机器学习（进阶）纳米学位

何龙
2018 年 10 月 1 日

I. 问题的定义

项目概述

该问题属于典型的有监督回归问题，通过商店提供的历史数据，对未来的商店销售额进行估计，销售额是一个数值型数据，而提供的训练数据中也提供了销售额数据，因此属于有监督的回归问题。目前拥有的相关数据分三部分：train.csv 表示用于训练的数据，test.csv 表示用于最终测试的数据，store.csv 表示 1115 家商店的数据。问题背景很简单，来源于欧洲 Rossmann 公司提供过去的数据，需求也很简单，就是对其下的商店进行销售额预测，以辅助商店经理对于商店的促销活动、人员时间表等提供一个更加合理的安排。

问题陈述

问题定义：通过结合训练数据、商店数据，处理数据，训练模型，并通过模型来最大程度的预测测试数据。

任务大纲：1.读取并预处理数据，2.特征工程，3.模型训练、测试，4.对比基准模型，5.模型优化，6.提交至 kaggle。

期望结果：模型预测测试数据的误差小于 20%。

评价指标

由于该问题需要提交至 kaggle，因此评价指标跟 kaggle 采用一致的最为合理，也就是 RMSPE（均方根百分误差），它的作用是评估所有预测数据与实际数值的平均误差的百分比形式，与 RMSE 相比引入了百分比的关系，避免了较小的误差被忽略。

公式：sqrt((sum(X_obs_i-X_model_i/X_obs_i)^2)/n)。

因为我们的目的是最小化预测值与实际值之间的差距，因此使用该评价指标是合理的，且该公式会避免出现负数，以免出现正负值中和的问题。

II. 分析

数据的探索

因为我们对数据的使用主要集中与训练阶段，因此我们主要关注、分析、挖掘的也是 train.csv 和 store.csv 文件，对于 train.csv，具有以下字段：

1. *Store*：表示商店编号。
2. *DayOfWeek*：表示星期几。
3. *Date*：表示日期，由年-月-日组成。
4. *Sales*：目标字段，表示销售额。
5. *Customers*：表示当天客户数量。
6. *Open*：表示是否开门。
7. *Promo*：表示是否有促销活动。
8. *StateHoliday*：表示国家假期。
9. *SchoolHoliday*：表示学校假期。

再来看看 store.csv 的字段：

1. *Store*：同上述。
2. *StoreType*：商店类型。
3. *Assortment*：分类级别。
4. *CompetitionDistance*：最近的竞争对手距离。
5. *CompetitionOpenSinceYear*：竞争对手开张年。
6. *CompetitionOpenSinceMonth*：竞争对手开张月。
7. *Promo2*：是否参与持续促销活动。
8. *Promo2SinceYear*：促销活动开始的年。
9. *Promo2SinceWeek*：促销活动开始的星期。
10. *PromoInterval*：促销活动的周期月份。

Train.csv 的基本信息：

```
RangeIndex: 1017209 entries, 0 to 1017208
Data columns (total 9 columns):
Store            1017209 non-null int64
DayOfWeek        1017209 non-null int64
Date             1017209 non-null datetime64[ns]
Sales            1017209 non-null int64
Customers        1017209 non-null int64
Open             1017209 non-null int64
Promo            1017209 non-null int64
StateHoliday     1017209 non-null object
SchoolHoliday    1017209 non-null int64
```

Store.csv 的基本信息：

```
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 10 columns):
Store                        1115 non-null int64
StoreType                    1115 non-null object
Assortment                   1115 non-null object
CompetitionDistance          1112 non-null float64
CompetitionOpenSinceMonth    761 non-null float64
CompetitionOpenSinceYear     761 non-null float64
Promo2                       1115 non-null int64
Promo2SinceWeek              571 non-null float64
Promo2SinceYear              571 non-null float64
PromoInterval                1115 non-null object
```

首先，最直观的感觉是两份数据可以通过 Store 字段链接到一起。其次在 train.csv 中没有 null 值，而 store.csv 中的 CompetitionDistance、 CompetitionOpenSinceYear、 CompetitionOpenSinceMonth 存在 null 值，且容易看到他们的缺失应该是技术原因导致的缺失，也就是说我们需要对其进行填充处理的，由于目前看不出其与其他字段的直接关系，因此填充方式选择 median 值填充。而对于 Promo2SinceYear、 Promo2SinceWeek 的缺失则不同，我们发现在他们缺失的数据中，都存在 Promo2 为 0 的情况，也就是说他们并不是技术原因导致缺失，而是由于商店不参与活动导致没有活动相关数据，因此不需要特别处理。

再来看一些有意思的统计数据：通常我们会认为商店的销售额跟假期应该有关系，那么事实是怎么样的呢？在学校是否放假的条件下，分别对应的销售额为：

学校放假平均销售额：*6476.52220712*
学校不放假平均销售额：*5620.97903381*

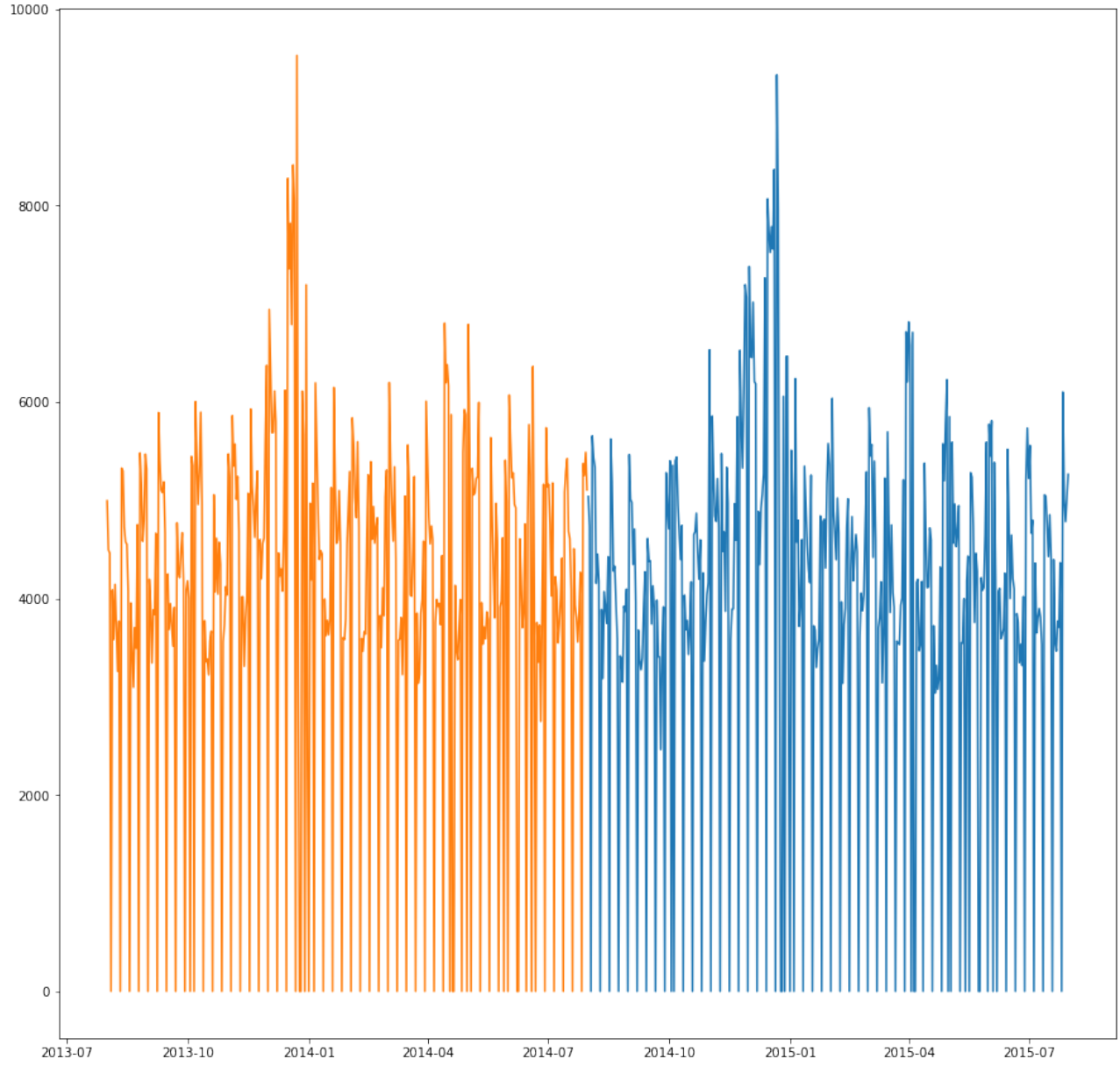
可以看到在学校放假或者不放假的情况下，销售额没有很明显的差异，这一点说明至少从平均值上看，放不放假是没有影响或者说影响不大的，那么国家假期呢：

