen, O. Shchur, S. S. Rangapuram, S. P. Arango, S. Kapoor, J. Zschiegner, D. C. Maddix, H. Wang, M. W. Mahoney, K. Torkkola, A. G. Wilson, M. Bohlke-Schneider, and Y. Wang. Chronos: Learning the language of time series, 2024. URL http://arxiv.org/abs/2403.07815.

A. F. Ansari, O. Shchur, J. Küken, A. Auer, B. Han, P. Mercado, S. S. Rangapuram, H. Shen, L. Stella, X. Zhang, M. Goswami, S. Kapoor, D. C. Maddix, P. Guerron, T. Hu, J. Yin, N. Erickson, P. M. Desai, H. Wang, H. Rangwala, G. Karypis, Y. Wang, and M. Bohlke-Schneider. Chronos-2: From univariate to universal forecasting, 2025. URL http://arxiv.org/abs/2510.15821.

J. D. Croston. Forecasting and stock control for intermittent demands. *Journal of the Operational Research Society*, 23(3):289–303, 1972. doi: 10.1057/jors.1972.50.

F. X. Diebold and R. S. Mariano. Comparing predictive accuracy. *Journal of Business & Economic Statistics*, 13(3):253–263, 1995. doi: 10.1198/073500102753410444. URL https://doi.org/10.1198/073500102753410444.

W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):120:5232–120:5270, 2022.

R. Fildes, P. Goodwin, M. Lawrence, and K. Nikolopoulos. Effective forecasting and judgmental adjustments: An empirical evaluation and strategies for improvement in supply-chain planning. *International Journal of Forecasting*, 25(1):3–23, 2009. doi: 10.1016/j.ijforecast.2008.11.010. URL https://www.sciencedirect.com/science/article/pii/S0169207008001362.

A. Garza, C. Challu, and M. Mergenthaler-Canseco. Timegpt-1, 2024. URL http://arxiv.org/abs/2310.03589.

T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007. doi: 10.1198/016214506000001437. URL https://doi.org/10.1198/016214506000001437.

S. Gu, B. Kelly, and D. Xiu. Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273, 2020. doi: 10.1093/rfs/hhaa009. URL https://doi.org/10.1093/rfs/hhaa009.

A. H. Huang, H. Wang, and Y. Yang. Finbert: A large language model for extracting information from financial text. *Contemporary Accounting Research*, 40(2):806–841, 2023. doi: 10.1111/1911-3846.12832. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/1911-3846.12832.

R. J. Hyndman and G. Athanasopoulos. 预测：方法与实践. OTexts, 2021. URL https://otexts.com/fpp3cn/.

R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006. doi: 10.1016/j.ijforecast.2006.03.001.

M. Jin, S. Wang, L. Ma, Z. Chu, J. Y. Zhang, X. Shi, P.-Y. Chen, Y. Liang, Y.-F. Li, S. Pan, and Q. Wen. Time-llm: Time series forecasting by reprogramming large language models. 2023. URL https://openreview.net/forum?id=Unb5CVPtae.

F. Li and T. Ruan. Recovery-informed forecasting strategy enhancement. *Annals of Tourism Research*, 118:104164, 2026. doi: 10.1016/j.annals.2026.104164. URL https://arxiv.org/abs/2603.01085.

L. Li, Y. Kang, F. Petropoulos, and F. Li. Feature-based intermittent demand forecast combinations: Accuracy and inventory implications. *International Journal of Production Research*, 61(22):7557–7572, 2023. doi: 10.1080/00207543.2022.2153941. URL https://arxiv.org/abs/2204.08283.

X. Li, Y. Kang, and F. Li. Forecasting with time series imaging. *Expert Systems with Applications*, 160:113680, 2020. doi: 10.1016/j.eswa.2020.113680. URL https://arxiv.org/abs/1904.08064.

Y. Liu, N. Bu, Z. Li, Y. Zhang, and Z. Zhao. At-fingpt: Financial risk prediction via an audio-text large language model. *Finance Research Letters*, 77:106967, 2025. doi: 10.1016/j.frl.2025.106967. URL https://www.sciencedirect.com/science/article/pii/S1544612325002314.

