上海大学

SHANGHAI UNIVERSITY

课程设计（论文）

UNDERGRADUATE COURSE (THESIS)

题目:显卡价格分析与预测

学院	计算机工程与科学学院
专业	智能科学与技术
学号	19122753
学生姓名	顾骏杰
指导教师	武星
起讫日期	2021.03.29 – 2021.05.31

目录

显卡价格分析与预测

摘要

显卡是计算机的基础且重要的组成部分，将计算机系统的显示信息进行转换以驱动显示器运行，控制显示器的正确显示。显卡是人机交互的重要设备之一。显卡的发展经历了 ISA、VESA、PCI、AGP、PCI-Express 五种接口类型，NVidia 和 AMD 的 PCI-E 显卡还支持多显卡技术（NVidia 的有桥/无桥 SLI，AMD 的 Crossfire），进一步增强了计算机的计算性能。

我们对于显卡的需求是客观存在的。但由于生产工艺的复杂，GPU 核心的产能有限，也没有专门的生产 GPU 核心的生产线。除了核心，显存等附属芯片同样产能有限，有线的产能还要分配给内存、手机等其他领域。受限的产能造成了显卡价格始终居高不下，并且还时常波动。因而，认为研究显卡价格具有实际意义，并能对显卡的消费行为提出一定的指导作用。

对显卡价格的分析，可以分为对显卡首发价格，即官方定价的分析，以及影响显卡价格波动的因素。对显卡官方定价可以从显卡本身的配置出发，而对价格波动则要结合当时实际情况。

关键词：显卡，大数据，分析预测

Analysis and forecast on GPU price

ABSTRACT

Video card is the foundation and important part of computer. It transforms the display information of computer system to drive the display to run and control the correct display of the display. Video card is one of the important equipment of human-computer interaction. The development of video card has experienced five interface types: ISA, VESA, PCI, AGP and PCI Express. NVIDIA and AMD PCI-E graphics cards also support multi graphics card technology (NVIDIA bridge / bridge free SLI, AMD Crossfire), which further enhances the computer computing performance.

Our demand for graphics card is objective. However, due to the complexity of production process, the capacity of GPU core is limited, and there is no special production line for GPU core. Besides the core, the capacity of the accessory chips such as video memory is also limited, and the capacity of the cable is also allocated to memory, mobile phones and other fields. The limited capacity has led to the high price of graphics cards and also fluctuates frequently. Therefore, it is of practical significance to study the price of graphics card and can provide some guidance for the consumption behavior of graphics card.

The analysis of the price of the card can be divided into the analysis of the first price of the card, that is, the official price, and the factors affecting the price fluctuation of the card. The official pricing of graphics card can be based on the configuration of the card itself, while the price fluctuation should be combined with the actual situation at that time.

Keywords: Graphic card, Big data, Analysis and forecast

第 1 章 绪论

本章主要描述了显卡价格构建的背景、意义，分析了相关现状，进而提出了本文所要研究的内容及目标。

§1.1 显卡价格分析的背景及意义

自 2020 年末，显卡价格开始全球性的上涨。受 2020 年疫情影响，导致全球供应链紧张，显卡芯片产能不足，市场供不应求。部分显卡的价格，即使是在已经面世两年后，仍然超过了首发售价。由于货源少的缘故，除了 RTX30 系列不断疯涨中，市场上 16 系列、20 系列或 RX5000 系列等显卡开始缺货并疯狂涨价中，这是十分不常见的事。除此之外，NVIDIA 的 30 系显卡的定价也引发了热烈讨论。足见人们对显卡的关注程度之深。因而，认为研究显卡价格具有实际意义，并能对显卡的消费行为提出一定的指导作用。

§1.2 显卡价格走势研究现状及存在的问题

显卡作为计算机的重要组成部分，一直受到广大主机厂商、零配件销售店以及众多自行组装电脑的用户的关注，可以说只要有接触显卡机会的人群对显卡价格都比较敏感。下面具体阐述研究现状以及存在的问题。

§1.2.1 研究现状

目前尚无系统的、科学的对显卡价格走势进行分析预测的学术组织或个人，现有研究基本基于经验与市场机制，将显卡作为高附加值工业制成品进行分析。

对显卡价格的分析，可以分为对显卡首发价格，即官方定价的分析，以及影响显卡价格波动的因素。对显卡官方定价可以从显卡本身的配置出发，而对价格波动则要结合当时实际情况。

对于显卡本身价格的分析可以表现出官方显卡本身的态度，一定程度上反映了显卡的制造成本、科技水平，同一世代不同价位的显卡表现出同一架构不同的周边配置和附件对价格的影响。通常情况下，同一世代的显卡的性能越好，显卡的价格越高。不同世代相近价格的显卡，世代较新的性能较高。性能相近的消费级显卡与专业级显卡，专业级显卡的价格更高。同一显卡在不同显卡制造商的不同型号上价格亦会有所浮动，可能通过减配降低价格，也可能用过更好的用料、散热、更高的核心频率甚至是灯效、联名等非性能因素而提升价格。

对于显卡实际价格的波动则是市场供求机制的简单体现。供不应求，则价格上升；供过于求则价格下降。一般来说，新显卡发售后，由于对新显卡的需求，

以及部分商家的囤货行为，显卡价格会首先上升。对于官方指导价而言信价比最高的显卡一般涨幅较为明显。待到显卡热潮过去，一般是数个月之后，显卡价格就会逐渐回落到首发价附近。在下一世代的显卡发售之后，上时代的显卡，由于新显卡供不应求，又可能会出现一次较小幅度的价格上升。等到新显卡的供货稳定后，经销商出于清除库存的原因，显卡价格会再次下跌，常常会低于首发售价。

§1.2.2 传统经验预测存在的问题

虽然经验预测在许多时候能够对显卡购买行为起到一定的指导，但是仍存在很多问题。

（1）不同购买力、购买方式得到的经验结论往往互相矛盾。

（2）过于主观，缺少数据支撑，建议内容模糊。

（3）抗干扰能力很差，没有对抗市场变化的手段。

§1.2.3 显卡价格模型构建的难点

尽管已经有了粗略的经验分析，但针对显卡的价格分析，仍然存在了以下一些技术难点：

（1）显卡消费人群复杂，制造厂商众多，难以得到准确的数据。

（2）显卡价格影响因素很多，且涉及方面广，不存在单一闭环。

§1.3 本文研究内容及目标

本文针对显卡价格的现状，分析得出显卡官方定价的影响因素，分析官方定价策略，建立显卡的硬件配置-性能-价格模型。再从社会角度分析显卡价格波动成因，并进行建模分析。

§1.3.1 研究内容

本文研究内容有以下几个方面。

（1）显卡价格与性能分析；

（2）显卡性能与配置分析；

（3）显卡硬件配置变化分析；

（4）显卡聚类分析；

（5）价格因素分析。

§1.3.2 研究目标

针对本文的研究内容，制定了以下几项指标：

（1）搜集显卡的硬件配置、性能跑分；

（2）搜集显卡的官方价格、市场价格；

（3）搜集 BTC 等虚拟货币的价格走势、世界金融的近期事件；

（4）根据已有的数据建立模型；

（5）验证提出的方法。

§1.4 本文组织结构

整篇论文分为五章。

第一章介绍了研究背景、研究意义，分析了显卡价格研究现状以及存在的问题和难点，并提出了本文的研究内容以及研究目标。

第二章主要介绍了本体的基本概念，并提出了影响显卡价格的因素。

第三章主要讲述影响显卡官方价格的因素，介绍了硬件配置对显卡价格的影响，建立显卡的配置-性能-价格模型。

第四章主要描述了虚拟货币对显卡价格的影响，分析显卡价格与虚拟货币价格之间的关系。

第五章对全文进行了总结，归纳了本文的主要工作与创新点，并指出了需要进一步研究的问题。

第 2 章 显卡硬件及其作用

本章具体描述了显卡的基本物理特性：介绍了显卡的基本概念，并引出后文所使用的相关显卡概念。

§2.1 显卡基本组成

本节介绍了显卡的基本概念以及显卡的基本组成部分。

§2.1.1 显卡的基本概念

显卡又称显示卡(Video card)，是计算机中一个重要的组成部分，承担输出显示图形的任务，对喜欢玩游戏和从事专业图形设计的人来说，显卡非常重要。主流显卡的显示芯片主要由 NVIDIA（英伟达）和 AMD（超微半导体）两大厂商制造，配置较高的计算机，都包含显卡计算核心。在科学计算中，显卡被称为显示加速卡。

显示芯片(Video chipset)是显卡的主要处理单元，因此又称为图形处理器(Graphic Processing Unit，GPU)，GPU 是 NVIDIA 公司在发布 GeForce 256 图形处理芯片时首先提出的概念。尤其是在处理 3D 图形时，GPU 使显卡减少了对 CPU 的依赖，并完成部分原本属于 CPU 的工作。GPU 所采用的核心技术有硬件 T&L（几何转换和光照处理）、立方环境材质贴图和顶点混合、纹理压缩和凹凸映射贴图、双重纹理四像素 256 位渲染引擎等，而硬件 T&L 技术可以说是 GPU 的标志。

显卡是插在主板上的扩展槽里的（一般是 PCI-E 插槽，此前还有 AGP、PCI、ISA 等插槽）。它主要负责把主机向显示器发出的显示信号转化为一般电器信号，使得显示器能明白个人计算机在让它做什么。显卡主要由显卡主板、显示芯片、显示存储器、散热器（散热片、风扇）等部分组成。显卡的主要芯片叫“显示芯片”（Video chipset，也叫 GPU 或 VPU，图形处理器或视觉处理器），是显卡的主要处理单元。显卡上也有和计算机存储器相似的存储器，称为“显示存储器”，简称显存。

早期的显卡只是单纯意义的显卡，只起到信号转换的作用；我们一般使用的显卡都带有 3D 画面运算和图形加速功能，所以也叫做“图形加速卡”或“3D 加速卡”。PC 上最早的显卡是 IBM 在 1981 年推出的 5150 个人计算机上所搭载的 MDA 和 CGA 两款 2D 加速卡 。

显卡通常由总线接口、PCB 板、显示芯片、显存、RAMDAC、VGA BIOS、VGA 功能插针、D-sub 插座及其他外围组件构成，显卡大多还具有 VGA、DVI 显示器接口或者 HDMI 接口及 S-Video 端子和 Display Port 接口。

§2.1.2 一般显卡结构

一般显卡的结构如下：

电容：电容是显卡中非常重要的组成部件，因为显示画质的优劣主要取决于电容的质量，而电容的好坏直接影响到显卡电路的质襞。

显存：显存负责存储显示芯片需要处理的各种数据，其容量的大小，性能的高低，直接影响着电脑的显示效果。新显卡均采用 DDR6/DDR5 的显存， 主流显存容量一般为 2GB ~ 8GB。

GPU 及风扇：GPU 即显卡芯片，它负责显卡绝大部分的计算工作，相当于 CPU 在电脑中的作用。GPU 风扇的作用是给 GPU 散热。

显卡接口：通常被叫做金手指，可分为 PCI、 AGP 和 PCI Express 三种，PCI 和 AGP 显卡接口都基本被淘汰， 市面上主流显卡采用 PCI Express 的显卡。

外设接口：显卡外设接口担负着显卡的输出任务，新显卡包括一个传统 VGA 模拟接口和一个或多个数字接口(DVI、HDMI 和 DP)。

桥接接口：中高端显卡可支持多块同时工作，它们之间就是通过桥接器连接桥接口。

§2.2 显卡主要逻辑组成部分及其作用

对于显卡首发价格，即官方指导价格的分析，应当从显卡本身出发，分析显卡本身的各项硬件因素。

§2.2.1 显卡架构

架构其实是影响显卡性能的最重要指标，新一代架构的出现往往也就意味着产品的迭代升级，比如从 Maxwell（麦克斯韦）→Pascal（帕斯卡）→Turing（图灵）→Ampere（安培）。

显卡架构是显示芯片各种处理单元的组成和工作模式，效率高的架构在同等参数下执行力更高，性能也更好。比如 Nvida 的开普勒架构，比费米架构的效率和功耗都进步了很多，同等参数的显卡比如费米架构的 GTX550Ti 和开普勒架构的 GTX650，在性能上 GTX650 就稍强，功耗也更低。显卡的驱动也很重要，显卡使用最新的驱动往往比老版本的驱动表现的要好一些。

§2.2.2 显卡制程

制程是指 IC 内电路与电路之间的距离。制程工艺的趋势是向密集度愈高的方向发展。密度愈高的 IC 电路设计，意味着在同样大小面积的 IC 中，可以拥有密度更高、功能更复杂的电路设计。微电子技术的发展与进步，主要是靠工艺技

术的不断改进，使得器件的特征尺寸不断缩小，从而集成度不断提高，功耗降低，器件性能得到提高。芯片制造工艺在 1995 年以后，从 500 纳米、350 纳米、250 纳米、180 纳米、150 纳米、130 纳米、90 纳米、65 纳米、45 纳米、32 纳米、28 纳米、22 纳米、14 纳米、10 纳米、7 纳米，一直发展到最新的 5 纳米、3 纳米。

一般以当前处理器的制程工艺乘以 0.714 即可得出下一代的制程工艺，如 10*0.714=7.14，即 7.14 纳米。

§2.2.3 核心数量

CUDA 是 NVidia 主推的并行计算架构。CUDA core，或者称为 sp，是主要的运算单元，内部有分别处理 int 和单精度 float 的部分，用于执行一些常用的运算指令。除了 sp 之外还有双精度单元用于科学计算以及特殊运算单元 sfu 来执行比较复杂的运算指令比如三角函数。一个特定计算量的任务，如果核心的数目越多，那么单位时间内执行的运算就越多，所以完成这个任务所花费的时间就越少，显然计算速度就越快。CUDA 运算速度只和核心频率有关，而 CUDA 核心数量则决定了显卡的计算力的强弱。

§2.2.4 显卡频率

（1） 基准频率

是指显示核心的工作频率，其工作频率在一定程度上可以反映出显示核心的性能，但显卡的性能是由核心频率、显存、像素管线、像素填充率等等多方面的情况所决定的，因此在显示核心不同的情况下，核心频率高并不代表此显卡性能强劲。

（2） 加速频率

表示在环境允许 GPU 进入超频模式时,可以达到的最高频率。

频率的数值并不是无休止的高才好，高频率必然带来高发热量高能耗，同时也容易出现花屏之类的问题。另外频率的高低在相同型号下才可进行比较，型号不同的显卡相互对比频率则没有任何意义。

§2.2.5 显存配置

§2.2.5.1 显存类型

（1） GDDR4

DDR4 相比 DDR3 最大的区别有三点：16bit 预取机制（DDR3 为 8bit），同样内核频率下理论速度是 DDR3 的两倍；更可靠的传输规范，数据可靠性进一步提升；工作电压降为 1.2V，更节能。

（2） GDDR5

第五版图形用双倍数据传输率存储器（Graphics Double Data Rate, version 5，简称 GDDR5），是一种高性能显卡使用的同步动态随机存取存储器，专为高带宽需求计算机应用所设计。由 AMD、SK Hynix、三星电子、NVIDIA、JEDEC 等联合制定，取代 GDDR3 以及 GDDR4 显示存储器。

（3） GDDR5X

美光科技于 2015 年 10 月宣布成功开发出 GDDR5X，比 GDDR5 更高、逼近现时 HBM 的带宽速度（于存储器总线维持 256 比特、等效时钟频率 14GHz 的条件下，至少可拥有 448GB/s 的带宽）。已知 GDDR5X 相较于 GDDR5 的改变，在于存储器预取从 8n 提升到 16n、更低的运行电压（和 DDR3L 相同的 1.35V）、其余部分与 GDDR5 的基本相同。

（4） GDDR6

GDDR6 英文全称为 Graphics Double Data Rate, version 6，属于第六代版图形用双倍数据传输率存储器。简单来说，也就是显卡的缓存。GDDR6 是目前最新六代技术，相比目前主流的 GDDR5 更先进，频率更高，更有利于提升显卡性能。GDDR6 显存采用双通道读写设计,显存速度高达 16Gbps, 工作电压相比上一代 GDDR5 显存有所降低,最高缓存容量能够高达 32Gb.

（5） HBM/HBM2

HBM 比起 GDDR5 拥有更高的带宽和比特，比特部分每一颗 HBM 存储器就高达 1024 位，存储器时钟频率只有 500 左右，电压也比 GDDR5 小，还能缩小存储器布置空间，不过制造困难成本也高，所以供应量非常少。在 HBM 发布之后，HBM 2 也成功开发出来，存储器比特提升至两倍。

目前在显存行业有两个方向，一是传统的 GDDR 继续演化，NVIDIA RTX 20 系列已经用上最新的 GDDR6，二就是高带宽的 HBM，已经进化到第二代，NVIDIA、AMD 的专业计算卡以及 AMD 的部分高端显卡都配备了它。

§2.2.5.2 数据位数

数据位数指的是在一个时钟周期之内能传送的 bit 数，它是决定显存带宽的重要因素，与显卡性能息息相关。当显存种类相同并且工作频率相同时，数据位数越大，它的性能就越高。

§2.2.5.3 显存容量

显存容量是显卡上本地显存的容量数，这是选择显卡的关键参数之一。显存容量的大小决定着显存临时存储数据的能力，在一定程度上也会影响显卡的性能。显存容量也是随着显卡的发展而逐步增大的，并且有越来越增大的趋势。显存容量从早期的 512KB、1MB、2MB 等极小容量，发展到 64MB、128MB、256MB、512MB、

768MB，一直到目前主流的 2GB、4GB、6GB 和高档显卡的 8GB、16GB、32GB 某些专业显卡甚至已经具有 48GB 的显存了。在显卡最大分辨率方面，最大分辨率在一定程度上跟显存有着直接关系，因为这些像素点的数据最初都要存储于显存内，因此显存容量会影响到最大分辨率。

§2.2.5.4 显存带宽

显存带宽就是显示芯片与显存之间的桥梁，带宽越大，则显示芯片与显存之间的通讯就越快捷。为了标示这宽度，显存带宽的单位为：字节/秒。显存的带宽与显存的位宽及显存的速度有关。

§2.3 本章小结

本章简述了显卡的基本概念与重要组成部分，介绍了各个部分的作用以及部分组件的发展历史，提及了不同等级部件对整体显卡性能的影响，但没有设计过多具体的、量化的内容。这部分将在下面的第三章进行详细介绍。

第 3 章 基于硬件的显卡价格分析方法

本章是全文的重点章节，全面阐述了本文的工作内容。通过对不同显卡的硬件数据进行分析，再与显卡的跑分、价格进行比较分析，建立显卡的硬件-性能-价格模型。本章主要反映的是显卡官方对显卡的价格策略。

§3.1 硬件-性能-价格模型概述

从本文开头就提到了显卡的硬件-性能-价格模型。在这里我们没有选择直接建立显卡的硬件-价格模型，主要原因在于显卡本身作为高附加值产品，硬件性能与价格并没有直接的联系（可以参考宝石界的克拉溢价效应），但一定与显卡的性能直接相关。而后，根据显卡性能，显卡厂商根据价格策略，对显卡进行定价，所以性能与价格之间存在着非线性的映射关系。因此直接建立显卡的硬件-价格模型欠妥。本文首先使用显卡硬件-性能-价格模型进行分析（下文简称显卡价格模型），了解其中的对应关系之后，再建立硬件到价格的直接模型。

§3.2 数据采集

数据是价格分析的基础，数据的质量决定了价格模型反映真实情况的能力和预测质量。本节将讨论显卡价格模型的数据采集。

§3.2.1 Kaggle 数据集 Computer parts - CPUs and GPUs

Kaggle 是由联合创始人、首席执行官安东尼·高德布卢姆 2010 年在墨尔本创立的，主要为开发商和数据科学家提供举办机器学习竞赛、托管数据库、编写和分享代码的平台。

Kaggle 数据集 Computer parts - CPUs and GPUs 中包含了大量 CPU 与 GPU 的硬件信息，包括基本的架构、频率，到的接口数量，再到最大分辨率。数据集非常完善，但是数据集中收录的显卡时间范围过广，因此不可直接用于分析。

