第 1 章 从菜鸟到高手的路径是什么

本章主要探讨如何帮助职场新人从迷茫中找到正确的学习方法，制订有效的成长计划，避免在职场中重复无效的努力，尽早走上正确且高效的成长之路。

作为曾经从非相关专业误打误撞进入数据分析师行业的人，笔者深知从菜鸟到专家的过程并非易事。经过数十年的努力与积累，虽然不能自称为高手，但对自己走过的弯路有了更深刻的理解。因此，借此机会回顾过去的经历，梳理出一条从菜鸟到高手的快速成长之路，帮助更多热爱数据分析的读者，使他们尽量少走弯路，这也是笔者最大的愿望。

许多初入职场的读者常常感到迷茫，不知如何开始制订清晰的学习与成长规划，更不清楚如何进入正确的成长轨道。实际上，成长路径可以分为以下4个层次：

（1）硬技能成长：学习数据分析的基础技能。
（2）软技能成长：培养数据分析的思维能力。
（3）通用链路技能：掌握Python数据分析技能学习的通用流程。
（4）职业心态建设：在不同的职业阶段保持积极的职业心态。

接下来，将详细介绍这4个成长路径的具体内容。

1.1 数据分析基础技能学习

许多热爱数据分析的人，一提到编程，往往会望而却步，尤其是许多非相关专业人士更是把编程能力视为一道高不可攀的入行门槛。实际上，基础编程技能在数据分析工作中确实至关重要，但并非像许多人想象的那样遥不可及。即便不懂编程，也并不意味着无法从事数据分析相关的工作。例如，许多人都能够熟练使用Excel软件，这就是进行数据分析的基础。如果

再学习一些基础的SQL（Structured Query Language，结构化查询语言）和Python技能，那么就更有机会进入这个行业。下面将简要介绍数据分析的基础技能——Excel、SQL和Python，帮助读者在学习过程中更有针对性。

1.1.1　Excel 能力

在谈论编程能力之前，我们先来看看Excel，相信很多人都很熟悉这个软件。在学习数据分析相关编程技能之前，熟练掌握Excel是必不可少的。不同岗位对Excel的技能要求差异较大，例如，财务人员在高效工作时对Excel的要求就比较高。

1. 要学习什么

根据不同的需求，Excel学习的内容也会有所不同。从入门数据分析的角度，有以下几个关键点：

- Excel基础使用能力：掌握添加和修改Excel文本内容、调整表格大小、修改字体等基本功能。
- Excel数据透视功能：多维度分析是常见的分析场景，因此掌握数据透视功能至关重要，包括不同维度的统计值和求和等（建议多加练习以便熟练掌握）。
- Excel图表生成功能：将透视后的数据表格以图表形式展示是汇报分析结果的重要手段，因此学会如何将数据转换为图表也十分重要。

2. 怎么学

关于学习方式，许多人会犹豫是否需要去培训机构。实际上，刚入门时可以优先利用一些免费的资源来了解数据分析的技能，并评估自己的学习能力，再决定是否参加培训。以下是一些学习建议：

- 基础技能图书：购买一本关于Excel技能的图书进行自学，通常能在短时间内获得显著提升。
- 网络免费教程：通过搜索引擎查找Excel基础教程，网络上拥有大量的免费资源，只需认真跟随学习即可。
- 实践练习：在网上寻找实际案例进行练习，并进行整理和分类，积累到自己的文档中，以备后续复习或直接使用。

3. 学到什么程度

明确学习内容和方式后，了解需要掌握的深度尤为重要。并非所有技能都需要掌握得非常熟练，这通常取决于实际工作的需求，因为每个人的工作需求不同，表1-1所示是根据常见需求和技能难易度的学习深度建议。

表 1-1　学习深度建议

工作使用频繁度	技能难易程度	学习深度建议
频繁使用	难	理解透彻
频繁使用	中	理解透彻
频繁使用	易	理解透彻
一般使用	难	记录到笔记中
一般使用	中	记录到笔记中
一般使用	易	尽量理解透彻
较少使用	难	记录到笔记中
较少使用	中	记录到笔记中
较少使用	易	记录到笔记中

从表中可以看出，对于频繁使用的技能，必须理解透彻，才能在实际工作中灵活运用，提高工作效率。如果频繁使用的技能难度较大，可以将其记录到笔记中，逐步熟悉和掌握。

总之，以上内容介绍了Excel在数据分析岗位入门中需要学习的内容、学习方式及学习深度。建议读者不要急于学习所有Excel功能，而是根据实际工作需求逐步掌握所需技能，并详细记录和总结，从而实现低成本、高效率的学习。

1.1.2　SQL 编程能力

随着互联网的不断发展，数据量也随之增长，Excel已无法满足数据分析的需求。许多使用过Excel的人都知道，当数据量超过10万条时，进行简单的统计分析就会变得非常缓慢。因此，将数据存储到数据仓库，并通过SQL进行数据提取和分析，已成为当前的首选方案。

1. 要学习什么

不同岗位对SQL的学习要求有所不同。例如，数据仓库工程师需要掌握的SQL技能远多于数据分析师。因此，数据分析师不必过于焦虑，只需要明确自己需要掌握的技能，便能轻松应对。

- SQL数据提取能力：对于数据分析师而言，数据提取是最基础的能力，必须掌握。
- SQL数据仓库管理能力：此能力取决于岗位需求。在许多大型公司中，数据仓库的管理通常由专门的团队负责。