国家放假平均销售额：5779.7781799
国家不放假平均销售额：5733.53062439

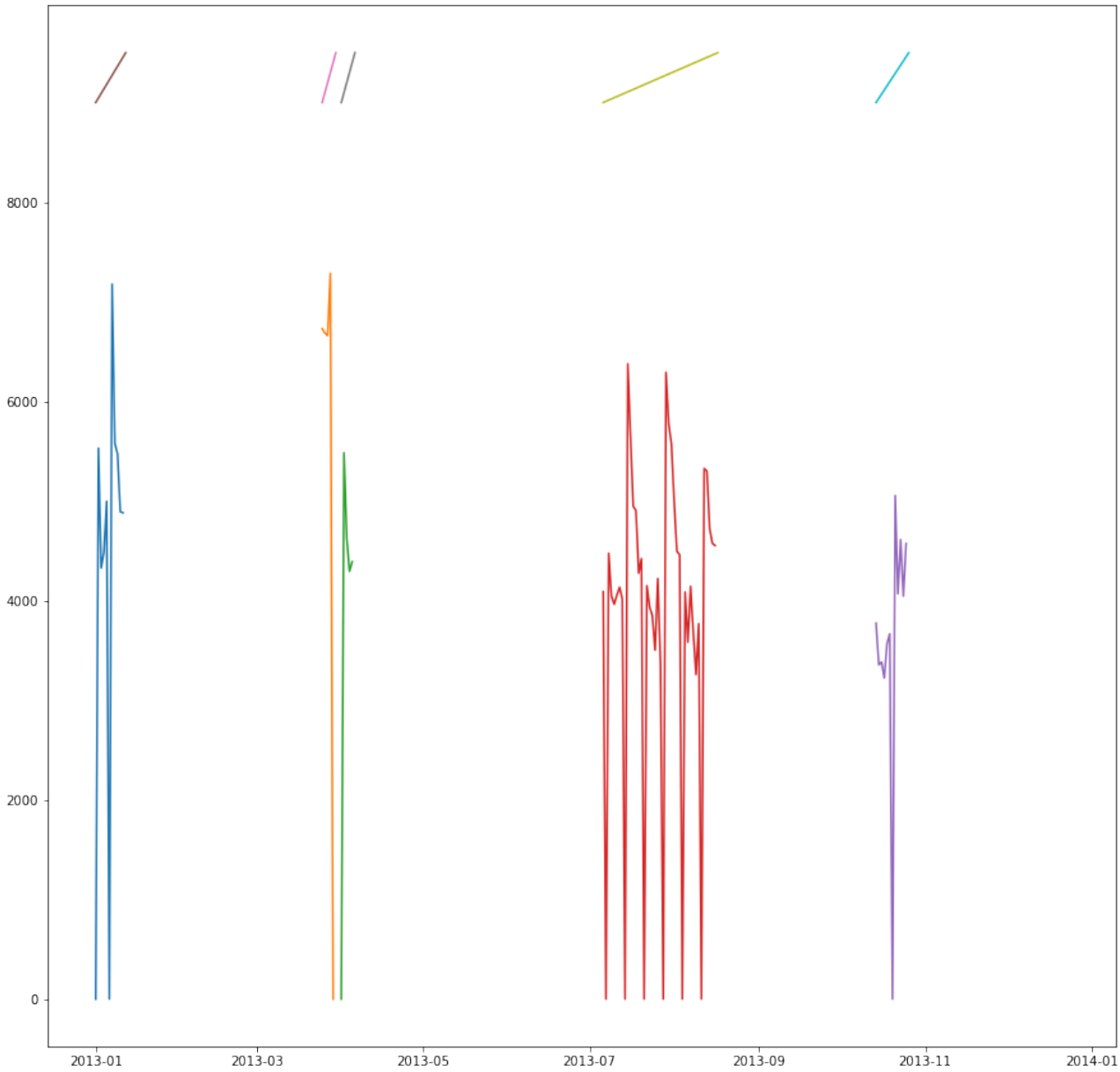
同样可以看到影响非常之小，这一点反应出来的跟我们的直觉可能不太一致。但是在数据中发现了一个非常有用的信息，那就是周末的销售额都是 0，也就是周末是不上班的，这一信息对于我们预测非常有帮助。而像 Date 信息、促销活动信息、竞争对手信息等都要进一步进行可视化验证其与我们预测值的相关性，综合分析才能判断是否需要挖掘更深层次的意义。

探索性可视化

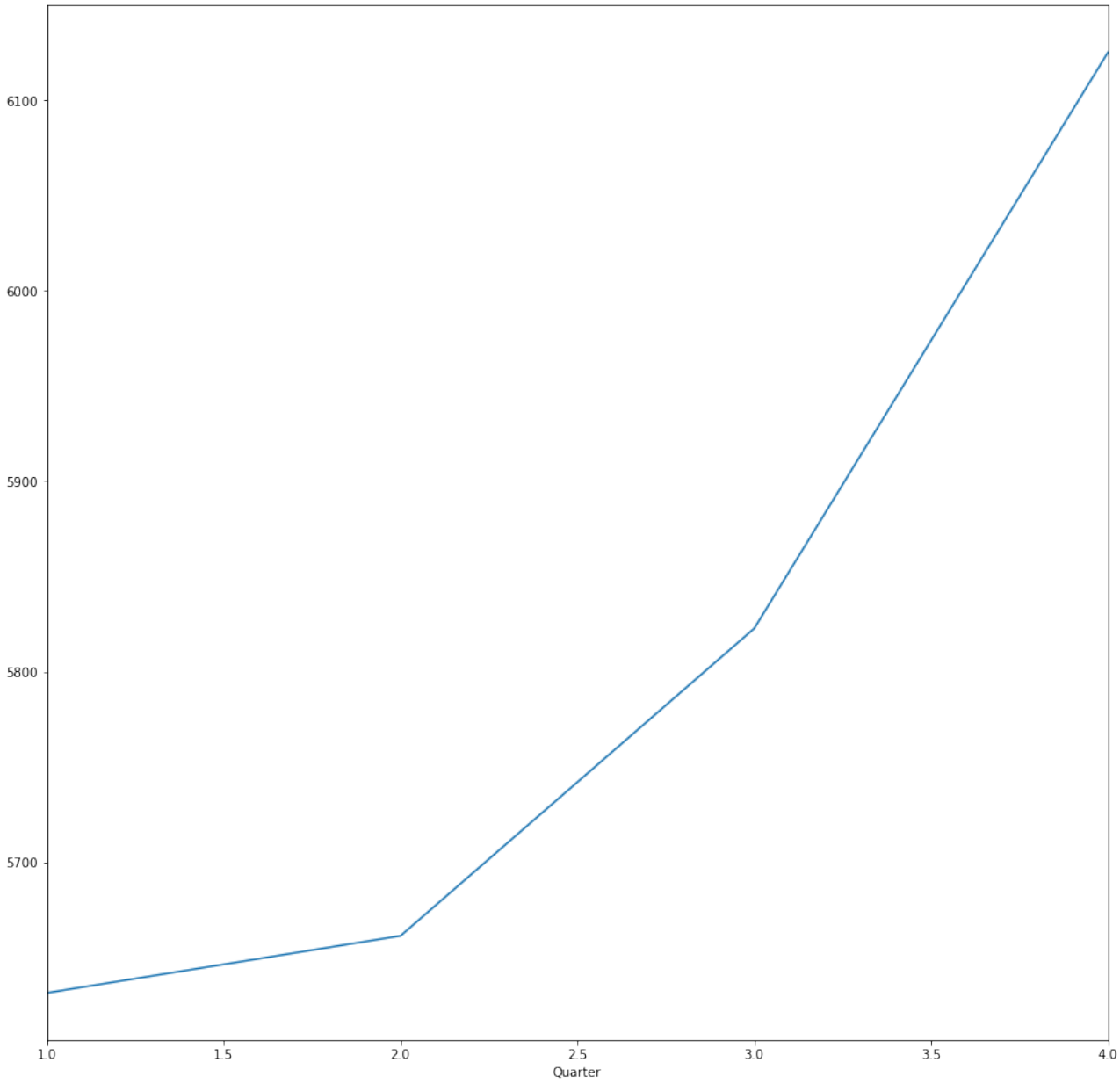
1．时间信息探索：之所以挖掘时间信息，主要原因有以下几个，首先数据的时间是连续完整的，Date 字段的训练数据从 13 年 1 月 1 号到 15 年 7 月 31 号，大约两年半的连续时间，时间的完整保证了我们对时间的挖掘可以推广到所有数据上，首先我们看看相邻的两年时间下的销售额变化图：



