

数据挖掘大作业实验报告

胡子骄 1120202062 07152002

丁励超 1120203541 07152002

内容1：数据可视化探索分析

1. 分析问题的背景介绍

篮球巨星科比于1996年以13顺位的选秀身份进入NBA联盟，一生都效力于洛杉矶湖人队。2016年宣布退役，职业生涯获奖无数，生涯总得分超33000分。

我们采用的数据集是NBA官方提供的科比球场表现数据集收录了科比自1996赛季~2016赛季共30697条职业生涯数据。每条数据都是一次出手，其中包括动作类型，投篮类型，投射距离，投射位置，是否命中等25个特征。

2. 数据集概括性介绍，包括数据集大小、属性名称、类型，数据质量。

科比数据集有30697行数据，25个属性值，有3个属性是浮点数类型，11个属性是整数类型，11个属性是object类型。

```
In [76]: data.shape
Out[76]: (30697, 25)

In [ ]: data.head()

action_type combined_shot_type game_event_id game_id lat loc_x loc_y lon minutes_remaining period playoffs season seconds_remaining
0 Jump Shot Jump Shot 10 20000012 33.9723 167 72 -118.1028 10 1 0 2000-01 2
1 Jump Shot Jump Shot 12 20000012 34.0443 -157 0 -118.4268 10 1 0 2000-01 2
2 Jump Shot Jump Shot 35 20000012 33.9093 -101 135 -118.3708 7 1 0 2000-01 2
3 Jump Shot Jump Shot 43 20000012 33.8693 138 175 -118.1318 6 1 0 2000-01 5
4 Driving Dunk Shot Dunk 155 20000012 34.0443 0 0 -118.2698 6 2 0 2000-01 1
```

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30697 entries, 0 to 30696
Data columns (total 25 columns):
 #   Column          Non-Null Count  Dtype  
--- 
 0   action_type     30697 non-null   object 
 1   combined_shot_type 30697 non-null   object 
 2   game_event_id    30697 non-null   int64  
 3   game_id          30697 non-null   int64  
 4   lat              30697 non-null   float64
 5   loc_x            30697 non-null   int64  
 6   loc_y            30697 non-null   int64  
 7   lon              30697 non-null   float64
 8   minutes_remaining 30697 non-null   int64  
 9   period           30697 non-null   int64  
 10  playoffs         30697 non-null   int64  
 11  season           30697 non-null   object 
 12  seconds_remaining 30697 non-null   int64  
 13  shot_distance    30697 non-null   int64  
 14  shot_made_flag   25697 non-null   float64
 15  shot_type        30697 non-null   object 
 16  shot_zone_area   30697 non-null   object 
 17  shot_zone_basic  30697 non-null   object 
 18  shot_zone_range  30697 non-null   object 
 19  team_id          30697 non-null   int64  
 20  team_name         30697 non-null   object 
 21  game_date         30697 non-null   object 
 22  matchup           30697 non-null   object 
 23  opponent          30697 non-null   object 
 24  shot_id           30697 non-null   int64  
dtypes: float64(3), int64(11), object(11)
memory usage: 5.9+ MB
```

属性名称和对应含义的介绍：

列名称	含义
action_type	动作类型，分为跳投(Jump Shot), 跑跳投(Running Jump Shot), 扣篮(Driving Dunk Shot), 上篮(Layup Shot), 突破上篮(Driving Layup Shot), 灌篮(Slam Dunk Shot), 转身跳投(turnaround jump shot)等等
combined_shot_type	组合投篮类型, 如跳投, 扣篮, 勾手, 擦板, 罚球
game_event_id	比赛的编号
lat	出手的纬度
loc_x	出手的x坐标
loc_y	出手的y坐标
lon	出手的经度
minutes_remaining	距离比赛结束, 还剩多少分钟
period	交手的场次, 取值为1~7
playoffs	是否是打季后赛
season	赛季, 如13~14赛季
seconds_remaining	距离比赛结束, 还剩多少秒
shot_distance	出手距离
shot_made_flag	是否命中 (研究目标)
shot_type	投射类型, 两分球还是三分球
shot_zone_area	投篮区域的表示方法一: 左侧, 右侧, 中场, 后场等
shot_zone_basic	投篮区域的表示方法二:中线, 禁区, 油漆区, 左侧底角, 右侧底角等
shot_zone_range	投篮区域的表示方法三:出手区域到篮板的距离, 小于8英尺, 816英尺, 1624英尺, 24英尺以上等
team_id	球队编号
team_name	球队名称
game_date	比赛日期
matchup	对阵双方
opponent	对手
game_id	比赛的编号
shot_id	镜头ID

查看缺失值的结果:

1 Number of instances = 30697