2. 怎么学

由于SQL是结构化查询语言，很多人会发现它相对容易学习。以下是一些学习建议：

- 图书学习：阅读基础的SQL图书，了解需要掌握的技能和数据提取的基本逻辑。
- 网络免费教程：充分利用网络上的免费资源，通过实践逐步理解并掌握SQL。

- 实践练习：建议下载并安装基础的数据仓库，亲自动手编辑SQL进行实操练习，从而提高学习效率。

3. 学到什么程度

SQL是用于管理关系数据库的标准语言，主要用于数据查询和程序设计。数据分析师的重点在于查询和分析，通过不同维度的数据进行统计分析。如果读者希望在数据分析行业长期发展，建议掌握以下学习重点并明确学习目标：

- 熟练掌握查询技能：必须熟练掌握单表和多表的关联查询、过滤等基本技能。
- 一般掌握数据仓库管理技能：了解数据仓库管理的基本知识，以便自主学习相关数据表的增、删、改、查技能。
- 提高数据查询效率等技能：可以根据实际情况学习提升数据查询效率的方法。如果公司有专门的团队支持，可以专注于数据分析中的数据提取工作。

以上是关于SQL在数据分析岗位入门中需要学习的内容、学习方法及学习程度的介绍。建议读者首先熟练掌握查询技能，以满足日常工作需求，然后逐步深入学习其他相关内容。

1.1.3　Python 编程能力

Excel主要用于快速统计和分析，SQL适合处理大规模数据查询与统计分析，而Python的应用范围更为广泛。那么，学习Python的意义是什么呢？在深入探讨之前，我们先对Python进行初步了解，明确学习的内容、方法以及学习的深度。

Python作为一种高级编程语言，广泛应用于软件开发和数据科学领域，并具有多样的优势。在TIOBE 2024年3月的编程语言排行榜中，Python凭借其强大、易学好用的特点位居第一，超越了C语言，如图1-1所示。

Programming Language	Ratings	Change
Python	15.63%	+0.80%
C	11.17%	-3.56%
C++	10.70%	-2.59%
Java	8.95%	-4.61%
C#	7.54%	+0.37%
JavaScript	3.38%	+1.21%
SQL	1.92%	-0.04%
Go	1.56%	+0.32%

图 1-1　2024 年 3 月编程语言排行榜

Python的主要应用领域包括：

- Web开发：Python在构建Web应用程序方面表现优异，常用框架如Django和Flask。
- 数据科学与人工智能：Python在数据分析、可视化、机器学习和人工智能领域的应用广泛，Pandas、NumPy、Scikit-learn和Matplotlib等库为这些领域提供了强大支持。
- 科学计算：Python在科学计算和工程领域也有广泛应用，SciPy和SymPy库提供了丰富的科学计算功能。
- 自然语言处理：Python在文本数据处理和自然语言处理方面表现出色，NLTK和spaCy等库为开发者提供了强大的工具和算法。

Python的优势包括：

- 简单易学：Python的语法简洁明了，类似自然语言，易于学习，特别适合初学者和非计算机专业人士。
- 多样的应用领域：Python可广泛应用于Web开发、数据科学、人工智能、机器学习、科学计算等多个领域，其灵活性和通用性使其成为全能的编程语言。
- 强大的生态系统：Python拥有庞大且活跃的社区，丰富的第三方库和工具（如NumPy、Pandas、TensorFlow、PyTorch等）大大简化了开发流程，提高了效率。
- 跨平台性：Python是跨平台的，可以在Windows、Linux、macOS等多种操作系统上运行，便于开发者在不同环境中部署应用程序。
- 快速开发：Python支持快速开发和迭代，动态类型和自动内存管理的特性使得原型构建和迭代开发迅速高效。
- 丰富的社区支持和文档：Python拥有庞大的开发者社区，丰富的文档、教程和问答网站（如Python官方文档、Stack Overflow等），方便开发者获取所需帮助和资源。
- 广泛的工具支持：Python支持多种集成开发环境（IDE），如PyCharm、Jupyter Notebook，还支持多种文本编辑器，如Sublime Text、VS Code，开发者可根据个人喜好选择合适的工具。

选择Python进行数据分析的原因如下：

- 简单易用：与R语言或其他数据分析语言相比，Python更加简洁易懂。
- 高效多功能：Python不仅可以快速进行统计分析、数据可视化，还能实现自动化预警和生成分析报告，广泛应用于数据挖掘和机器学习领域。
- 丰富的生态支持：众多第三方库使得Python在数据分析方面既高效又便捷。

通过以上介绍，我们对Python的应用和优势有了初步了解，接下来将具体探讨入门Python时应学习的内容、方法及深度。

1. 要学什么

在数据分析领域，学习Python时可以重点关注以下几个方面：

- 基础语法：掌握基本语法以进行数据读取和统计分析。
- 数据可视化：学会使用Python生成可视化图表，方便制作数据分析报告。
- 自动化处理：利用Python自动化生成分析报告（结合实际工作需求进行学习）。
- 数据挖掘：掌握特征挖掘和模型建立的技能（结合实际工作需求进行学习）。

2. 怎么学

针对Python的学习，不同难度的内容可以采用不同的学习方法：

- 网络学习：通过图书学习基础语法仍然是最常见的选择。
- 网络免费教程：利用免费的网络在线教程也能有效掌握基础语法、可视化和自动化等内容，满足入门需求。
- 实践练习：对于数据挖掘等进阶内容，建议结合自己的能力，通过自学和网络平台竞赛实践相结合或通过培训机构和实践相结合的方式进行学习，成长会更快。

3. 学到什么程度

由于Python的应用范围非常广泛，学习的深度应根据实际工作需求进行调整。以下是针对不同岗位的学习层次进行区分：

- 业务数据分析师：需要掌握Python的基本语法，能够进行数据分析、可视化和自动化处理。
- 数据挖掘工程师：需要深入学习Python在机器学习算法和特征挖掘方面的应用。
- 数据产品经理：需要具备Python基础知识，了解爬虫技术、机器学习和特征挖掘，能够独立探索和分析产品。