S. Makridakis, F. Petropoulos, and Y. Kang. Large language models: Their success and impact. *Forecasting*, 5(3):536–549, 2023. doi: 10.3390/forecast5030030. URL https://www.mdpi.com/2571-9394/5/3/30.

Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers, 2023. URL http://arxiv.org/abs/2211.14730.

F. Petropoulos, D. Apiletti, V. Assimakopoulos, M. Z. Babai, D. K. Barrow, S. Ben Taieb, C. Bergmeir, R. J. Bessa, J. Bijak, J. E. Boylan, J. Browell, C. Carnevale, J. L. Castle, P. Cirillo, M. P. Clements, C. Cordeiro, F. L. Cyrino Oliveira, S. De Baets, A. Dokumentov, J. Ellison,

P. Fiszeder, P. H. Franses, D. T. Frazier, M. Gilliland, M. S. Gönül, P. Goodwin, L. Grossi, Y. Grushka-Cockayne, M. Guidolin, M. Guidolin, U. Gunter, X. Guo, R. Guseo, N. Harvey, D. F. Hendry, R. Hollyman, T. Januschowski, J. Jeon, V. R. R. Jose, Y. Kang, A. B. Koehler, S. Kolassa, N. Kourentzes, S. Leva, F. Li, K. Litsiou, S. Makridakis, G. M. Martin, A. B. Martinez, S. Meeran, T. Modis, K. Nikolopoulos, D. Önkal, A. Paccagnini, A. Panagiotelis, I. Panapakidis, J. M. Pavía, M. Pedio, D. J. Pedregal, P. Pinson, P. Ramos, D. E. Rapach, J. J. Reade, B. Rostami-Tabar, M. Rubaszek, G. Sermpinis, H. L. Shang, E. Spiliotis, A. A. Syntetos, P. D. Talagala, T. S. Talagala, L. Tashman, D. Thomakos, T. Thorarinsdottir, E. Todini, J. R. Trapero Arenas, X. Wang, R. L. Winkler, A. Yusupova, and F. Ziel. Forecasting: Theory and practice. *International Journal of Forecasting*, 38(3):705–871, 2022. doi: 10.1016/j.ijforecast.2021.11.001. URL https://arxiv.org/abs/2012.03854.

N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

X. Shi, S. Wang, Y. Nie, D. Li, Z. Ye, Q. Wen, and M. Jin. Time-moe: Billion-scale time series foundation models with mixture of experts, 2024. URL https://arxiv.org/abs/2409.16040v1.

R. J. Shiller. *Narrative Economics: How Stories Go Viral and Drive Major Economic Events*. Princeton University Press, 2019.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023. URL http://arxiv.org/abs/1706.03762.

S. Wang, H. Wu, X. Shi, T. Hu, H. Luo, L. Ma, J. Y. Zhang, and J. Zhou. Timemixer: Decomposable multiscale mixing for time series forecasting. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*. arXiv, 2024. doi: 10.48550/arXiv.2405.14616. URL http://arxiv.org/abs/2405.14616.

X. Wang, R. J. Hyndman, F. Li, and Y. Kang. Forecast combinations: An over 50-year review. *International Journal of Forecasting*, 39(4):1518–1547, 2023. doi: 10.1016/j.ijforecast.2022.11.005. URL https://arxiv.org/abs/2205.04216.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. doi: 10.1109/4235.585893.

G. Woo, C. Liu, A. Kumar, C. Xiong, S. Savarese, and D. Sahoo. Unified training of universal time series forecasting transformers, 2024. URL http://arxiv.org/abs/2402.02592.

B. Zhang, Y. Kang, A. Panagiotelis, and F. Li. Optimal reconciliation with immutable forecasts. *European Journal of Operational Research*, 308(2):650–660, 2023. doi: 10.1016/j.ejor.2022.11.035. URL http://arxiv.org/abs/2204.09231.

H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference*

on Artificial Intelligence, volume 35, pages 11106–11115, 2021. doi: 10.1609/aaai.v35i12.17325. URL https://ojs.aaai.org/index.php/AAAI/article/view/17325.