```

2 Number of attributes = 25
3 Number of missing values:
4 action_type          0
5 combined_shot_type   0
6 game_event_id        0
7 game_id               0
8 lat                   0
9 loc_x                 0
10 loc_y                0
11 lon                  0
12 minutes_remaining    0
13 period                0
14 playoffs              0
15 season                0
16 seconds_remaining     0
17 shot_distance         0
18 shot_made_flag        5000
19 shot_type              0
20 shot_zone_area         0
21 shot_zone_basic        0
22 shot_zone_range        0
23 team_id                0
24 team_name              0
25 game_date              0
26 matchup                0
27 opponent                0
28 shot_id                0
29 dtype: int64

```

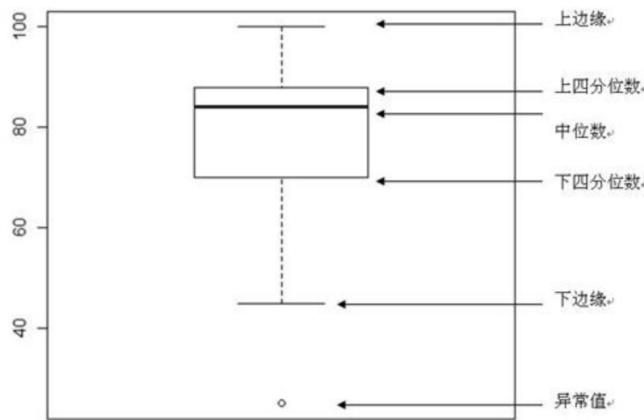
1.完整性：总共有30697条数据，shot_made_flag属性存在5000条缺失，其他属性均完整。

2.准确性：来自官方数据，真实程度较高。

3.可视化探索分析（画图并给出分析结论），至少包括以下内容：

- 绘制至少三个属性的箱线图并进行分析分布情况，特点。

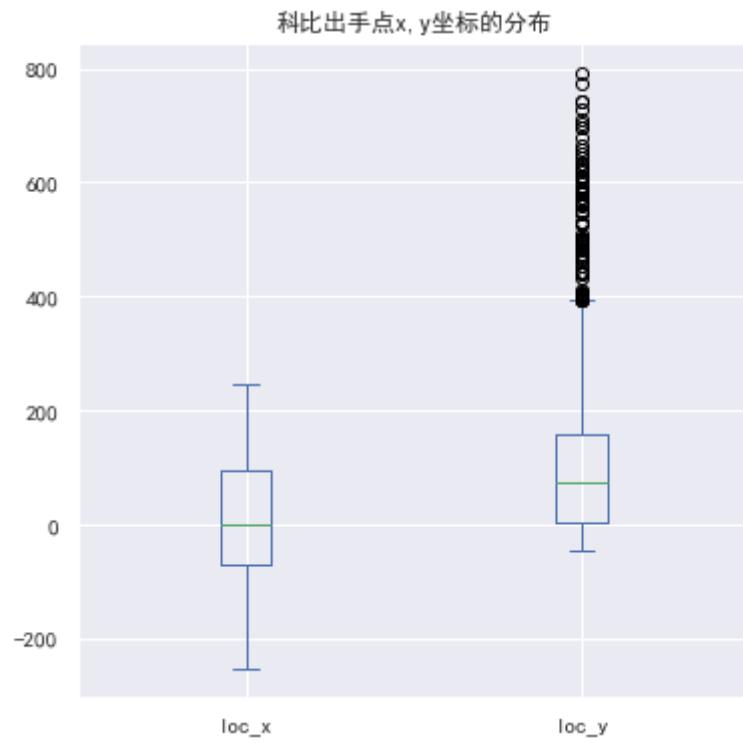
箱线图的作用：直观地识别数据中异常值；直观地判断数据离散分布情况，了解数据分布状态。



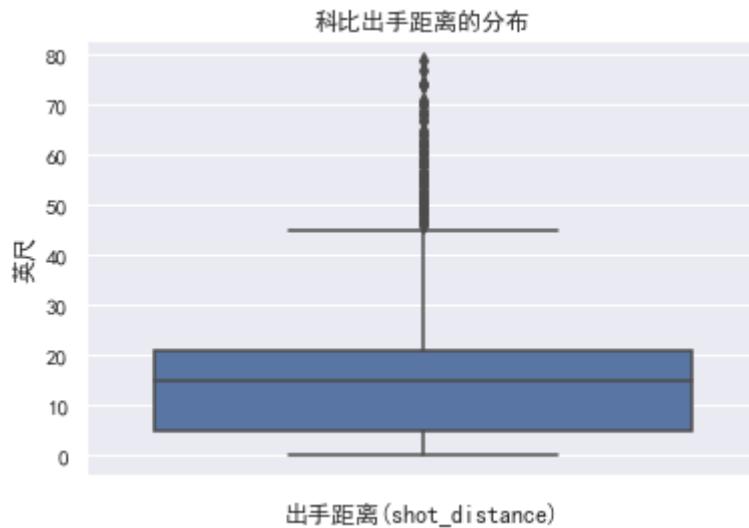
从图中可以得到如下信息：

- 中位数：常用于度量数据的中心。
 - 四分位数极差 (IQR, interquartile range)
- IQR表示数据集中间 50% 的数据，即第一个四分位数与第三个四分位数之间的距离 (Q3-Q1)。
- 最小观察值（下边缘） = Q1 - 1.5 IQR。

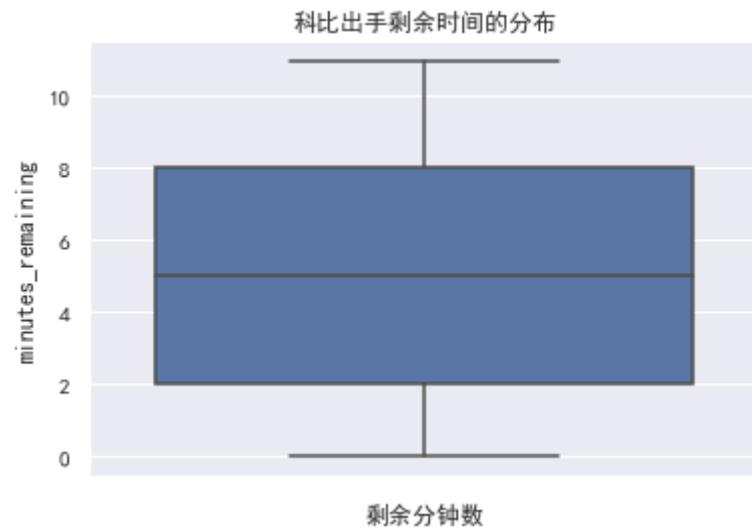
- 最大观察值（上边缘）= $Q3 + 1.5 \text{ IQR}$ 。
- 超出最大或者最小观察值的数据个别的绘制出。



`loc_x` 属性分布较为集中，没有离群点，中位数位于0处，表示投篮在左右两边的次数基本相同，且大多数靠近零点；`loc_y` 属性呈现偏态分布，原因是有一部分球的出手位置的 y 轴坐标显著的较远。

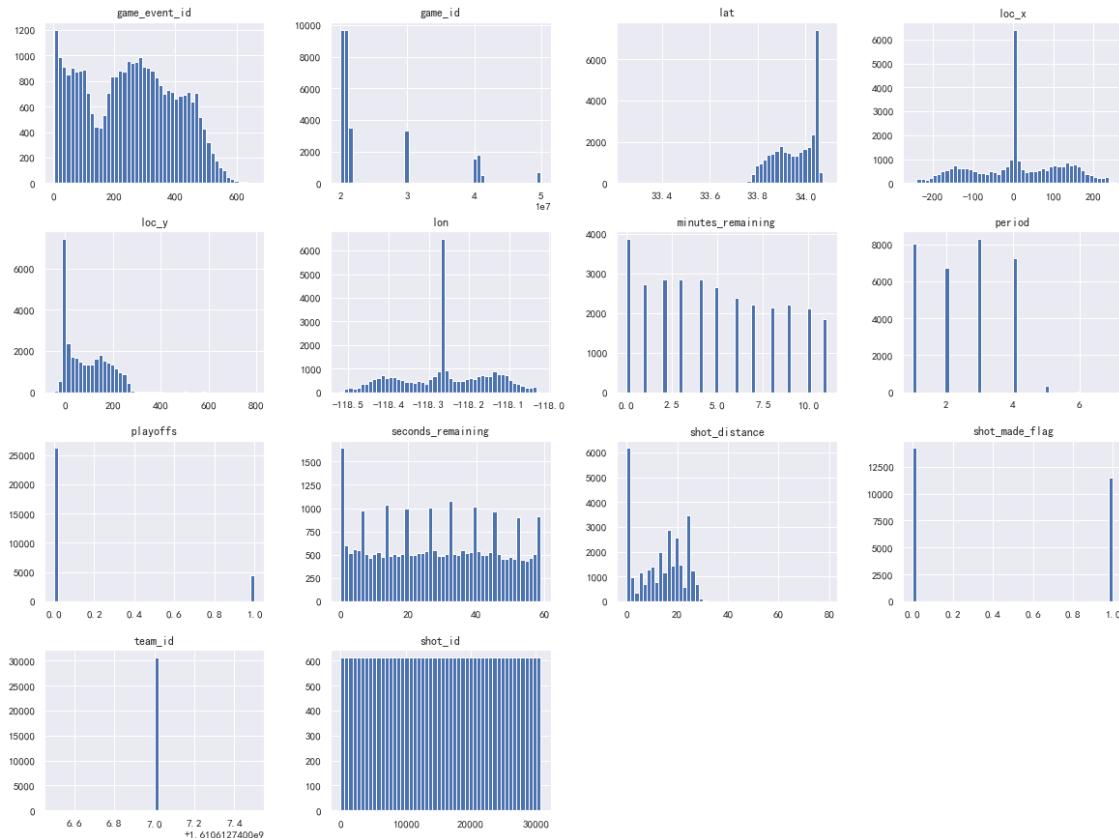


可以看到科比出手距离箱线图和`loc_y` 属性的箱线图大致相同，从0到7.9英尺范围内均有分布，25%到75%的数据集中在0.5-2.1英尺之间，中位数偏小，一部分的数据点分布在箱线图之外，呈现偏态分布。



剩余时间的分布较为集中，无离群点。

- 计算数据集的统计特征

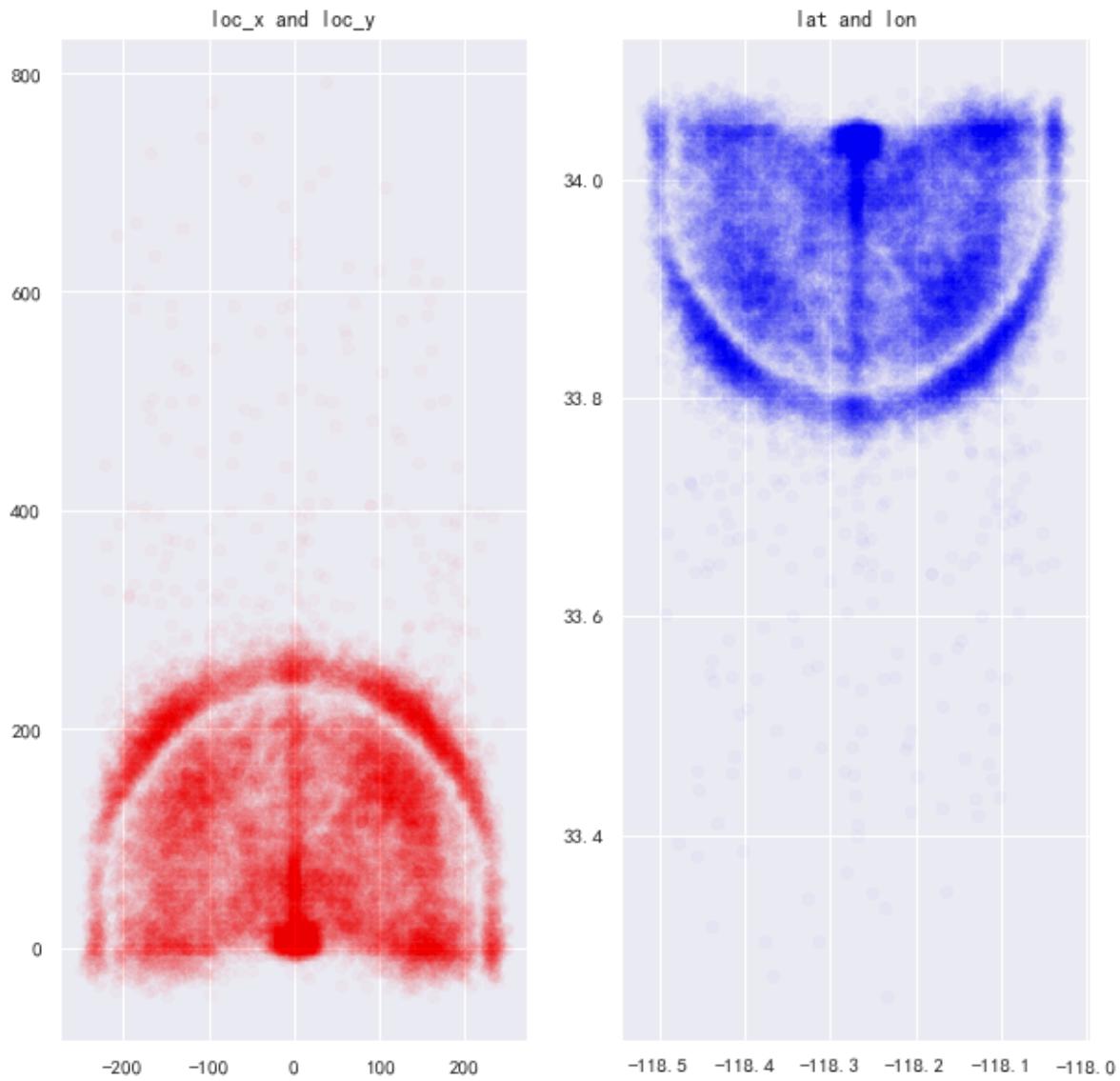


从图中可以得到如下信息：

- id特征展示参加比赛的场次标号，属于标签属性。
- 位置信息重复，表现为loc_x和lon属性重复，loc_y和lat属性重复。
- 场次为5的实例很少，但根据常识可知不是异常值。
- 季后赛占比较低。
- 选择合适的可视化图表（至少三种）实现数据集的可视化。

出手位置

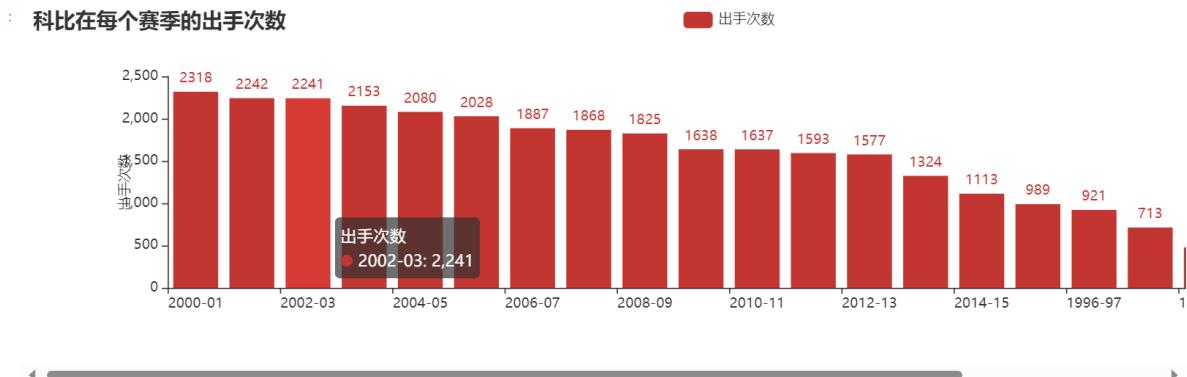
散点图的密度展示了在不同地方投篮次数的多少，越密集的地方表示该位置出手的次数越多。



出手次数

由于pyecharts 库有着良好的交互性，精巧的图表设计，我们选择用这个库函数来制作大部分图表以便分析者即时交互的查看数据，其交互功能在此无法展示，您可以通过运行代码来感受。

我们对科比在每个赛季出手的次数做一个统计，按照出手次数的多少降序排列，我们对season（赛季）这一栏进行统计，查看有多少个赛季，之后将各个赛季出手次数统计并用柱状图绘制出来。

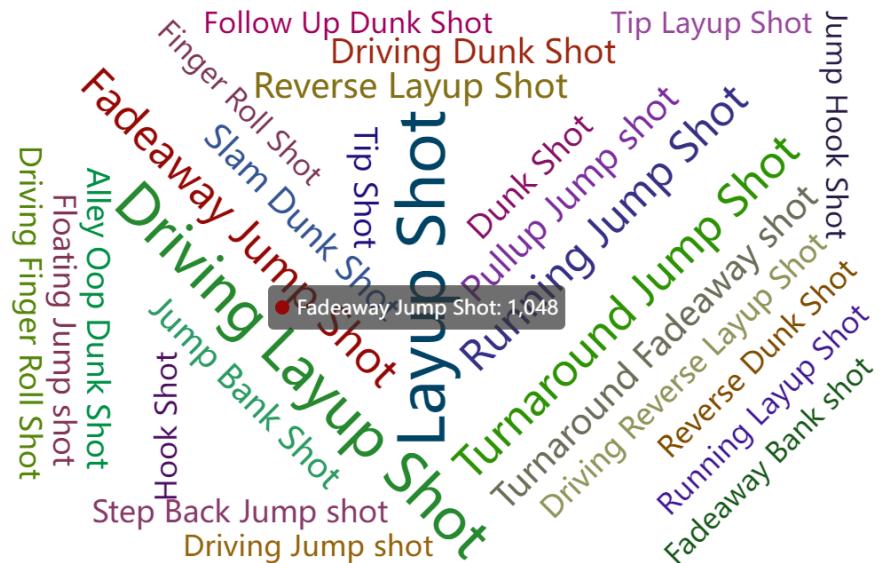