总的来说，Excel、SQL和Python作为数据分析的核心技能，都需要熟练掌握基础知识。虽然学习门槛并不高，但在此基础上，结合自身实际工作场景进行针对性学习，将更有效地提升工作效率和技能水平。

1.2　数据分析思维能力培养

在成长为一名优秀数据分析师的过程中，除了掌握各种技巧和编程能力外，数据分析思维的软实力才是核心竞争力。因此，有意识地培养自己的分析思维至关重要。

那么，如何培养数据分析思维呢？许多分析师在开始数据分析时通常遵循以下思路：

（1）观察数据趋势的变化。

（2）分析用户的基本维度变化，如年龄、性别等。

（3）深入分析交叉维度，找出问题所在。

这三步是许多数据分析师的入门方法，如同“三板斧”一般，如图1-2所示，这种分析方法对于一般问题有效。然而，面对不同的业务场景时，这种思路可能并不适用。如果缺乏其他分析思路，重新寻找切入点可能让人感到无从下手。因此，培养数据分析能力，尤其是分析思维能力，变得尤为重要。

图 1-2　数据分析师入门“三板斧”

总体而言，数据分析思路对数据分析师来说，就像探宝专家手中的地图。只有拥有清晰完整的地图，才能快速准确地找到目标，进而获得有价值的洞见。否则，分析得出的结论可能既不准确也不实用，对公司而言毫无价值。因此，掌握正确的分析思路是每位希望成为优秀分析师的必备技能。

当我们面对老板布置的业务分析任务时，如何找到正确的分析思路呢？根据笔者多年的经验，正确的分析思路源于对业务的深刻理解，同时需要从更高的视角（包括业务、公司和行业）出发，通过严谨的逻辑和量化分析的方法来解决问题。

下面是笔者结合多年工作经验，提出的由底层到上层、由浅入深的4个层面分析思路，旨在帮助读者提升数据分析能力。

1.2.1　需求层面：角色转换

每个公司的数据分析师所需要分析的内容来自不同的业务部门，涉及销售、运营、财务、产品、技术等多个领域。作为分析师，我们应主动进行角色转换（见图1-3），站在各部门的视角，识别数据分析的关键点，帮助他们发现产品、运营流程和销售业绩中的问题。同时，提出切实可行的建设性建议，以提升工作效率和营收，而非被动地等待需求，成为单纯的数据提

取工具。如果最终沦为工具，那将是每位数据分析师最为悲惨的境遇。

图 1-3　数据分析师角色转换

笔者曾任职于一家电商公司，担任数据分析师。在该公司，运营团队会定期开展活动，并在节假日进行短信营销。然而，由于缺乏商业智能（Business Intelligence，BI）工具的支持，每次营销只能采用全量推送的方式。这种粗放式的操作不仅耗费大量成本，而且对提升营销效果的帮助有限。

从成本的角度分析，我们发现每次短信营销活动的成本高达10万元，一年中类似的活动超过十次，导致全年短信营销费用超过百万元。然而，通过数据分析，我们注意到转化率主要集中在一线和三线城市的年轻人群体。这表明，全量推送存在严重的资源浪费。

为了优化这一问题，我们进行了A/B测试，尝试精准筛选目标用户群体并定向投放。结果表明，仅投入1万元进行精准投放，其转化率与全量投放的效果几乎相当。最终，我们为运营部门节约了近100万元的预算。

现在看来，这是一种简单而高效的精细化运营策略。但在互联网行业刚起步的阶段，这种意识并非人人都具备。许多时候，问题的复杂性并非核心障碍，关键在于思维的灵活性和对数据价值的敏锐洞察。通过数据驱动决策，不仅能显著降低成本，还能大幅提升运营效率，为业务带来更大价值。

只有通过角色转换，深入理解不同角色的日常工作，我们才能发现有价值的分析主题，找到有效的分析思路，解决真正有意义的问题。如果被动等待问题找上门，往往会导致临时思考分析思路时手足无措。因此，主动出击，提前发现问题并制定解决方案，才是更为高效的关键策略。

1.2.2　业务层面：核心指标

除了主动进行角色转换以探索需求外，还需要深入了解各业务部门的核心指标。如图1-4所示，每个业务部门通常都有其特定的KPI，例如，销售部门关注营收指标，运营部门关注效率指标，产品部门则关注访问量或产品异常率等指标。当公司本月的营收总额下降时，常见的分析思路是顺着营收来源逐步拆解相关指标。首先明确营收的主要渠道，然后逐一排查这些渠道在哪个时间段出现了问题。这种“顺藤摸瓜”的分析方法直观且简单，但关键在于聚焦核心指标并进行深入分析。

图 1-4　关注核心 KPI

需要注意的是，在发现问题后必须进一步提供解决方案和建议，而不是简单地将问题抛给业务部门去解决。这种做法是不负责任的行为，每一位数据分析师都应具备闭环思维。虽然这个理念被广泛提及，但真正落实并不容易。例如，当我们发现某个广告渠道的投放效果不佳时，可以对比其他广告渠道的用户画像，识别出最佳的广告投放转化人群，同时提出相应的算法建模方案。只有在方案落地并确认效果达到预期后，才能算作一个完整的分析闭环。

1.2.3　战略层面：明确方向

在担任数据分析师的几年里，笔者逐渐感到自己的分析思路遇到了瓶颈。除了日常的需求分析和指标分析外，发现自己无法为公司提供更大的价值。经过反思后，才意识到问题出在对自己设限制。曾经，笔者从未主动去了解公司的战略层面，总认为自己只是处于底层的业务分析师，没有必要去了解公司的战略部署。这种错误的想法使笔者错失了许多重要的提升机会。

