南開大學
Nankai University

基于 LSTM 的时间序列销量预测

来自 kaggle 网站 Store Sales-Time Series Forcasting 竞赛

Python 与机器学习课程报告

李 任 2013455@nankai.edu.cn

目录

一、项目描述

回归预测类问题、时间序列分析是机器学习所试图解决的热门问题。本报告以 kaggle 网站 Stores Sales-Times Series Forecasting 竞赛数据集为基础，对数据集预处理、描述性统计与可视化，筛选出与预测标签关联性较大的属性与数据，作为训练模型的输入参数，并根据数据集特征选取合适模型，比较不同模型训练效果，最后进行回归预测，得出模型效果评估。

二、数据描述

Stores Sales-Times Series Forecasting 数据集，包含 7 个文件，holidays_events.csv、oil.csv、sample_submission.csv、stores.csv、test.csv、transactions.csv。撇除训练模型无关的文件，本项目所用到文件为：holidays_events.csv、oil.csv、stores.csv、transactions.csv。文件中保存的 DataFrame 数据含义如下：

图表 1：数据含义

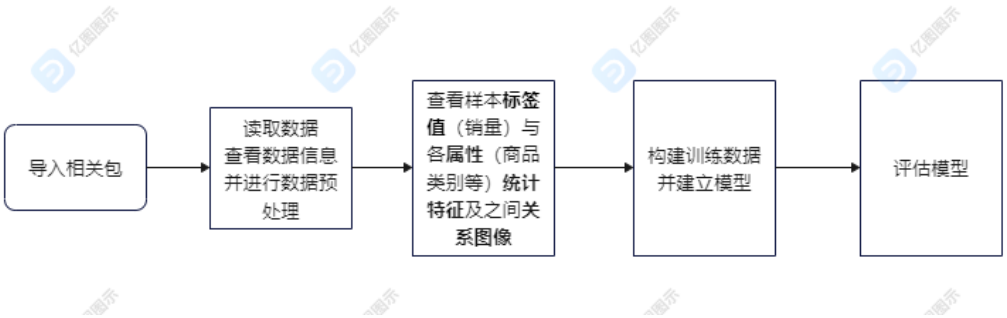
csv 文件	变量名称	变量含义
train.csv	id	序号
	date	日期
	srore_nbr	商店序号
	family	商品种类
	sales	商品销量
	onpromotion	该类商品促销数
holidays_events.csv	date	日期
	type	类别
	locale	节日地点类别
	locale_name	节日地点名称
	description	节日描述
	transferred	节日是否延迟
	store_nbr	商店序号
	city	所在城市

stores.csv	state	所在州
	type	商店类型
	cluster	商店聚类
	date	日期
oil.csv	dcoilwtico	油价
	date	日期
Transactions.csv	store_nbr	商店序号
	transactions	商店交易量

其中，train.csv 文件包含数据样本 3000888 条，涉及数据量极大。

三、实验过程描述

过程简述：



(一) 导入相关包

```
import pandas as pd#线性代数运算
import numpy as np

from sklearn.model_selection import train_test_split #划分数据
import tensorflow as tf#机器学习、神经网络
from pylab import mpl
from sklearn.linear_model import LinearRegression#线性回归
from sklearn.ensemble import RandomForestRegressor#随机森林
from tensorflow.keras.layers import LSTM, Dense, Dropout#LSTM长短周期记忆网络
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_log_error#模型评价指标
import matplotlib.pyplot as plt#绘图
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib
import seaborn as sns
import statsmodels.api as sm#统计分析
```