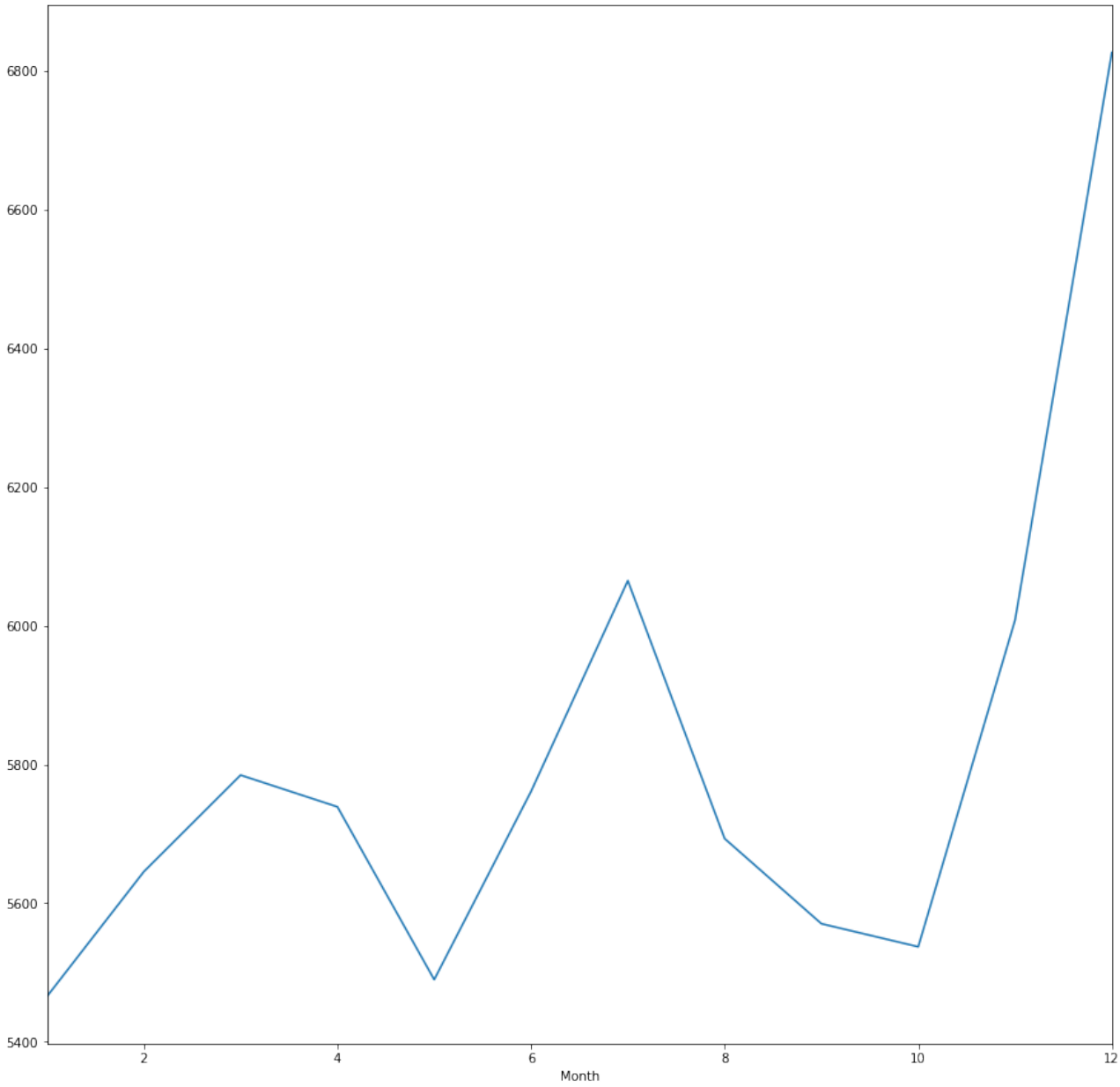
可以看到趋势惊人的一致，这说明了销售额是一个时间周期强相关的属性，因此对 Date 的挖掘一定要彻底，再来看看深层次一点的挖掘，即假期期间的销售额变化，注意图中上方的小短线表示假期的持续时间，对应下面的线表现期间的销售情况：



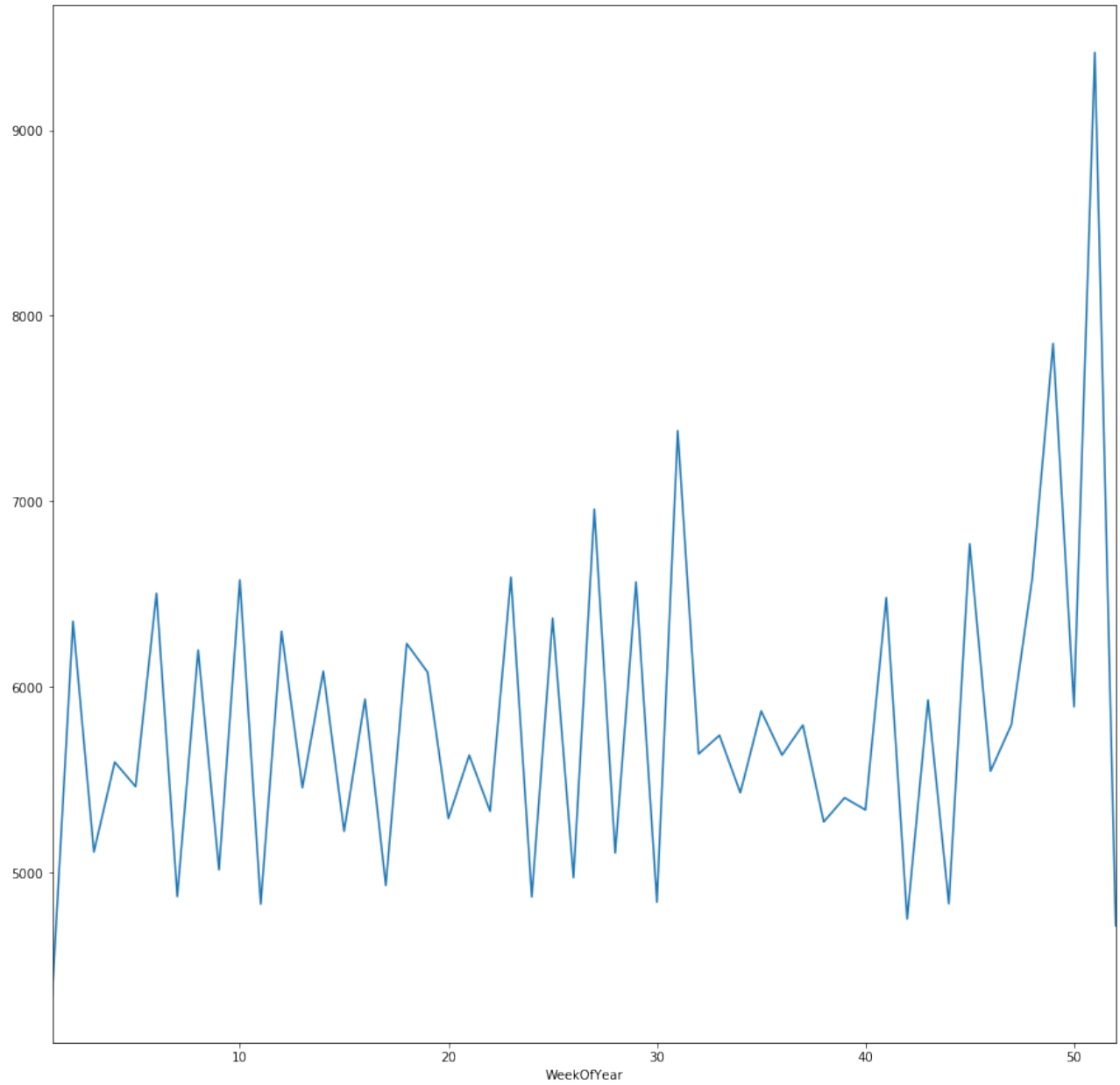
无法看到明显的趋势或相关性，这一点结合之前的学校假期、国家假期与平时的销售对比数据，说明假期对结果的影响很小，不需要特别挖掘处理，当然周日另算，下面看看挖掘后的季度、月份、`WeekOfYear` 的销售额对比图，先看看季度的：