正如俗话所说：“不想当将军的士兵不是好士兵。”在数据分析领域也一样，可以认为，“不想了解公司战略的分析师，不是一个称职的分析师”。

我们不必制定惊天动地的战略规划，但有必要在充分了解公司战略后，通过数据驱动的方式更好地支持战略的执行，如图1-5所示。实际上，每项业务都是公司整体战略的重要支撑，因此我们的工作与战略发展息息相关。例如，当公司处于高速发展时期，其整体战略必然是快

速扩张。在这种情况下，各业务目标也应与之相匹配，可能需要增加预算以支持发展。然而，如果我们只关注预算分析，研究如何节约成本，这样的思路显然与公司战略背道而驰。即使我们的分析报告再好再准确，对公司而言也可能意义不大。

图 1-5　明确企业战略方向

作为分析师，我们的职级和能力可能不足以直接影响和改变公司战略，但我们可以尽力为战略的实施提供支持。例如，在公司扩张时强烈要求节约成本，或在稳定期盲目建议扩张并不是最佳选择，反而可能会为公司带来风险。

因此，若希望提升分析的价值，不妨深入了解公司战略，明确分析思路和方向。

1.2.4　行业层面：洞察影响

在深入了解公司的战略后，需要进一步了解公司所处行业的发展动态，这样才能进行更有价值的分析，并为公司创造更大利润，如图1-6所示，即使公司不是行业巨头，通常也会受到行业变化的影响，但在这种情况下，掌握行业的变化趋势仍能帮助我们提前做好分析准备，为公司的战略决策提供有价值的支持。哪怕我们的分析不能够直接影响战略，但仍然能为公司战略部署提供重要参考。

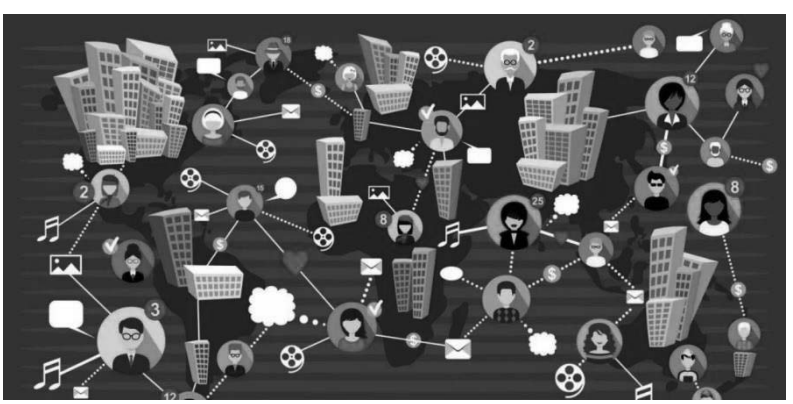

图 1-6　洞察行业影响

例如，在第一代智能手机出现时，可以分析随着销量变化，市场份额如何变迁。同时，通过用户的购买数据分析，可以洞察手机用户的心智变化，从而预测未来智能手机的受欢迎趋势。此外，智能手机的发展将对公司销量和利润产生怎样的影响，也可以提出相关建议，提出可能的尝试和战略调整。想象一下，如果将这样的市场分析报告和可行性解决方案呈现给老板时，将会产生怎样的积极效果。

因此，要打破分析瓶颈，就需要密切关注行业动态。可以通过各类行业报告和专业网站获取信息，这将有助于我们积累更多的分析能力和思路。

总而言之，数据分析能力并不是凭空而来的，而是通过学习、探索和沉淀逐渐获得的。要提升分析能力，就必须深入了解不同的角色、业务、战略和行业，从而不断拓宽全局视角，发现更多有价值的分析思路。正是由于分析师在数据敏感度、经验和行业认知上的差异，才导致薪资待遇的跨度也很大。

在数据分析师行业，既要懂得分析、了解行业，又要具备一定的商业头脑，只有这样才能在未来创造更大的可能性。

1.3　Python 数据分析通用链路技能

在掌握了硬技能学习和软实力培养之后，接下来需要深入了解实际工作中涉及的具体任务。虽然不同岗位的职责有所不同，但对于数据分析而言，始终离不开从数据收集、分析到输出的完整链路。随着时代的发展，使用Python进行数据分析的频率和深度只会越来越高。因此，前期了解Python在数据分析全链路中的具体应用显得尤为重要。

当然，掌握Python技能的程度应根据实际工作需求来界定。不同工作场景所需的技能环节各不相同，例如：

- 数据分析：频繁使用Python进行统计分析。
- 数据挖掘：经常使用Python进行特征指标挖掘。
- 数据建模：使用Python进行数据建模。
- 机器学习：运用Python进行机器学习。

因此，我们需要系统地整理自己的学习链路。通过拆解不同的链路，可以更有针对性地学习Python。

通用的全链路数据分析过程中通常涉及的Python学习内容，可以分为6个主要部分：数据收集、数据预处理、数据分析、数据挖掘、数据可视化和数据分析报告。

1.3.1　数据收集

许多刚入职场的新人主要收集公司各业务部门的数据，而很少接触公开合规的数据。实

际上，这些公开的数据也是数据分析的重要原材料，因此在获取这些数据时，需要学习Python的数据爬取技能。

对于喜欢自学的读者，可以尝试爬取一些公开的经济数据、商品评论等进行分析和挖掘，这将有助于提升Python的应用能力。因此，数据收集作为数据分析的第一步，至关重要。具体内容将在后续章节中详细介绍。