可以看到科比出手次数最多的赛季是2000年（01号赛季）随后逐年以微小的幅度递减，出手次数最少的时期则集中于他刚刚进入NBA联盟的1996-1999年。对比客观事实来看，1996-1997赛季，科比是NBA新人经常作为替补，出场机会不高，并且因为左边的髌屈肌拉伤和流感缺席两场比赛，1997-1998赛季，科比因脚踝扭伤缺席了3场比赛。随后科比经历了2012-2013赛季跟腱断裂，2013-2014赛季膝盖骨折，2014-2015赛季旋转轴肌腱撕裂。从整体来看，在每个赛季中科比出手的次数都是非常多的，可以说明科比是球队里的主力之一。

动作类型

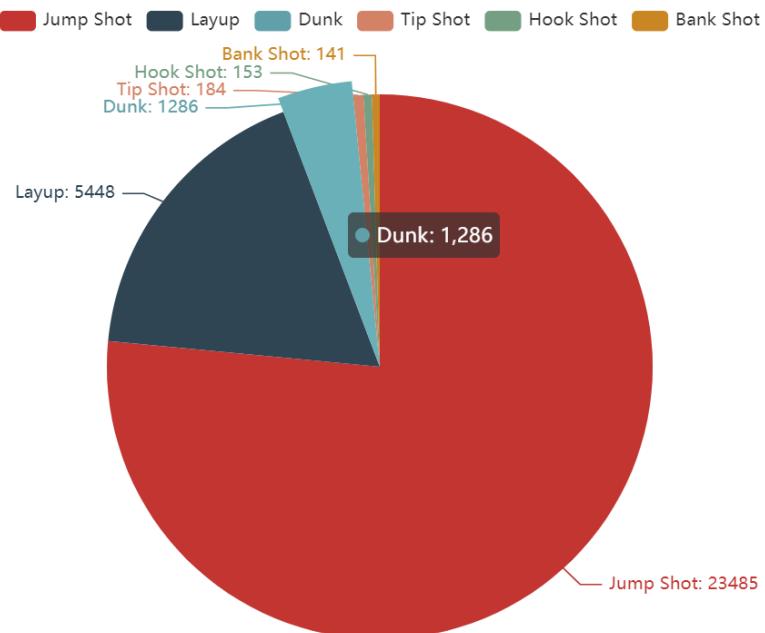
对于展示科比动作类型的属性action_type，我们为其做词云图。在科比如此繁多的动作中，使用最多的是Layup Shot（上篮），共有2567次，其次是Driving Layup Shot（突破上篮），共有1978次。您可以在代码这种使用我们的交互式图表来实时查看任意动作出现的次数。

WordCloud_action_type

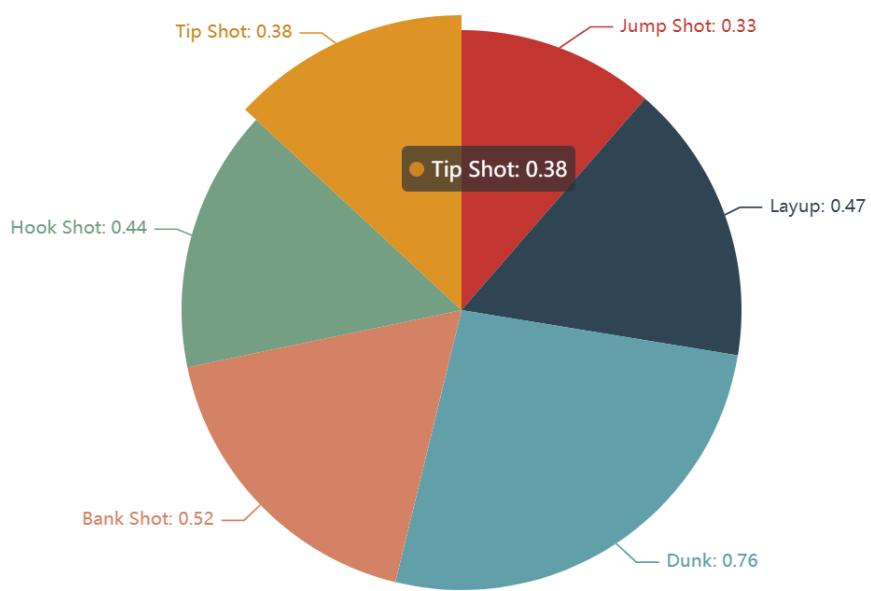


接着对科比的进攻方式与出手命中率做统计，将每次出手的方式combined_shot_type统计出来，绘制一个饼图，再对每次出手方式不同出手是否命中做一个饼图，观察进攻进攻方式与命中率。

科比进攻的方式



各种进攻方式命中率

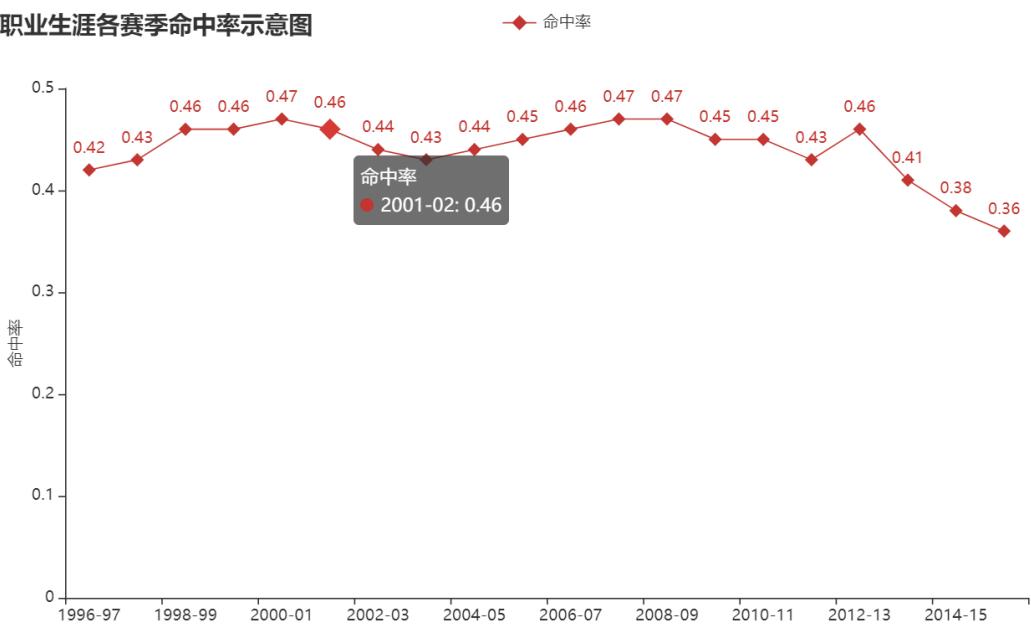


从图中可以看出，科比最常采用的进攻类型是Jump Shot（跳投），如我们熟悉的翻身跳投，后仰和急停干拔都属于跳投的范畴，其次常采用的是Layup（上篮），而命中率最高的动作类型是Dunk（扣篮）和Bank Shot（擦板）。

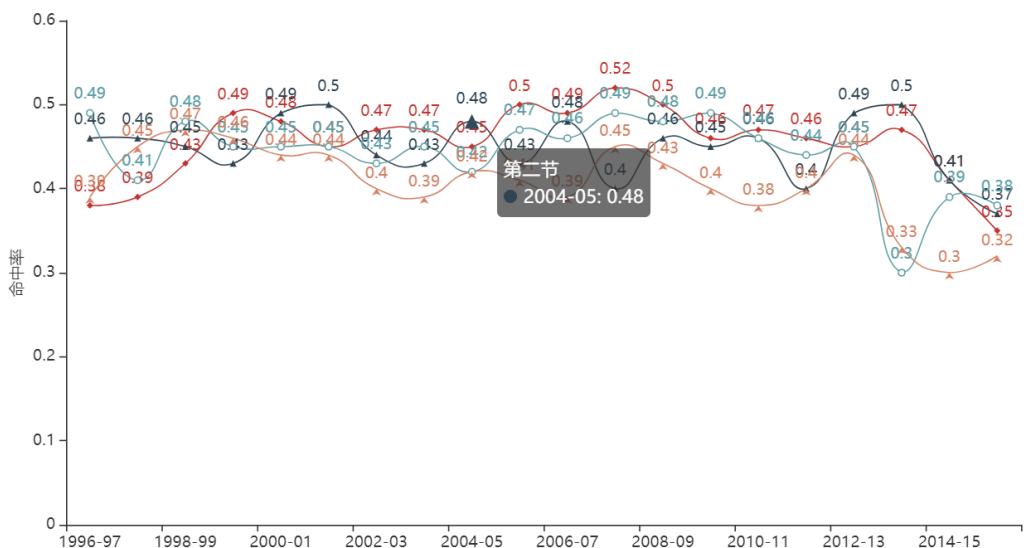
命中率

科比投篮命中率在前五年呈现上涨趋势，中间部分因为伤病原因有所波动，在经过2013年的跟腱断裂后命中率逐年走低。

科比职业生涯各赛季命中率示意图

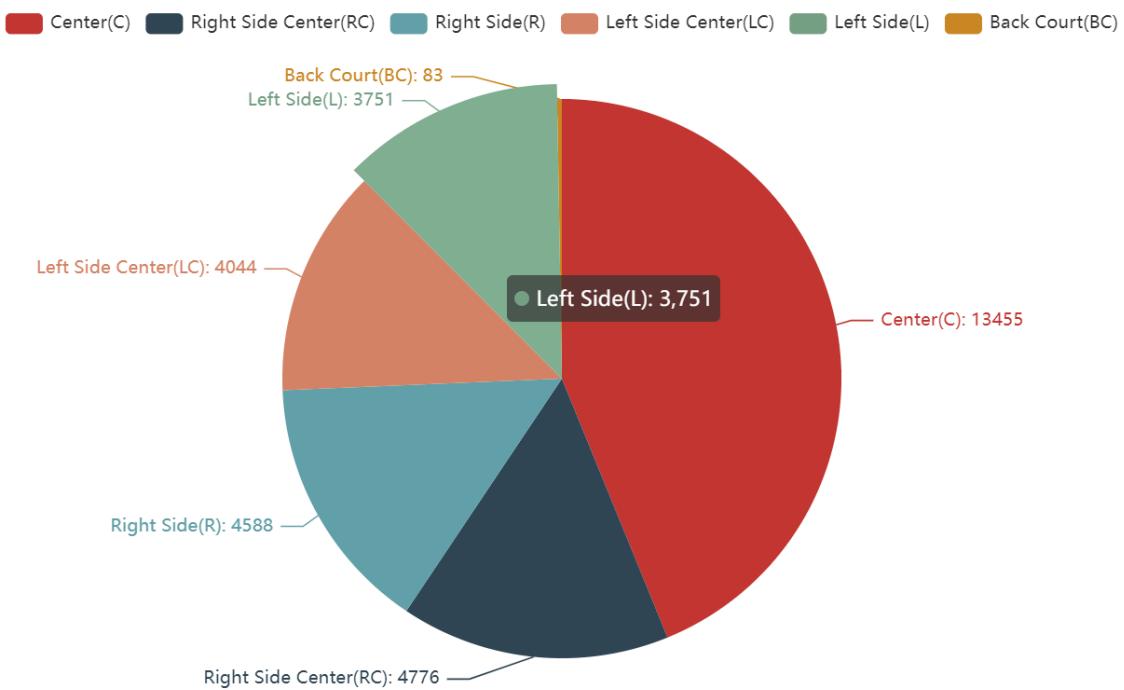


科比职业生涯各节命中率示意图



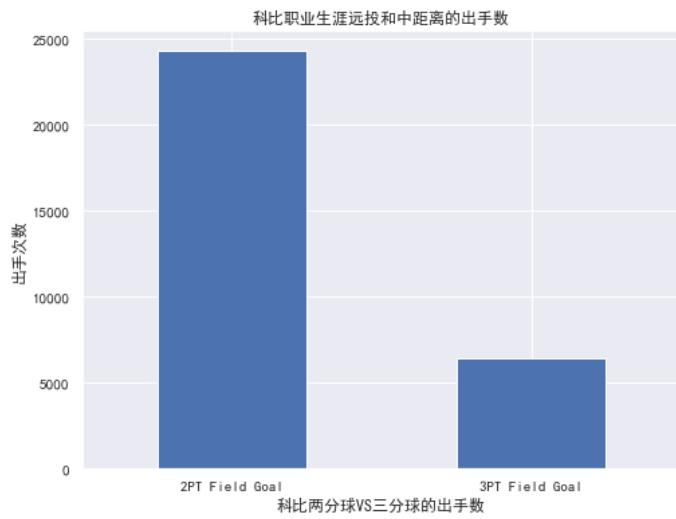
可以看出科比在第四节的命中率最低，这也可以说说明体力对科比投篮命中率有很大的影响。

出手位置



大部分的出手区域在中心位置，左右出手位置的次数基本持平。

科比远投和中距离的出手数：

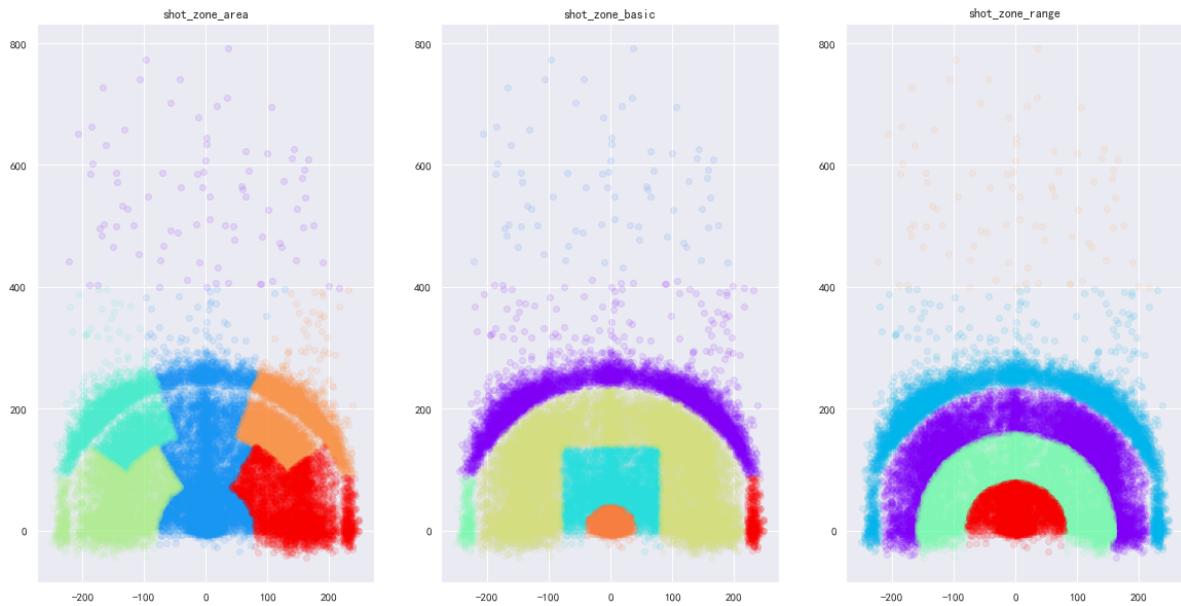


由上图可以看出，科比的出手主要以中距离进攻为主，像我们比较熟悉的急停跳投，翻身跳投，干拔跳投等。

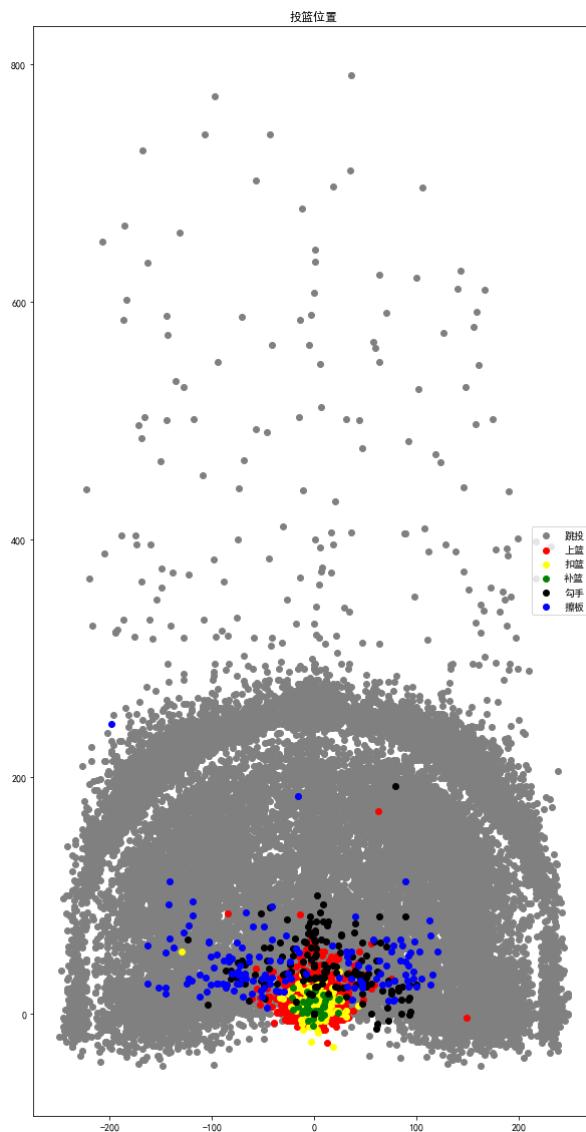
数据集中的shot_zone_area, shot_zone_basic, shot_zone_range三个属性可以看作一个属性，都是代表投球位置的。这里将三个属性都按具体位置的类别绘制散点图。对比NBA真实赛场：



第一个图是shot_zone_area属性的，图中红色和橙色是右侧投球区，墨绿色和绿色是左侧投球区，蓝色是中间投球区；第二个图是shot_zone_basic属性的，图中橙色是篮板底下、蓝色是除进攻有理区（合理冲撞区：站在圆弧线以内，只要防守队员与进攻队员有身体接触，都可能会判防守犯规。而在线外，防守队员则可以提前站好位，如果进攻队员用身体接触了防守队员，会判进攻队员撞人）外的禁区，紫色是底线之外的三分，红色是右边底线三分，绿色是左边底线三分；第三个图是shot_zone_range属性的，图中红色是除进攻有理区外的禁区，绿色是进攻有理区，紫色是中距离，蓝色是三分线外。这三个图都是对科比投球位置的一个统计。



我们选取combined_shot_type属性中各个出手方式，有跳投、上篮、扣篮、补篮、勾手、擦板，不同的出手方式以不同的颜色通过散点图的方式将出手位置在图上绘制出来，得到以下散点图。从图中可以看出篮球场大致的外观，其中红黄绿聚集的地方是球框底下，这里适合上篮、扣篮和补篮，再看灰色点跳投，灰色点之间存在一条明显的分界线，而这条分界线就是三分线，科比出手的位置可以说是遍布整个球场，这也说明科比出手最多的方式是跳投。



至此，数据可视化的任务基本完成。

内容2：数据清洗

从各种渠道获得的源数据大多是“脏”数据，不符合人们的需求，如数据中含有唯一数据或重复数据、异常值数据，以及数据不完整等。而我们在使用数据的过程中对数据的要求是具有一致性、准确性、完整性、时效性、可信性、可解释性，因此对数据集进行清洗。

1. 特征选择与构建

- 特征选择

特征选择的降维方式好处是可以保留原有维度特征的基础上进行降维，既能满足后续数据处理和建模需求，又能保留维度原本的业务含义，以便于业务理解和应用。对于业务分析性的应用而言，模型的可理解性和可用性很多时候要有限于模型本身的准确率、效率等技术指标。例如，决策树得到的特征规则，可以作为选择用户样本的基础条件，而这些特征规则便是基于输入的维度产生。

我们根据一定的规则和经验，直接在原有的维度中挑选一部分参与到计算和建模过程，用选择的特征代替所有特征，不改变原有特征，也不产生新的特征值，这里我们主要通过人工选择删除对最终分析结果无影响的特征。