能看出有一个非常明显的上升趋势，说明销售额是一个季节性强相关的属性，再来看看月份的：

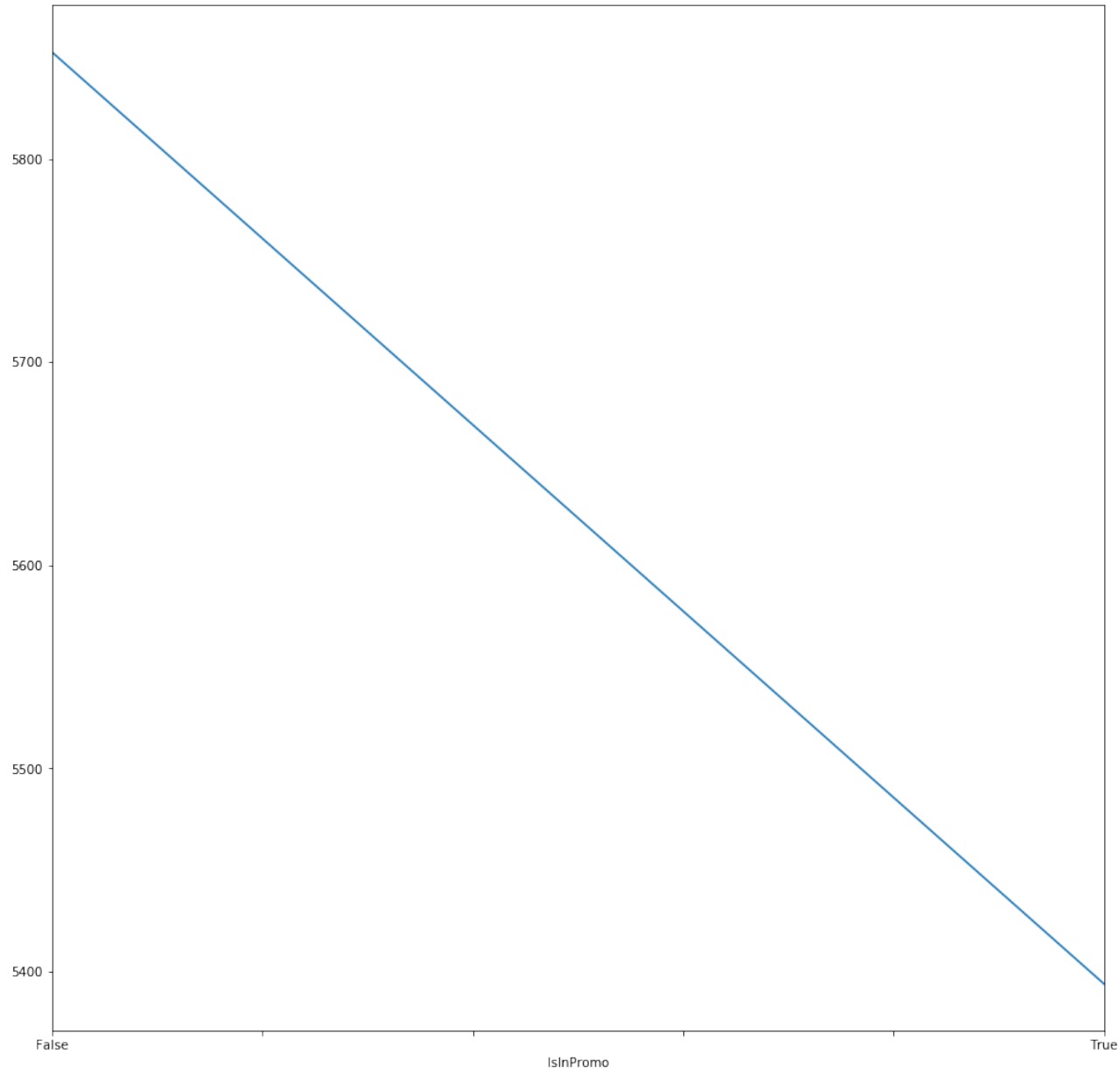


同样可以看出 7,11,12 月份明显高于其他月份，同时一年中，有 3 个销售额上升的情况出现，这一特点的原因不能肯定，但是应该与季节相关的可能性较大。最后看看 WeekOfYear 的：

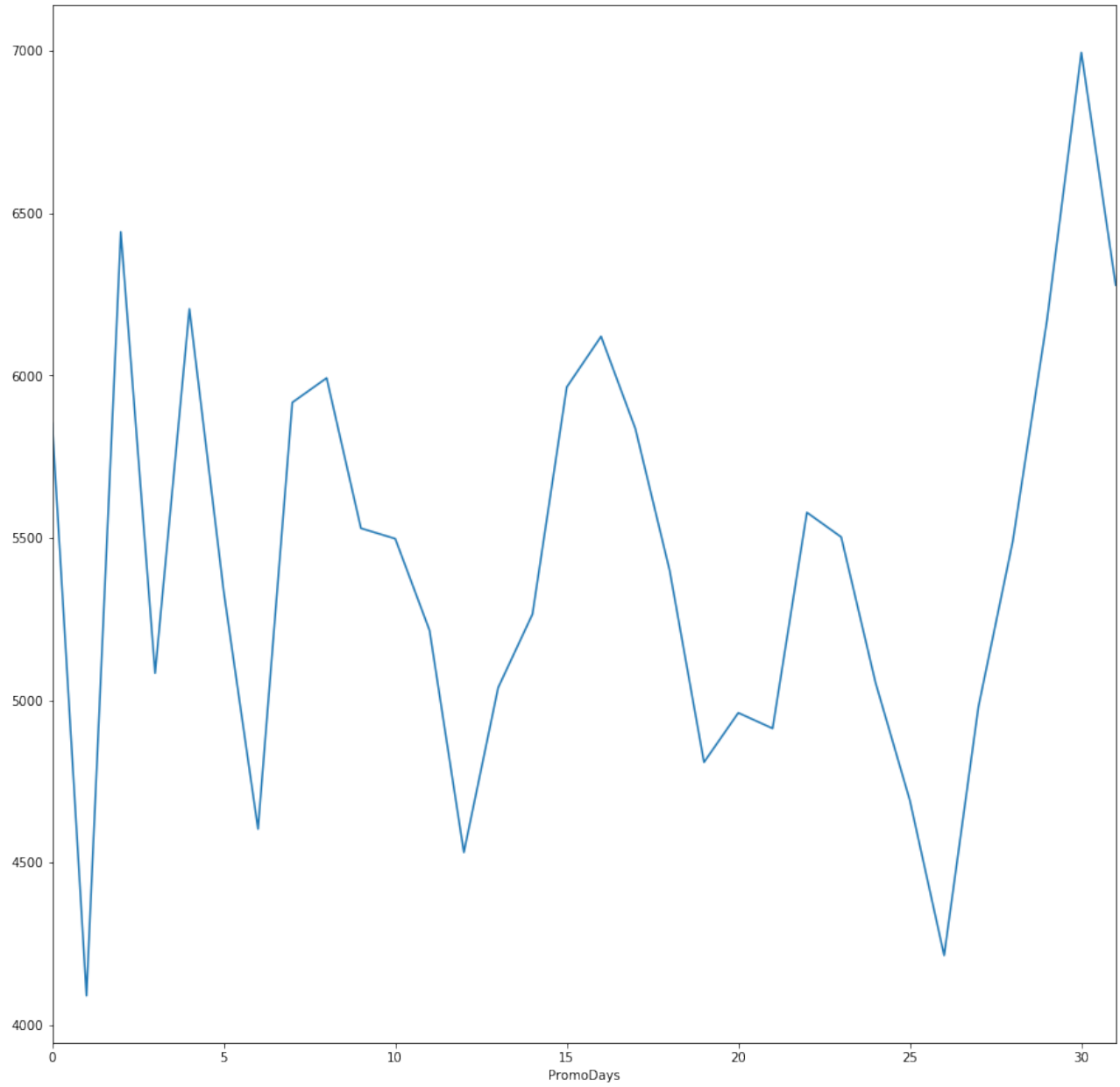


这个情况就不太好总结趋势，不过可以看做是对月份的一个细化表现，整体上跟月份上表现的趋势是一致的，只是将这一趋势细化到以星期为单位的数据上，最后我们可以肯定的是，时间字段对于结果影响非常大。

2．持续促销活动的探索：商店重要的促进销售的手段就是搞促销活动，因此我们没理由不对持续促销相关数据进行探索分析，先来看看是否处于促销月对于销售额的影响（PS：是否处于促销月是我们针对持续促销挖掘的新特征）：