1.3.2　数据预处理

在获取数据后，我们所得到的通常是未经加工处理的原始数据。首先需要做的就是对这些数据进行各种加工和处理，将其转换为可用于分析的数据，这个过程称为数据预处理。

数据预处理的常见操作包括：

- 处理重复值
- 处理缺失值
- 处理空格
- 处理异常值

……

任何可能影响分析结果的未经处理的数据都需要进行预处理。在实际操作中，可能会遇到多种特征处理的情况，届时可根据需求灵活处理。

1.3.3　数据分析

在数据收集和预处理完成后，接下来就是数据分析。分析的第一步通常是现状分析，用于了解数据的基本情况，主要包括：

- 数据量：数据的总数量。
- 数据维度：数据包含的维度数量。
- 有效数据：可用于分析的有效数据量。
- 数据范围：数据覆盖的时间周期。

……

这些基本信息都可以通过Python快速分析完成。

对数据基本情况的了解是现状分析的关键。之后可以进行不同维度的分析或交叉维度分析，主要包括：

- 单维度趋势分析
- 多维度交叉分析

不同的应用场景会采用多种分析方法。

1.3.4　数据挖掘

数据挖掘在金融风控行业广泛应用，特征挖掘可以通过Python实现。同时，利用Python调用各种算法库进行特征建模也非常便捷。此外，通过Python还可以用来进行计算评估，最终将结果以可视化报告的形式呈现。

1.3.5　数据可视化

数据分析后的可视化是一个重要环节，它能帮助需求方更好地理解数据。可视化是最有效的表达方式之一。

Python的Matplotlib库能够轻松生成各种可视化报表和图形，例如：

- 柱状图
- 饼状图
- 折线图
- 排序图
- 堆砌图

……

不同类型的数据结果可以通过不同的可视化图形呈现，使信息更加直观易懂。

1.3.6　数据分析报告

提到数据分析报告，许多职场人士会选择使用PPT。虽然PPT能够以美观清晰的方式展现汇报信息，但它更侧重于结构化的视觉展示。相比之下，Word文档在细节和清晰度方面可能更具优势。

使用Python生成数据分析报告是一个高效的选择。Python的自动化功能可以显著提高工作效率，尤其是在日常的日报或周报中，通过编程实现一键生成报告，能够大幅节约时间。

如果需要定制化的数据分析报告，可以先使用Python生成可视化图表，再补充分析观点。这种方式同样高效，前提是需要熟练掌握相关技能。

以上是数据分析工作中，各岗位可能涉及的通用链路技能。不同环节需要掌握不同的技能，且根据岗位需求，学习的深度会有所不同。如果计划在数据分析行业长期发展，那么提前了解并掌握全链路的基本技能，将有助于更早地应用于实际工作中。

1.4　保持最佳的职业心态

在数据分析行业工作过程中，难免会遇到各种挑战。这些问题可能会直接影响我们对未

来发展的信心，甚至产生离开这个行业的念头。在这种情况下，如果缺乏良好的职业心态，即使具备扎实的编程能力和敏锐的分析思维，也可能难以坚持下去。

1.4.1　遇到问题

在工作中，我们可能遭遇如下问题：

（1）被视为单纯的数据提取工具，每天重复相似的工作，令数据分析的价值感变得微乎其微。

（2）工作仅五年便感觉遇到瓶颈，未来发展前景似乎黯淡无光。

（3）对职业寿命的焦虑，让你意识到与医生、律师等职业相比，自己在行业中的发展空间并不乐观。

以上问题是大部分数据分析师面临的三大挑战。除此之外，每个人还可能面临不同的困扰，此处不一一列举。

1.4.2　面对和理解问题

作为曾经亲历这些问题的人，我深知，如果不及时调整心态，往往会半途而废。基于个人经验，以下是对上述三个问题的分析，这些问题恰好反映了职业发展过程中不同阶段的心态挑战：

- 职业初期（1~3年）：进入数据分析行业之初，由于经验尚浅，往往难以主动发现和解决问题，导致频繁被动接受数据提取任务。时间一长，发现自己似乎只是在重复相同的工作，价值感逐渐减弱。
- 职业中期（3~6年）：在这段时间里，优秀的数据分析师能够迅速从初级晋升为高级分析师，但如果希望进一步提升，就会发现这并非易事。此时，需要拓宽知识面，学习更多技能才能实现突破。
- 职业后期（6年以上）：通常在8~10年间，数据分析师进入职业中后期。如果未能达到专家级别，未来的发展机会将大大缩减。即使达到了专家水平，面对未来的职业规划也可能感到迷茫。

在不同的职业阶段，遇到这些问题是非常普遍的现象，尤其是在互联网等行业，数据分析师面临的问题往往会更早显现，初期难以适应也是正常现象。

1.4.3　解决问题：保持最佳的职业心态

在过去十多年中，笔者经历了许多挑战，也得到了许多同事的帮助。在此，笔者希望通过梳理这些经历，帮助更多的人在数据分析行业中保持最佳的职业心态。有时感到郁闷、迷茫

或焦虑，往往不是因为能力不足，而是因为没有认真分析自己内心深处的问题。

1. 主动思考，打破工具人设

如果感觉自己被视为工具人，不妨反思一下，是他人将你视为工具人，还是自己将自己视作工具人？当我们接到数据需求时，是否主动思考以下几个问题：

- 数据需求合理性：这个数据需求是否合理且必要？如果不必要，可以选择不做，从而减少重复工作。
- 数据需求重复性：这个数据需求是否经常被重复提出？如果是，可以考虑通过固化和自动化来提升反馈效率。
- 数据需求拓展性：这个数据需求是否有潜力进一步拓展成更大、更有价值的需求？如果能够反馈给业务方，将有助于改变他人对你角色的看法。

需要记住，主动思考是打破工具人设的关键。

2. 挖掘能力洼地，突破瓶颈

作为一名数据分析师，虽然你可能已经熟悉编程，也能应对基本的业务分析，但这并不意味着已经到达了终点。每个人都有不完美之处，努力思考并找到自己的能力洼地，可以有效突破瓶颈。以下是一些寻找能力洼地的思考维度，可供读者参考：