```
#删除对最终结果无影响的id特征
drop_ids = ['game_event_id', 'game_id', 'team_id', 'shot_id']
for feature in drop_ids:
    raw = raw.drop(feature, axis = 1)

#lat, lon, loc_x, loc_y表达的是相同的含义，删除lat, lon特征
raw = raw.drop(['lat', 'lon'], axis = 1)

#action_type和combined_shot_type表达的含义相近，删除action_type
raw = raw.drop(['action_type'], axis = 1)

#shot_zone_area, shot_zone_basic, shot_zone_range表达的是相同的含义，保留一个
raw = raw.drop(['shot_zone_basic', 'shot_zone_range'], axis = 1)

#team_name和game_date对最终结果没有影响，删除这两个特征
raw = raw.drop(['team_name', 'game_date'], axis = 1)

#matchup和opponent表达的是相同的意思，保留opponent
raw = raw.drop('matchup', axis = 1)
```

- 特征构建

数据聚合指的是我们将两个或多个对象的值组合为单个对象，可以用于初步构建新的特征。好处有：(1)减少要处理的数据的大小(2)改变分析的粒度(从细尺度到粗尺度)(3)提高数据的稳定性。

```
In [3]: #创建一个新的特征time_remaining，用于替代minutes_remaining和seconds_remaining
raw['time_remaining'] = raw['minutes_remaining']*60 + raw['seconds_remaining']

In [4]: #删除minutes_remaining和seconds_remaining特征
raw = raw.drop(['minutes_remaining', 'seconds_remaining'], axis = 1)
```

查看经过特征选择与构建后的数据集：