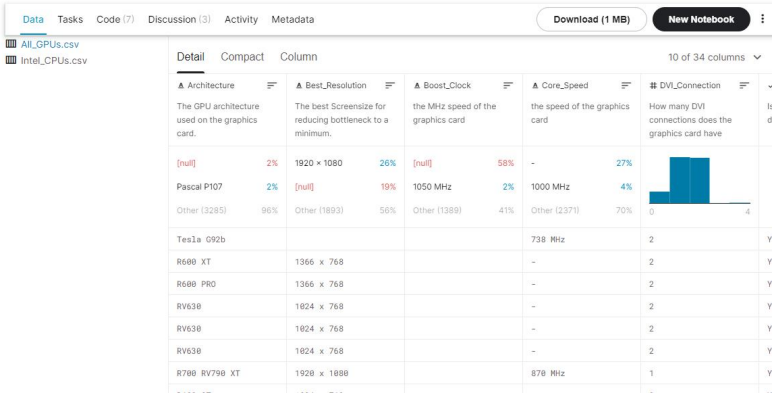


图 3-1 Kaggle 数据集 Computer parts - CPUs and GPUs

§3.2.2 NVIDIA 官网

作为全球最大的显卡生产制造商之一，NVIDIA 的显卡在我们生活的许多方面都有应用在 NVIDIA 的官方网站上可以获得 NVIDIA GeForce 6 系列到 30 系列的全部显卡的重要信息。

图 3-2 NVIDIA 官网的显卡信息

有了官方数据后，应当对数据进行规整化，以利于后期的处理。

表 3-1 显卡数据（部分）

名称	架构	CUDA 核心	加速频率	基准频率	内存速度	显存位宽	显存带宽	跑分	价格
GEFORCE GTX 1030 SDDR4	Pascal	384	1379		1	64	16.8		579
GEFORCE GTX 1030 GDDR5	Pascal	384	1468		3	64	48	2621	
GEFORCE GTX 1050 2GB	Pascal	640	1455	1354	7	128	112	4461	
GEFORCE GTX 1050 3GB	Pascal	768	1581	1392	7	96	84	5229	
GEFORCE GTX 1050 TI	Pascal	768	1392	1290	7	128	112	6346	1499
GEFORCE GTX 1060 3GB	Pascal	1152	1708	1506	8	192	192	6696	1599
GEFORCE GTX 1060 6GB	Pascal	1280	1708	1506	8	192	192	9955	1839
GEFORCE GTX 1070	Pascal	1920	1683	1506	8	256	256	13353	3099
GEFORCE GTX 1070 TI	Pascal	2432	1683	1607	8	256	256	14195	3999
GEFORCE GTX 1080	Pascal	2560	1733	1607	10	256	320	14944	4499

GEFORCE GTX 1080 TI	Pascal	3584	1582		11	352	484	17925	6699
GEFORCE GTX 1650 GDDR5	Turing	896	1665	1485	8	128	128	6968	1599
GEFORCE GTX 1650 GDDR6	Turing	896	1590	1410	12	128	192	7746	1899
GEFORCE GTX 1650 SUPER	Turing	1280	1725	1530	12	128	192	9868	2099
GEFORCE GTX 1660	Turing	1408	1785	1530	8	192	192	11595	1999
GEFORCE GTX 1660 SUPER	Turing	1408	1785	1530	14	192	336	12668	2199
GEFORCE GTX 1660 TI	Turing	1536	1770	1500	12	192	288	11974	2299
GEFORCE RTX 2060	Turing	1920	1680	1365	14	192	336	13864	3199
GEFORCE RTX 2060 SUPER	Turing	2176	1650	1470	14	256	448	16455	3499
GEFORCE RTX 2070	Turing	2304	1620	1410	14	256	448	16105	3899
GEFORCE RTX 2070 SUPER	Turing	2560	1770	1605	14	256	448	18112	4599
GEFORCE RTX 2080	Turing	2944	1710	1515	14	256	448	18610	6499
GEFORCE RTX 2080 SUPER	Turing	3072	1815	1650	15.5	256	496	19449	7299
GEFORCE RTX 2080 TI	Turing	4352	1545	1350	14	352	616	21662	8999
GEFORCE RTX 3060	Ampere	3584	1780	1320	14	192	360	16660	2499
GEFORCE RTX 3060 TI	Ampere	4864	1670	1410	14	256	448	19652	2999
GEFORCE RTX 3070	Ampere	5888	1730	1500	14	256	448	21689	3899
GEFORCE RTX 3080	Ampere	8704	1710	1440	19	320	760	24184	5499
GEFORCE RTX 3090	Ampere	10496	1700	1400	19	384	912	25557	11999
GEFORCE GTX 950	Maxwell	768	1188	1024	6	128	105	5398	1199
GEFORCE GTX 960	Maxwell	1024	1178	1024	7	128	112	6023	1699
GEFORCE GTX 970	Maxwell	1664	1050	1050	7	256	224	9722	2399
GEFORCE GTX 980	Maxwell	2048	1126	1126	7	256	224	11222	3999
GEFORCE GTX 980 TI	Maxwell	2816	1075	1000	7	384	336	13877	4699
GEFORCE GTX TITAN X	Maxwell	3072	1075	1000	7	384	336	13001	3899
GEFORCE GTX 750	Kepler	512	1085	1020	5	128	80	3399	799
GEFORCE GTX 750 TI	Kepler	640	1085	1020	5	128	86	3934	999
GEFORCE GTX 760	Kepler	1152	1033	980	6	256	162	4778	1099
GEFORCE GTX 770	Kepler	1536	1085	1046	7	256	224	5867	1299
GEFORCE GTX 780	Kepler	2304	900	863	6	384	288	7985	1499
GEFORCE GTX 780 TI	Kepler	2880	928	875	7	384	336	9194	1899
GEFORCE GTX TITAN	Kepler							8528	2099
GEFORCE GTX	Kepler	2880	980	889	7	384	336	9211	2799

TITAN BLACK									
GEFORCE GTX TITAN Z	Kepler	5760	876	705	7	768	672	9133	5999
GEFORCE GTX 650	Kepler	384	1058	1058	5	128	80	1761	599
GEFORCE GTX 650 TI	Kepler	768	928	928	5	128	86	2539	999
GEFORCE GTX 660	Kepler	960	1033	980	6	192	144	3960	799
GEFORCE GTX 660 TI	Kepler	1344	980	915	6	192	144	4344	1099
GEFORCE GTX 670	Kepler	1344	980	915	6	256	192	5337	999
GEFORCE GTX 680	Kepler	1536	1058	1006	6	256	192	5483	1099
GEFORCE GTX 690	Kepler	3072	1019	915	6	512	384	5702	1699

§3.2.3 比价网站

由于项目的时间问题，从零开始采集数据虽然可行，但是时间段过短，对部分关键时间点的采集是缺失的，会有很大可能改变最后得出的结论。因此，需要获取项目开始时刻之前的显卡价格数据。可以借助比价网站的历史价格查询功能实现。

目前提供这一服务的网站有很多，包括优惠买、慢慢买、西贴、过客等，网站会定时从众多网购平台，包括淘宝、京东、天猫等电商，获取商品的价格，存入数据库中。用户使用网站时，提交一个指向商品的 url，网站返回它所采集的历史数据。依靠这一功能，我们可以较为便捷的获取到显卡的历史价格数据。