- 角色转换：作为数据分析师，可以尝试转换为产品角色，从新的视角积累经验，找到能力突破的机会。
- 思维转换：如果我们的分析主要围绕商品销量，不妨学习一些财务知识，可以从财务的角度进行分析，寻找新的突破点。
- 技能转换：尽管我们可能已熟悉现有技能（如Python分析和可视化），但为了更好地提升数据分析能力，可以尝试学习网络爬虫技能，便于自行获取数据，助力进一步突破技能瓶颈。

切忌过早满足于现有状态，可以从不同角度寻找能力洼地并重点提升自己的数据分析能力。

3. 学会能力迁移，拓展行业边界

当你在数据分析领域达到一定高度后，可能会发现自己的价值或薪资难以提升。但这并不意味着前景有限。学会能力迁移将为你打开更多的行业机会。以下是一些能力迁移的建议：

- 行业复用：在不同的行业中，数据分析能力的许多技能（如Python数据分析和挖掘、分析思维等）都是可以复用的，无须担心技能会迅速被淘汰。
- 降维拓展：许多传统行业的数字化转型仍处于早期阶段，运用互联网数据分析或产品

分析的能力，可以在这些行业实现降维扩展，发挥更大潜力。

- 创新探索：随着自媒体和知识付费的流行，数据分析能力不仅能为内容创造带来优势，还能与多个行业融合，创造出新的自媒体玩法。

无论在哪个领域，许多技能和思维方式都是通用的。例如解决问题的能力、学习能力、沟通与协作能力，甚至在一些早期行业进行降维打击。因此，不要给自己设限，勇于拓展自己的行业边界。

1.5　本章小结

要想从数据分析菜鸟成长为高手，并非一蹴而就，当然也不需要将所有技能都学习到80分以上才算合格。基础技能需要熟练掌握，而高阶技能则应有所了解。例如，Excel基础技能、SQL基本查询能力和Python基础编程能力，这些通常被视为入门门槛。只要愿意学习，数月内就能完全上手。笔者就是跨专业自学过来的，实际上并没有想象中那么难。

真正难的是数据分析的思维能力，但这种能力也是可以逐步培养的。可以尝试给自己不同的角色和定位，刻意练习分析思维，不断积累经验，一定会有意想不到的收获。

在学习或工作过程中，了解Python数据分析的通用链路非常重要。虽然不需要对每个环节都深入掌握，但应尽早了解并学习。在实际工作中，根据自己的需求深入学习相关部分，这样不仅能打通全链路技能，还可以在某个环节形成自己的强项，这对升职加薪会有很大帮助。

最后，在未来的职业生涯中，保持良好的心态是制胜的关键。只要不断拓展自己，未来就会有无限的可能性。

第 2 章

NumPy基础

2

本章将介绍在Python数据分析过程中常用的NumPy基础编程。掌握NumPy的基本概念，并能够进行简单的数据查询和计算，将显著提升实际数据处理的效率。

2.1 NumPy 简介

NumPy（Numerical Python）是Python数据分析中不可或缺的基础库。它支持高效的多维数组和矩阵运算，并提供了丰富的数学函数库来进行数组运算。NumPy在处理大量数组数据时速度极快，且无须使用Python循环，从而显著提高计算效率。

NumPy最重要的特性之一是其N维数组对象——ndarray。该对象是同类型数据的集合，索引从0开始。ndarray对象用于存储同类型元素的多维数组，数组中的每个元素在内存中占有相同大小的存储区域。

本节将重点介绍NumPy在数据分析中常用的一些语法和功能。

重要提醒：本书主要介绍数据分析师成长过程中需要重点学习的知识点，对于没有任何编程基础的读者，建议先学习一些基础编程知识，然后阅读本书，以便快速理解数据分析师成长过程中的核心内容。

2.2 NumPy 结构

创建一个ndarray数组只需调用NumPy的array函数。每个函数都有具体的名称和相应的参

数，根据不同的参数实现不同的功能，具体格式如下：

```
numpy.array(object, dtype=None, copy=True, order=None, subok=False, ndmin=0)
```

相关参数说明如下：

- object：表示要转换为数组的对象或嵌套数列。
- dtype：数组元素的数据类型（可选）。
- copy：是否需要复制对象（可选）。
- order：创建数组的存储顺序，C表示行优先，F表示列优先，A表示任意顺序（默认值）。
- subok：如果为True，则返回一个与基类类型一致的数组。
- ndmin：指定生成数组的最小维度。

在上述参数中，常用的是object和dtype，其他参数很少用到。因此，在学习过程中，不必熟知每一个参数。以下是一个简单的代码示例：

```
import numpy as np
a = np.array([1,2,3],dtype=np.int32)
print (a)
```

输出结果如下：

```
[1 2 3]
```

以上代码输出了一个简单的一维数组。如果要生成更复杂的数组，需要修改各种参数。后续将详细介绍常见的数据生成和处理方法。

2.3　数据类型及转换

在对NumPy数组进行加、减、乘、除等操作之前，了解数组的数据类型非常重要，这有助于更好地进行数据交互。数据类型的初始化代码如下：

```
import numpy as np

data1=np.array([1,2,3],dtype=np.float64)
data2=np.array([1,2,3],dtype=np.int32)
print(data1.dtype)
print(data2.dtype)
```

输出结果如下：

```
float64
int32
```

如果需要转换数组的数据类型，可以使用astype方法：

```
import numpy as np

data1 = np.array([1,2,3,4])
print(data1.dtype)
data2 = data1.astype(np.float64)
print(data2.dtype)
```