```
1 <class 'pandas.core.frame.DataFrame'>
2 RangeIndex: 30697 entries, 0 to 30696
3 Data columns (total 12 columns):
4 #   Column           Non-Null Count  Dtype  
5 ---  -- 
6 0   combined_shot_type 30697 non-null   object 
7 1   loc_x              30697 non-null   int64  
8 2   loc_y              30697 non-null   int64  
9 3   period             30697 non-null   int64
```

```

10   4  playoffs           30697 non-null  int64
11   5  season            30697 non-null  object
12   6  shot_distance     30697 non-null  int64
13   7  shot_made_flag    25697 non-null  float64
14   8  shot_type          30697 non-null  object
15   9  shot_zone_area    30697 non-null  object
16  10  opponent           30697 non-null  object
17  11  time_remaining    30697 non-null  int64
18 dtypes: float64(1), int64(6), object(5)
19 memory usage: 2.8+ MB

```

2. 缺失值处理

经过上一步的特征选择与构建，现在数据集有8个数值属性，4个object属性，30697条数据。我们发现存在缺失值的属性是shot_made_flag，由于这个属性是“是否成功投篮”的目标属性，不能进行简单的填补，因此这里我们选择丢弃该属性，行数由原来的30697条减少到25687条。

```

1 | Number of rows in original data = 30697
2 | Number of rows after discarding missing values = 25697

```

3. 异常值识别和处理

离群点(Outlier)属于观测量，既有可能是真实数据产生的，也有可能是噪声带来的，但是总的来说是和大部分观测量之间有明显不同的观测值。为了丢弃异常值，我们可以计算每个属性的Z值，并删除那些包含异常高或低Z值属性的实例（例如，如果 $Z > 3$ 或 $Z \leq -3$ ）。z-score 的计算定义如下：

$$z = (x - \mu) / \sigma$$

这里的 x 为原始分值， z 为经过转换后的z-score， μ 为总体样本空间的分值均值， σ 则为总体样本空间的标准差。去除 $Z > 3$ 或者 $Z \leq -3$ 的属性值后，行数由原来的25697减少为23612条。但是对于偏态数据用分箱则效果更好。

4. 噪声识别和处理

噪声是指被观测的变量的随机误差或方差。

分箱方法是一种简单常用的预处理方法，通过考察相邻数据来确定最终值。按照属性值划分的子区间，如果一个属性值处于某个子区间范围内，就称把该属性值放进这个子区间所代表的“箱子”内。把待处理的数据（某列属性值）按照一定的规则放进一些箱子中，考察每一个箱子中的数据，采用某种方法分别对各个箱子中的数据进行处理。

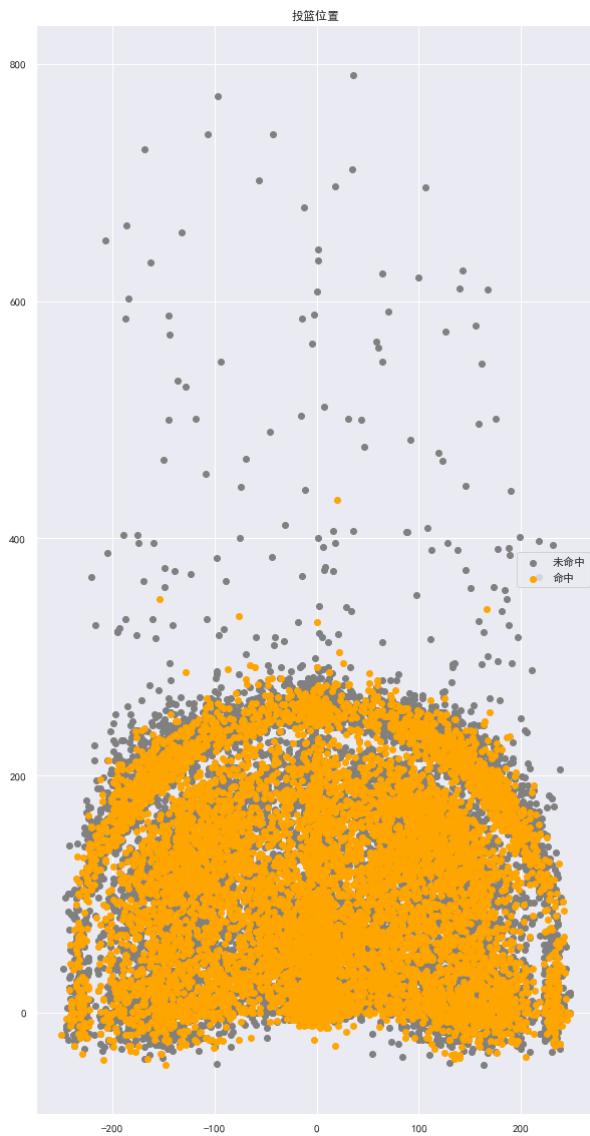
采用分箱时要确定的两个问题是：

- 如何分箱；
- 如何对每个箱子中的数据进行平滑处理。

方法：等深分箱法、等宽分箱法、用户自定义区间法和最小熵法。

等宽分箱法容易受异常值影响，分箱后可能分配不均匀。等深/等频分箱法则容易将相同的值分到不同的箱中。

但是经过之前的可视化对于数据集进行探索性分析后我们发现，数据集中的数据都是合理的，投篮距离中少部分的“离群点”和是否命中的属性值之间有着一定的联系，并且经人工核对是真实存在的数据，不应该作为异常点和噪声来进行舍弃，因此我们可以选择保留所有的数据。



5.数据去重

重复数据指多次出现的数据。若重复数据在整体样本中所占权重比其他数据大，容易导致结果的倾向性，因此对于重复数据常用的预处理方法是剔除，或者按比例降低其权重，进行数据的重新布局，形成概率分布。

- 对于一般数量可控的重复数据，通常采用的方法是简单的比较算法剔除。
- 对于重复的可控数据而言，一般通过代码实现对信息的匹配比较，进而确定剔除不需要的数据。

我们发现数据共有3行重复，经过数据去重后，行数由23612条减少到23609条。

```

1 | Number of duplicate rows = 3
2 | Number of rows before discarding duplicates = 25697
3 | Number of rows after discarding duplicates = 25694

```

6.特征扩充

独热编码：我们对combined_shot_type, shot_type, shot_zone_area, opponent, season这五个个标称属性做独热编码。离散特征通过one-hot编码映射到欧式空间，离散特征的某个取值就对应欧式空间的某个点，独热编码解决了分类器不好处理属性数据的问题，它的值只有0和1，不同的类型存储在垂直的空间。当类别的数量很多时，特征空间会变得非常大，我们可以再用主成分分析法来减少维度。

	loc_x	loc_y	period	playoffs	shot_distance	shot_made_flag	time_remaining	combined_shot_type_Bank Shot	combined_shot_type_Dunk	combined_shot_type_Hc Si
1	-157	0	1	0	15	0.0	622	0	0	
2	-101	135	1	0	16	1.0	465	0	0	
3	138	175	1	0	22	0.0	412	0	0	
4	0	0	2	0	0	1.0	379	0	1	
5	-145	-11	3	0	14	0.0	572	0	0	

5 rows × 74 columns

至此我们对科比数据集的数据清洗基本完成，属性个数由12个增加到74个，行数由30697条减少到23609条。

7.简述PCA原理并使用PCA对属性做主成分分析

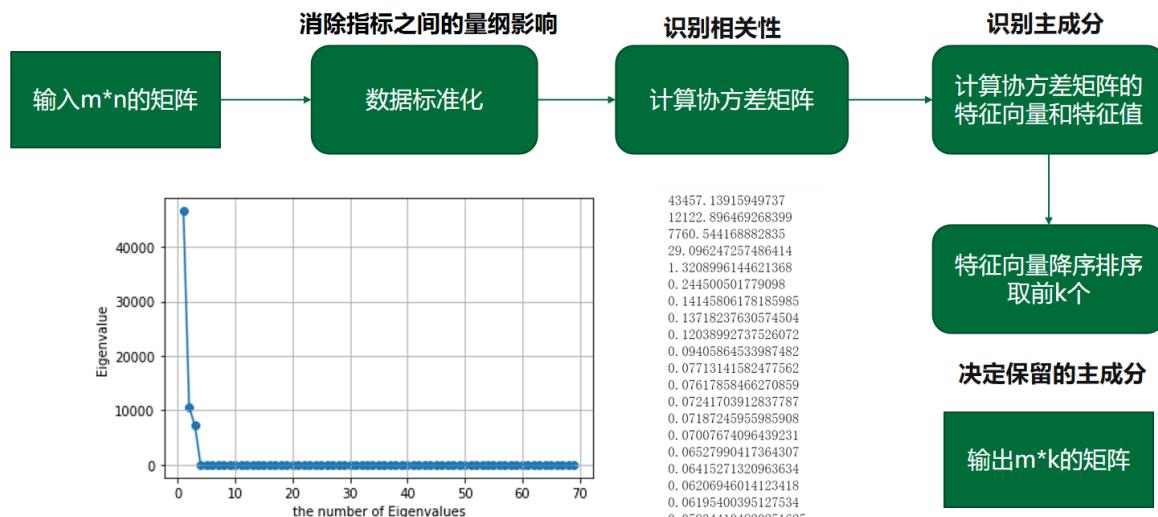
PCA (Principal Component Analysis) 是一种常用的数据降维方法。PCA通过线性变换将原始数据变换为一组各维度线性无关的表示，将数据从原来的高维空间投影到低维空间来减少数据中属性数量，可用于提取数据的主要特征分量。由PCA创建的新属性(也称为分量)具有以下属性：

(1)它们是原始属性的线性组合

(2)它们彼此正交(垂直)

(3)它们捕获数据中最大的变化量。

从原始数据集中我们选择一个子集来做主成分分析：



过程如图所示，其中我们打印出协方差矩阵由大到小的特征值序列：