导入 pandas、numpy 进行线性代数运算；导入 seaborn、matplotlib 包进行可视

化；导入 statsmodel 进行统计分析；导入 sklearn 中 linearregression 等相关包进行机器学习模型训练。

(二) 读取数据集并进行数据预处理

1. 读取需要用到的 csv 文件并查看缺失值

```
"""
读取数据
"""
oil_data = pd.read_csv(r"D:\My_coding\kaggle_dataset\store-sales-time-series-forecasting\o
holiday_data = pd.read_csv(r"D:\My_coding\kaggle_dataset\store-sales-time-series-forecasti
store_data = pd.read_csv(r"D:\My_coding\kaggle_dataset\store-sales-time-series-forecasting
train_data = pd.read_csv(r"D:\My_coding\kaggle_dataset\store-sales-time-series-forecasting
transaction_data = pd.read_csv(r"D:\My_coding\kaggle_dataset\store-sales-time-series-forec
sample = pd.read_csv(r'D:\My_coding\kaggle_dataset\store-sales-time-series-forecasting\sam
#查看训练数据情况
print('训练数据信息:\n',train_data.info())
print(train_data.head())
print('油价信息n',oil_data.info())
print(oil_data.head())
print('假日信息:\n',holiday_data.info())
print(holiday_data.head())
print('有 {} 种不同商品类别'.format(train_data.family.nunique()))
print('有 {} 个不同的商店'.format(train_data.store_nbr.nunique()))
print('有 {} 个不同的地点'.format(store_data.city.nunique()))
print('有 {} 个不同的假日'.format(holiday_data.description.nunique()))
#103个假日，实在太多，转化为onehot编码数据量过大，
#而且一般而言假日与促销与否关系很大，此处直接舍去假日，不作为训练参数
#检查缺失值
print('训练数据缺失值:\n',train_data.isna().sum())
print('油价数据缺失值:\n',oil_data.isna().sum())
print('假日数据缺失值:\n',holiday_data.isna().sum())
print('商店数据缺失值:\n',store_data.isna().sum())
print('交易数据缺失值:\n',transaction_data.isna().sum())
```

将数据，并提取至 dataframe 数据结构中，查看各个数据集信息及缺失值状况，根据数据状况进行缺失值填充等数据预处理。

2.数据预处理

```
'''
数据预处理
'''

#只有油价有缺失值,缺失值数量不多，直接插值

oil_data['date']=pd.to_datetime(oil_data['date'])
oil_data = oil_data.set_index('date')
oil_data=oil_data.resample('1D').sum()
oil_data.reset_index()
oil_data['dcoilwtico'] = np.where(oil_data['dcoilwtico']==0, np.nan, oil_data['dcoilwtico'])
oil_data['interpolated_price'] = oil_data.dcoilwtico.interpolate()
oil_data = oil_data.drop('dcoilwtico',axis=1)
#撤除延迟的节日
holiday_data=holiday_data.drop(holiday_data[holiday_data['transferred']==True].index)
'''
```

在查看数据信息及缺失值后，发现仅有油价存在缺失值，且缺失值较总数来说很少，故直接对其插值填充。

(三) 查看数据标签值与各属性的统计特征、相关性

查看商品销量与各个属性之间关系，确定模型训练时需要选择哪些样本标签或特征。

1. 查看油价随时间的变化趋势图

```
'''
查看数据标签值与各个属性的统计特征、相关性
'''
#油价时间趋势
fig,ax = plt.subplots(figsize=(15,5))
plt.plot(oil_data['interpolated_price'])
ax.xaxis.set_major_locator(mdates.MonthLocator(interval=1))
plt.xticks(rotation=70)
plt.title("油价时间趋势")
plt.show()
#看到2015年01月后油价大幅下降，2016年02月油价下降幅度也较大
```

2. 查看商品总销量、月度销量的时间趋势图并对商品按销量进行排序

```
#计算不同商品每月销量
train_data['date']=pd.to_datetime(train_data['date'])

print('训练数据按日期排列：\n',train_data.groupby('date').sales.sum().sort_values().head(10))


#查看所有商品总销量的时间趋势
fig, ax = plt.subplots(figsize=(18, 7))
ax.set(title="所有商品总销量时间趋势")
total_sales = train_data.groupby('date').sales.sum()
plt.plot(total_sales)

ax.xaxis.set_major_locator(mdates.MonthLocator(interval=1))
plt.xticks(rotation=70)
plt.axvline(x=pd.Timestamp('2016-04-16'),color='r',linestyle='--',linewidth=4,alpha=0.3)
plt.text(pd.Timestamp('2016-04-20'),1400000,'地震',rotation=360,c='r')
plt.axvline(x=pd.Timestamp('2015-01'),color='g',linestyle='--',linewidth=4,alpha=0.3)
plt.text(pd.Timestamp('2015-01'),1400000,'油价大幅下降',rotation=360,c='g')
plt.axvline(x=pd.Timestamp('2016-01'),color='g',linestyle='--',linewidth=4,alpha=0.3)
plt.text(pd.Timestamp('2016-01'),1400000,'油价大幅下降',rotation=360,c='g')
plt.show()
```

```
#不同商品月度销量的时间趋势
date_fam_sale =train_data.groupby(['date','family']).sum().sales
unstack = date_fam_sale.unstack()
unstack = unstack.resample('M').sum()
fig, ax = plt.subplots(figsize=(20, 10))
ax.set(title="不同商品月度总销量时间趋势")
plt.stackplot(unstack.index,unstack.T,labels=unstack.T.index)
ax.xaxis.set_major_locator(mdates.MonthLocator(interval=1))
plt.axvline(x=pd.Timestamp('2016-04-16'),color='black',linestyle='--',linewidth=4,alpha=0.5)
plt.text(pd.Timestamp('2016-04-20'),30000000,' 地震',rotation=360,c='black',size=16)
plt.xticks(rotation=90)
plt.axvline(x=pd.Timestamp('2015-01'),color='g',linestyle='--',linewidth=4,alpha=0.3)
plt.text(pd.Timestamp('2015-01'),30000000,'油价大幅下降',rotation=360,c='g')
plt.axvline(x=pd.Timestamp('2016-01'),color='g',linestyle='--',linewidth=4,alpha=0.3)
plt.text(pd.Timestamp('2016-01'),30000000,'油价大幅下降',rotation=360,c='g')
plt.legend(loc='lower center',bbox_to_anchor=(0.5,-0.2),ncol=11)
plt.show()



#月度数据显示，明显，虽然各个商品的销量随时间变化而变化，但不同商品销量的排序一直没有改变
```

```
#对不同商品销量进行排序
train_data[['store_nbr','family']].astype('category')
month_family = train_data.groupby('family').agg('sum')#不同商品总销量
fig, ax = plt.subplots(figsize=(7,7))
plt.barh(month_family.sales.sort_values().index,month_family.sales.sort_values())
ax.set(title='不同商品总销量排序(所有时间)')
plt.show()
```