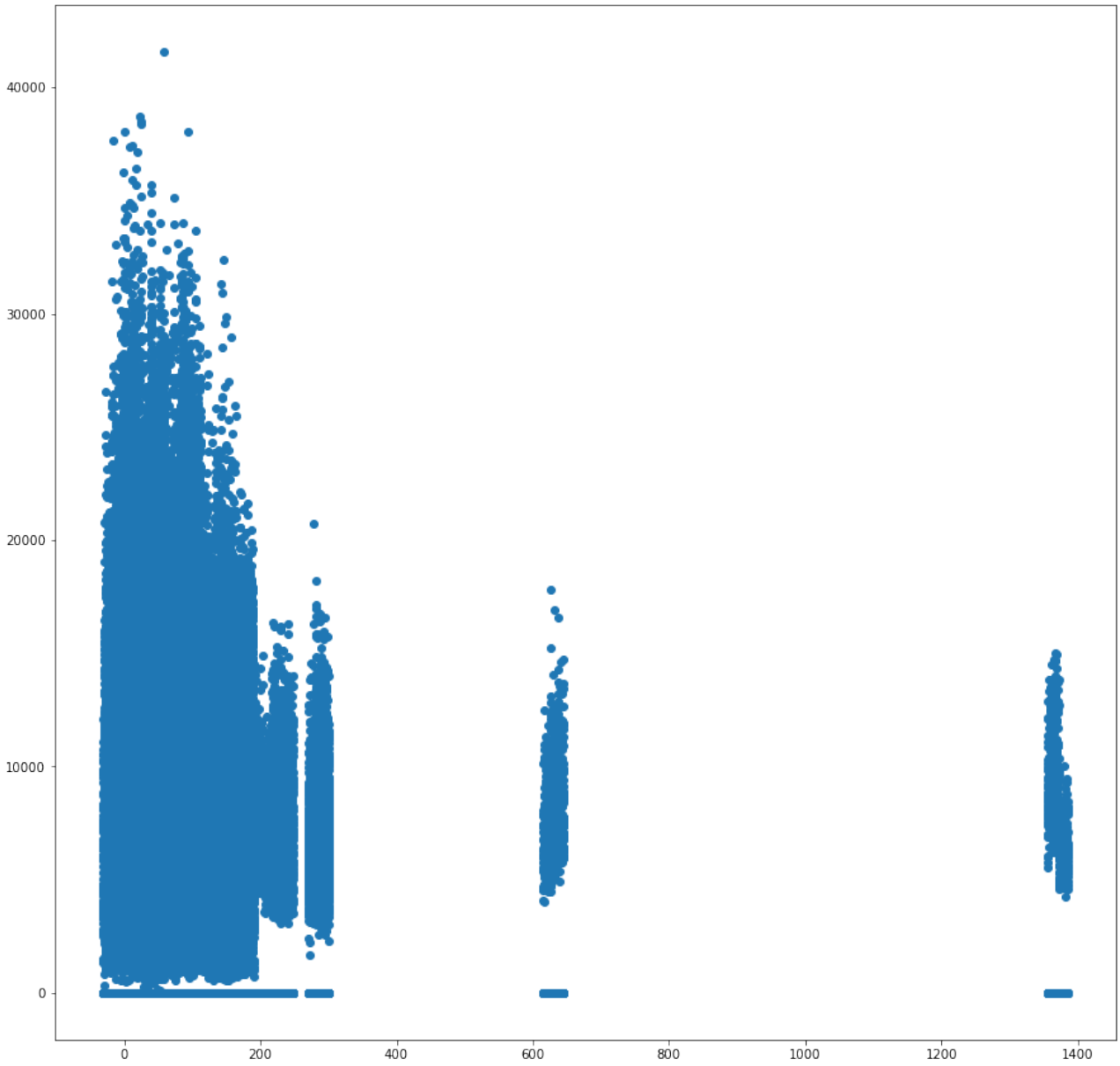


这就非常清楚的表达了持续促销对销售额的影响，下面再细化以下，看看促销时间跟销售额的影响：



这张图很有意思，首先大方向上，是促销的时间越长，效果越差，这一点也符合我们的认知，毕竟你天天搞促销，但是后面人们的购买欲望也降低了，而且该买的促销前几天也都买了对吧，但是发现了一点特殊的就是，促销的最后几天销售额有明显的上升，估计是大家看促销快完了再不买就没得买了吧，怪不得路上的店铺每次经过喇叭里都喊着"最后一天，最后一天"，看来道理都隐藏在生活中。

3．竞争对手信息的探索：一家商店是否有竞争对手，以及两家店的距离，对方的营业时间都会对我们的销售额造成影响，这是很明显的道理，下面让我们可视化分析一下，开张营业时间的销售额对比图：



可以看到，对手的营业时间越短，我们的销售额越高，这一结果基本符合我们的预测，毕竟一家新店对我们销售额的影响肯定不如一家“老字号”。

算法和技术

采用 XGBoost [1] +校正系数 [2] +多模型融合[3]+增量训练验证数据的组合来实现模型的训练以及优化，下面分别介绍三者的使用原因、参数细节。

XGBoost：继承自 GDBT 的一种提升算法，相比较 Adaboost 而言，主要区别在于它评估模型的不足使用的是梯度下降的方法，因此支持更多的目标函数（包括我们的评估指标：RMSPE），因此更适合我们的项目，同时 XGBoost 支持并行、速度更快，且提供了更多的超参数可以设置，方便我们调参。下面是我最终选用的一组超参数：

```
params = {
    'objective':'reg:linear',
    'booster':'gbtree',
    'eta':.03,
    'max_depth':10,
    'subsample':.9,
    'colsample_bytree':.7
}
num_boost_round = 5000
```

输出选择 reg:linear，即线性回归，符合我们的项目需求，booster 选用默认的 gbtree 即可。eta、max_depth 为实验所得的经验值，num_boost_round 设置为 5000，试验中发现在 1000,3000 的情况下依然有提升，而 5000 时迭代出现多次在 4500 次左右由于 100 次性能无提升而提前终止的情况，5000 次在验证集的表示是 0.1021，已经是很好的结果了，因此选择了 5000 作为迭代次数，而 early_stopping_rounds 设置为 100 保证了少量的连续迭代无法提升不会终止模型，但是在 100 次都没有提升的情况下会终止模型，避免模型过拟合，在验证集的性能降低的情况出现。

校正系数：这一做法的主要思想是，如果我们的预测数值相对于实际数值存在整体偏高或者偏低的情况时，我们可以通过校正系数来统一进行预测结果的校正来减小误差，试验中发现在校正系数为 0.99 时，RMSPE 值从 0.1021 降低到了 0.1011，只是增加了一个系数就降低了 0.001，还是很不错的处理方式。

多模型融合：训练多个模型，根据模型的 RMSPE 分数进行加权融合，目的是让模型具有更好的泛化能力。

增量训练验证数据：保证了要预测的测试数据在时间上是紧挨着整个训练数据的（因为已经将验证数据也加入了训练中），同时也是最大化的利用可用的数据进行训练。

基准模型

由于是一个回归类问题，且数据分布符合正态分布（TODO 补一张数据分布图），因此基准模型我采用训练数据的 Sales 的 mean 值，毕竟这种数据分布下，猜测 mean 值是最傻瓜保守的一种合理做法，得到的结果如下：

基准模型预测值：5773.45927746
基准模型的 RMSPE：0.439804931616

误差百分比为%43，基本符合预期，后续只要模型的得分低于该分数，就说明我们的模型预测比傻瓜化的一直猜 mean 要更准确。

III. 方法

数据预处理

数据预处理 0：类型转换，train 和 test 的 Date 字段使用 to_datetime 转换为 DateTime，方便后面可视化时的显示效果，store 的 PromoInterval 转换为 Str，虽然现实还是 Object，但是后续可以直接用 in 操作了，这两个转换都是为了后续一些操作上的遍历，并不是非转换不可。

数据预处理 1：空值填充，之前的数据探索中我们知道，test 的 open 字段、store 的 CompetitionOpenSinceMonth、CompetitionOpenSinceYear、CompetitionDistance、Promo2SinceYear、Promo2SinceWeek，下面我们依次描述对每个字段的填充，以及选择这种填充方式的原因：

1. Open 字段：由于如果是 close 的话，那么必然会预测为 0，因此此处填充均使用 1，表示营业，同样的数据中也可以看到 1 的数量远大于 0，从众数的角度也是 1 更合理。
2. CompetitionDistance：典型的数值型字段缺失，通常情况下在无法通过其他字段来推测时，我们直接采用 mean 值填充，是较为傻瓜合理的办法。
3. CompetitionOpenSinceYear、CompetitionOpenSinceMonth：同 2，但是有一点要注意，因为是年份和月份，因此最好取整数填充，方便后续如果想作为枚举型数据处理的需求。
4. Promo2SinceYear、Promo2SinceWeek：比较特别的一种缺失，它们两个的缺失并不是由于技术性、人为性错误导致数据收集上的问题，而是由于 Promo2 字段为 0 导致的，也就是当对应商店不参与持续促销活动时，这两个字段为空，那么这种情况我们是不能也不需要处理的，贸然处理会造成相反的效果，但是保持空值对算法貌似不太友好，不过后面我们的特征工程会处理它们。

数据预处理 2：数据拼接，train 和 store，test 和 store，目的是在一个整体的数据上结合 train 和 store 的所有特征进行分析、挖掘，实现 1+1 大于 2 的效果。

数据预处理 3：新特征的获取：

1. 时间相关：通过 Date 一个字段，挖掘出 Year、Quarter、Month、Day、WeekOfYear、IsWorkDay6 个字段。
2. 促销相关：根据 1 中挖掘的 Month 字段，通过 Promo2 判断商店是否参与了持续促销，如果参与了再判断当前数据的 Month 是否在 PromoInterval 的促销月中，挖掘出 IsInPromo 字段。而通过 IsInPromo 字段判断是否处于促销月，如果是，那么再通过 Day 字段获取该月已经持续的天数来挖掘出当前处于持续促销活动的第几天，即 PromoDays 字段。
3. 竞争对手相关：对于竞争对手相关字段，CompetitionOpenSinceYear、CompetitionOpenSinceMonth 两个字段并不是常规意义上的数值型字段，关系由大小决定，而是时间点字段，这种字段直接使用的话很难保证实际效果，但是我们可以从这两个字

段中挖掘出竞争对手针对于当前数据的总营业时间，单位为月，而这一字段明显是一个典型的字段关系由数值大小来表示的字段，即 CompetitionOpenMonths。