输出结果如下：

```
int64
float64
```

通过astype方法，我们可以将data1的数据类型从int64转换为float64。
下面的代码示例将浮点数转换为整数类型（注意，小数部分将被截断）：

```
import numpy as np

data = np.array([1.5,-2.3,3.4,4.7])
print(data.astype(np.int32))
```

输出结果如下：

```
[ 1 -2  3  4]
```

可以看到，输出结果中小数部分被删除了。
此外，如果需要将包含数字的字符串元素转换为数字类型，也可以使用astype方法：

```
import numpy as np

data = np.array(['1.5','-2.3','3.4','4.7'], dtype = np.string_)
print(data.astype(float))
```

输出结果如下：

```
[ 1.5 -2.3  3.4  4.7]
```

通过astype方法进行转换时，会生成一个新的数组，而原数组的内容不会发生改变。

2.4　生成各种数组

在NumPy中，生成各种数据主要使用array函数。无论是单维数组、多维数组，还是随机数组，都可以通过该函数或其相关扩展功能来创建。针对不同的分析场景，我们需要创建不同的数组用于实践。下面先来看一个简单的代码示例：

```
import numpy as np
```

```
data = [2,3.5,5,0,3]
arr_data = np.array(data)
print(arr_data)
```

输出结果如下：

```
[2.  3.5  5.  0.  3. ]
```

以上代码生成了一个一维数组。如果要想创建一个多维数组，可以使用嵌套序列，代码如下：

```
import numpy as np

data = [[0,1,2,3],[4,5,6,7]]
arr_data = np.array(data)
print(arr_data)
```

输出结果如下：

```
[[0 1 2 3]
 [4 5 6 7]]
```

可以看到，data列表成功生成了一个二维数组。我们可以通过shape属性来确认数组的维度：

```
print(arr_data.shape)
```

输出结果如下：

```
(2, 4)
```

以上输出中的2代表行数，4代表列数，即生成了一个2行4列的数组。接下来，介绍几种常见的定制化数组的创建方法：

- 要创建一个全0的数组，可以使用zeros方法。
- 要创建一个全1的数组，可以使用ones方法。
- 要创建一个未初始化数值的数组，可以使用empty方法。

```
import numpy as np

data1 = np.zeros(5)          # 创建全0的数组
data2 = np.zeros((3,4))      # 创建全0的多维数组

data3 = np.ones(5)           # 创建全1的数组
data4 = np.ones((3,4))       # 创建全1的多维数组

data5 = np.empty((2,3,3))    # 创建未初始化的多维数组
print(data1)
print(data2)
```

```
print(data3)
print(data4)
print(data5)
```

输出结果如下：

```
[0. 0. 0. 0. 0.]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[1. 1. 1. 1. 1.]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
[[[-3.10503618e+231 -3.10503618e+231 -5.43472210e-323]
  [ 0.00000000e+000  2.12199579e-314  0.00000000e+000]
  [ 0.00000000e+000  0.00000000e+000  1.77229088e-310]]

 [[ 3.50977866e+064  0.00000000e+000  0.00000000e+000]
  [            nan             nan  3.50977942e+064]
  [ 6.93487456e-310 -3.10503618e+231 -3.10503618e+231]]]
```

由于NumPy专注于数值计算，默认的数据类型为float64（浮点型），除非另有指定。

2.5　数组计算

使用NumPy数组进行计算的效率通常高于通过for循环进行计算，因为NumPy数组支持逐元素操作。接下来，我们将详细介绍数组的基本加、减、乘、除计算。基础计算在数据分析过程中常常用于数据预处理，因此需要熟练掌握。

```
import numpy as np

data1 = np.array([[1, 2, 3], [2, 3, 4]])
print(data1)

# 加法
print(data1 + data1)

# 减法
print(data1 - data1)

# 乘法
print(data1 * data1)
```

```
# 除法
print(data1 / 2)
```

输出结果如下：

```
[[1 2 3]
 [2 3 4]]
[[2 4 6]
 [4 6 8]]
[[0 0 0]
 [0 0 0]]
[[ 1  4  9]
 [ 4  9 16]]
[[0.5 1.  1.5]
 [1.  1.5 2. ]]
```

如上代码所示，数组的计算是逐一对应进行的。例如，在除以2的运算中，数组中的每一个元素都被除以2。

2.6　索引和切片

在数据分析中，全面了解数据，特别是识别和处理异常值至关重要。为了快速定位具体的异常数据，通常需要借助索引进行高效的数据筛选。因此，为了能够更精准地掌握数据现状并为后续的计算分析奠定基础，熟练运用数据的索引与切片技术至关重要。

NumPy的索引和切片是操作数组的重要功能，允许用户以灵活的方式访问和修改数组中的元素。

- 索引：使用方括号（[ ]）加上索引号，引用数组中特定位置的元素。索引的作用是从数组中提取特定的元素，重新组成一个子数组。
- 切片：访问数组的连续元素子集。切片操作使用冒号(:)表示范围，语法为start:stop:step。其中start是起始位置，stop是结束位置（不包括），step是步长。切片操作可以应用于一维或多维数组。

1. 一维数组的索引和切片

一维数组的索引和切片代码如下：

```
import numpy as np

data = np.arange(15)
print(data)
```

```
print(data[4])      # 按照索引进行数据查询
print(data[3:6])
data[3:6] = 10      # 对区间数据进行赋值
print(data)
```

输出结果如下：

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
4
[3 4 5]
[ 0  1  2 10 10 10  6  7  8  9 10 11 12 13 14]
```

由上述代码可见，NumPy的索引方式与Python列表相似，均从0开始。当对data[3:6]进行赋值时，整个切片的值会被修改，且这一变化反映在原数组中，最后对应位置的值被修改为10。

使用切片索引[:]可以引用数组的所有值，代码如下：