图 3-3 比价网查询显卡历史价格

dat.csv - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
"2019,2,17",2149.00,"购买1件,页面价:2199.00,优惠券：满1999减50"
"2019,2,18",2199.00,""
"2019,2,19",2199.00,""
"2019,2,20",2199.00,""
"2019,2,21",2199.00,""
"2019,2,22",2199.00,""
"2019,2,23",2199.00,""
"2019,2,24",2199.00,""
"2019,2,25",2148.00,"购买1件,页面价:2198.00,优惠券：满2099减50"
"2019,2,26",2198.00,""
"2019,2,27",2198.00,""
"2019,2,28",2149.00,"购买1件,页面价:2199.00,优惠券：满2099减50"
"2019,2,29",2199.00,""
"2019,2,30",2199.00,""
"2019,2,31",2199.00,""
"2019,3,1",2179.00,"购买1件,页面价:2199.00,优惠券：满1749减20"
"2019,3,2",2179.00,"购买1件,页面价:2199.00,优惠券：满1000减20"
"2019,3,3",2079.00,"购买1件,页面价:2199.00,满减：满2199减100,优惠券：满1000减20"
"2019,3,4",2199.00,""
"2019,3,5",2199.00,""
"2019,3,6",2199.00,""
"2019,3,7",2199.00,""
"2019,3,8",2199.00,""
"2019,3,9",2199.00,""
"2019,3,10",2199.00,""
"2019,3,11",2079.00,"购买1件,页面价:2099.00,优惠券：满1000减20"
"2019,3,12",2099.00,""
"2019,3,13",2099.00,""
"2019,3,14",2099.00,""
"2019,3,15",2099.00,""
"2019,3,16",2099.00,""
"2019,3,17",2039.00,"购买1件,页面价:2199.00,满减：满1749减140,优惠券：满1000减20"
"2019,3,18",2199.00,""
"2019,3,19",2129.00,"购买1件,页面价:2199.00,满减：满1749减50,优惠券：满1000减20"
"2019,3,20",2079.00,"购买1件,页面价:2199.00,满减：满1749减100,优惠券：满1000减20"
"2019,3,21",2199.00,""
"2019,3,22",1979.00,"购买1件,页面价:2099.00,满减：满1749减100,优惠券：满1000减20"
"2019,3,23",2099.00,""
"2019,3,24",2099.00,""
"2019,3,25",2099.00,""
```

第 29 行，第 23 列 100% Windows (CRLF) UTF-8

图 3-4 爬取页面得到的数据

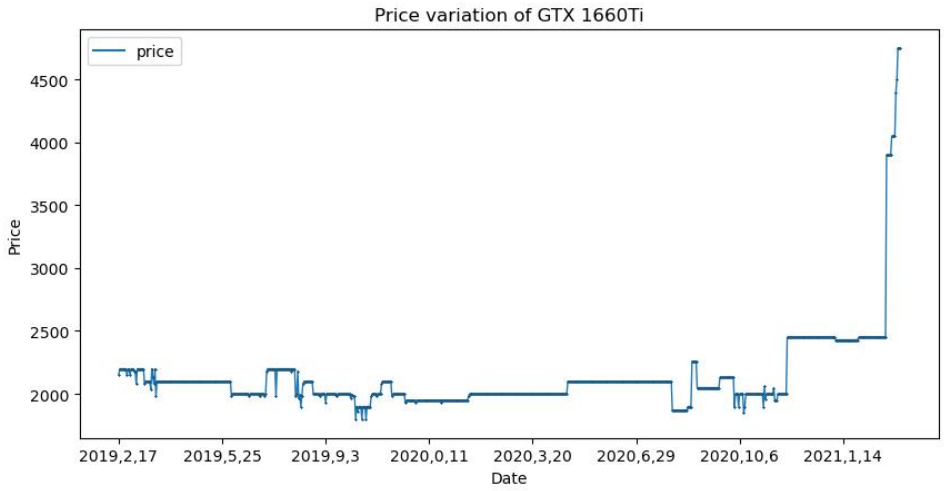
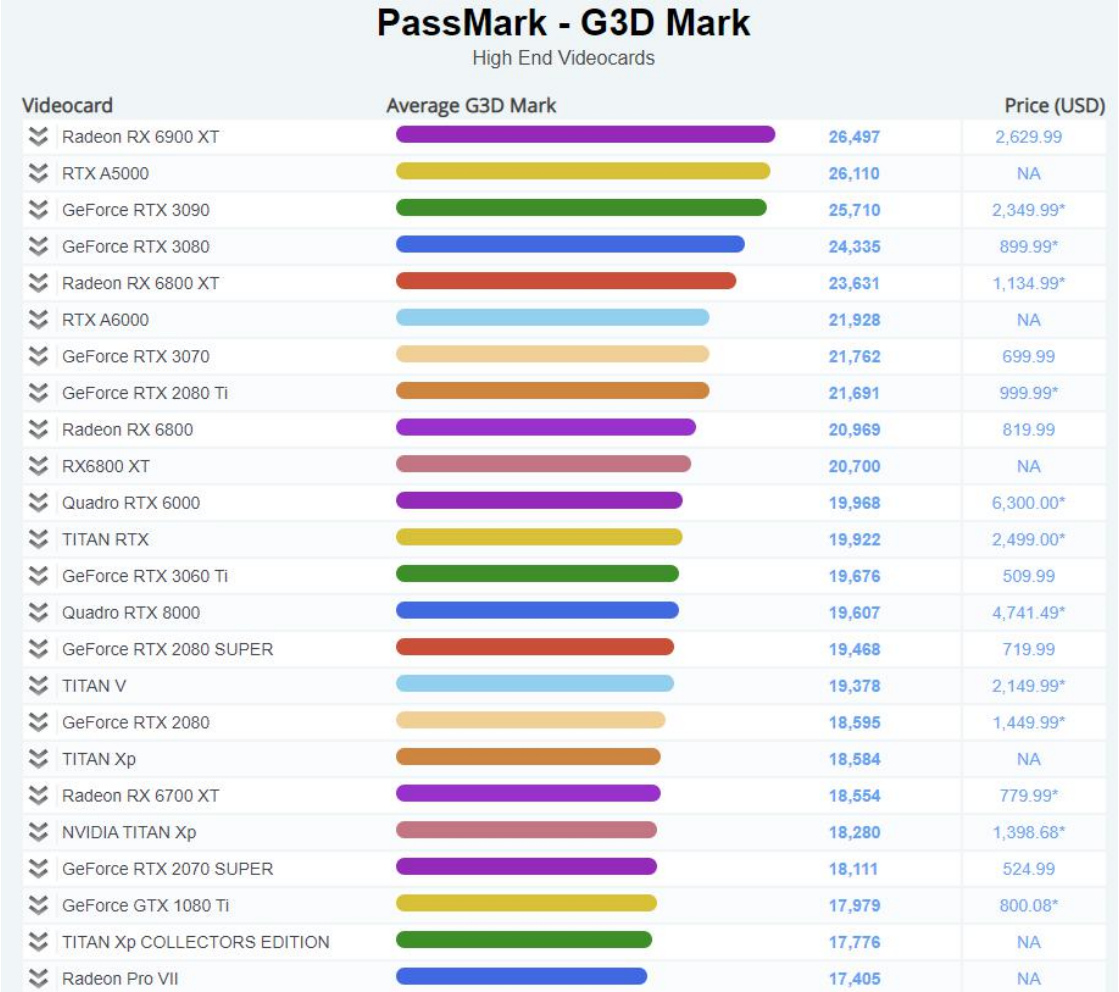


图 3-5 根据数据重建的折线图

§3.2.4 性能查询网站

从 PassMark Software - Video Card (GPU) Benchmarks - High End Video Cards (videocardbenchmark.net)中，可以较为便捷的得到显卡跑分数据。



PassMark - G3D Mark

High End Videocards

Videocard	Average G3D Mark	Price (USD)
Radeon RX 6900 XT	26,497	2,629.99
RTX A5000	26,110	NA
GeForce RTX 3090	25,710	2,349.99*
GeForce RTX 3080	24,335	899.99*
Radeon RX 6800 XT	23,631	1,134.99*
RTX A6000	21,928	NA
GeForce RTX 3070	21,762	699.99
GeForce RTX 2080 Ti	21,691	999.99*
Radeon RX 6800	20,969	819.99
RX6800 XT	20,700	NA
Quadro RTX 6000	19,968	6,300.00*
TITAN RTX	19,922	2,499.00*
GeForce RTX 3060 Ti	19,676	509.99
Quadro RTX 8000	19,607	4,741.49*
GeForce RTX 2080 SUPER	19,468	719.99
TITAN V	19,378	2,149.99*
GeForce RTX 2080	18,595	1,449.99*
TITAN Xp	18,584	NA
Radeon RX 6700 XT	18,554	779.99*
NVIDIA TITAN Xp	18,280	1,398.68*
GeForce RTX 2070 SUPER	18,111	524.99
GeForce GTX 1080 Ti	17,979	800.08*
TITAN Xp COLLECTORS EDITION	17,776	NA
Radeon Pro VII	17,405	NA

图 3-6 显卡跑分图

§3.3 数据预处理

在已经得到所需数据的情况下，需要对数据进行处理。数据预处理（data preprocessing）是指在主要的处理以前对数据进行的一些处理。现实世界中数据大体上都是不完整，不一致的脏数据，无法直接进行数据挖掘，或挖掘结果差强人意。为了提高数据挖掘的质量产生了数据预处理技术。　　数据预处理有多种方法：数据清理，数据集成，数据变换，数据归约等。这些数据处理技术在数据挖掘之前使用，大大提高了数据挖掘模式的质量，降低实际挖掘所需要的时间。

对于原始数据应主要从完整性和准确性两个方面去审核。完整性审核主要是检查应调查的单位或个体是否有遗漏，所有的调查项目或指标是否填写齐全。准确性审核主要是包括两个方面：一是检查数据资料是否真实地反映了客观实际情况，内容是否符合实际；二是检查数据是否有错误，计算是否正确等。审核数据准确性的方法主要有逻辑检查和计算检查。逻辑检查主要是审核数据是否符合逻辑，内容是否合理，各项目或数字之间有无相互矛盾的现象，此方法主要适合对

定性（品质）数据的审核。计算检查是检查调查表中的各项数据在计算结果和计算方法上有无错误，主要用于对定量（数值型）数据的审核。

§3.3.1 表格数据清理

在上文 3.2.2 章节中，采集了 NVIDIA 官网上给出的关于显卡硬件配置的数据，并整理成了一张表格。不难发现，表格中存在部分位置空缺。整理如下：

表 3-2 显卡缺失数据

名称	架构	CUDA 核心	加速频率	基准频率	内存速度	显存位宽	显存带宽	跑分	价格
GEFORCE GTX 1030 SDDR4	Pascal	384	1379	NaN	1	64	16.8	NaN	579
GEFORCE GTX 1030 GDDR5	Pascal	384	1468	NaN	3	64	48	2621	NaN
GEFORCE GTX 1050 2GB	Pascal	640	1455	1354	7	128	112	4461	NaN
GEFORCE GTX 1050 3GB	Pascal	768	1581	1392	7	96	84	5229	NaN
GEFORCE GTX 1080 TI	Pascal	3584	1582	NaN	11	352	484	17925	6699
GEFORCE GTX TITAN	Kepler	NaN	NaN	NaN	NaN	NaN	NaN	8528	2099

可以发现，存在部分数据的缺失，这些数据在 NVIDIA 官网上并没有被列出。由于显卡的数量并不是很多，所以不选择直接抛弃这些数据的全部，采用补足策略。

对基准频率或加速频率其一的缺失，直接使用没有缺失的部分代替。观察到缺失基准/加速频率的 GPU 均为跑分较低或较高的，性能较低的 GPU 视作缺失动态加速功能，跑分较高的视作散热或供电不足以支持更高的核心频率。

对于跑分缺失，整张表格只有 GTX1030SDDR4 出现这一情况。显卡本身的持有量很少，缺乏足够的测评。这里使用 GTX1030GDDR5 的跑分*CUDA 核心数量比来代替。

对于 GTXTITAN，缺失的数据过多，跳过配置-性能阶段的分析，利用存在的性能-价格数据参与性能-价格模型的建立。

§3.3.2 显卡价格数据清理

如上文图 3-5 中，因为电商不时的促销行为，得到的数据会有许多不在主趋势线上的数据。但是显卡在当天的价格下降客观存在，因此不能直接讲这些离群点直接删除。由于本项目目的在于获取两者之间的关系，因而可以认为短时间的、高频率的数据并不重要。可以通过一个一维卷积将这些短期趋势合并到长期趋势中，柔化得到的折线。在这里使用的卷积核为 u(10)-u(0)，u 为单位阶跃函数。

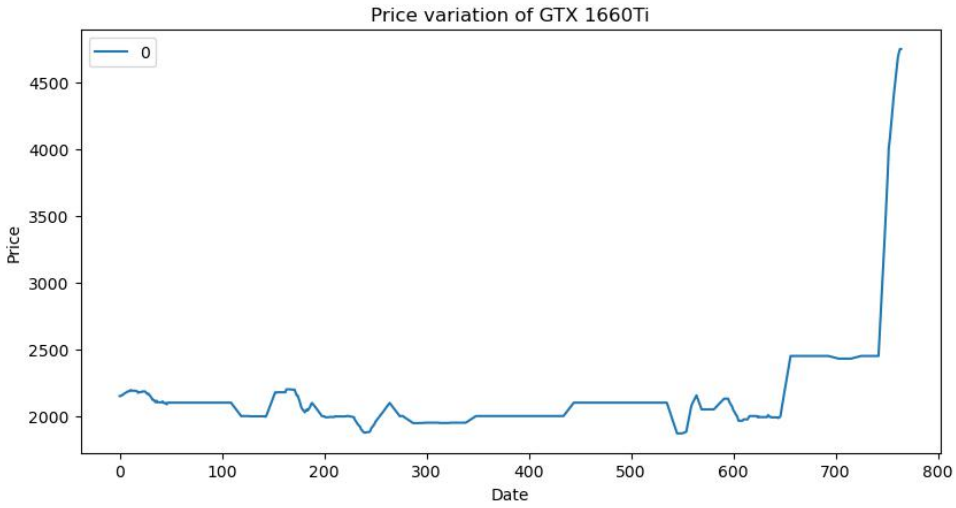