数据预处理 4：无用字段的抛弃，无用字段主要包括 Date、Promo2SinceWeek、Promo2SinceYear、PromoInterval、CompetitionOpenSinceMonth、CompetitionOpenSinceYear、Store，可以看到，都是由于我们的预处理操作后，被使用、挖掘过的字段，因此它们的价值在新字段中更好的体现了，所以可以抛弃，而不必担心整体信息的丢失。

数据预处理 5：枚举性字段的整体映射，包括 StateHoliday、StoreType、Assortment，可以看到特征挖掘后，枚举字段也少了很多，那么对于这三个字段，由于其取值中有 a、b 等字符型数据，因此我们通过通过映射表{'a':1, 'b':2, 'c':3, 'd':4, 'e':5, 'f':6, '0':0, 1:1, 2:2, 3:3, 4:4, 5:5, 6:6, 0:0}将其映射至数字上。

数据预处理 6：提取目标字段，将 Sales 从 train 数据中提取出来，如果不提取，那么结果就是在训练集、验证集上表现完美的模型，在测试集上结果大打折扣，甚至因为字段的不匹配导致模型无法预测测试数据。

数据预处理 7：数据集划分，使用训练数据的最后 n 周作为验证集，这保证了有效利用销售额的周期性变化特点，同样也为我们后续预测接下来的测试数据做了准备（测试数据在时间上是紧挨着验证该数据的）。

执行过程

模型执行过程如下：

1．模型构建：指定最初的模型参数、迭代次数以及转换训练数据的格式为 xgboost 需要的 DMatrix。

2．模型训练：利用 xgb 的 train 方法通过指定的参数使用指定的训练数据进行模型训练，返回一个 Booster 对象。

3．使用 Booster 的 predict 进行验证集上的预测，并通过 RMSPE 方法计算分数。

4．模型保存到磁盘，方便后续使用，毕竟训练一次都要 1H 以上。

5．模型优化：

1．参数优化。

2．校正系数。

3．多模型融合。

4．增量训练验证集数据。

6．提交 kaggle。

在使用 XGBoost 库的过程中，最大的困难来源于对库的不了解以及网上资料的困乏，最大的信息源就是它的官方文档，庆幸的是官方文档的资料基本满足我们的需求。参数优化中，困难主要来自于对参数的调整，由于每次优化后训练都耗时很长，因此每次调整都要特外谨慎，在这个过程中也更加的了解了 XGBoost 的算法原理，以及各个参数对结果的影响。

获取校正系数的函数的代码片段：

```
    for actor in [0.990+i/1000. for i in range(20)]:
        results[actor]=rmspe(y_pred=np.expm1(pred_valid)*actor,
y_real=np.expm1(y_valid))
    sorted(results.items(),key = lambda x:x[1],reverse = True)[-1]
```

多模型融合的权重计算的代码片段：

```
    for i in range(len(actor_scores)):
        weights.append(actor_scores[i][1])
    weights = [sum(weights)-w for w in weights]
    weights = [1.*w/sum(weights) for w in weights]
```

完善

参数优化

初始模型训练情况如下图所示：

训练情况

1. ps_first =

```
{
'max_depth':5,
'learning_rate':.3,
'n_estimators':5000,
'objective':'reg:linear',
'booster':'gbtree',
'gamma':0,
'min_child_weight':1,
'subsample':.8,
'colsample_bytree':.8,
'colsample_bylevel':.8,
'random_state':6,
'silent':True,
}
```

结果：在1077次时达到最优迭代，RMSPE为0.160147，花费时间1026s，且连续100次误差不能降低导致训练提前终止，因此对于学习0.3来说，n_estimators设置为1100是合适的。

可以看到，初始的模型参数，基本都是常用的初始数值，其中 learning_rate 选择 0.3，是一个比较大的学习率，目的是在后续参数调整中速度比较快。下面看看初始模型在验证集的表现：

```
XGBoost Model Valid Start....
Valid RMSPE:0.164605545389
XGBoost Model Valid End, Time: 3.010702 s....
```

可以看到，初始模型的验证集得分为 0.1646，对于一个初始模型来说，算是很不错的得分吧。。。下面我们来看看每一步的优化。

Step 1：首先对 max_depth 和 min_child_weight 进行网格组合搜索最优取值，之所以选择这两个是因为这两个参数对结果影响较大，参数调整遵循先大后小的原则，如下：

```
XGBRegressor Grid Search Start....
Best: -0.038621 using {'max_depth': 7, 'min_child_weight': 1}
XGBRegressor Grid Search End, Time: 27851.039786 s....
```

可以看到，最优组合为 max_depth=7&min_child_weight=1，也可以看到整整用了 27851s，而使用这两个新取值的模型在验证集的表现如下：

```
XGBoost Model Valid Start....
Valid RMSPE:0.165398297166
XGBoost Model Valid End, Time: 1.720675 s....
```

可以看到，RMSPE 值反而下降了，不过因为我们整体参数还在调节中，因此暂时先不管它，现在再来调一下 gamma 值，如下：

```
XGBRegressor Grid Search Start....
Best: -0.038621 using {'gamma': 0.0}
XGBRegressor Grid Search End, Time: 9285.333055 s....
```

可以看到，最优取值为 0，既然我们之前取得就是 0，那么就不需要取验证集测试了，再来调下 subsample 和 colsample_bytree：

```
XGBRegressor Grid Search Start....
Best: -0.039613 using {'subsample': 0.8, 'colsample_bytree': 0.9}
XGBRegressor Grid Search End, Time: 26338.595597 s....
```

可以看到，最优取值为 subsample=0.8&colsample_bytree=0.9，看下这种取值在验证集的表现：

```
XGBoost Model Valid Start....
Valid RMSPE:0.178430845007
XGBoost Model Valid End, Time: 0.787057 s....
```

可以看到，RMSPE 值又高了，不过不要紧，这部分调参只是在基础上选择最优即可。到底几个关键参数都已经经过了调整，那么我们应该回过来调整我们的学习率，最初是选了一个很大的学习率，现在应该逐步降低：

```
0.01: Valid RMSPE:0.168352504187, n_estimators=9408
0.02: Valid RMSPE:0.170173284035, n_estimators=4627
0.03: Valid RMSPE:0.162580754407, n_estimators=3888
0.05: Valid RMSPE:0.16030973754, n_estimators=2557
0.1: Valid RMSPE:0.162610144828, n_estimators=1443
0.2: Valid RMSPE:0.160755814105, n_estimators=1051
0.3: Valid RMSPE:0.178430845007, n_estimators=238
```

结果很明显了，在学习率为 0.05、估计器个数为 2557 时，验证集 RMSPE 为 0.1603，明显低于初始模型的 0.1646，说明我们的参数优化是有效果的。

校正系数

下面我们应用校正系数，前面也提到，应用校正系数的前提是预测数据与实际数据有明显偏差，如图：

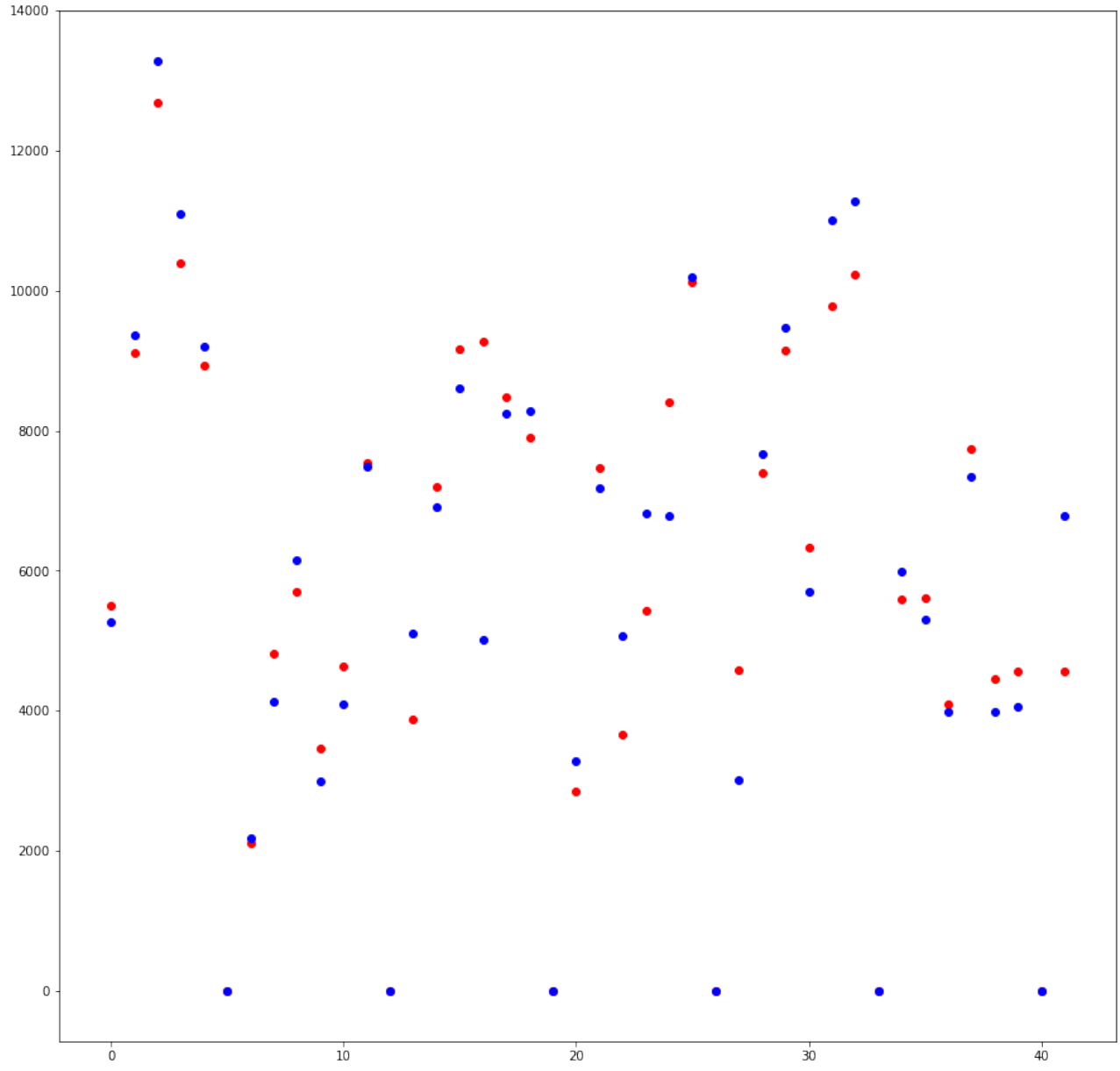
```
np.mean(np.abs(np.expm1(pred_valid_opt)-np.expm1(y_valid)))
```

```
693.13059095674203
```