```

1 43457.13915949737
2 12122.896469268399
3 7760.544168882835
4 29.096247257486414
5 1.3208996144621368
6 0.244500501779098
7 0.14145806178185985
8 0.13718237630574504
9 0.12038992737526072
10 0.09405864533987482
11 0.07713141582477562
12 0.07617858466270859
13 0.07241703912837787
14 0.07187245955985908
15 0.07007674096439231
16 0.06527990417364307
17 0.06415271320963634

```

据此我们可以选择降序排序的前3个特征向量为主成分。

内容3：异常检测

一、异常检测的相关原理及概念

在数据挖掘中，异常检测（离群点检测）是对不符合预期模式或数据集中其他项目的项目、事件或观测值的识别。通常异常项目会转变成银行欺诈、结构缺陷、医疗问题、文本错误等类型的问题。

假定给定一个统计过程来产生数据对象集，离群点是一个数据对象，它显著不同于其他数据对象，好像它是被不同的机制产生的一样。特别是在检测滥用与网络入侵时，有趣性对象往往不是罕见对象，但却是超出预料的突发活动。这种模式不遵循通常统计定义中把异常点看作是罕见对象，于是许多异常检测方法（特别是无监督的方法）将对此类数据失效，除非进行了合适的聚集。相反，聚类分析算法可能可以检测出这些模式形成的微聚类。

离群点可以分为3类：

- 全局离群点，它显著的偏离数据集中的其余对象，是最简单的一类离群点。
- 情境离群点，如果关于对象的特定情境，它显著的偏离其他对象。
- 集体离群点，数据对象的一个子集作为整体显著偏离整个数据集。

有三大类异常检测方法：

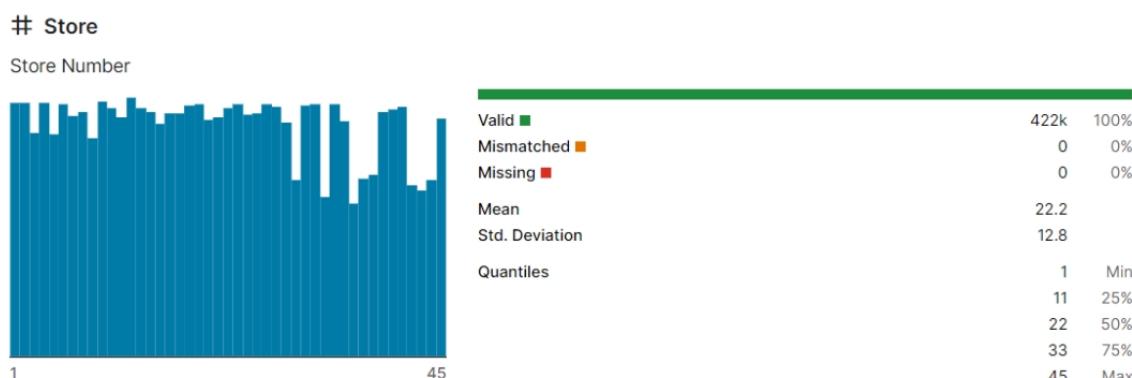
- 在假设数据集中大多数实例都是正常的前提下，**无监督**异常检测方法能通过寻找与其他数据最不匹配的实例来检测出未标记测试数据的异常。
- 监督式**异常检测方法需要一个已经被标记“正常”与“异常”的数据集，并涉及到训练分类器（与许多其他的统计分类问题的关键区别是异常检测的内在不均衡性）。
- 半监督式**异常检测方法根据一个给定的正常训练数据集创建一个表示正常行为的模型，然后检测由学习模型生成的测试实例的可能性。

二、算法描述与相关案例

数据集描述：

美国领先的零售商店之一沃尔玛希望准确预测销售额以及消费者需求，某些事件和假期对每周的销售的影响。现在有沃尔玛 45 家门店的销售数据以及一些颇具参考意义的属性，同样其中也不乏出现异常销量的异常点。

其中我们需要的所有特征的数据内容如下所示：



📅 Date

Weekly Date



Valid	422k	100%
Mismatched	0	0%
Missing	0	0%
Minimum	5Feb10	
Mean	18Jun11	
Maximum	26Oct12	

IsHoliday

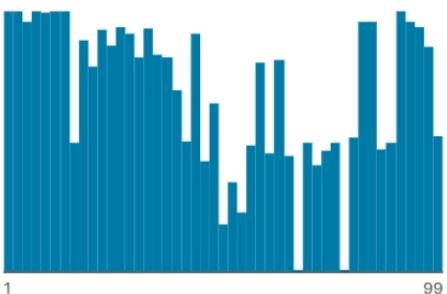
Holiday IF True = 1 if else = 0



Valid	422k	100%
Mismatched	0	0%
Missing	0	0%
Mean	0.07	
Std. Deviation	0.26	
Quantiles	0	Min
	0	25%
	0	50%
	0	75%
	1	Max

Dept

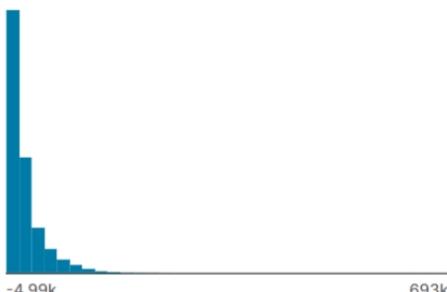
Department Number in each store



Valid	422k	100%
Mismatched	0	0%
Missing	0	0%
Mean	44.3	
Std. Deviation	30.5	
Quantiles	1	Min
	18	25%
	37	50%
	74	75%
	99	Max

Weekly_Sales

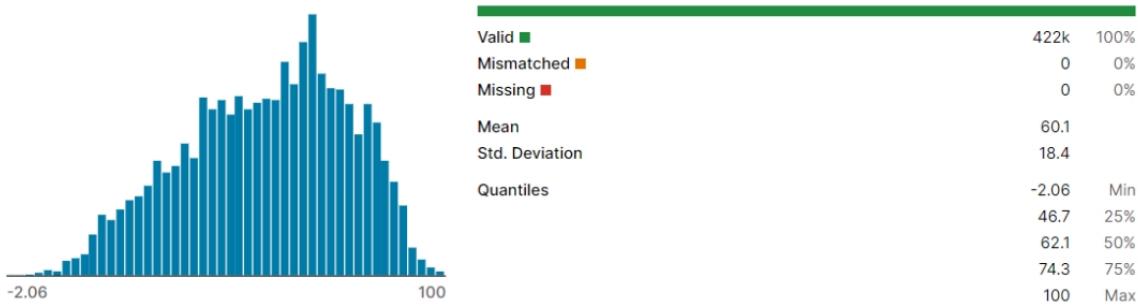
In Dollars



Valid	422k	100%
Mismatched	0	0%
Missing	0	0%
Mean	16k	
Std. Deviation	22.7k	
Quantiles	-4.99k	Min
	2.08k	25%
	7.61k	50%
	20.2k	75%
	693k	Max

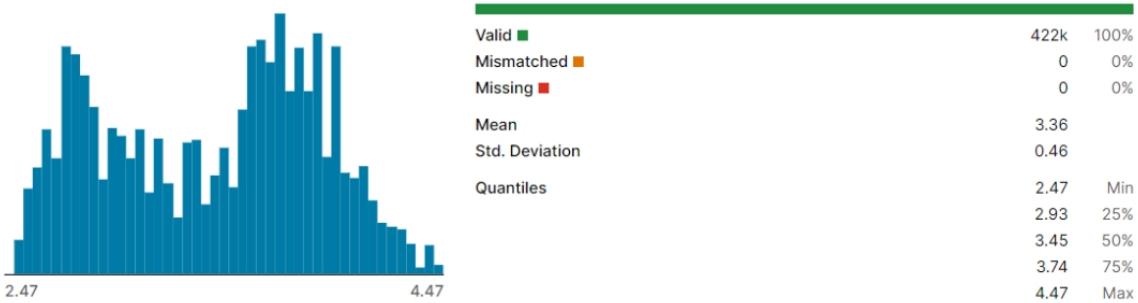
Temperature

In Degree Celsius



Fuel_Price

In Dollars



1) 孤立森林 (Isolation Forrest)

孤立森林是一种无监督的异常检测算法。它使用隔离（数据点与其余数据的距离）检测异常，而不是对正常点进行建模。2007年，它最初由Fei Tony Liu开发，作为他博士研究的原始想法之一。这项研究的意义在于它偏离了当时支持大多数现有异常检测器的主流哲学，在主流哲学中，所有的正常实例都在异常被识别为不符合正常实例分布的实例之前被分析。隔离森林引入了一种不同的方法，该方法使用二叉树显式隔离异常，展示了一种更快的异常检测器的新可能性，该检测器直接针对异常而不分析所有正常实例。该算法具有线性时间复杂度、低常数和低内存需求，适用于大量数据。

使用隔离森林进行异常检测是一个由两个主要阶段组成的过程：

1.训练数据集用于构建 iTree。

2.测试集中的每个实例都通过前一阶段的 iTrees 构建，并使用下面描述的算法为实例分配适当的“异常分数”。

一旦测试集中的所有实例都被分配了一个异常分数，就可以将分数大于预定义阈值的任何点标记为“异常”，这取决于应用分析的域。

异常值的计算方法为：

$$s(x, m) = 2^{\frac{-E(h(x))}{c(m)}}$$

其中 x 是节点， m 是样本的大小， $E(h(x))$ 是平均树根到节点 x 的路径长度，

$$c(m) = \begin{cases} H(m-1) - \frac{2(m-1)}{n}, & m > 2 \\ 1, & m = 2 \\ 0, & \text{otherwise} \end{cases}$$

异常值 s 如果接近 1，那么它就很有可能是异常值；如果 s 在 0.5 附近，那么很可能是一个正常值；若一个数据集内所有点的异常值 s 都集中在大约 0.5 左右，那么该数据集没有异常。

伪代码：

Algorithm 1 : $iForest(X, t, \psi)$

Inputs: X - input data, t - number of trees, ψ - sub-sampling size

Output: a set of t $iTrees$

```

1: Initialize  $Forest$ 
2: set height limit  $l = ceiling(\log_2 \psi)$ 
3: for  $i = 1$  to  $t$  do
4:    $X' \leftarrow sample(X, \psi)$ 
5:    $Forest \leftarrow Forest \cup iTree(X', 0, l)$ 
6: end for
7: return  $Forest$ 
```

Algorithm 2 : $iTree(X, e, l)$

Inputs: X - input data, e - current tree height, l - height limit

Output: an $iTree$

```

1: if  $e \geq l$  or  $|X| \leq 1$  then
2:   return  $exNode\{Size \leftarrow |X|\}$ 
3: else
4:   let  $Q$  be a list of attributes in  $X$ 
5:   randomly select an attribute  $q \in Q$ 
6:   randomly select a split point  $p$  from  $max$  and  $min$  values of attribute  $q$  in  $X$ 
7:    $X_l \leftarrow filter(X, q < p)$ 
8:    $X_r \leftarrow filter(X, q \geq p)$ 
9:   return  $inNode\{Left \leftarrow iTree(X_l, e + 1, l),$ 
10:                      $Right \leftarrow iTree(X_r, e + 1, l),$ 
11:                      $SplitAtt \leftarrow q,$ 
12:                      $SplitValue \leftarrow p\}$ 
13: end if
```

应用场景：

孤立森林算法内存要求低，处理速度快，时间复杂度是线性的，可以处理高维数据和大数据，可以进行在线预测；即能发现群异常数据，也能发现散点异常数据，也能处理训练数据中不包含异常数据的情况。目前在工业界的应用范围比较广。常见的场景包括：网络安全中的攻击检测、金融交易欺诈检测、疾病侦测、噪声数据过滤（数据清洗）等。

在python中实现孤立森林：

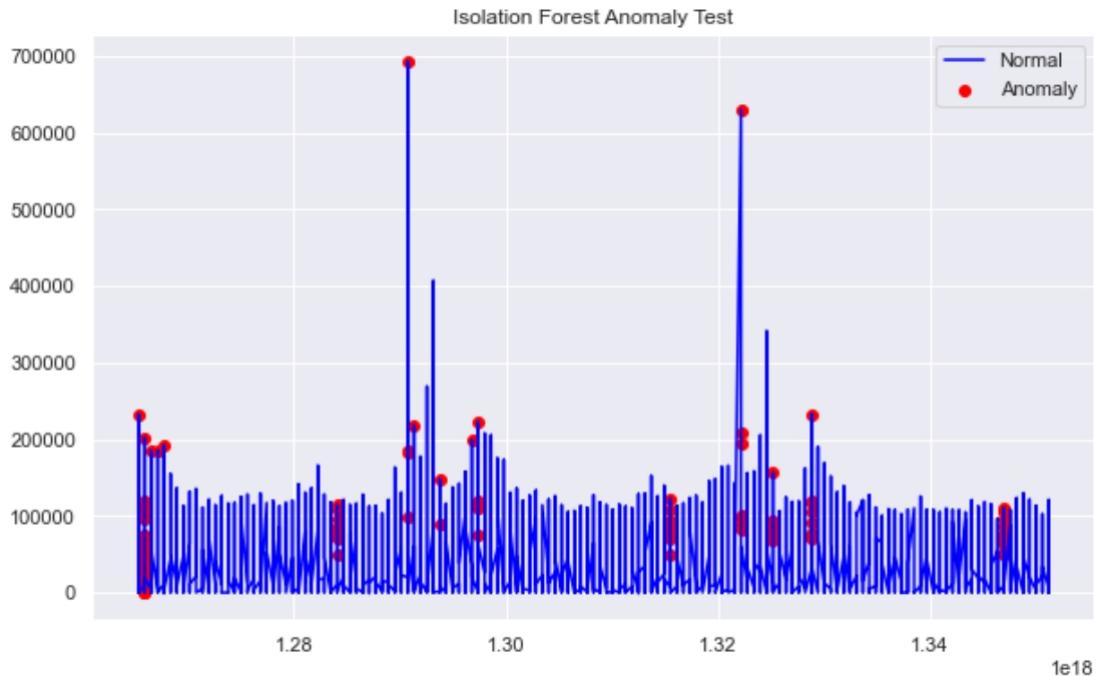
```