图 3-7 卷积处理后的折线图

§3.4 硬件-性能分析

数据分析是本章节重点。本节通过数据分析，将会给出显卡的基本价格模型，探究各个要素在决定显卡价格上所起的作用。

§3.4.1 数值相关性分析

在统计学中，交叉表是矩阵格式的一种表格，显示变量的多变量频率分布。交叉表被广泛用于调查研究，商业智能，工程和科学研究。它们提供了两个变量之间的相互关系的基本画面，可以帮助他们发现它们之间的相互作用。卡尔·皮尔逊首先在“关于应变的理论及其关联理论与正常相关性”中使用了交叉表。

多元统计学的一个关键问题是找到高维应变表中包含的变量的（直接）依赖结构。如果某些有条件的独立性被揭示，那么甚至可以以更智能的方式来完成数据的存储。为了做到这一点，可以使用信息理论概念，它只能从概率分布中获得信息，这可以通过相对频率从交叉表中容易地表示。

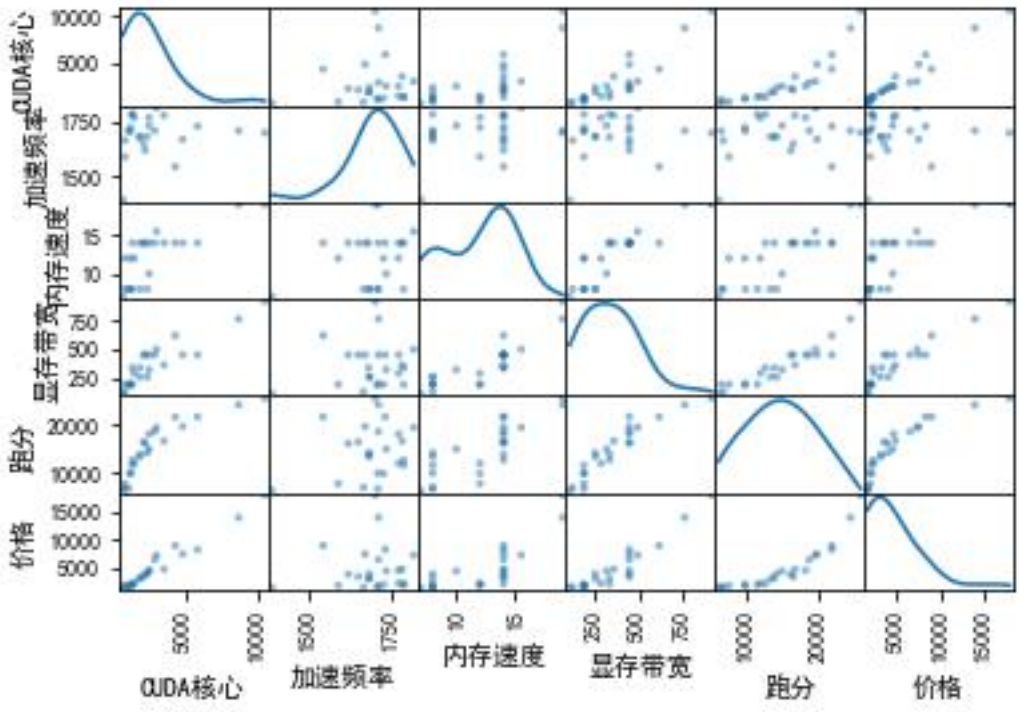


图 3-8 显卡数据交叉表

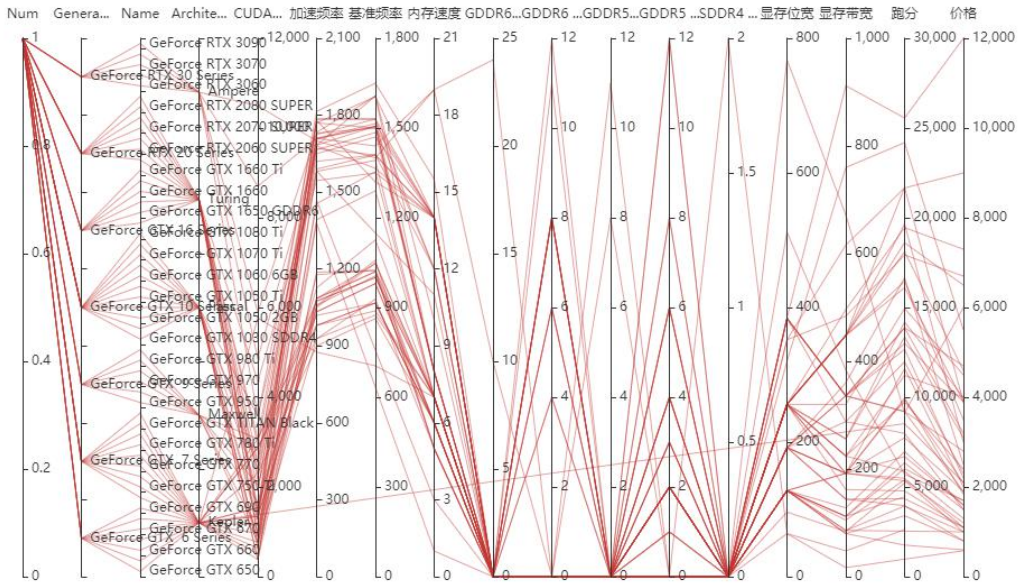


图 3-9 显卡数据平行表

上图列举了部分采集到的影响显卡价格的硬件相关信息，以及他们互相之间的关系。从上图中可以粗略地得出：显卡的价格和显卡的 CUDA 核心数量、显卡加速频率、显存速度、显存带宽、显卡本身的跑分都有一定的关系，其中，CUDA 核心数量、显存带宽、显卡跑分呈现正相关，显卡的加速频率以及显存带宽虽然不直接影响显卡价格，但是决定显卡价格的上界。

针对所列出的全部显卡进行交叉表分析后，我们也可针对每一世代的显卡进行类似的分析，得到不同世代显卡的共性与不同点。

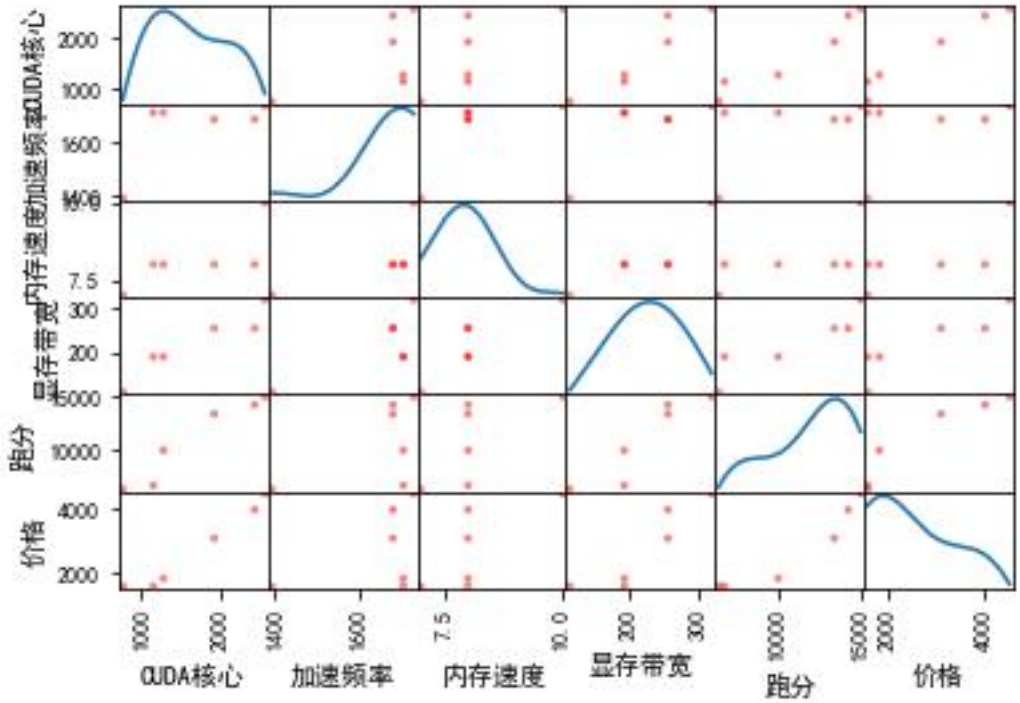


图 3-10 10 系显卡(Pascal 架构)数据交叉表

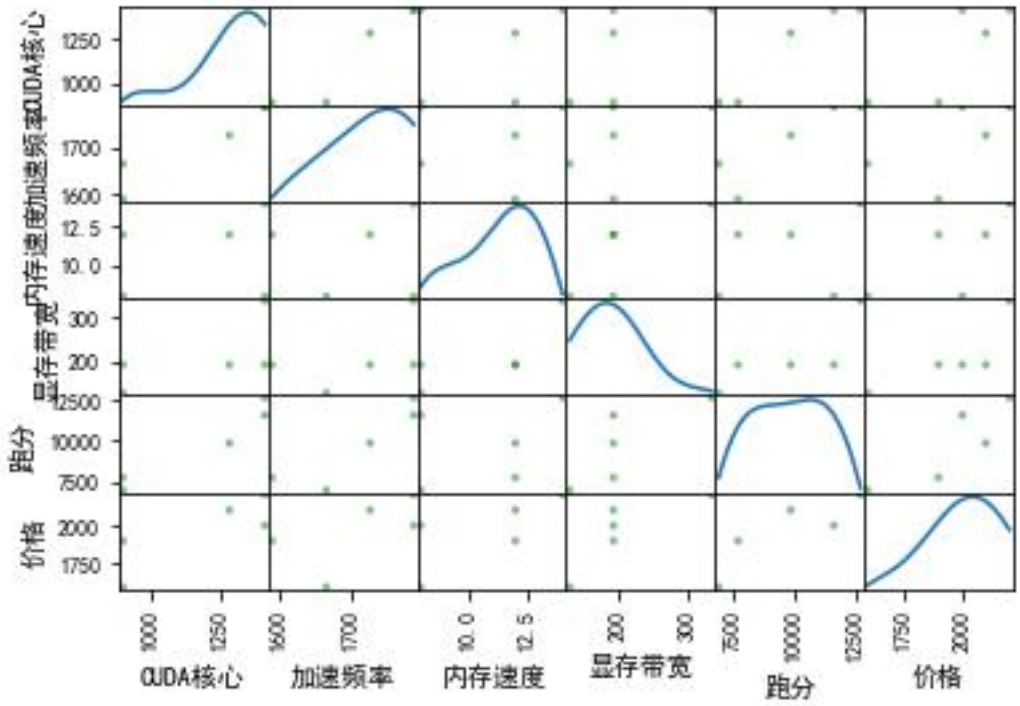


图 3-11 16 系显卡(Turing 架构)显卡数据交叉表

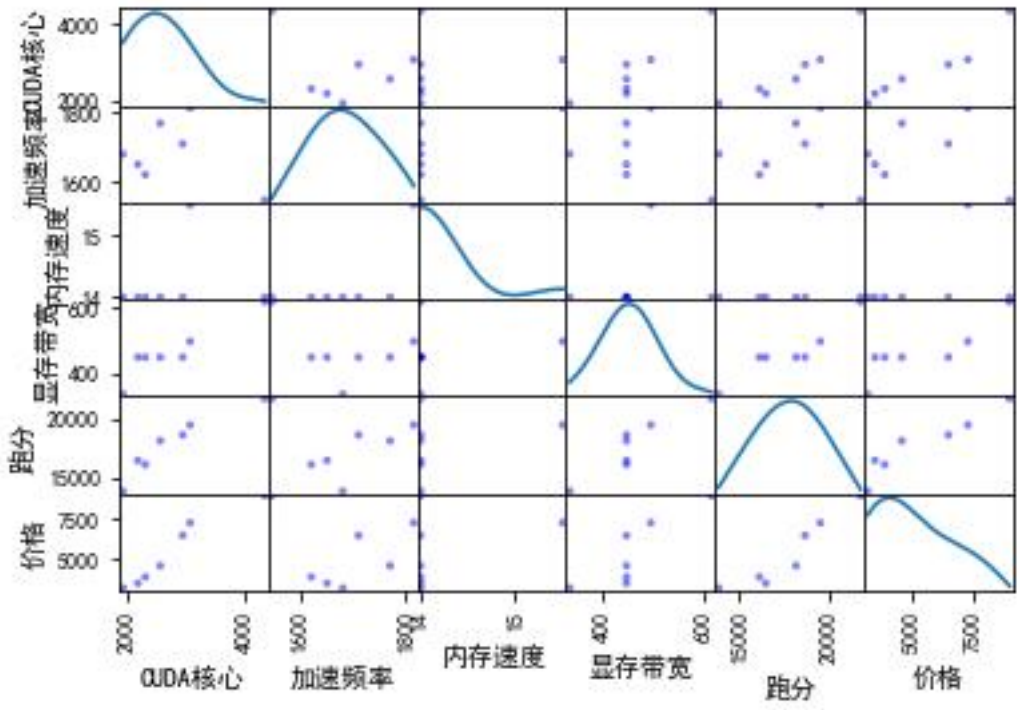


图 3-12 20 系显卡(Turing 架构)数据交叉表

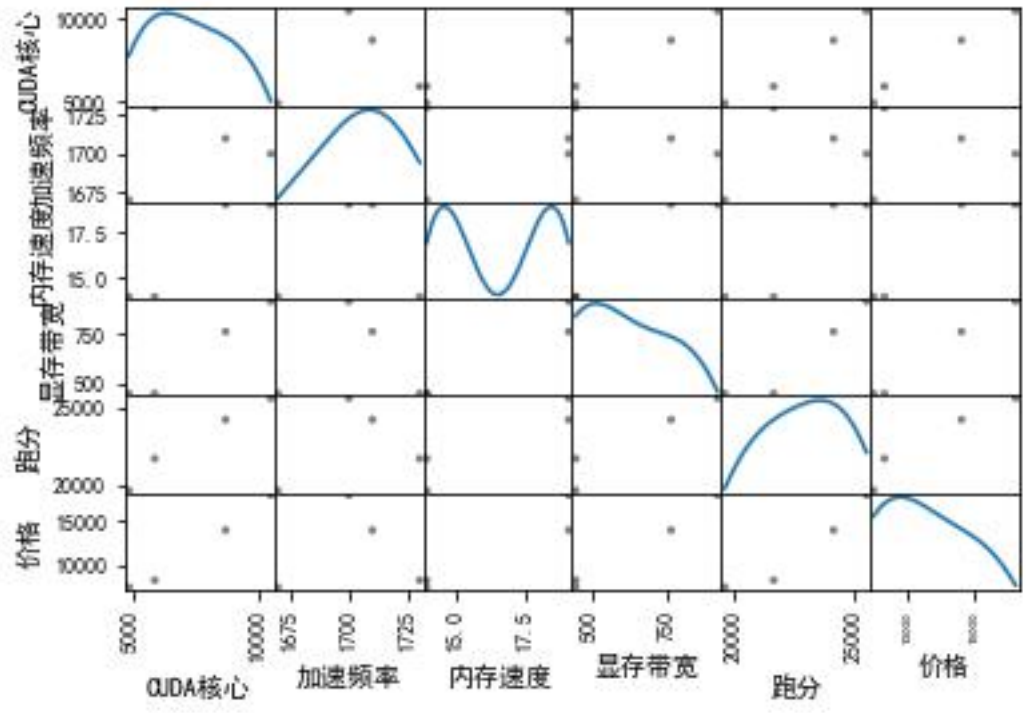


图 3-13 30 系显卡(Ampere 架构)数据交叉表

从图中，我们可以很容易的看出在总表中，数据项之间的关系依然被保留。不同点在于，不同世代的显卡的内存速度、显存带宽等值得分布不同，同一世代的显卡，取值固定在几个相同的离散值。越新的显卡，这些离散的数值越大，反应显卡性能越高。

§3.4.2 数值分组分析

继续上一小节的内容，根据交叉表分析得出的结论，我们了解到不同显卡世代存在着数值上的差异。在这一节中，我们将把一个世代的显卡作为一个整体进行分析。对每一世代的显卡，分别根据其硬件信息，绘制相应的箱型图，得到其中的差异以及发展趋势。

自然的，首先我们将把一个系列(Series)的显卡作为一个类别进行分析。

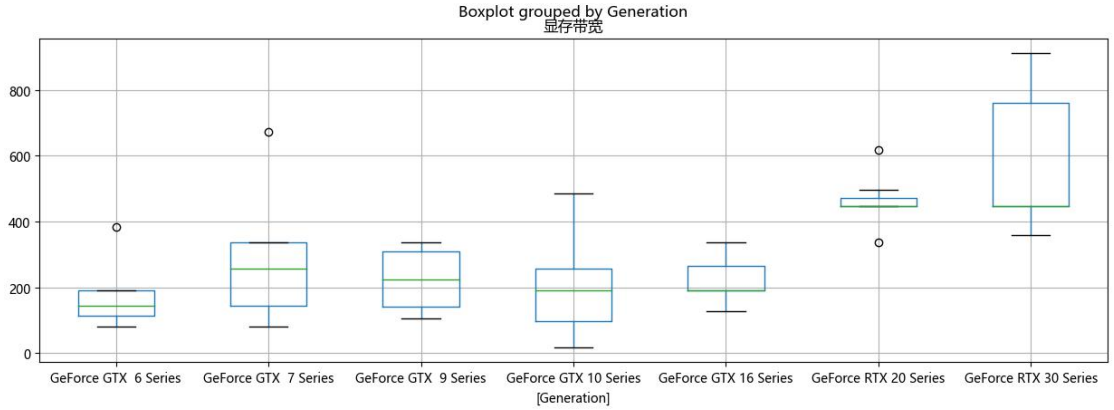


图 3-14 不同系列显卡显存带宽对比

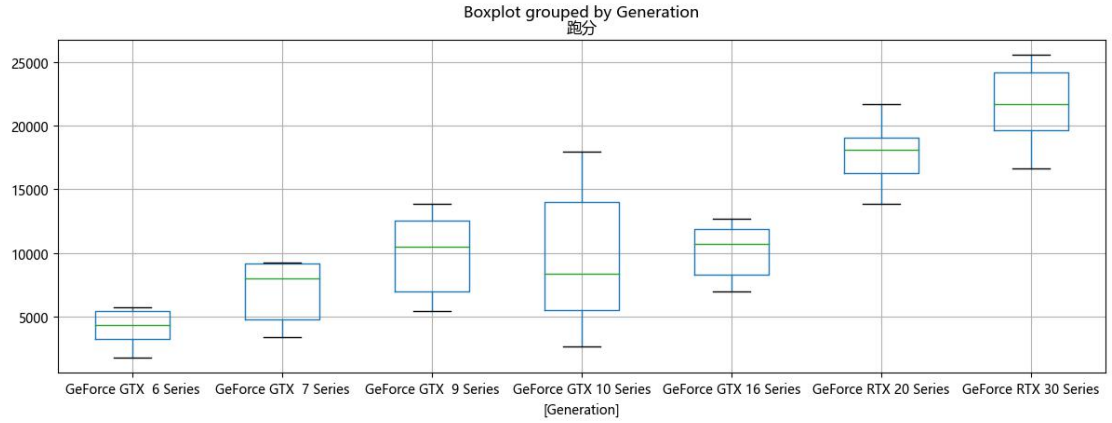


图 3-15 不同系列显卡跑分对比

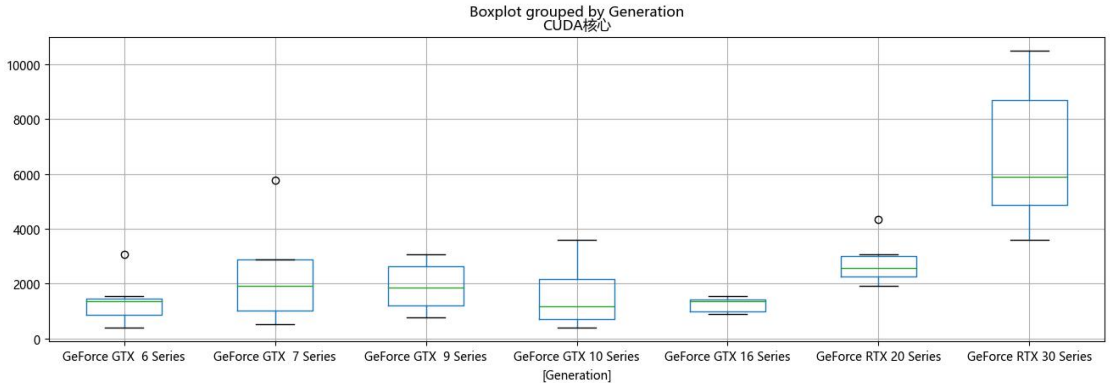


图 3-16 不同系列显卡 CUDA 核心数量对比

由得到的箱型图可以证明，不同世代的显卡之间确实存在着一定的硬件与性

能差异。然而上图也表现出了趋势的不定，代与代之间不总是产生一种显著的差异，甚至没有体现出单调上升的特性。这是有原因的。查阅资料可以得到 GTX16 系显卡与 RTX20 系显卡发售时间比较接近，可以近似的看做两者位于同一世代，RTX20 系列是 GTX16 系列的高位版本,且两者同为 Turing 架构。

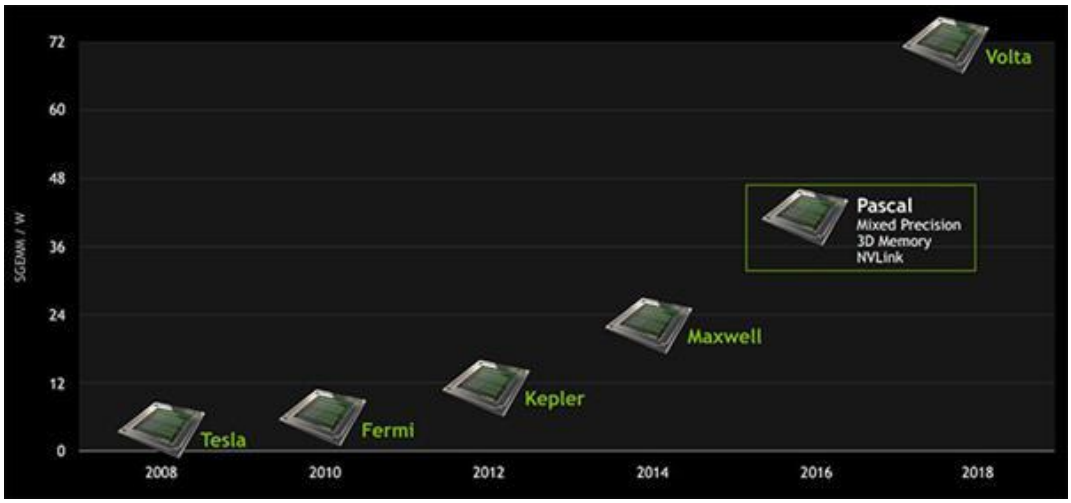


图 3-17 NVIDIA 显卡架构发展历史(2008-2018)

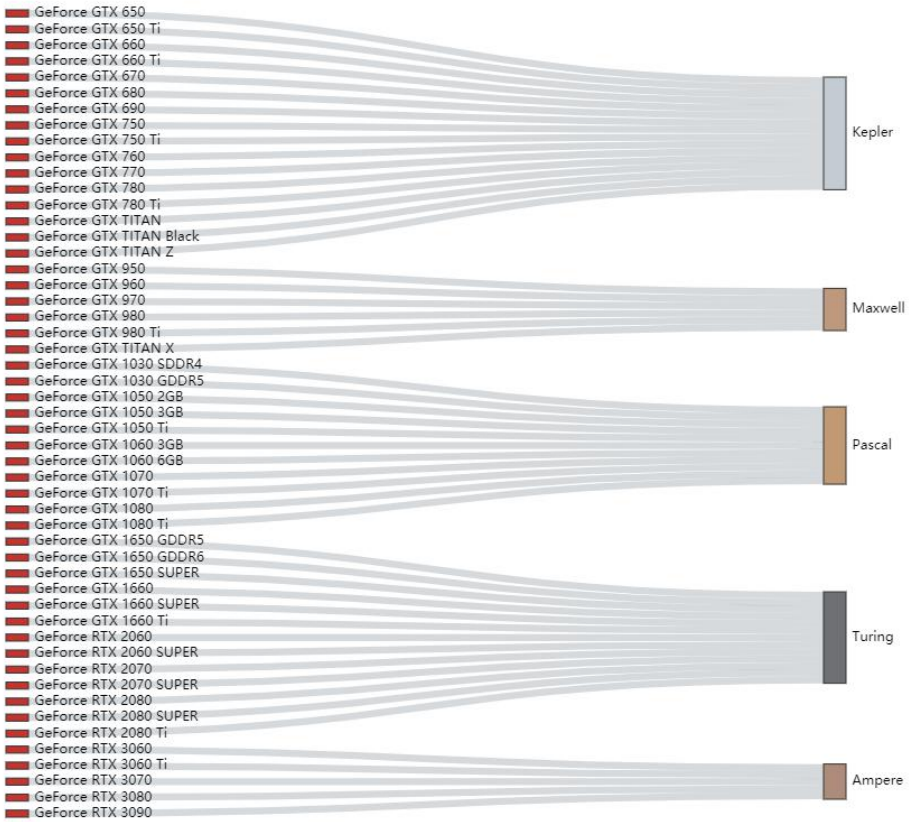


图 3-18 NVIDIA 显卡型号与架构对应图

因此我们对数据进行修正，不再采用代数作为分组关键词，而是采用架构作

为关键词。得到的结果如下：

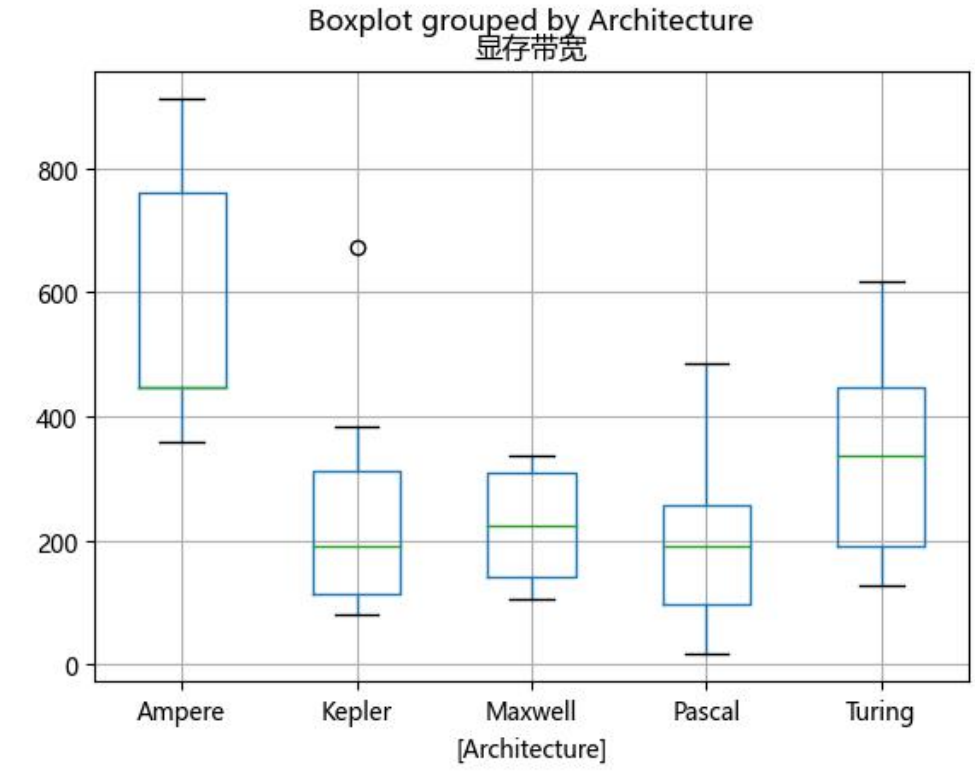


图 3-19 不同架构显卡显存带宽对比

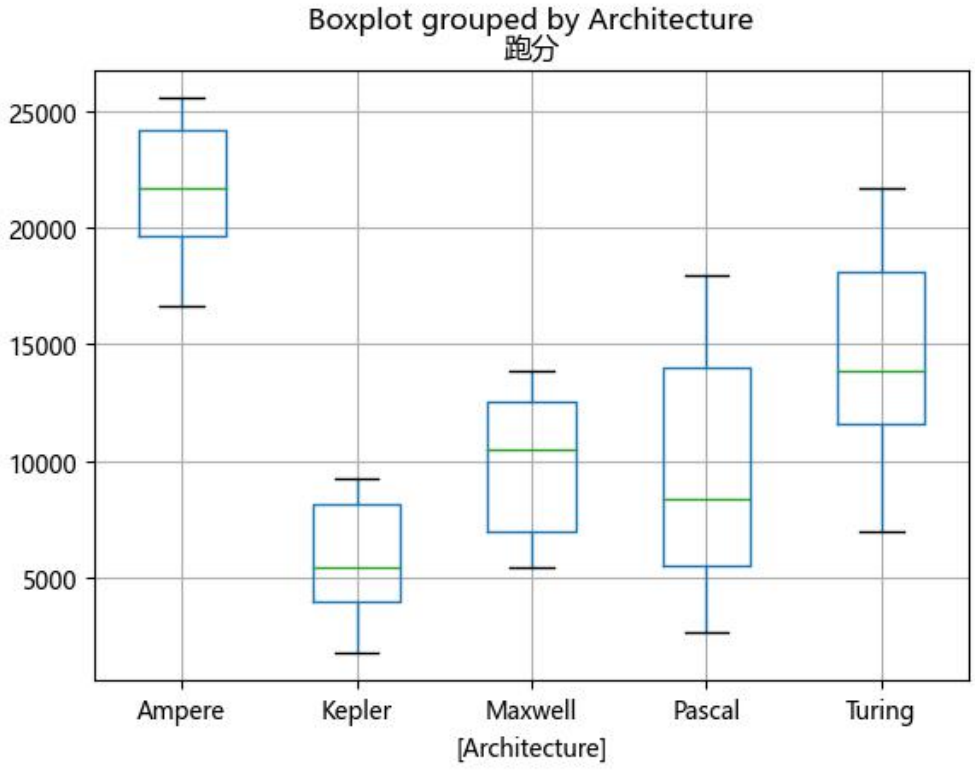


图 3-20 不同架构显卡跑分对比

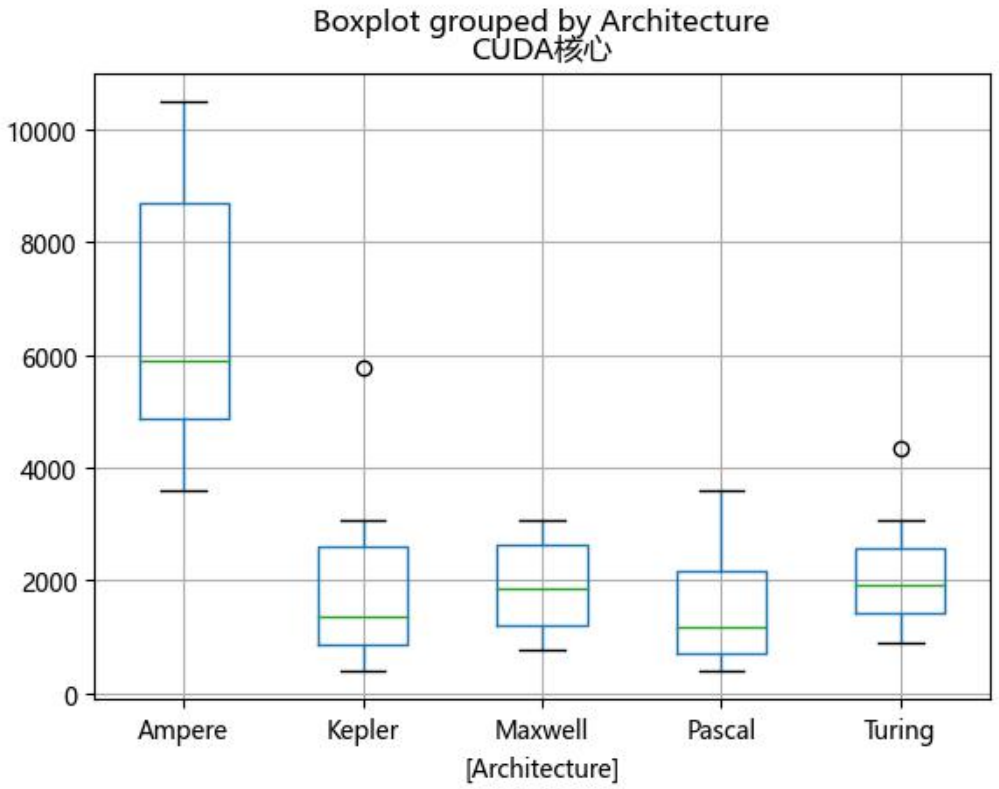


图 3-21 不同架构显卡 CUDA 核心数量对比

需要注意，NVIDIA 的显卡架构为：Tesla→Fermi→Kepler→Maxwell→Pascal→Turing→Ampere。上面图 3-16 到图 3-18 中并未严格按照时间顺序排序。

从上图中，我们可以更明确的得出结论：随着显卡架构的提升，CUDA 核心数量除最新的 Ampere 架构外不发生大的变化，显存带宽缓慢增长，跑分稳步提升。显卡架构是衡量显卡的一大重要指标，更好的架构（包括了更先进的制程）是推动显卡性能发展的最大驱动力。

§3.4.3 硬件定量分析

在完成定性分析后，我们借助回归进行定量分析。

线性回归是利用数理统计中回归分析，来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法，运用十分广泛。其表达形式为 y = w'x+e，e 为误差服从均值为 0 的正态分布。回归分析中，只包括一个自变量和一个因变量，且二者的关系可用一条直线近似表示，这种回归分析称为一元线性回归分析。如果回归分析中包括两个或两个以上的自变量，且因变量和自变量之间是线性关系，则称为多元线性回归分析。

读取已有的数据，进行线性回归分析,即可得到各变元对结果的贡献。结果如下：

-21704.279346231735 [1.13436531e+00 4.84316229e+00 1.81041942e+00 1.17413241e+03 -3.69098688e+02 1.48418891e+02 4.09481917e+02 7.10583827e+02 0.00000000e+00 2.90022674e+01]

Predict score is 0.9221450427992408

实际情况下，考虑到显卡性能与硬件不一定是线性关系，所以不使用线性回归模型建立最终模型。考虑非线性回归模型。

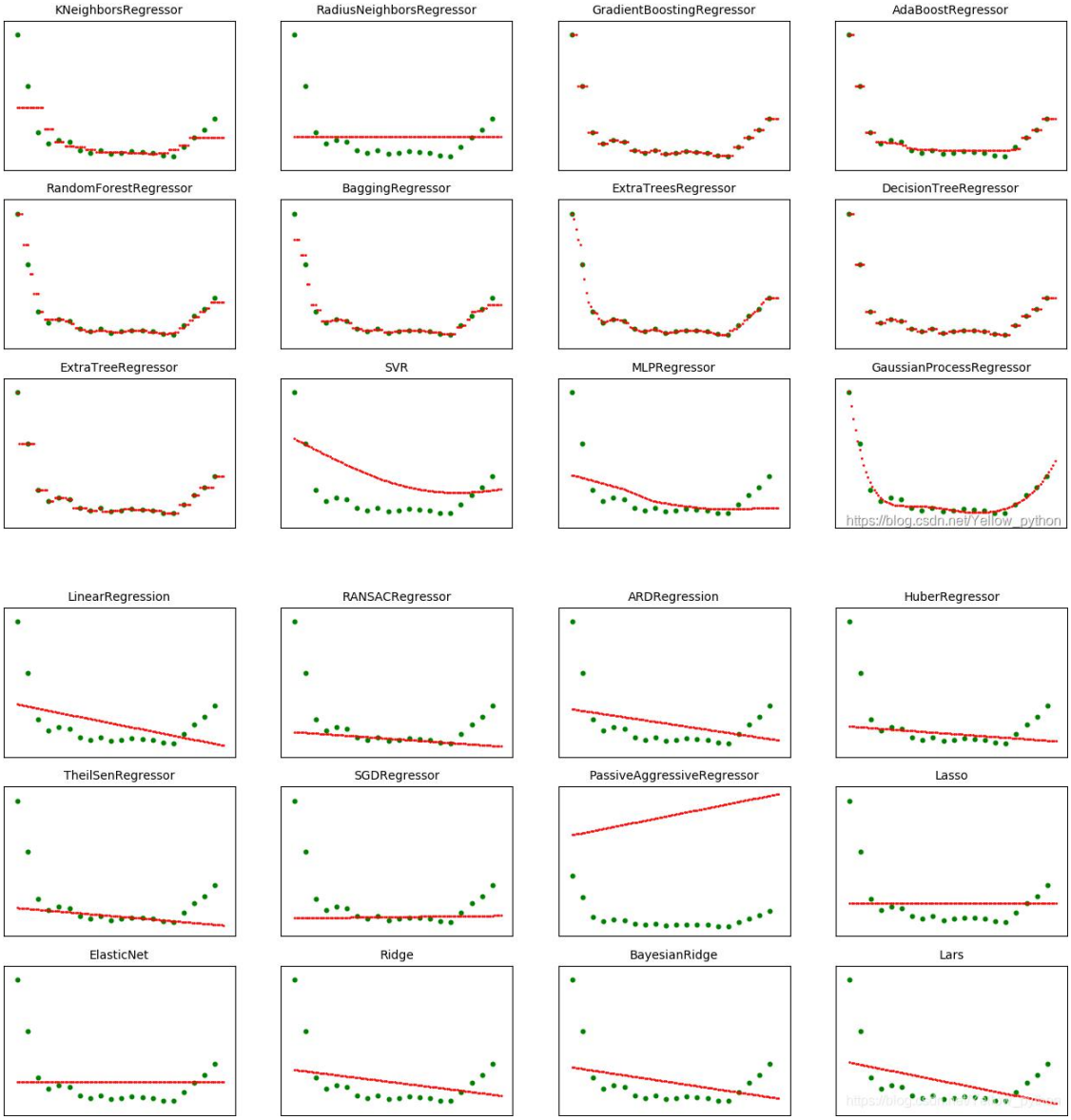


图 3-22 sklearn 提供的回归模型

（1）随机森林（Random Forests）

随机森林是一种重要的基于 Bagging 的集成学习方法，可以用来做分类、回归等问题。

随机森林由 LeoBreiman（2001）提出，从原始训练样本集 N 中有放回地重复随机抽取 k 个样本生成新的训练样本集合，然后根据自助样本集生成 k 个分类

树组成随机森林，新数据的分类结果按分类树投票多少形成的分数而定。其实质是对决策树算法的一种改进，将多个决策树合并在一起，每棵树的建立依赖于一个独立抽取的样品，森林中的每棵树具有相同的分布，分类误差取决于每一棵树的分类能力和它们之间的相关性。特征选择采用随机的方法去分裂每一个节点，然后比较不同情况下产生的误差。能够检测到的内在估计误差、分类能力和相关性决定选择特征的数目。单棵树的分类能力可能很小，但在随机产生大量的决策树后，一个测试样品可以通过每一棵树的分类结果经统计后选择最可能的分类。

随机森林有许多优点：具有极高的准确率,随机性的引入，使得随机森林不容易过拟合,随机性的引入，使得随机森林有很好的抗噪声能力,能处理很高维度的数据，并且不用做特征选择,既能处理离散型数据，也能处理连续型数据，数据集无需规范化,训练速度快，可以得到变量重要性排序,容易实现并行化

随机森林的缺点：当随机森林中的决策树个数很多时，训练时需要的空间和时间会较大,随机森林模型还有许多不好解释的地方，是个黑盒模型。

（2）极端随机森林

极端随机森林同样是一种多棵决策树集成的分类器，与随机森林分类器比较，主要有两点不同：对于每个决策树的训练集，RF 采用的是随机采样 bootstrap 来选择采样集作为每个决策树的训练集，而 extra tree 一般不采用随机采样，即每个决策树采用原始训练集。Random Forest 应用的是 Bagging 模型，Extra Tree 使用的所有的样本，只是特征是随机选取的，因为分裂是随机的，所以在某种程度上比随机森林得到的结果更加好。在选定了划分特征后，RF 的决策树会基于信息增益，基尼系数，均方差之类的原则，选择一个最优的特征值划分点，这和传统的决策树相同。但是 Extra tree 比较的激进，会随机的选择一个特征值来划分决策树。从第二点可以看出，由于随机选择了特征值的划分点位，而不是最优点位，这样会导致生成的决策树的规模一般会大于 RF 所生成的决策树。也就是说，模型的方差相对于 RF 进一步减少，但是 bias 相对于 RF 进一步增大。在某些时候，Extra tree 的泛化能力比 RF 更好。

（3）锑度回升

梯度提升回归树是一种从它的错误中进行学习的技术。它本质上就是集思广益，集成一堆较差的学习算法进行学习。