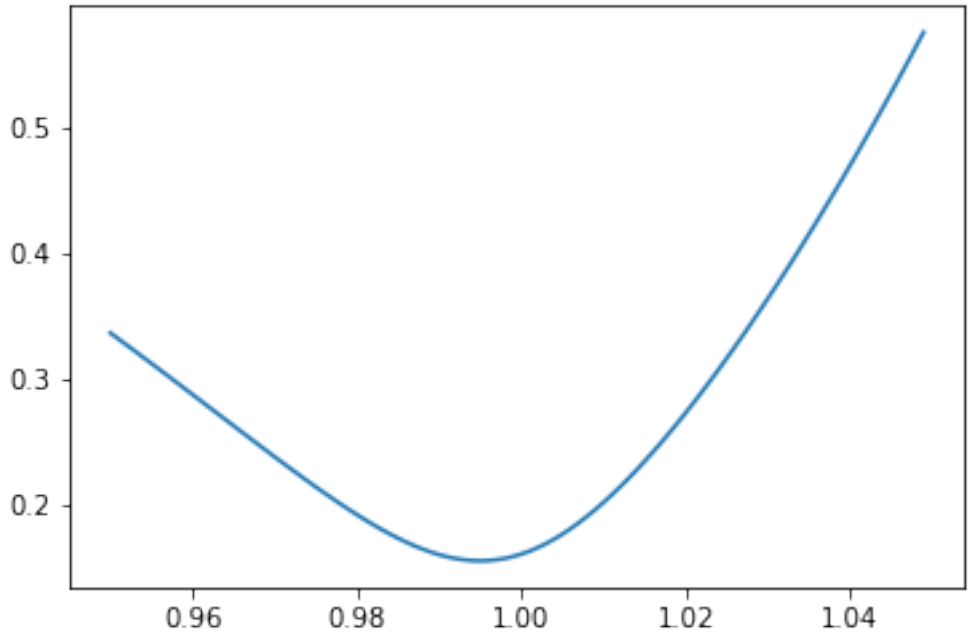
```
np.mean(np.expm1(pred_valid_opt)-np.expm1(y_valid))
```

```
-96.321284492246519
```

可以看到，整体上偏差为-96，也就是说预测数据普遍偏低，从散点图上看：



可以看到，直观感觉也是整体偏低的，因此我们可以通过赋予其一个校正系数来校正这一问题，如下：



计算得到 0.995 为最佳校正系数，小于 1 也符合我们的数据整体偏低的看法，校正后在验证集的表现：

```
校正前：0.160303987825
校正后：
(0.995, 0.15503830432208604)
```

可以看到，校正后提升了大约 0.005，只是一个校正系数就起到这么大的作用。

全数据训练

之前的所有训练都是通过划分数据集得到训练集和验证集来训练的，因为最终要放到 kaggle 去测试，因此最后一步优化我们将训练集+验证集统一作为训练数据去训练我们的模型，同时应用之前的所有优化手段。

IV. 结果

模型的评价与验证

XGBoost 模型通过数据训练进行有监督的回归学习，输入数据使用了 kaggle 提供的 train 和 store，输出为数值型的 Sale，满足项目需求，基准模型的 RMSPE：0.4245，模型达到了 0.155，非常大的提升，因此所选的模型是合理可用的，与期待结果一致，最终参数如下：

优化后的参数

可以看到优化后的RMSPE值从0.1646减低到了0.1603，对应参数为：

```
ps_opt_final = {
    'max_depth':7,
    'learning_rate':.05,
    'n_estimators':2557,
    'objective':'reg:linear',
    'booster':'gbtree',
    'gamma':grid_result2.best_params_['gamma'],
    'min_child_weight':1,
    'subsample':grid_result3.best_params_['subsample'],
    'colsample_bytree':grid_result3.best_params_['colsample_bytree'],
    'random_state':6,
    'silent':True,
}
```

训练时间：3391s，验证时间：20s。

在将个别数据使用异常值替代后与正常输入数据对比，

1.原始输入

```
DayOfWeek                 5
Open                      1
Promo                     1
StateHoliday              0
SchoolHoliday             1
StoreType                 3
Assortment                1
CompetitionDistance    1270
Promo2                    0
Year                   2015
Quarter                   3
Month                     7
Day                      31
WeekOfYear               31
IsWorkDay              True
IsInPromo             False
PromoDays                 0
CompetitionOpenMonths    82
Name: 0, dtype: object
```

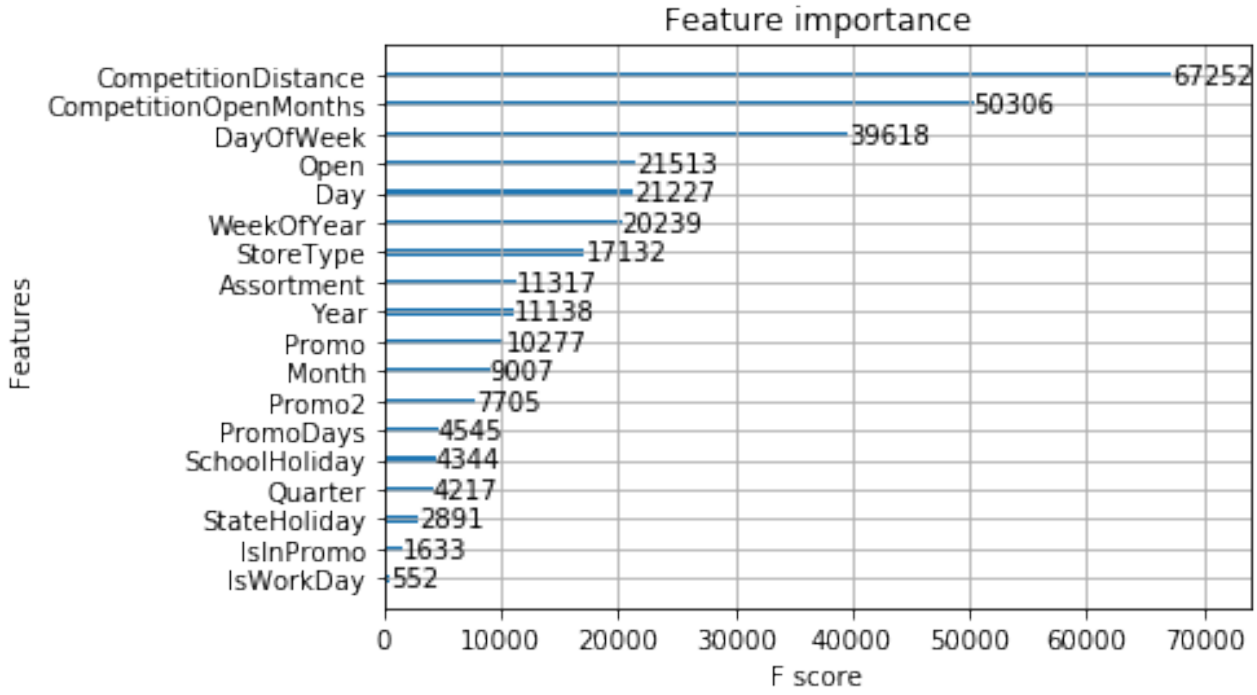
，输出：5510.4448，真实值：5262.9999，

2.修改 Year 为 0，输出：5763.11

3.修改 CompetitionDistance 为-100，输出：8143.42

4.修改 DayOfWeek 为 8，输出：5692.17

可以看到，在对输入数据进行了小范围的异常化后，对于输出的影响还是有的，影响的大小由对应字段对预测结果的重要程度而定，字段的重要性如下：



可以看到，对于预测结果而言，CompetitionDistance 是最为重要的字段，而 IsWorkDay 是相对最不重要的字段，上面的异常值测试也发现了 CompetitionDistance 被修改为-100 时，对预测结果的影响是最大的，而其他两个字段的影响则很小，应该可以说，鲁棒性还是不错的。