data = df[['IsHoliday', 'Dept', 'Weekly_Sales', 'Temperature', 'Fuel_Price', 'Unemployment']]
scaler = StandardScaler()
np_scaled = scaler.fit_transform(data)
data = pd.DataFrame(np_scaled)
isoforest = IsolationForest(n_estimators=100, max_samples='auto',
                            contamination=outliers_fraction, max_features=6)

df['anomaly_iso'] = isoforest.fit_predict(data)
```

其中n_estimators代表*ITree*的数量，contamination=outlier_fraction代表异常点的比例，max_features指选取特征值的数量；最后我们按照时间序列作图。

结果如下图所示：



异常点产生的原因在于孤立森林不同于分类、聚类等算法，它基于一个事实就是异常点很少且很容易被分离出来；图中这些异常点的来源也就是在进行特征分类时，按照我们设想的异常点比例，被更早或者更快区分离出来的点。

2) OneClassSVM

sklearn提供了一些机器学习方法，可用于奇异（Novelty）点或者异常（Outlier）点检测，包括OneClassSVM, Isolation Forest, Local Outlier Factor (LOF) 等，其中OneClassSVM可以用于Novelty Detection，而后两者可用于Outlier Detection。

严格来说，OneClassSVM不是一种异常点检测方法，而是一种奇异点检测方法：它的训练集不应该掺杂异常点，因为模型可能会去匹配这些异常点。但在数据维度很高，或者对相关数据分布没有任何假设的情况下，OneClassSVM也可以作为一种很好的异常点检测方法。

在one-class classification中，仅仅只有一类的信息是可以用于训练，其他类别的（总称outlier）信息是缺失的，也就是区分两个类别的边界线是通过仅有的一类数据的信息学习得到的。

其算法的主要步骤为：假设产生的超球体参数为中心 o 和对应的超球体半径 $r > 0$ ，超球体体积 $V(r)$ 被最小化，中心 o 是支持向量的线性组合；跟传统SVM方法相似，可以要求所有训练数据点 x_i 到中心的距离严格小于 r 。但是同时构造一个惩罚系数为 C 的松弛变量 ξ_i ，优化问题如下所示：

$$\begin{aligned} & \underset{r, o}{\min} V(r) + C \sum_{i=1}^m \xi_i \\ & \|x_i - o\|_2 \leq r + \xi_i, i = 1, 2, 3 \dots m \\ & \xi_i \geq 0, i = 1, 2, \dots m \end{aligned}$$

采用拉格朗日对偶求解之后，可以判断新的数据点 z 是否在内，如果 z 到中心的距离小于或者等于半径 r ，则不是异常点，如果在超球体以外，则是异常点。**由于OneClassSVM需要正常的数据集来进行训练，参考一些专家对于该数据集异常检测的建议，我们手动标注训练集中Weekly_Sales中大于350k的即为异常数据。**

应用场景：

OneClassSVM有能力捕获数据集的形状，因此对于强非高斯数据有更加优秀的效果，例如两个截然分开的数据集。严格来说，一分类的SVM并不是一个异常点监测算法，而是一个奇异点检测算法：它的训练集不能包含异常样本，否则的话，可能在训练时影响边界的选取。但是，对于高维空间中的样本数据集，如果它们做不出有关分布特点的假设，OneClassSVM将是一大利器。典型的案例有：信用卡贷款中的违约客户、肿瘤检测中的恶性肿瘤预测、产品质量检验中的残次品率等等。

在python中实现OneClassSVM的主要方法如下图所示：

```
data = df[['IsHoliday', 'Dept', 'Weekly_Sales', 'Temperature', 'Fuel_Price', 'Unemployment']]
data = data.loc[data['Weekly_Sales']<=350000]
scaler = StandardScaler()
np_scaled = scaler.fit_transform(data)
data = pd.DataFrame(np_scaled)
# train oneclassSVM
svm = OneClassSVM(nu=outliers_fraction, kernel="rbf", gamma=0.01)
svm.fit(data)
data = df[['IsHoliday', 'Dept', 'Weekly_Sales', 'Temperature', 'Fuel_Price', 'Unemployment']]
scaler = StandardScaler()
np_scaled = scaler.fit_transform(data)
data = pd.DataFrame(np_scaled)
df['anomaly_svm'] = svm.predict(data)
```

其中nu=outliers_fraction代表我们期望的异常点（奇异点）比例。

kernel是核函数，其中我们比较常用的有：

$$\text{Linear kernel: } k(x, y) = \langle x, y \rangle$$

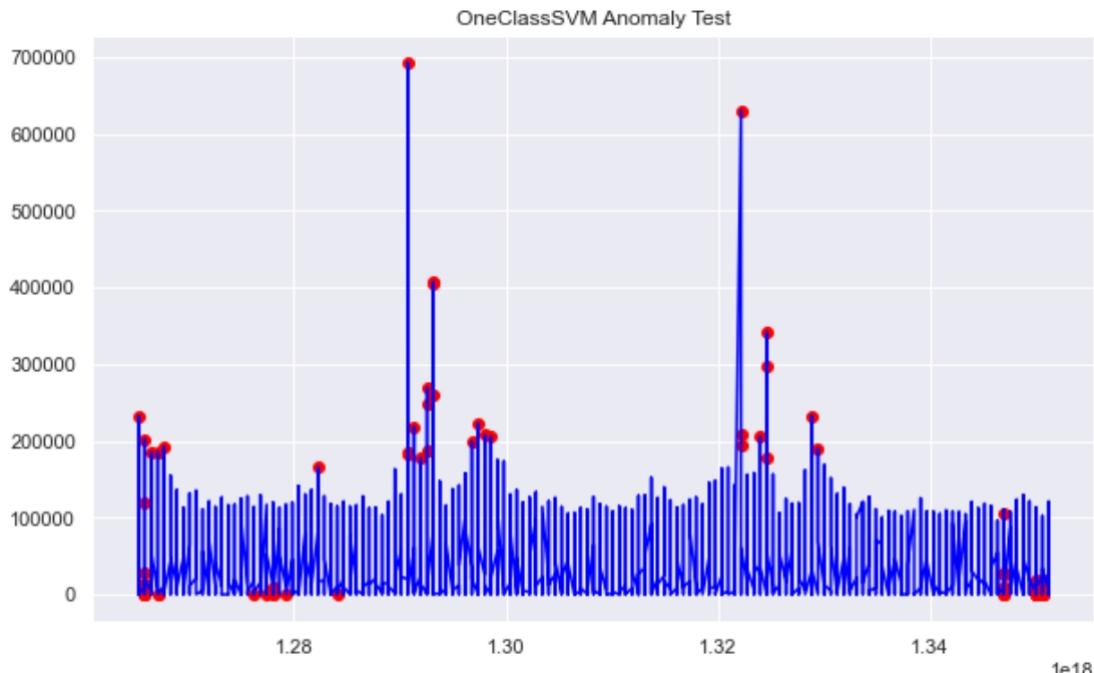
$$\text{Polynomial kernel: } k(x, y) = (\langle x, y \rangle + c)^d$$

$$\text{Gaussian Radial Basis Function kernel (RBF): } k(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}$$
$$d \in \mathbb{Z}^+, \sigma \in \mathcal{R} - \{0\}$$

kernel函数的主要作用是将低维难以分开的特征投影到高维空间，得到与原点距离最大的超平面；同样也有将高维数据投影到低维空间的PCA kernel。这里我们使用的是RBF。

而 γ （代码中的gamma）代表了 RBF Kernel 将样本投影到高维空间的缩放比例， γ 值设定的越小代表样本在高维空间的越分散，即 σ 越大，在训练时的作用可能会造成准确率较低，但预测未知样本的泛化能力强；反之 γ 值设定的越大，样本在高维空间会挤在一起，所获得的 support vectors 就会较少，训练时准确率较高，但预测未知的泛化能力弱。

训练好模型再对数据集时间序列作图之后，我们得到异常检测的结果是：



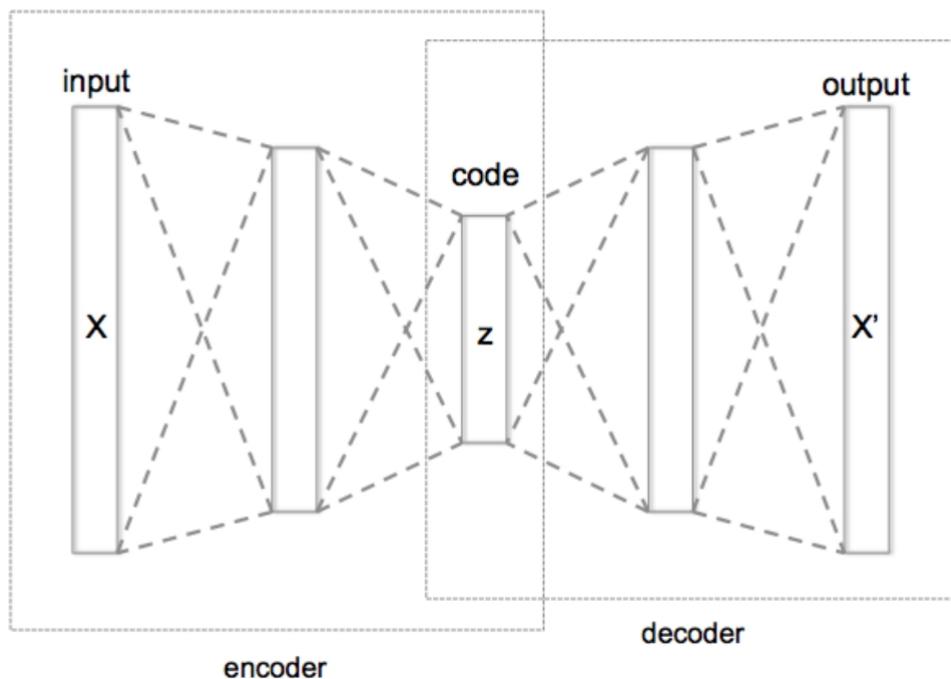
这些异常点产生的原因是OneClassSVM中使用RBF的Kernel函数，将样本投影到高维空间；然后，根据我们的数据集以及期望的异常点的比例，得到一个决策边界；最后将训练好的模型去适用数据集，在决策边界内的点是正常的，在决策边界外的就是异常点。

3) AutoEncoder

Auto-Encoder，中文称作自编码器。它基于反向传播算法与最优化方法（如梯度下降法），利用输入数据本身作为监督，来指导神经网络尝试学习一个映射关系，从而得到一个重构输出。在时间序列异常检测场景下，异常对于正常来说是少数，所以我们认为，如果使用自编码器重构出来的输出跟原始输入的差异超出一定阈值（threshold）的话，原始时间序列即存在了异常。

通过算法模型包含两个主要的部分：Encoder（编码器）和Decoder（解码器）。

编码器的作用是把高维输入 编码成低维的隐变量 从而强迫神经网络学习最有信息量的特征；解码器的作用是把隐藏层的隐变量 还原到初始维度，最好的状态就是解码器的输出能够完美地或者近似恢复出原来的输入，即



如图所示，

(1)从输入层 -> 隐藏层的原始数据X的编码过程：

$$h = g_{\theta_1}(x) = \sigma(W_1x + b_1)$$

(2)从隐藏层 -> 输出层的解码过程：

$$\hat{x} = g_{\theta_2}(x) = \sigma(W_2x + b_2)$$

那么算法的优化目标函数就写为：

$$\text{Minimumloss} = \text{dist}(X, X^R)$$

其中dist为二者的距离度量函数，通常用MSE（均方差）表示。

应用场景：降维，异常检测，图像去噪，图像压缩，图像生成。

我们根据正常数据训练Autoencoder，现在我们有了手动标注好正常数据的数据集，我们用tensorflow来实现AutoEncoder。由于我们输入的数据的特征维度一共是6个，那我们AutoEncoder的编码解码过程为：X->4->2->4->X'，实现的过程如下：