GBDT 是基于 Boosting 思想的，Adaboosting 是最著名的 Boosting 算法，其基本思想是使用多个弱分类器来构建一个强分类器。Adaboosting 构造方法是一个迭代的过程，大致思路是：针对同一个训练集训练多层的弱分类器，每层使用训练集训练一个弱分类模型，我们从训练出的模型中得到预测结果。之后根据训练集

中样本分类是否正确、总体分类的准确率来确定每个样本上应重新分配的权值，将修改过权重后的新数据集训练一个下层的分类器。这样不断进行训练直到有很少的错分样本，最后将每层的分类器有权重分配的融合在一起，这样下来就组成了最终的决策分类器。

每次迭代需要三次计算。

第一次计算分类误差率：

$$e_m = P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i)$$

第二次计算分配给当前分类器的系数：

$$\alpha_m = \frac{1}{2}\log\frac{1-e_m}{e_m}$$

第三个是计算新的数据集的权值，其中 Zm 为规划因子：

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2}, \ldots, w_{m+1,N})$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} e^{-\alpha_m y_i G_m(x_i)}$$

$$Z_m = \sum_{i=1}^{N} W_{mi} e^{-\alpha_m y_i G_m(x_i)}$$

最终得到将所有层分类器的组合：

$$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$$

对所采集数据进行上述三种回归分析，评估性能。

随机森林回归的默认评估值为： 0.9383967247806242

随机森林回归的 R_squared 值为： 0.9383967247806242

随机森林回归的均方误差为: 1211000.6066666667

随机森林回归的平均绝对误差为: 738.4999999999999

极端随机森林回归的默认评估值为： 0.978700394240416

极端随机森林回归的 R_squared 值为： 0.9383967247806242

极端随机森林回归的均方误差为: 1211000.6066666667

极端随机森林回归的平均绝对误差为: 738.4999999999999

梯度提升回归的默认评估值为： 0.9767755934882469

梯度提升回归的 R_squared 值为： 0.9383967247806242

梯度提升回归的均方误差为: 1211000.6066666667

梯度提升回归的平均绝对误差为: 738.4999999999999

故在建立模型的过程中，采用极端随机森林回归算法或梯度提升回归算法效果较好。

§3.5 性能-价格分析

在已经建立显卡硬件-性能映射之后，我们接着进行显卡的性能-价格模型。

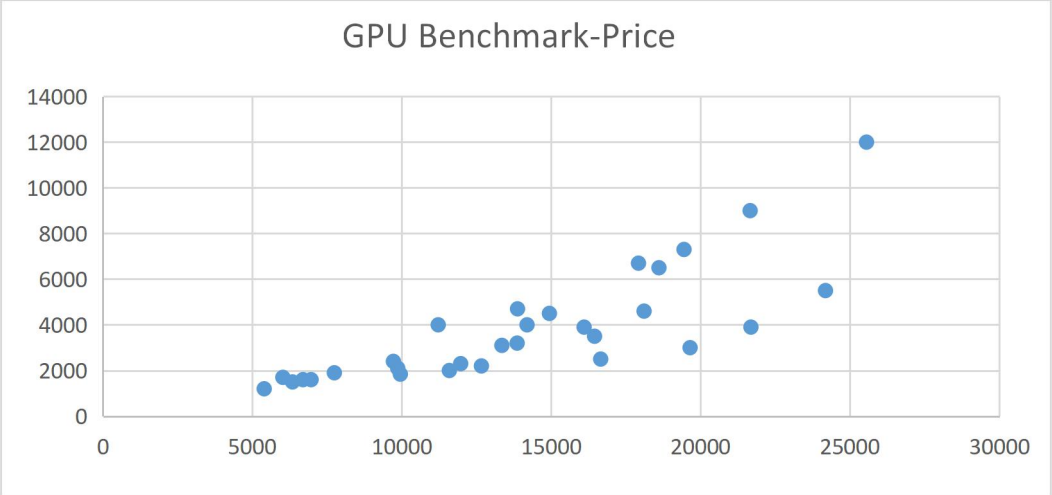


图 3-23 显卡跑分-价格散点图

§3.5.1 聚类分析

聚类分析指将物理或抽象对象的集合分组为由类似的对象组成的多个类的分析过程。它是一种重要的人类行为。聚类分析的目标就是在相似的基础上收集数据来分类。聚类源于很多领域，包括数学，计算机科学，统计学，生物学和经济学。在不同的应用领域，很多聚类技术都得到了发展，这些技术方法被用作描述数据，衡量不同数据源间的相似性，以及把数据源分类到不同的簇中。

k 均值聚类算法（k-means clustering algorithm）是一种迭代求解的聚类分析算法，其步骤是，预将数据分为 K 组，则随机选取 K 个对象作为初始的聚类中心，然后计算每个对象与各个种子聚类中心之间的距离，把每个对象分配给距离它最近的聚类中心。聚类中心以及分配给它们的对象就代表一个聚类。每分配一个样本，聚类的聚类中心会根据聚类中现有的对象被重新计算。这个过程将不断重复直到满足某个终止条件。终止条件可以是没有（或最小数目）对象被重新分配给不同的聚类，没有（或最小数目）聚类中心再发生变化，误差平方和局部最小。

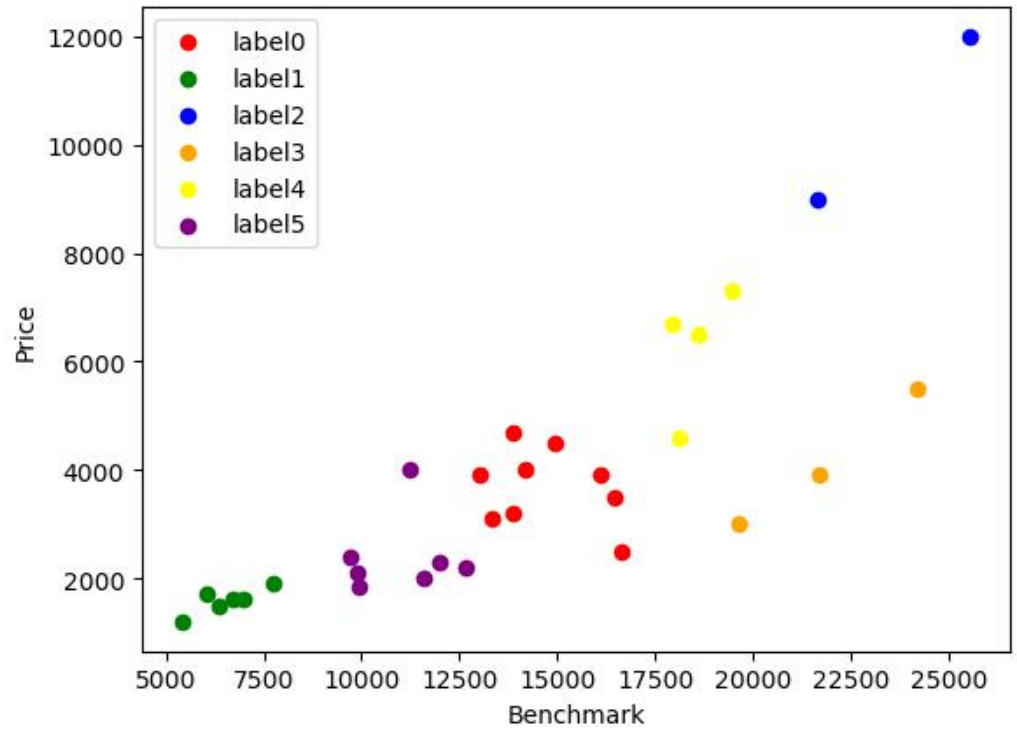


图 3-24 显卡 k-means 分类(取 k=6)

经反复实验，发现取 k=6 时得到的聚类效果最好。须注意因为我们建立的是性能-价格模型，在这里跑分为横坐标，价格为纵坐标。图标原点不是单位原点，但任然可以近似地认为斜率代表性价比。斜率越小，性价比越高。

由聚类得到的显卡分类结果，我们可以得到显卡的六分类标准：

第一类：低价格，低性能，性价比一般。称这种显卡为入门级显卡，以 NVIDIA 命名体系一般以 10、20 或 30 结尾，性能仅持平或略高于核芯显卡。图中以 label1 标定。

第二类：较低性能，较低价格，性价比较高。称这种显卡为甜点级显卡，以 NVIDIA 命名体系一般以 50 或 60 结尾。此类显卡是显卡消费主力之一，能够支持 3D 图像应用的基本流畅运行，足够支持基本的图片编辑、视频剪辑任务。图中以 label5 标定。

第三类：中等性能，中等价格，性价比一般。称这种显卡为主流级显卡，也是显卡的主要消费类别。按 NVIDIA 命名体系，一般以 70、部分以 Ti 和 SUPER 结尾。足够支持市面 3D 应用于游戏的完全的、流畅的运行。足够支持图片编辑、视频剪辑任务，图中以 label0 标定。

第四类：高性能，高价格，较低性价比。称这种显卡为发烧级显卡，按 NVIDIA 命名体系以 80、部分 Ti 后缀，部分 TITAN 前缀。对本世代几乎所有应用都有良好的性能支持，甚至在后一世代应用推出后依然不会失去流畅运行的能力。图中

以 label4 标定。

第五类：高性能，中等价格，一般性价比。同上文第四类十分相似，但一般从别的显卡升级而来，且拥有更先进的制程与架构。图中以 label3 标定。

第六类：超高性能，超高价格，较低性价比。称这种显卡为旗舰级显卡，代表当前代所能达到的 GPU 最高水平。拥有最高的性能密度。

§3.5.2 线性与非线性回归分析

探究显卡的性能与价格之间的关系，可以转化为如何由显卡性能获得显卡的价格。我们可以在前文基础上，进行一元回归分析，获得两者之间的联系。

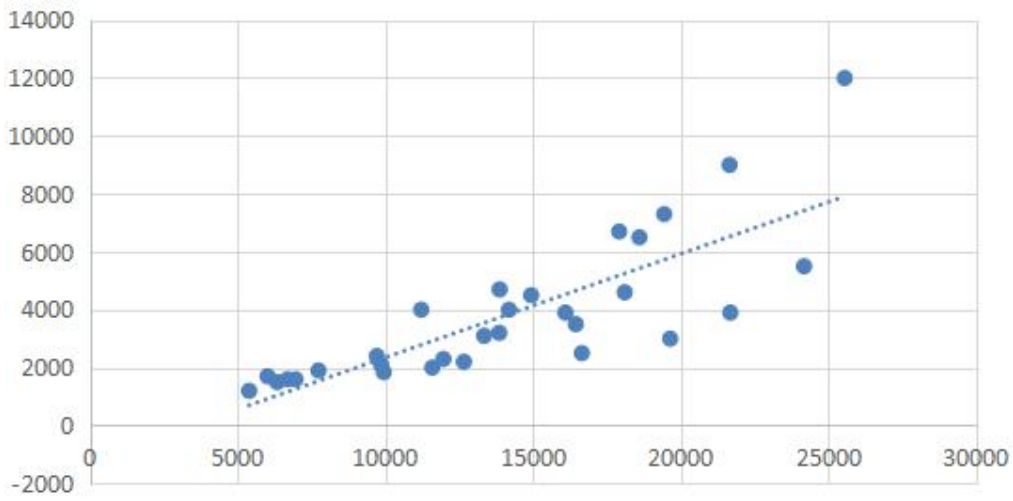


图 3-25 线性回归

线性回归公式为：

$$y = 0.3579x - 1233.5, \quad R^2 = 0.6504$$

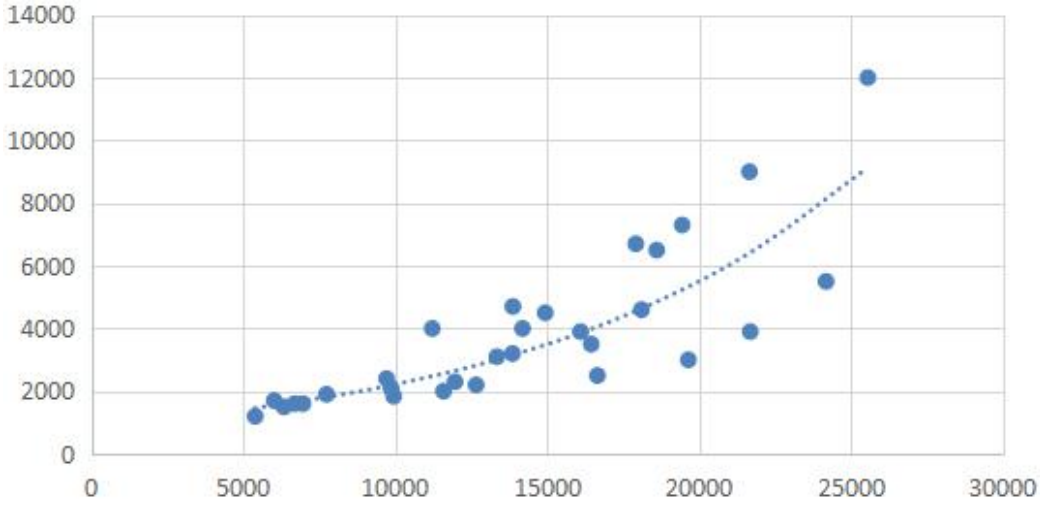
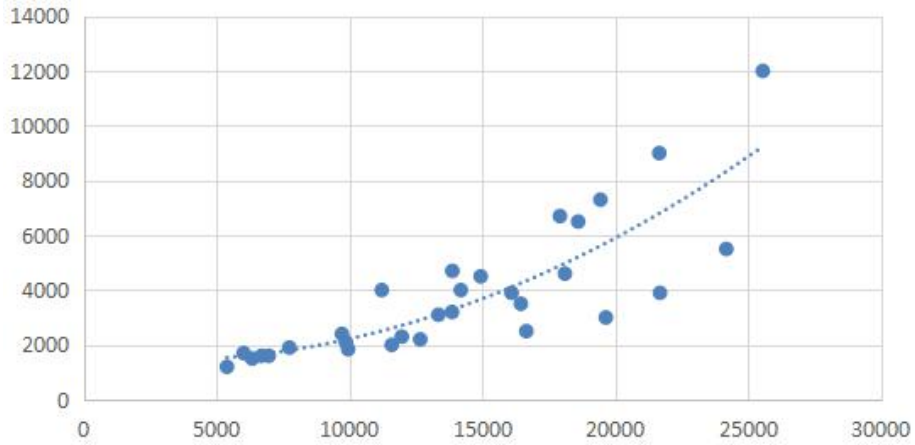


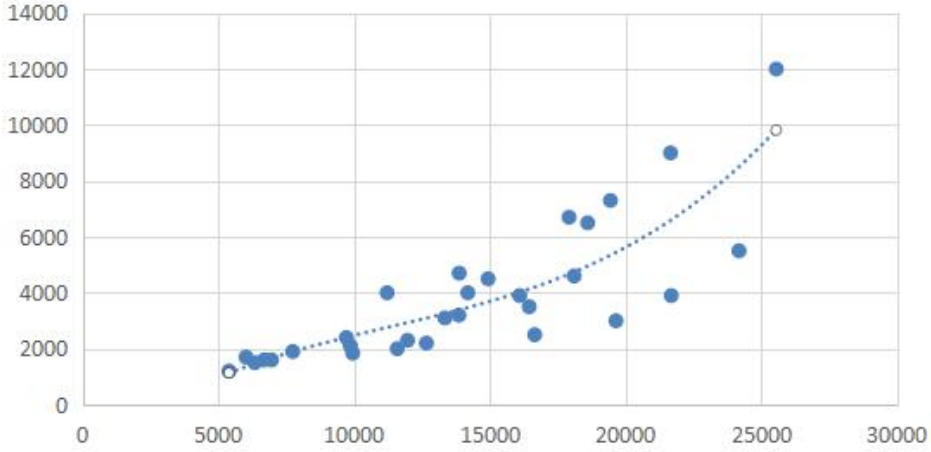
图 3-26 指数回归

指数回归公式为：

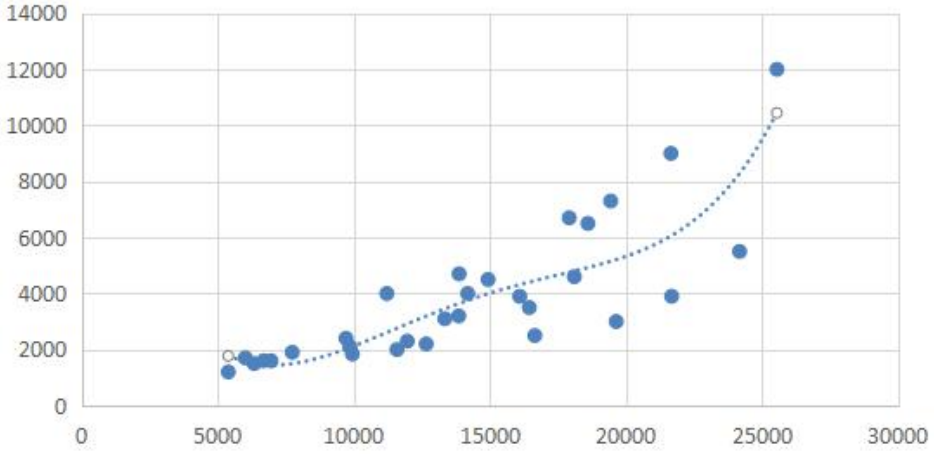
$$y = 892.15e^{9E-05x}, \quad R^2 = 0.7668$$



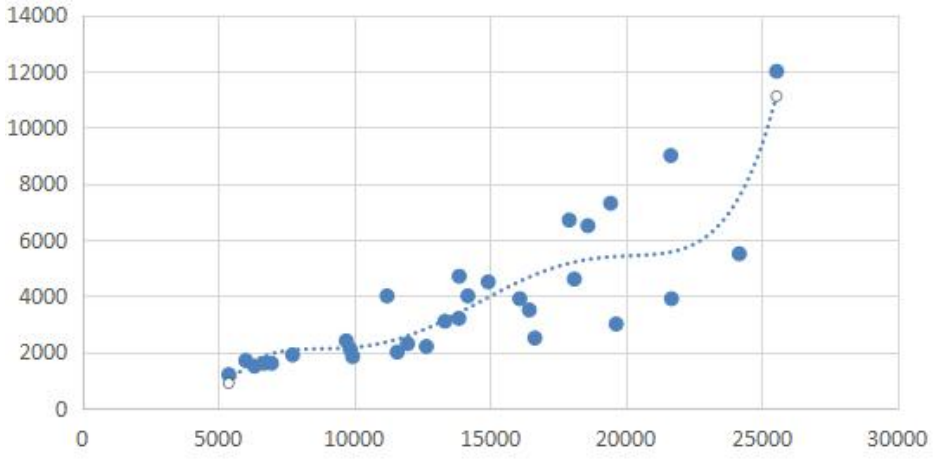
(a)



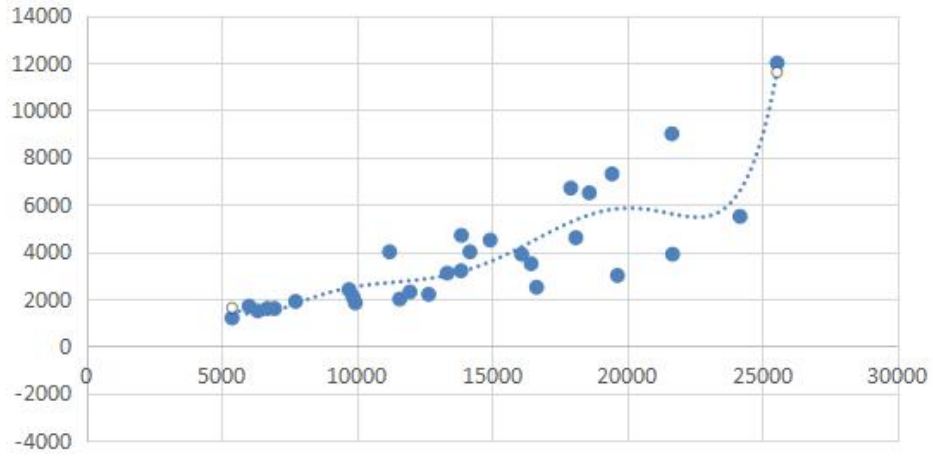
(b)



(c)



(d)



(e)

图 3-27 多项式回归

多项式回归公式为：

$$s = 2 \quad y = 1E-05x^2 - 0.0816x + 1537.6, \quad R^2 = 0.6895$$

$$s = 3 \quad y = 1E-09x^3 - 5E-05x^2 + 0.7472x - 1790.1, \quad R^2 = 0.6980$$

$$s = 4 \quad y = 4E-13x^4 - 2E-08x^3 + 0.0005x^2 - 3.7117x + 11566$$
$$R^2 = 0.6980$$

$$s = 5 \quad y = 9E-17x^5 - 7E-12x^4 + 2E-07x^3 - 0.0024x^2 + 14.547x - 32185, \quad R^2 = 0.7393$$

重新对之前数据分系列进行回归分析。得到

GTX 9 系列：

$y = -2E-12x^4 + 6E-08x^3 - 0.0008x^2 + 4.4346x - 8516.2 \quad R^2 = 0.9373$

GTX 10 系列：

$y = 4E-05x^2 - 0.5923x + 3551.3, \quad R^2 = 0.9939$

GTX 16 系列：

$y = -3E-12x^4 + 1E-07x^3 - 0.002x^2 + 14.341x - 36270$ $R^2 = 0.878$

RTX 20 系列：

$y = -3E-08x^3 + 0.0015x^2 - 27.417x + 161478$ $R^2 = 0.9577$

GTX 16 + RTX 20 系列：

$y = 2E-09x^3 - 2E-05x^2 + 0.0763x + 1809.1$ $R^2 = 0.9655$

RTX 30 系列：

$y = 6E-08x^3 - 0.0037x^2 + 72.724x - 476290$ $R^2 = 0.9722$

§3.5.3 Keras 神经网络回归分析