数据在训练集和验证集的划分是按照时间前后的，即取最后 n 周作为验证集数据，因为我们知道销售额是有周期性的，那么训练数据、验证数据、测试数据均有时间上的前后关系可以帮助我们利用这一周期性的特点，因此在验证集上的 RMSPE 得分是客观可信的，而在 kaggle 测试集上的误差得分要高于验证集的得分，其中最可能的原因是销售额数据本身是有时间前后关联性的，比如用去年的数据估计今年的销售额，误差会明显低于取前年的数据估计今年的销售额，因此我们用 13,14,15 前半年的数据同样预测这之间的某些时间的销售额，这个误差也要低于取预测 15 年后半年的销售额，这个结果是可以预见，并且合理的，对于模型来说，只要我们持续使用最新的数据去不断的训练它，那么它的预测结果就能一直保持一个很低的误差率，因此我们的模型是合理的可靠的。

合理性分析

基准模型在 Kaggle 测试集上的得分：Public-0.38919，Private-0.40736，而我们的模型优化后最佳情况得分：Public-0.14474，Private-0.16027，可以看到使用模型预测的误差要远远小于基准模型定义的使用 mean 值预测，而使用 mean 值预测是人们在不确定事物面前选择的一种常用的预测手段。

我们模型的最终目的是通过更加合理、准确的预测，帮助商店经理更好、更专注的进行人员、促销活动等的安排，提高效率以及销售额，那么事实证明我们的模型相比较之前傻瓜式的使用 mean 值猜测的方式确实有很大提升，不管是更低的误差率还是更快的预测速度，因此我们的模型确实解决了问题，或者说一定程度的解决了问题，当然还是有很大的提升空间，即更准确的预测。

V. 项目结论

结果可视化

全数据训练，它的优点不仅仅在于加大了数据量，同时由于我们的数据是时间序列数据，那么使用与待预测数据相连的数据去训练模型会起到更大的帮助，而从 kaggle 得分上也能看出，当我们的模型仅仅是经过优化+校正系数时的得分：

而当在此基础上加入验证数据形成全数据训练时：

可以看到，没有任何优化手段，而仅仅是增加了一部分数据，就有了其他优化手段都不及的性能提升，也充分说明了数据的重要性，相比于一个优化，数据+特征功能对于整体的影响要大得多。

对项目的思考

项目的整体流程：

1. 数据读取：读入 *kaggle* 提供的训练、测试、商店数据。
2. 数据异常值分析、处理：对于 *store* 和 *test* 的异常数据进行了针对处理。
3. 数据链接：分别将 *trian* 和 *test* 与 *store* 链接到一起进行后续工作。
4. 枚举字段统一映射：将各种枚举字段的取值都映射到数值型的 *0,1,2,3...* 上。
5. 特征工程：主要是提取新特征，比如年、月、日、对手开张时长、是否促销月、是否周末等。
6. 基准模型建立、评分。
7. *XGBoost* 模型构建、训练、评分。
8. 参数调优。
9. 校正系数。
9. 全数据训练。
10. 最终模型在测试集上预测，并将结果整理成 *kaggle* 所需格式上传。

其中特征工程是最有趣的环节，一个个脑海中的想法被各种可视化验证有效或者无效，很多认为理所当然的想法，比如销售额跟假期相关被认证是关系不大的，比如竞争对手越远销售额越高，也被证实是错误的，而一些出乎意料的特征却跟预测结果关系很大，比如销售额是有周期性的，也就是日期强相关，竞争对手越近，销售额反而越高等等，这真的是一个很相像的挖掘过程，一铲子下去出来的是泥土还是黄金，没人知道。

最困难的部分应该是模型调参、优化的部分，毕竟在确定好适合的迭代次数（5000）之后，每次运行时间都比较长，而大部分运行的结果表现优化手段没有起到作用，因此这部分需要对算法非常了解，对运行结果非常敏感，才能做出针对性调整，否则就是做一堆无用功不说，结果没有任何提升。

还有一个比较困难的部分我认为是数据处理不彻底，比如很多数据上的瑕疵是在训练时，或者验证后才发现，而一旦数据有问题，那就意味着整个流程都要从头再来一遍，这也是一个相当大的工程，这一点也说明了前期工作的重要性。

总的来说最终结果是符合我的预期的，毕竟在有限的时间内，这个结果已经很不错了，通过这个模型在这个项目上的表现，我们可以判断同类的场景下的类似问题，都可以通过 XGBoost 为主，辅以各种优化手段来达到预测目的。

需要作出的改进

0. 整个执行流程中的中间变量一定要单独保存下来，不然丢失在茫茫的内存中再找可就要再执行一遍。
1. 最好应该将数据的处理、分析、特征挖掘以及最后的模型相关的操作分开来，这样既能避免每一大部分之间互相影响，又能更好的针对每一个步骤来选择工具和方法，比如对于数据的处理，目前同样是在 Jupyter 中使用 Python 完成，但是根据 Kaggle 第一名的分享得到的灵感，其实这部分用 SQL 来解决更加遍历，一个是 SQL 语句非常灵活、快速，另一个是数据库本身支持很多数据相关的操作，更加方便我们对数据进行分析处理。
2. 更重视特征工程阶段，最开始做项目时，主要精力放在模型上，必须模型的选择、超参数的设置、各种优化手段等等，但是对数据的特征挖掘做的很少，最终 RMSPE 分数在 0.24 左右就很难再降低了，后面同样是根据 kaggle 第一名的分享，以及各种网上大神的分享中，开始慢慢重视特征挖掘过程，开始从 Date、促销活动、对手商店等各个基础字段中挖掘更深层次的、对预测结果影响更大的特征出来，最后加入这些特征做预测时，模型没有经过太大预测的情况下在验证集上即获得了 0.102 的好成绩，虽然在 kaggle 中一开始只得

到了 0.16，但是已经比之前经过各种优化的模型还要好了，也证明了好的训练数据源对结果的影响远远大于各种 trick 的优化（同样过度的优化存在过拟合的问题）。
3．根据数据选模型，而不是根据模型改数据，根据我们数据的特点、期望答案、挖掘出的特征特点等等，我们最终选择了比较适合的 XGBoost 算法，而不是像一开始的直接定了 Adaboost，但实际上 Adaboost 在该项目上并不是那么合适，其中最重要的是我们的评估函数 RMSPE 可以直接在 XGBoost 上应用。
4．要更加重视前期调研工作，包括项目背景了解、前人们的经验分享等等，都会对我们做项目提供非常大的帮助，也能帮助我们少走很多弯路，很多错误是没有必要重复犯的，当然了踩过的坑可能印象更深刻，也是有好处的。
5．应用统一标准的归一化到各个数值性字段上，项目中没有应用归一化的主要顾虑在于各个数值型数据的最大最小值无法判断估计，担心贸然的归一化处理带来未知的问题。
6．使用 CNN 实现销售额预测，相比较而言，CNN 由于其强大的抽象能力，在特征表达上要更加优秀，不知道与我们的 XGBoost 对比会有怎样的结果，当然，由于各种原因，此次项目中没有进行这样的对比很遗憾，期望后续可以完成。
7．竞争对手的信息挖掘，我认为只是挖掘了对手的开张营业时间是不够的，或许可以根据它的 Distance 进行对手细分，比如每 5km 表示一类对手，可想而知，如果对手超过 10km，或许跟我们的营业额就没有直接关系了，类似 kaggle 泰坦尼克幸存率预测项目中的年龄处理，并不是直接当做枚举进行 OneHot，也不是当做数值型数据，而是根据年龄段，以及社会上常用的年龄分为幼儿、少年、青年、中年、老年等 4 个字段，更符合实际项目需求，事实也证明这样处理效果更好，同时不会造成维度灾难。
8．滑动窗口统计时间序列数据。

参考文献

【1】XGBoost 官方文档：https://xgboost.readthedocs.io/en/latest/

【2】XGBoost 参数优化：https://blog.csdn.net/aicanghai_smile/article

【3】Pandas 官方文档：http://pandas.pydata.org/pandas-docs/stable/api.html

【4】Kaggle 项目地址：https://www.kaggle.com/c/rossmann-store-sales