```
import numpy as np

data = np.arange(15)
print(data[:])
```

输出结果如下：

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

2. 二维数组的索引和切片

二维数组的索引与一维数组有显著区别。一维数组的索引返回的是具体位置的单个值，而二维数组的索引返回的是一维数组。代码如下：

```
import numpy as np

data = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5]])
print(data[2])
```

输出结果如下：

```
[3 4 5]
```

要选择二维数组中的具体元素，可以使用以下两种方式：

```
print(data[0][0])
print(data[0,0])
```

输出结果如下：

```
1
1
```

由上述代码可以看出，data[0][0]表示先取第一个数组，再取该数组中的第一个元素。无

论是使用两个方括号，还是使用逗号隔开，都可以用来索引具体元素。

3. 多维数组的索引和切片

多维数组的索引逻辑与二维数组相似，逐层获取数据即可。代码如下：

```
import numpy as np

data = np.array([[[1, 2, 3], [2, 3, 4]], [[3, 4, 5], [4, 5, 6]]])
print(data)
print(data[0])          # 输出一个2×3的数组
print(data[0][0])
print(data[0][0][0])
print(data[0, 0, 0])
```

输出结果如下：

```
[[[1 2 3]
  [2 3 4]]

 [[3 4 5]
  [4 5 6]]]
[[1 2 3]
 [2 3 4]]
[1 2 3]
1
1
```

由上述代码可以看出，多维数组是数组的嵌套结构，只需按照顺序逐层索引即可。

对数组进行切片时，需要特别小心。下面是通过二维数据进行切片的详细示例：

```
import numpy as np

data = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5]])
print(data)

# 选择前两行的数据
print(data[:2])

# 选择前两行，第2列及之后的数据
print(data[:2, 1:])

# 选择第2行的前两列数据
print(data[1, :2])

# 选择第3列的前两行数据
print(data[:2, 2])
```

```
# 选择所有行的第1列数据
print(data[:, :1])

# 将0赋值给前两行第2和第3列的元素
data[:2, 1:] = 0
print(data)
```

输出结果如下：

```
[[1 2 3]
 [2 3 4]
 [3 4 5]]
[[1 2 3]
 [2 3 4]]
[[2 3]
 [3 4]]
[2 3]
[3 4]
[[1]
 [2]
 [3]]
[[1 0 0]
 [2 0 0]
 [3 4 5]]
```

从上述输出结果可以看出，使用索引可以对数组中任意位置的数据进行切片。如果对切片进行赋值，整个切片的值会被重新赋值，原数组的数据也会相应地被修改。

2.7　布尔索引

除了使用具体的数值来索引元素外，我们还可以通过传入布尔值数组进行索引。在数据分析中，通过布尔索引可以更方便地洞察数据的特征。因此，充分了解布尔索引非常重要。

```
import numpy as np

names = np.array(['a', 'b', 'c'])
data = np.random.randn(3, 4)
print(names)
print(data)
print(names == 'a')
print(data[names == 'a'])
```

输出结果如下：

```
['a' 'b' 'c']
[[ 0.62882063 -0.9443457   0.78765372  0.20187566]
 [-0.39464619  0.21552156  0.8627636   2.10803587]
 [ 1.42294934 -0.37465153 -0.25897518 -2.26696677]]
[ True False False]
[[ 0.62882063 -0.9443457   0.78765372  0.20187566]]
```

由输出结果中可以看到，根据布尔值数组，我们成功提取了第一行的数据。需要注意的是，布尔值数组的长度必须与原数组的行数一致。

在进行行列切片时，同样可以使用布尔值进行索引：

```
print(data[names == 'a',2:])
print(data[names == 'a',3])
```

输出结果如下：

```
[[0.57400185 0.86182715]]
[0.86182715]
```

在某些特殊情况下，可能需要执行取反操作，即选择除某一行或某一列之外的所有数据。可以使用“!=”运算符或者在条件前加上“~”运算符来实现取反运算：

```
print(data[~(names == 'a')])
condition = names == 'a'
print(data[~condition])
```

输出结果如下：

```
[[ 1.68083323 -0.87480113  2.10586516  0.61812069]
 [ 3.09885395 -1.44095797 -1.11527903  1.61043083]]
[[ 1.68083323 -0.87480113  2.10586516  0.61812069]
 [ 3.09885395 -1.44095797 -1.11527903  1.61043083]]
```

如果想选择多个名字，可以使用条件组合：

```
condition = (names == 'a') | (names == 'b')
print(data[condition])
```

输出结果如下：

```
[[-1.75475781 -1.37914091 -0.38530713  1.03460284]
 [-1.84118133 -0.53665693 -0.61262528  0.21221634]]
```

在组合条件时，使用“&”运算符表示“与”运算，使用“|”运算符表示“或”运算。

如果想修改数组中符合条件的数据，可以在条件判断表达式后进行赋值：

```
data[data<0] = 0
print(data)
```

输出结果如下：

```
[[0.         1.46486123 0.         1.4938428 ]
 [0.76859983 1.44274013 0.4706168  0.        ]
 [0.475637   0.         1.35437378 0.24853394]]
```

如上所示，所有小于0的数值都被设置为0。

2.8　本章小结

尽管NumPy为数值数据操作提供了基础的计算功能，但在大多数情况下，我们更倾向于使用Pandas进行数据分析和统计，尤其是在处理表格数据时。Pandas提供了更多针对特定场景的函数，例如时间序列操作等，而这些功能是NumPy所不具备的。因此，本章仅介绍NumPy的基础用法，包括计算和索引等内容，帮助读者对NumPy有一个初步了解。接下来，我们将重点探讨Pandas的各种使用场景。