在上一节中，采用的各种回归方式均不能达到良好的效果。为了更好的拟合模型，采用深度学习方法，利用 Keras 神经网络进行数据拟合。

TensorFlow™是一个基于数据流编程（dataflow programming）的符号数学系统，被广泛应用于各类机器学习（machine learning）算法的编程实现，其前身是谷歌的神经网络算法库 DistBelief 。Tensorflow 拥有多层级结构，可部署于各类服务器、PC 终端和网页并支持 GPU 和 TPU 高性能数值计算，被广泛应用于谷歌内部的产品开发和各领域的科学研究。

Keras 是一个用 Python 编写的高级神经网络 API，它能够以 TensorFlow, CNTK, 或者 Theano 作为后端运行。Keras 的核心数据结构是 model，一种组织网络层的方式。最简单的模型是 Sequential 顺序模型，它由多个网络层线性堆叠。

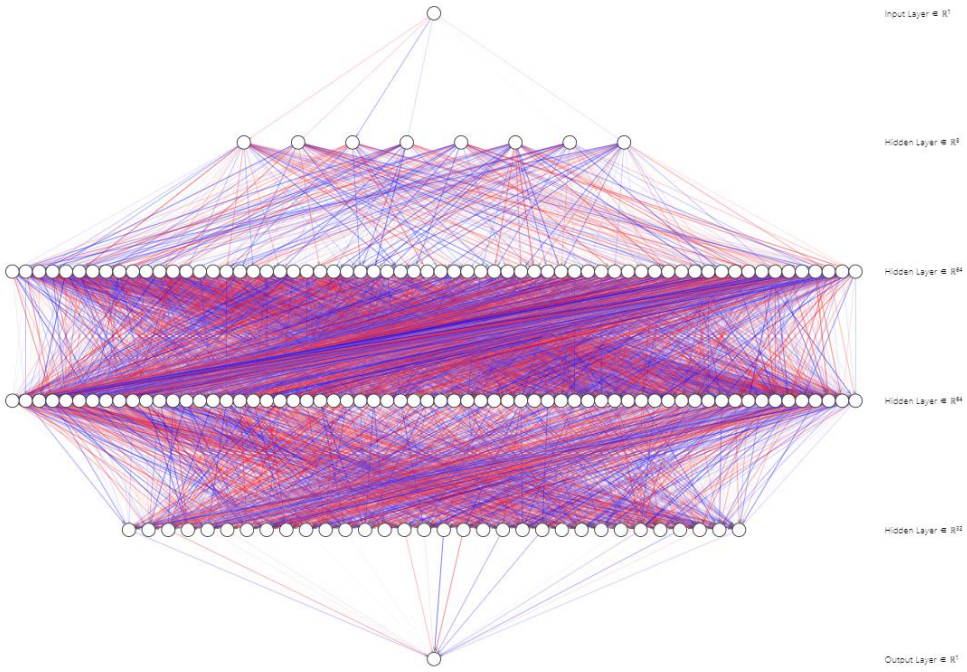


图 3-28 Keras 神经网络 Sequential 顺序网络模型

在 Sequential 顺序模型上使用采集的数据进行训练，使用高效的 ADAM 优化算法以及优化的最小均方误差损失函数。得到以下结果：

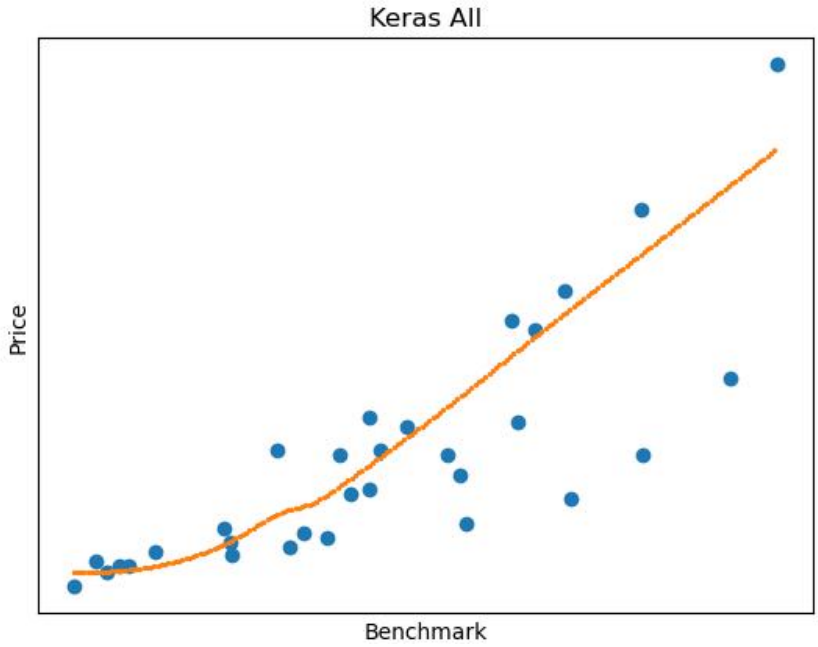


图 3-29 Keras 神经网络回归结果

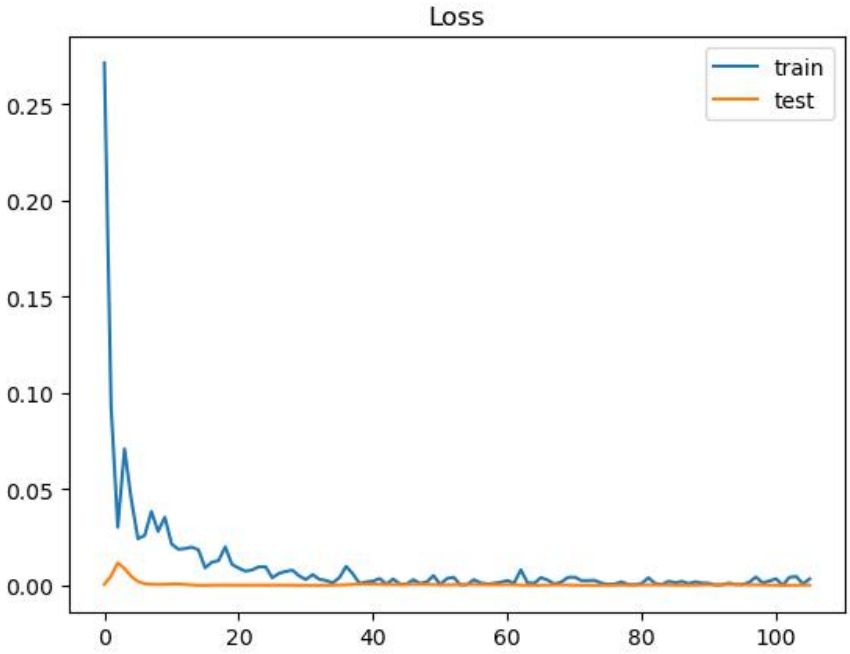


图 3-30 Keras 神经网络 Loss 曲线

从上图可看出，使用神经网络对显卡的总体性能-价格关系进行回归分析依旧不能得到较好的结果。尝试对每一代显卡单独进行回归分析。

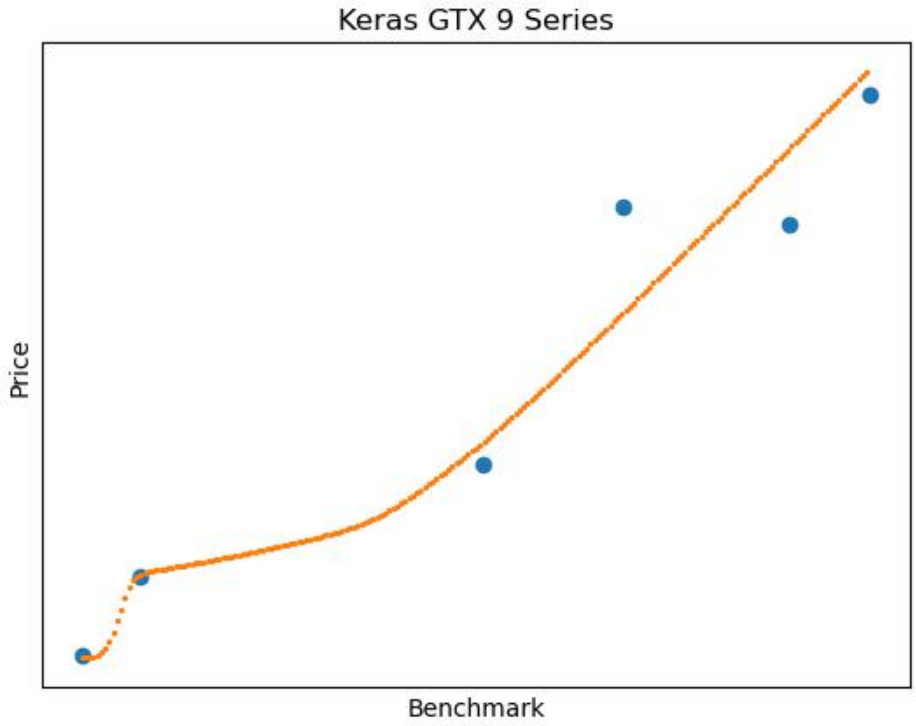


图 3-31 GTX 9 系显卡回归分析(Maxwell 架构)

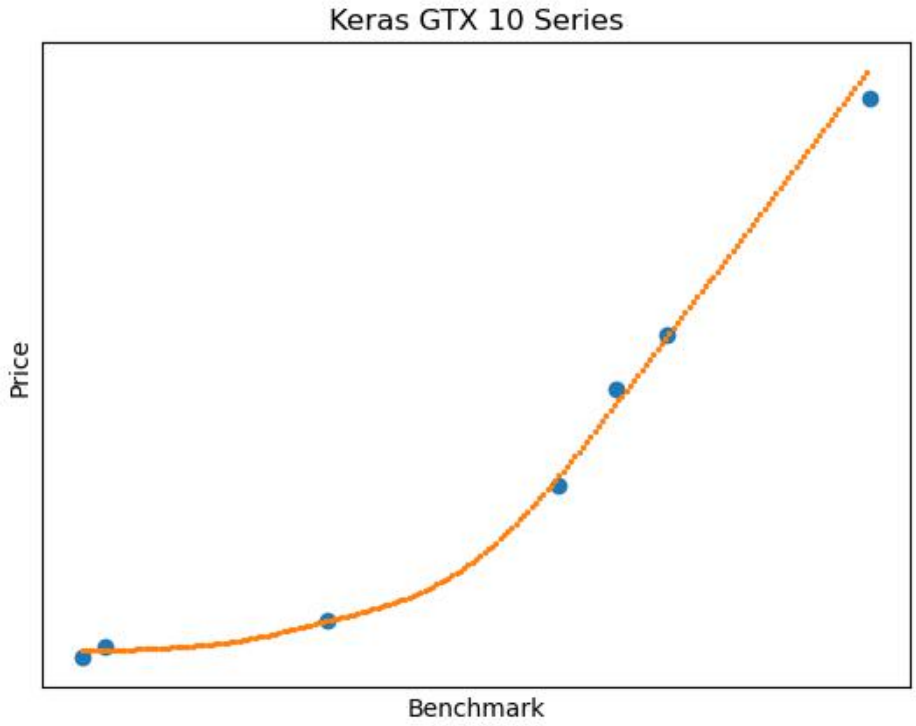


图 3-32 GTX 10 系显卡回归分析(Pascal 架构)

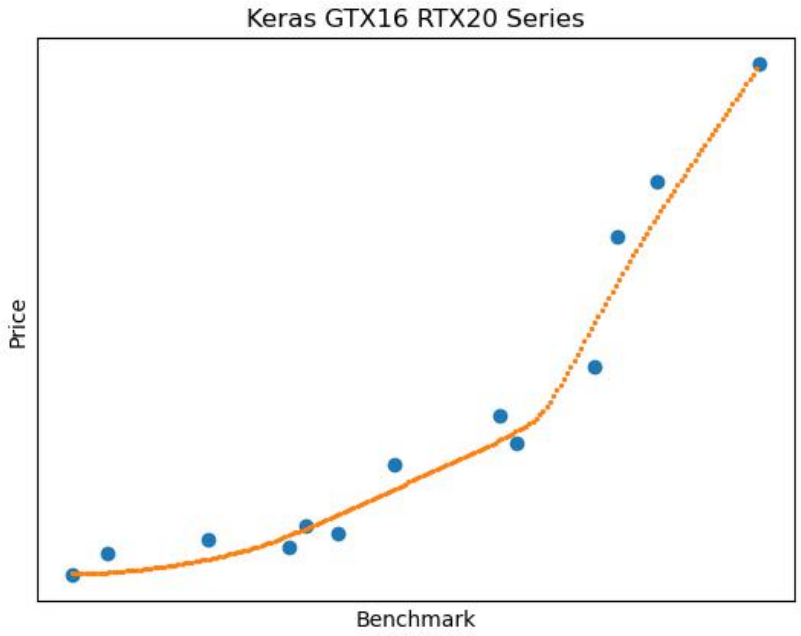


图 3-33 GTX 16 RTX 20 系显卡回归分析(Ampere 架构)

对不同代数的显卡性能-价格进行回归分析后，得到了较好的结果。说明每代显卡之内存在着固定的映射关系，但显卡整体却几乎不存在此种关系。

§3.6 显卡硬件-价格模型

同样采用 Keras 神经网络模型，但是将采集到的所有性能变量结果全部输入全连接神经网络中。网络模型发生一定变更。

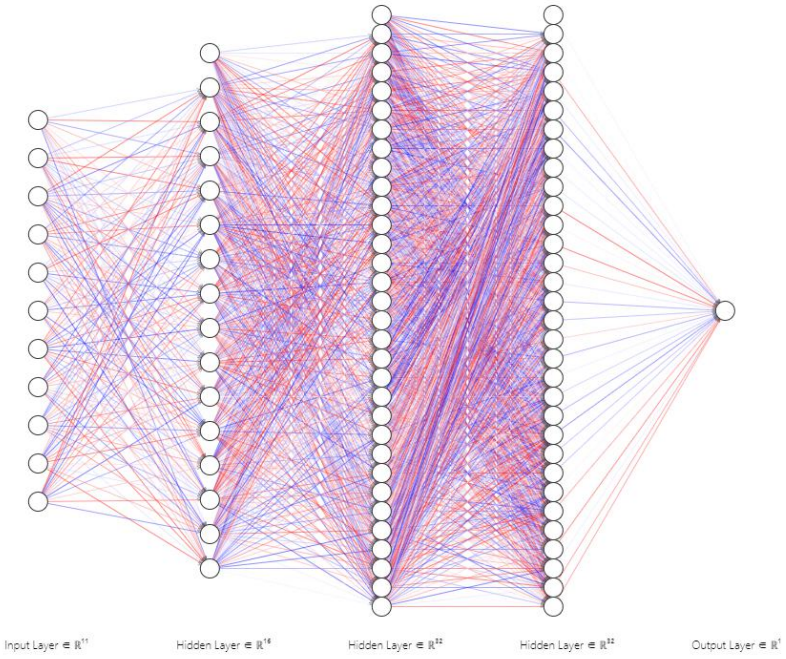


图 3-34 Keras 神经网络结构

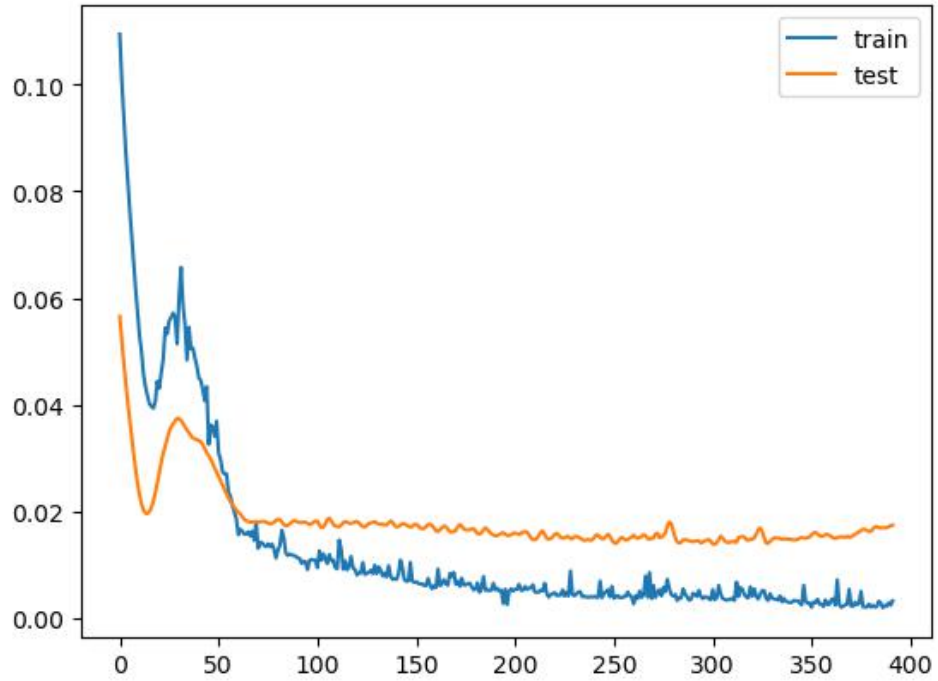


图 3-35 Loss 曲线

Test RMSE=0.132，并没有取得比硬件-价格-性能模型更好的结果。出于可解释性原因，应当采用硬件-价格-性能模型。

§3.7 本章小结

再此章节中，文章建立了显卡的硬件-性能-价格模型，得到了从显卡硬件到显卡价格的完整流程。

（1）显卡的硬件-性能模型：

y = -21704.279346231735+1.13436531e+00*x1+4.84316229e+00*x2+1.81041942e+00*x3+1.17413241e+03*x4-3.69098688e+02*x5+1.48418891e+02*x6+4.09481917e+02*x7+7.10583827e+02*x8+0.00000000e+00*x9+2.90022674e+01*x10

在实际分析中，应当采用如极端森林、梯度回升等集成模型以提高性能。

（2）显卡的性能-价格模型：

GTX 9 系列：

$y = -2E-12x^4 + 6E-08x^3 - 0.0008x^2 + 4.4346x - 8516.2 \quad R^2 = 0.9373$

GTX 10 系列：

$y = 4E-05x^2 - 0.5923x + 3551.3, \quad R^2 = 0.9939$

GTX 16 系列：

$y = -3E-12x^4 + 1E-07x^3 - 0.002x^2 + 14.341x - 36270 \quad R^2 = 0.878$

RTX 20 系列：

$y = -3E\text{-}08x^3 + 0.0015x^2 - 27.417x + 161478$ $R^2 = 0.9577$

GTX 16 + RTX 20 系列：

$y = 2E\text{-}09x^3 - 2E\text{-}05x^2 + 0.0763x + 1809.1$ $R^2 = 0.9655$

RTX 30 系列：

$y = 6E\text{-}08x^3 - 0.0037x^2 + 72.724x - 476290$ $R^2 = 0.9722$

每代显卡之内存在着固定的映射关系，但显卡整体却几乎不存在此种关系。

第 4 章 显卡价格波动分析

本章主要介绍显卡发售后价格的波动情况，以及造成显卡价格波动的因素。

§4.1 数据采集

如上文 3.2.3 节所言，本章中所提到的数据均来自比价网站，以官方指导售价作为参考。

表 4-1 显卡价格数据（部分）

显卡名	原价	现价	涨幅
3060	2499	6899	176.07%
3060ti	2999	8299	176.73%
3070	3899	10678	173.87%
3080	5499	17095	210.87%
3090	11999	23379	94.84%
6700XT	3699	6899	86.51%
6800	4599	8999	95.67%
6800XT	5199	10499	101.94%
6900XT	7999	12999	62.51%
950	260	600	130.77%
960 2GB	360	800	122.22%
970	600	1250	108.33%
980ti	1199	2299	91.74%
1060 3GB	900	1400	55.56%
1060 6GB	1300	2300	76.92%
1070	1500	3500	133.33%
1070 Ti	1800	4000	122.22%
1080	2100	4500	114.29%
1080 Ti	3200	6600	106.25%
2060	2000	3999	100.00%
2060 Super	2500	4499	80.00%
2070 Super	3300	6199	81.82%
2080 Super	3900	7299	79.49%
2080 Ti	8000	12099	25.00%
RX580	700	3000	328.57%
5500XT	1500	4000	166.67%
5700	2000	8000	300.00%
5700XT	2600	9000	246.15%
Radeon 7	3300	10000	203.03%

此外，我们也需要采集显卡的价格波动曲线，以及虚拟货币的价格曲线。这些都可以较为便利的从网上获取。

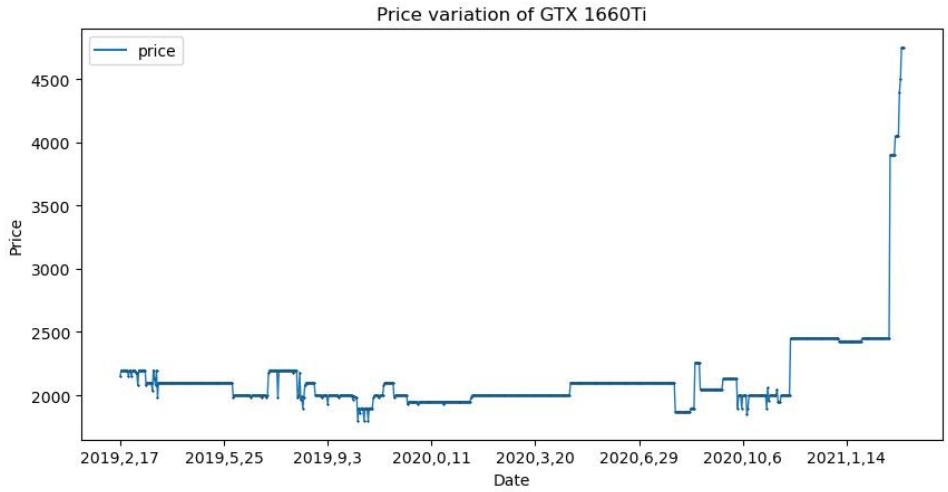


图 4-1 显卡价格曲线

图 4-2 BTC 历史价格曲线

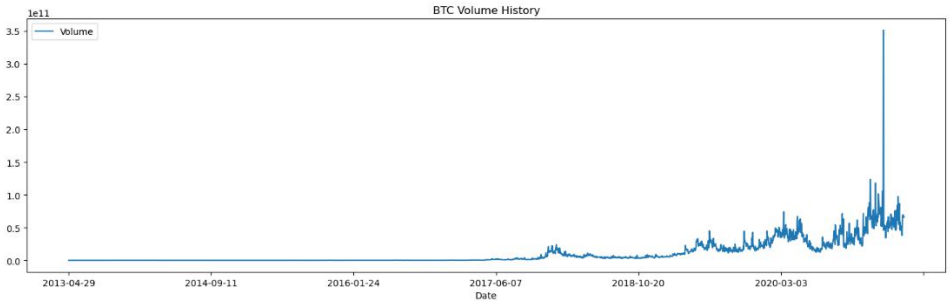


图 4-3 BTC 历史容量曲线

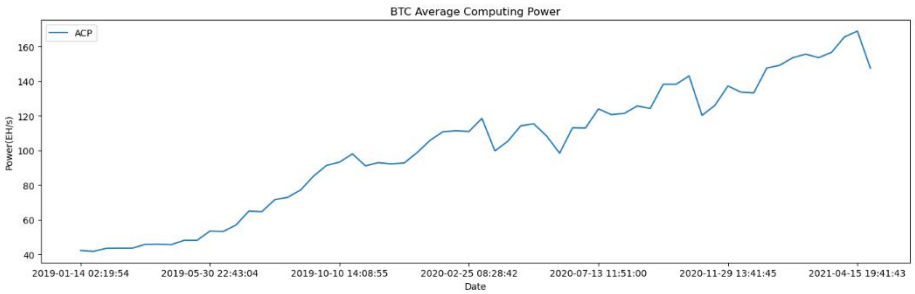


图 4-4 BTC 历史算力曲线

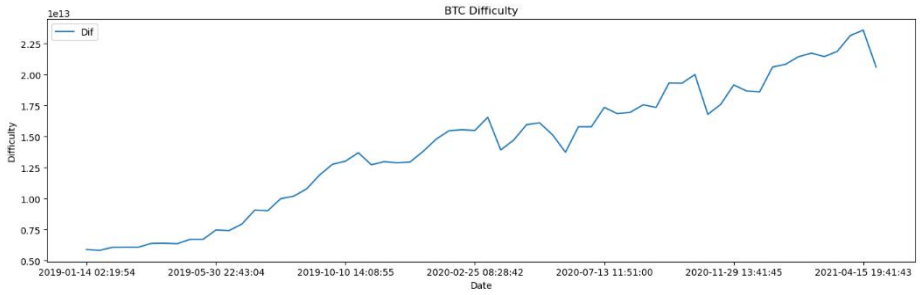


图 4-5 BTC 历史难度曲线

需要注意的是，BTC 的算力已经主要由 ASCI 提供，与显卡脱钩。我们仍旧研究 BTC 的原因是，BTC 是虚拟货币的风向标，BTC 的波动势必影响如 ETH 等强烈依赖于显卡的其他虚拟货币；且对于 BTC 的研究较为成熟。

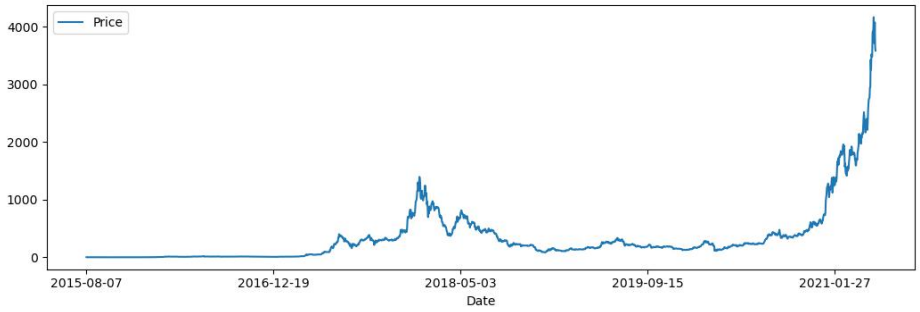


图 4-6 ETH 历史价格曲线

将图 4-2 与图 4-6 比较，不难看出两者之间的关系。

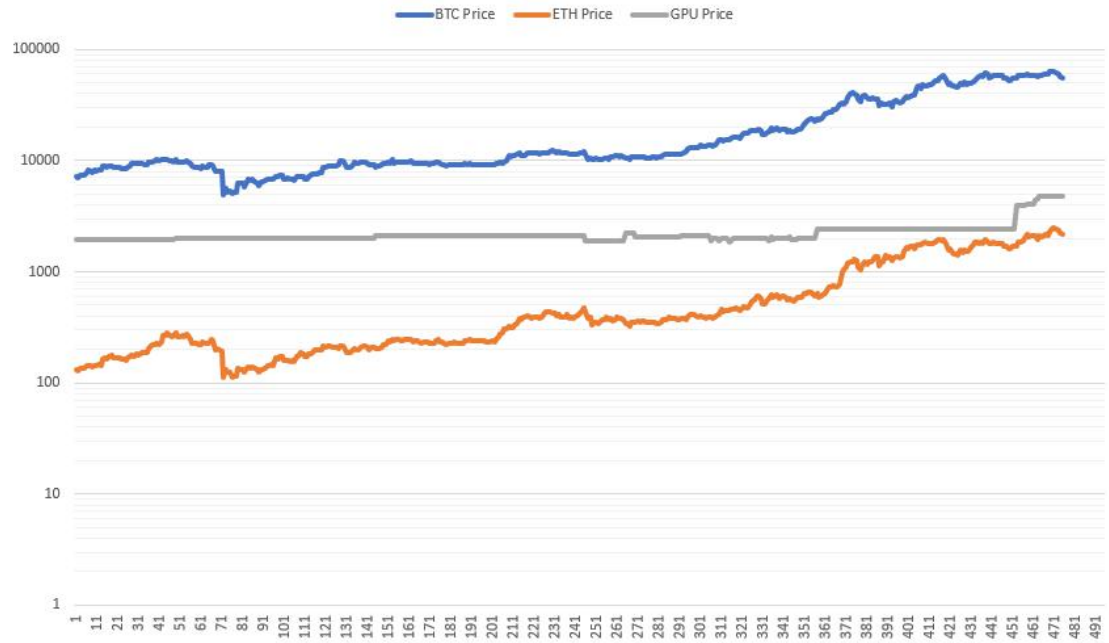


图 4-7 BTC，ETH，GPU 历史价格曲线（对数坐标）

§4.2 数据分析

§4.2.1 相关性分析

Pearson 相关系数（Pearson Correlation Coefficient）是用来衡量两个数据集合是否在一条线上面，它用来衡量定距变量间的线性关系。

Pearson 相关系数衡量的是线性相关关系。若 r=0，只能说 x 与 y 之间无线性相关关系，不能说无相关关系。相关系数的绝对值越大，相关性越强：相关系数越接近于 1 或-1，相关度越强，相关系数越接近于 0，相关度越弱。

通常情况下通过以下取值范围判断变量的相关强度：

0.8-1.0 极强相关

0.6-0.8 强相关

0.4-0.6 中等程度相关

0.2-0.4 弱相关

0.0-0.2 极弱相关或无相关

Pearson 相关系数可以由公式计算得到：

$$r = \frac{N\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{N\sum x_i^2 - (\sum x_i)^2}\sqrt{N\sum y_i^2 - (\sum y_i)^2}}$$

计算 Pearson 相关系数可以得到：