```
class AutoEncoder(Model):
    """
    Parameters
    -----
    output_units: int
        Number of output units

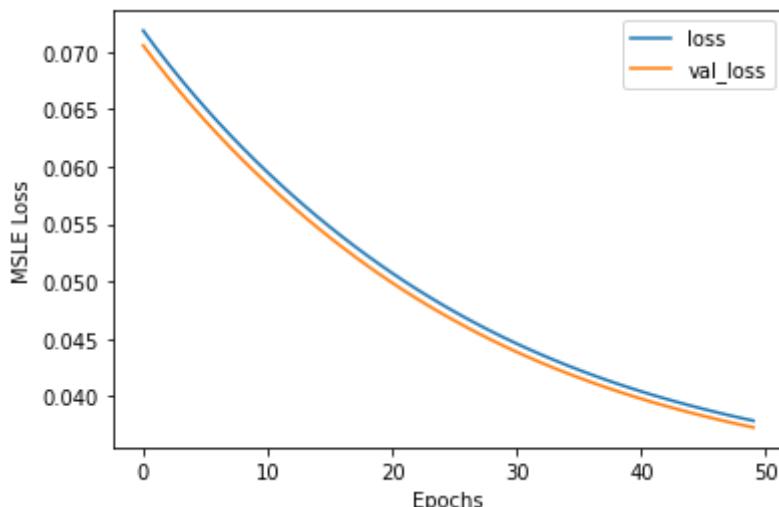
    code_size: int
        Number of units in bottle neck
    """

    def __init__(self, output_units, code_size=8):
        super().__init__()
        self.encoder = Sequential([
            Dense(4, activation='relu'),
            Dropout(0.1),
            Dense(2, activation='relu'),
            Dropout(0.1),
            Dense(code_size, activation='relu')
        ])
        self.decoder = Sequential([
            Dense(2, activation='relu'),
            Dropout(0.1),
            Dense(4, activation='relu'),
            Dropout(0.1),
            Dense(output_units, activation='sigmoid')
        ])

    def call(self, inputs):
        encoded = self.encoder(inputs)
        decoded = self.decoder(encoded)
        return decoded

print(x_train_scaled.shape)
model = AutoEncoder(output_units=x_train_scaled.shape[1])
# configurations of model
model.compile(loss='msle', metrics=['mse'], optimizer='adam')
```

训练完毕后输出loss与value loss曲线：



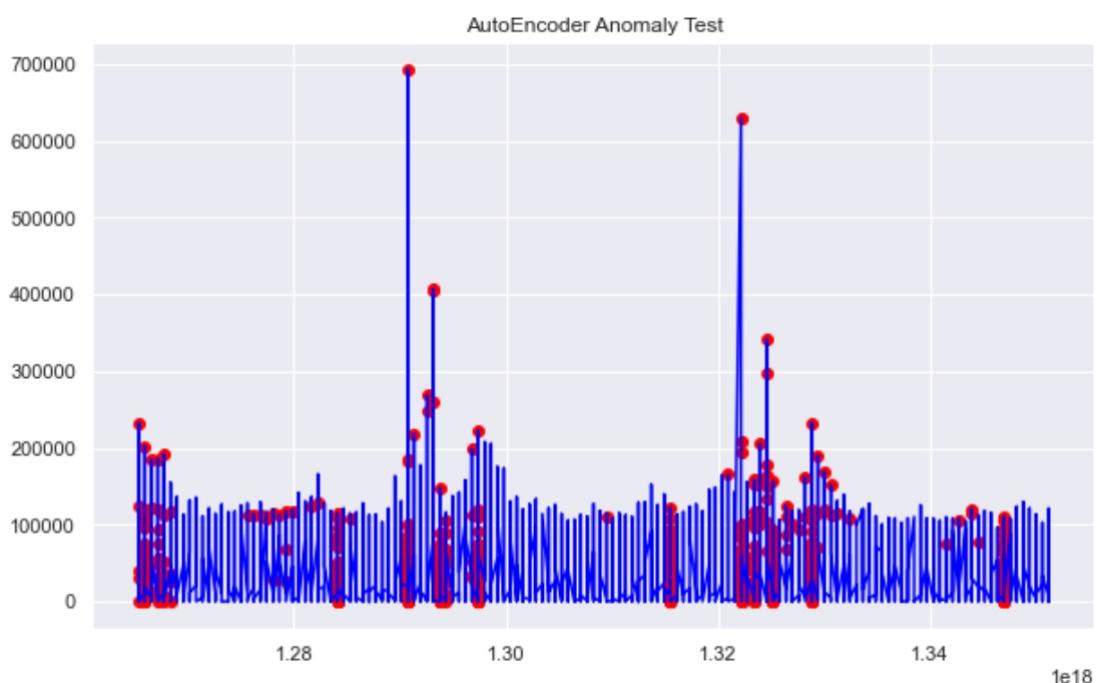
阈值的定义采用重构数据误差的均值+标准差的方法：

```
def find_threshold(model, x_train_scaled):
    reconstructions = model.predict(x_train_scaled)
    # provides losses of individual instances
    reconstruction_errors = tf.keras.losses.msle(reconstructions, x_train_scaled)
    # threshold for anomaly scores
    threshold = np.mean(reconstruction_errors.numpy()) + np.std(reconstruction_errors.numpy())
    return threshold
```

最终训练结果的阈值以及准确率是：

```
Threshold: 0.04124190425336151
65/65 [=====] - 0s 1ms/step
accuracy: 0.911778962675715
323/323 [=====] - 0s 1ms/step
```

将该部分的数据重新用训练好的模型按照时间序列作图：



异常点的检测符合我们的预期，从这里看来训练结果还是比较令人满意的。

4) 异常检测结果的分析比较

孤立森林：无监督方法，孤立森林的思想是基于异常点很少，而且容易分离，这也就意味着孤立森林对于全局稀疏点有着优秀的识别率；但这也意味着孤立森林无法对局部稀疏点敏感；其次对于维度很高的数据，由于孤立森林切割数据是随机选取特征，可能数据都被孤立之后仍有特征未被使用，可靠性在高维下有所降低。

OneClassSVM：半监督方法，OneClassSVM透过这些正常样本的特征去学习一个决策边界，再透过这个边界去判别新的资料点是否与训练数据类似，超出边界即视为异常。该算法在捕获数据集的形状上非常优秀，严格来说该算法是奇异点的检测，在做不出对样本数据集分布特点的假设时OneClassSVM相当实用；同样，OneClassSVM可能会把异常点划分在决策边界内所以需要使用正常的数据集去训练，并且在处理大量数据的时候，Kernel计算时间较长，故不适合处理海量数据。

AutoEncoder：半监督方法，AutoEncoder基于反向传播算法与最优化方法（如梯度下降法），利用输入数据本身作为监督，来指导神经网络尝试学习一个映射关系，从而得到一个重构输出。其优点是泛化性强，缺点也很明显，须使用正常数据来训练。

三、未来挑战

- 深度有监督方法：当前深度有监督异常检测算法的精度虽然很高，但是由于这类方法高度依赖于大量有标注的正常样本和异常样本，因此在实际中很难找到能够应用的场景。如果想要扩大适用范围，未来如何在小样本条件下，得到泛化性能强的模型，是这类方法需要突破的重点和难点。
- 深度无监督方法：当前深度无监督异常检测算法对于数据的要求较低，适用范围也较大，但是由于这类方法抗干扰能力较差，实际中误报数量往往较多，影响了实际使用时的用户感受。因此，如何进一步建模噪声干扰或抑制噪声干扰，是这类方法未来研究的重点。
- 群体异常检测：群体异常检测是近年来刚刚兴起的研究方向，目前还处于起步阶段，未来的研究方向、方法仍然存在很大的不确定性，但是该方向有望成为未来的热点研究方向之一。

四、参考文献

数据集：<https://www.kaggle.com/c/expedia-personalized-sort/data>

孤立森林：https://blog.csdn.net/weixin_43999733/article/details/104355598

<https://cloud.tencent.com/developer/article/1814601>

oneclasssvm：https://blog.csdn.net/qq_19446965/article/details/118742147

<https://cloud.tencent.com/developer/article/1010835>

AutoEncoder：<https://www.tensorflow.org/tutorials/generative/autoencoder>

人员分工

胡子骄：整体任务规划和分配，任务一二的数据集选取，任务一图表可视化探索性分析，任务二数据清洗与特征工程任务，任务一二代码实现，异常检测未来挑战相关原理分析，文档撰写与整理，PPT制作。

丁励超：任务三的数据集选取，问题背景介绍，数据集统计特征与相关性计算，PCA原理，任务三异常检测算法构建，任务三代码实现，任务三异常检测结果对比分析，文档撰写与PPT制作。