```
Pearson Correlation Coefficient B_E (0.9868019372059657,
0.0)
Pearson Correlation Coefficient E_G (0.7666954569558987,
4.431944317877678e-93)
Pearson Correlation Coefficient B_G (0.7436935392775914,
9.36433883411633e-85)
```

可知：

BTC 价格与 ETH 价格极强相关。

BTC 价格与 GPU 价格强相关。

ETH 价格与 GPU 价格强相关。

可发现显卡价格与虚拟货币之间确实存在着关联。

§4.2.2 回归分析

由表 4-1 可知，显卡的原价和现价存在着一定程度的差异，回归分析的目的在于从显卡的原价得到显卡的当前价格。

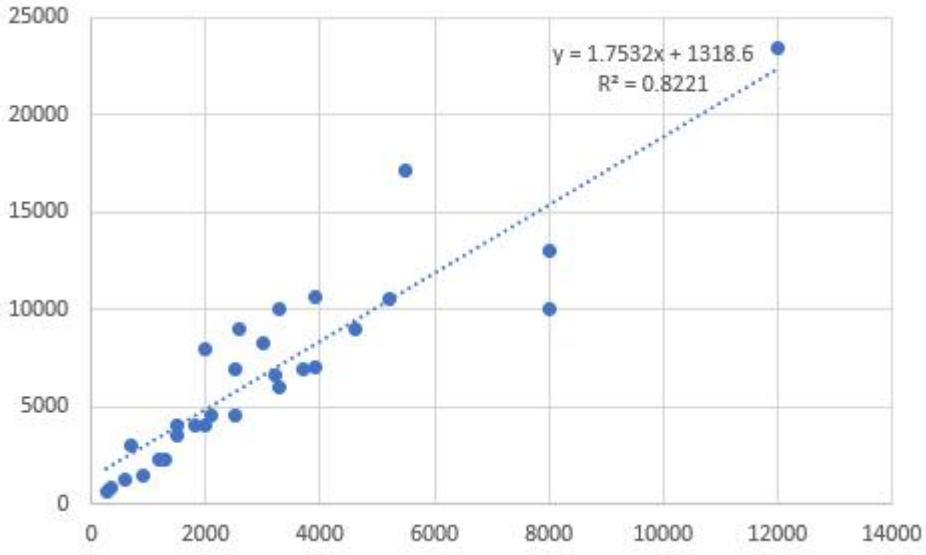


图 4-8 显卡原价-现价对比

由图可知，显卡的涨幅比较接近。除少数离群点外，涨幅基本在原价的 175%左右。也就是说，显卡平均涨了 2.5 倍多。

线性回归的 R 值仍然较大，我们采用神经网络进行更好的拟合。

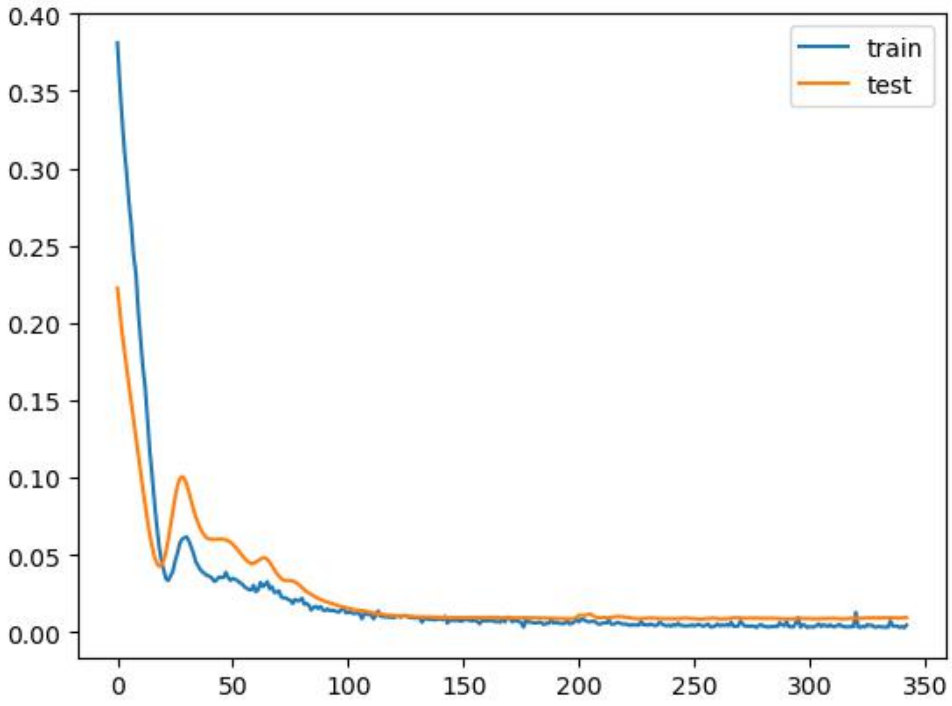


图 4-9 Keras 神经网络 Loss 曲线

我们以显卡的官方价格、跑分作为模型的输入项，拟合显卡的市场流通价格。结果如下（数据经过归一化处理）。

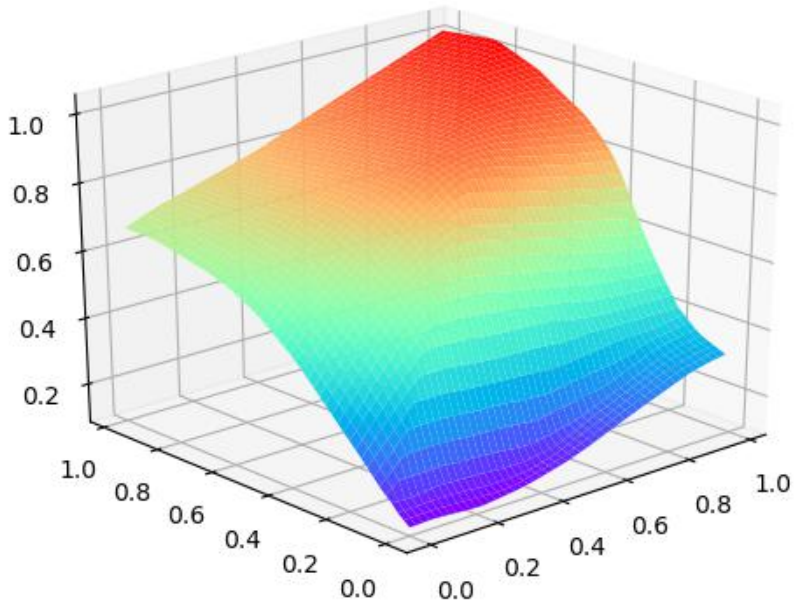


图 4-10 拟合模型

容易看出，无论是价格还是跑分，显卡价格升高均是对中端部分影响最大，而对低端与高端影响较小。

§4.3 本章小结

本章主要介绍了显卡的价格波动成因，2020 年以来的显卡价格波动情况，以及虚拟货币对显卡的影响。同时，建立了显卡的历史价格性能-价格模型，对显卡价格分析有一定的积极作用。

第 5 章 总结与展望

本章对全文的主要工作和创新点作了总结，并提出需要进一步研究和改进之处。

§5.1 本文总结

本文针对现有显卡价格分析与预测模型，对模型进行分析优化，完成了新的显卡硬件-性能-价格模型和显卡价格变动模型，并对显卡价格变动原因做了一定程度的分析。

§5.1.1 本文的主要工作

本文主要研究的是显卡的价格波动，主要工作内容有以下几个方面：

（1）显卡硬件-性能模型的建立：基于采集显卡数据进行数据分析，初步形成了一个效果尚可的显卡硬件指标-性能的映射。

（2）显卡性能-价格模型的建立：基于采集显卡数据进行数据分析，形成了一个针对单代显卡的性能-价格回归模型。

（3）虚拟货币-显卡价格的关系分析：针对 2020 年 1 月 1 日到 20221 年 4 月中旬 500 天左右的数据，明确了比特币、以太坊与显卡价格的联系性。

（4）不同显卡涨幅分析：将不同官方建议价、性能跑分的显卡进行分析，得到了基于初始价和性能的涨幅预测模型。

§5.1.2 本文的主要创新点

本文显卡价格预测与分析创新点主要有以下两项：

（1）实现了显卡价格模型的建立。

（2）实现了基于学习的显卡价格涨幅预测模型。

§5.2 展望

虽然本文实现了显卡价格模型的构建，但仍存在不少需要改进的地方：

（1）本文所基于的数据量相对而言还是不够大。显卡作为高附加值工业制成品，往往一年只能推出一代。而过早的数据因为时代的不同，只能弃而不用。加之显卡本身的不可预测性，本文的结果强烈的依赖于近五年的显卡模型，依赖于半导体的发展。如果显卡制造商与 OEM 改变价格策略或是科技发展速度发生变化，模型的抗干扰能力与传统经验预测相比，并没有出现很大的提升。

（2）显卡发售后的价格，相比其他商品而言，被人为操纵的概率要大的多。全球半导体产量、经济状况、虚拟货币价格变化都会对显卡的价格造成一定程度

的影响，而人为的干预很难也没有必要被整合进模型中。故实际进行显卡的价格预测，如果要实际使用的话，还要结合强化学习、对抗学习等其他方式，对模型进行进一步补全。

致谢

首先要衷心感激导师武星一学期来为我创造了良好的学习环境和浓厚的学习氛围，老师的学识渊博、平易近人、严谨的治学态度和踏实的工作作风都给我留下了深刻的印象，是我今后在事业和生活中的楷模。也要感谢同学们一学期的陪伴与支持，在讨论中互相都得到了很多新的感悟与启发。导师和同学们将不断激励我在以后的工作学习中不断努力、顽强拼搏，做一个对社会和国家有用的人。

参考文献

[1] 惠一峻. 浅析显卡价格暴涨的相关因素[J]. 数码世界, 2018(5).

[2] 刘刚, 刘娟, 唐婉容. 比特币价格波动与虚拟货币风险防范——基于中美政策信息的事件研究法[J]. 广东财经大学学报, 2015, 030(003):30-40.

[3] 孙方江. 比特币价格剧烈波动现象引发的监管思考[J]. 金融会计, 2016, 1(03):52-56.

[4] 王任, 贺雅琴. 数字货币以太坊价格存在泡沫吗?——基于 GSADF 方法的实证研究[J]. 金融与经济, 2018, 000(010):9-16.

[5] 孙吉贵, 刘杰, 赵连宇. 聚类算法研究[J]. 软件学报, 2008, 19(1):48-61.

[6] 杨莉, 邱家驹, 江道灼. 基于 BP 网络的下一交易日无约束市场清算价格预测模型[J]. 电力系统自动化, 2001, 25(19):11-14.

[7] 周开利. 神经网络模型及其 MATLAB 仿真程序设计[M]. 清华大学出版社, 2005.

[8] 杜泽桉. 利用 Keras 高级神经网络实现商品销量的预测[J]. 电子制作, 2019, No.368(Z1):78-81.

[9] 高云, 彭炜. 基于 Keras 的分类预测应用研究[J]. 山西大同大学学报:自然科学版, 2019(5):26-30.

[10] Dillon J V, Langmore I, Tran D, et al. Tensorflow distributions[J]. arXiv preprint arXiv:1711.10604, 2017.

[11] Ketkar N. Introduction to keras[M]//Deep learning with Python. Apress, Berkeley, CA, 2017: 97-111.

[12] Fackler P L, Goodwin B K. Spatial price analysis[J]. Handbook of agricultural economics, 2001, 1: 971-1024.

[13] Park B, Bae J K. Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data[J]. Expert systems with applications, 2015, 42(6): 2928-2934.

[14] Velankar S, Valecha S, Maji S. Bitcoin price prediction using machine learning[C]//2018 20th International Conference on Advanced Communication Technology (ICACT). IEEE, 2018: 144-147.

[15] Jordan M I, Mitchell T M. Machine learning: Trends, perspectives, and prospects[J]. Science, 2015, 349(6245): 255-260.

附录：部分源程序清单

```
//1 显卡价格卷积处理
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import matplotlib.pyplot as plt
import scipy.signal
data = pd.read_csv('dat.csv', index_col=0)
data = DataFrame(data)
data.dropna(axis=1, inplace=True)
data.plot()
x = data.iloc[:, 0]
print(x)
h = np.array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1])
a = scipy.signal.convolve(x, h)
a = DataFrame(a[9:-10])
print(a)
a.plot()
plt.show()

//2 交叉表的构建
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plt
import numpy as np
from pylab import mpl
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn import model_selection
mpl.rcParams['font.sans-serif'] = ['MicroSoft YaHei']
data = pd.read_excel('GPU.xlsx', 'Sheet2', index_col=0)
ndata = data[['CUDA 核心', '加速频率', '内存速度', '显存带宽', '跑分', '价格
']]
X = data.iloc[:, 3:13].values.astype(float)
plt.figure()
pd.plotting.scatter_matrix(ndata, diagonal='kde')
plt.xticks(fontsize=5)
plt.yticks(fontsize=5)

//3 箱型图的绘制
import pandas as pd
import matplotlib.pyplot as plt
from pylab import mpl
mpl.rcParams['font.sans-serif']=['SimHei']
```

```
data = pd.read_excel('GPU.xlsx', 'Sheet2', index_col=0)
data.dropna(inplace=True)
ndata = data[['CUDA 核心','加速频率','内存速度','显存带宽','跑分','价格']]
data10 = ndata.iloc[0:6, :]
data16 = ndata.iloc[6:11, :]
data20 = ndata.iloc[12:19, :]
data30 = ndata.iloc[20:24, :]
print(data30)
plt.figure(figsize=(10,10), dpi=300)
pd.plotting.scatter_matrix(ndata, diagonal='kde')
pd.plotting.scatter_matrix(data10, diagonal='kde', color='r')
pd.plotting.scatter_matrix(data16, diagonal='kde', color='g')
pd.plotting.scatter_matrix(data20, diagonal='kde', color='b')
pd.plotting.scatter_matrix(data30, diagonal='kde', color='k')
plt.xticks(fontsize=5)
plt.yticks(fontsize=5)
plt.savefig('out10.jpg', dpi=300)
```

//4 线性回归算法

```
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plt
import numpy as np
from pylab import mpl
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn import model_selection
mpl.rcParams['font.sans-serif'] = ['MicroSoft YaHei']
data = pd.read_excel('GPU.xlsx', 'Sheet1', index_col=0)
data.dropna(inplace=True)
X = data.iloc[:, 3:13].values.astype(float)
y = data.iloc[:, 14].values.astype(float)
#y = data.iloc[:, 15].values.astype(float)
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
 test_size=0.35, random_state=4)
linregTr = LinearRegression()
linregTr.fit(X_train, y_train)
print(linregTr.intercept_, linregTr.coef_)
y_train_pred = linregTr.predict(X_train)
y_test_pred = linregTr.predict(X_test)
train_err = metrics.mean_squared_error(y_train, y_train_pred)
test_err = metrics.mean_squared_error(y_test, y_test_pred)
print("train err = " + str(train_err), "\ntest err = " + str(test_err))
pred_score = linregTr.score(X_test, y_test)
print("Predict score is " + str(pred_score))
```

```
//5 随机森林、极端森林、梯度上升回归
from sklearn import model_selection
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor,
 GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_
error
import numpy as np
import pandas as pd
'''
随机森林回归
极端随机森林回归
梯度提升回归
通常集成模型能够取得非常好的表现
'''
# 1 准备数据
# 读取 GPU 信息
data = pd.read_excel('GPU.xlsx', 'Sheet2', index_col=0)
data = data.iloc[0:29, 0:16]
data.dropna(inplace=True)
X = data.iloc[:, 3:13].values.astype(float)
y = data.iloc[:, 14].values.astype(float)
#y = data.iloc[:, 15].values.astype(float)
y = np.ravel(y)
# 2 分割训练数据和测试数据
# 随机采样 25%作为测试 75%作为训练
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
 test_size=0.25, random_state=4)
# 3 训练数据和测试数据进行标准化处理
ss_X = StandardScaler()
X_train = ss_X.fit_transform(X_train)
X_test = ss_X.transform(X_test)
ss_y = StandardScaler()
y_train = ss_y.fit_transform(y_train.reshape(-1, 1))
y_test = ss_y.transform(y_test.reshape(-1, 1))
# 4 三种集成回归模型进行训练和预测
# 随机森林回归
rfr = RandomForestRegressor()
# 训练
rfr.fit(X_train, y_train.ravel())
# 预测 保存预测结果
rfr_y_predict = rfr.predict(X_test)
# 极端随机森林回归
etr = ExtraTreesRegressor()
# 训练
etr.fit(X_train, y_train.ravel())
```

```
# 预测 保存预测结果
etr_y_predict = rfr.predict(X_test)
# 梯度提升回归
gbr = GradientBoostingRegressor()
# 训练
gbr.fit(X_train, y_train.ravel())
# 预测 保存预测结果
gbr_y_predict = rfr.predict(X_test)
# 5 模型评估
# 随机森林回归模型评估
print("随机森林回归的默认评估值为：", rfr.score(X_test, y_test))
print("随机森林回归的 R_squared 值为：", r2_score(y_test, rfr_y_predict))
print("随机森林回归的均方误差
为:", mean_squared_error(ss_y.inverse_transform(y_test),ss_y.inverse_tra
nsform(rfr_y_predict)))
print("随机森林回归的平均绝对误差
为:", mean_absolute_error(ss_y.inverse_transform(y_test),ss_y.inverse_tr
ansform(rfr_y_predict)))
# 极端随机森林回归模型评估
print("极端随机森林回归的默认评估值为：", etr.score(X_test, y_test))
print("极端随机森林回归的 R_squared 值为:", r2_score(y_test, gbr_y_predict))
print("极端随机森林回归的均方误差
为:", mean_squared_error(ss_y.inverse_transform(y_test),ss_y.inverse_tra
nsform(gbr_y_predict)))
print("极端随机森林回归的平均绝对误差
为:", mean_absolute_error(ss_y.inverse_transform(y_test),ss_y.inverse_tr
ansform(gbr_y_predict)))
# 梯度提升回归模型评估
print("梯度提升回归回归的默认评估值为：", gbr.score(X_test, y_test))
print("梯度提升回归回归的 R_squared 值为:", r2_score(y_test, etr_y_predict))
print("梯度提升回归回归的均方误差
为:", mean_squared_error(ss_y.inverse_transform(y_test),ss_y.inverse_tra
nsform(etr_y_predict)))
print("梯度提升回归回归的平均绝对误差
为:", mean_absolute_error(ss_y.inverse_transform(y_test),ss_y.inverse_tr
ansform(etr_y_predict)))
```

```
//6 k-means 算法
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
import pandas as pd
from pandas import DataFrame
data = pd.read_excel('/GPU.xlsx', 'Sheet3', index_col=0)
```

```
values = data.iloc[:, 14:16].astype(float)
values.dropna(inplace=True)
X = DataFrame(values)
#构造聚类器
estimator = KMeans(n_clusters=6)
#聚类
estimator.fit(X)
#获取聚类标签
label_pred = estimator.labels_
#绘制 k-means 结果
x0 = X[label_pred == 0]
x1 = X[label_pred == 1]
x2 = X[label_pred == 2]
x3 = X[label_pred == 3]
x4 = X[label_pred == 4]
x5 = X[label_pred == 5]
plt.scatter(x0.iloc[:, 0], x0.iloc[:, 1], c = "red", marker='o', label='
label0')
plt.scatter(x1.iloc[:, 0], x1.iloc[:, 1], c = "green", marker='o', label
='label1')
plt.scatter(x2.iloc[:, 0], x2.iloc[:, 1], c = "blue", marker='o', label=
'label2')
plt.scatter(x3.iloc[:, 0], x3.iloc[:, 1], c = "orange", marker='o', labe
l='label3')
plt.scatter(x4.iloc[:, 0], x4.iloc[:, 1], c = "yellow", marker='o', labe
l='label4')
plt.scatter(x5.iloc[:, 0], x5.iloc[:, 1], c = "purple", marker='o', labe
l='label5')
plt.xlabel('Benchmark')
plt.ylabel('Price')
plt.legend(loc=2)
plt.show()

//7 Keras 神经网络回归
import matplotlib.pyplot as plt
from math import sqrt
from matplotlib import pyplot
import pandas as pd
from numpy import concatenate
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from sklearn import preprocessing
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import Adam

'''
```

```
keras 实现神经网络回归模型
'''
'''
# 读取数据
path = 'data.csv'
train_df = pd.read_csv(path)
# 删掉不用字符串字段
dataset = train_df.drop('jh', axis=1)
# df 转 array
values = dataset.values
# 原始数据标准化，为了加速收敛
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
y = scaled[:, -1]
X = scaled[:, 0:-1]
'''
data = pd.read_excel('C:/Users/Fraxinus/Desktop/BigData/GPU.xlsx', 'Shee
t3', index_col=0)
#data.dropna(inplace=True)
values = data.iloc[22:34, 14:16].astype(float)
values.dropna(inplace=True)
print(values)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
X = scaled[:, 0]
y = scaled[:, 1]
# 随机拆分训练集与测试集
from sklearn.model_selection import train_test_split
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.25
)
n = 200
w = [i/n*(max(X)) for i in range(n)]
# 全连接神经网络
model = Sequential()
#input = X.shape[1]
model.add(Dense(units=8, input_dim=1))
model.add(Activation('relu'))
# 隐藏层 128
model.add(Dense(1024))
model.add(Activation('relu'))
# Dropout 层用于防止过拟合
model.add(Dropout(0.1))
# 隐藏层 256
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.1))
```

```
# 隐藏层 128
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.1))
# 没有激活函数用于输出层，因为这是一个回归问题，我们希望直接预测数值，而不需要采用激活函数进行变换。
model.add(Dense(1))
# 使用高效的 ADAM 优化算法以及优化的最小均方误差损失函数
model.compile(loss='mean_squared_error', optimizer=Adam())
# early stoppping
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=90, verbose=2)
# 训练
history = model.fit(train_X, train_y, epochs=1000, batch_size=20, validation_data=(test_X, test_y), verbose=2,
                    shuffle=False, callbacks=[early_stopping])
#history = model.fit(train_X, train_y, epochs=1000, batch_size=20, validation_data=(X, y), verbose=2,
#                     shuffle=False, callbacks=[early_stopping])
# loss 曲线
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
# 预测
yhat = model.predict(test_X)
z = model.predict(w)
pyplot.xticks(())
pyplot.yticks(())
pyplot.scatter(X, y)  # 样本点
pyplot.scatter(w, z, s=2)  # 预测线
pyplot.show()
rmse = sqrt(mean_squared_error(test_y, yhat))
print('Test RMSE: %.3f' % rmse)
plt.plot(test_y)
plt.plot(yhat)
pyplot.show()
```

//8 Keras 神经网络回归

```
import matplotlib.pyplot as plt
from math import sqrt
from matplotlib import pyplot
import pandas as pd
from numpy import concatenate
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from sklearn import preprocessing
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import Adam
'''
keras 实现神经网络回归模型
'''
data = pd.read_excel('GPU.xlsx', 'Sheet3', index_col=0)
#data.dropna(inplace=True)
values = data.iloc[:, 3:16].astype(float)
values.dropna(inplace=True)
print(values)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
X = scaled[:, 0:11]
y = scaled[:, 12]
# 随机拆分训练集与测试集
from sklearn.model_selection import train_test_split

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.25
)
# 全连接神经网络
model = Sequential()
#input = X.shape[1]
model.add(Dense(units=8, input_dim=11))
model.add(Activation('relu'))
# 隐藏层 16
model.add(Dense(16))
model.add(Activation('relu'))
# Dropout 层用于防止过拟合
#model.add(Dropout(0.1))
# 隐藏层 32
model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dropout(0.01))
# 隐藏层 32
model.add(Dense(32))
model.add(Activation('relu'))
#model.add(Dropout(0.1))
# 没有激活函数用于输出层，因为这是一个回归问题，我们希望直接预测数值，而不需要采
用激活函数进行变换。
model.add(Dense(1))
# 使用高效的 ADAM 优化算法以及优化的最小均方误差损失函数
model.compile(loss='mean_squared_error', optimizer=Adam())
# early stoppping
```

```
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=90, verbose=
2)
# 训练
history = model.fit(train_X, train_y, epochs=1000, batch_size=20, valida
tion_data=(test_X, test_y), verbose=2,
                    shuffle=False, callbacks=[early_stopping])
#history = model.fit(train_X, train_y, epochs=1000, batch_size=20, valid
ation_data=(X, y), verbose=2,
#                     shuffle=False, callbacks=[early_stopping])
# loss 曲线
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
yhat = model.predict(test_X)
rmse = sqrt(mean_squared_error(test_y, yhat))
print('Test RMSE: %.3f' % rmse)
plt.plot(test_y)
plt.plot(yhat)
pyplot.show()
# 预测 y 逆标准化
inv_yhat0 = concatenate((test_X, yhat), axis=1)
inv_yhat1 = scaler.inverse_transform(inv_yhat0)
inv_yhat = inv_yhat1[:, -1]
# 原始 y 逆标准化
test_y = test_y.reshape((len(test_y), 1))
inv_y0 = concatenate((test_X, test_y), axis=1)
inv_y1 = scaler.inverse_transform(inv_y0)
inv_y = inv_y1[:, -1]
# 计算 RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
plt.plot(inv_y)
plt.plot(inv_yhat)
```

//9 比特币历史数据获取分析

```
import time
import requests
import json
import csv
import pandas as pd
import matplotlib.pyplot as plt
time_stamp = int(time.time())
print(f"Now timestamp: {time_stamp}")
# 1367107200
```

```
request_link = f"https://web-api.coinmarketcap.com/v1/cryptocurrency/ohl
cv/historical?convert=USD&slug=bitcoin&time_end={time_stamp}&time_start=
1367107200"
print("Request link: " + request_link)
r = requests.get(url = request_link)
#print(r.content)
# 返回的数据是 JSON 格式，使用 json 模块解析
content = json.loads(r.content)
#print(type(content))
quoteList = content['data']['quotes']
# 将数据保存为 BTC.csv
with open('BTC.csv','w' ,encoding='utf8',newline='') as f:
    csv_write = csv.writer(f)
    csv_head = ["Date","Price","Volume"]
    csv_write.writerow(csv_head)
    for quote in quoteList:
        quote_date = quote["time_open"][:10]
        quote_price = "{:.2f}".format(quote["quote"]["USD"]["close"])
        quote_volume = "{:.2f}".format(quote["quote"]["USD"]["volume"])
        csv_write.writerow([quote_date, quote_price, quote_volume])
print("Done")
series = pd.DataFrame()
df = pd.read_csv("BTC.csv")
series['Date'] = df['Date'].tolist()
series['Price'] = df['Price'].tolist()
series['Volume'] = df['Volume'].tolist()

ax = plt.gca()
series.plot(kind='line', x='Date', y='Price', ax=ax)
plt.show()
plt.cla()
ax = plt.gca()
series.plot(kind='line', x='Date', y='Volume', ax=ax)
plt.show()
series = pd.DataFrame()
df = pd.read_csv("BTC_Difficulty.csv")
series['Date'] = df['Time'].tolist()
series['ACP'] = df['AverageComputingPower(EH/s)'].tolist()
series['Dif'] = df['Difficulty'].tolist()
reverse_series = series.iloc[::-1]
ax = plt.gca()
reverse_series.plot(kind='line', x='Date', y='ACP', ax=ax)
plt.show()
plt.cla()
ax = plt.gca()
reverse_series.plot(kind='line', x='Date', y='Dif', ax=ax)
plt.show()
```

//10 Pearson 分析

```
import numpy as np
from scipy.stats import pearsonr
import pandas as pd
import matplotlib.pyplot as plt
series = pd.DataFrame()
df = pd.read_csv("BTCETHComp.csv")
series['Date'] = df['Date'].tolist()
series['BTC'] = df['BTC Price'].tolist()
series['ETH'] = df['ETH Price'].tolist()
series['GPU'] = df['GPU Price'].tolist()
ax = plt.gca()
series.plot(kind='line', x='Date', y='BTC', ax=ax)
series.plot(kind='line', x='Date', y='ETH', ax=ax)
series.plot(kind='line', x='Date', y='GPU', ax=ax)
print("Pearson Correlation Coefficient B_E", pearsonr(series['BTC'], ser
ies['ETH']))
print("Pearson Correlation Coefficient E_G", pearsonr(series['ETH'], ser
ies['GPU']))
print("Pearson Correlation Coefficient B_G", pearsonr(series['BTC'], ser
ies['GPU']))
plt.show()
```

//11 三维图形绘制

```
import matplotlib.pyplot as plt
from matplotlib import pyplot
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
#省略神经网络训练部分
fig = plt.figure()
ax = plt.axes(projection='3d')
xx = np.arange(0, 1, 0.02)
yy = np.arange(0, 1, 0.02)
mX, mY = np.meshgrid(xx, yy)
Z = np.zeros((50, 50))
for i in range(0, 50, 1):
    for j in range(0, 50, 1):
        tx = float(i/50.0)
        ty = float(j/50.0)
        Z[i][j] = model.predict(np.array([[tx, ty]]))
    print('batch complete')
Z.reshape(2500, 1)
ax.plot_surface(mX, mY, Z, cmap='rainbow')
plt.show()
```