3. 查看不同商店月度销量的时间趋势图并对商店按销量进行排序

```
#不同商店月度销量的时间趋势
date_fam_sale =train_data.groupby(['date','store_nbr']).sum().sales
unstack = date_fam_sale.unstack()
unstack = unstack.resample('M').sum()
fig, ax = plt.subplots(figsize=(20, 10))
ax.set(title="不同商店月度总销量时间趋势")
plt.stackplot(unstack.index,unstack.T,labels=unstack.T.index)
ax.xaxis.set_major_locator(mdates.MonthLocator(interval=1))
plt.axvline(x=pd.Timestamp('2016-04-16'),color='black',linestyle='--',linewidth=4,alpha=0.5)
plt.text(pd.Timestamp('2016-04-20'),30000000,' 地震',rotation=360,c='black',size=16)
plt.axvline(x=pd.Timestamp('2015-01'),color='g',linestyle='--',linewidth=4,alpha=0.3)
plt.text(pd.Timestamp('2015-01'),30000000,'油价大幅下降',rotation=360,c='g')
plt.axvline(x=pd.Timestamp('2016-01'),color='g',linestyle='--',linewidth=4,alpha=0.3)
plt.text(pd.Timestamp('2016-01'),30000000,'油价大幅下降',rotation=360,c='g')
plt.xticks(rotation=90)
plt.legend(loc='best',bbox_to_anchor=(0.5,-0.2),ncol=11)
plt.show()

#月度数据显示，明显，虽然各个商店的销量随时间变化而变化，但不同商品销量的排序改变幅度较小
#对不同商店销量排序
month_store = train_data.groupby('store_nbr').agg('sum')#不同商品总销量
fig, ax = plt.subplots(figsize=(7,7))
plt.barh(month_store.sales.sort_values().index,month_store.sales.sort_values())
ax.set(title='不同商店总销量排序(所有时间)')
plt.show()
#不同商店销量差别较大，考虑可能是商店所在城市有所影响
```

4. 查看促销活动与商品销量之间散点图

```
#考虑促销活动对销量的影响

import matplotlib.pyplot as plt


y = month_family['sales'] # 因变量
x = month_family['onpromotion'] # 自变量
x = sm.add_constant(x) # 截距项
model = sm.OLS(y, x).fit() # 构建最小二乘模型并拟合

predicts = model.predict() # 模型的预测值

plt.scatter(x.onpromotion, y, cmap='plasma',label='实际值') # 散点图
plt.plot(x.onpromotion, predicts, color = 'red', label='预测值')
plt.legend() # 显示图例，即每条线对应 label 中的内容
plt.xlabel('促销次数')
plt.ylabel('商品销量')
plt.title('不同商品促销次数与销量关系')
plt.show()
```

5. 查看交易量与每个商店不同商品销量之间散点图

```
#考虑销量与交易量关系，将两个dataframe合并，撇除无用属性
transaction_data = transaction_data.set_index('date')
transaction_data.index = pd.to_datetime(transaction_data.index)
def transaction_sales_dic(transaction_df,sale_dic):
    transaction_dic = {}
    sale_dict = sale_dic.copy()

    for i in transaction_df['store_nbr'].unique():
        store_transacion = transaction_df.loc[transaction_df['store_nbr'] == i]
        transaction_dic[i] = store_transacion['transactions']

    for i in sale_dict.keys():
        sale_dict[i] = sale_dict[i].groupby(['date','store_nbr']).sales.sum()
        sale_dict[i] = sale_dict[i].reset_index()
        sale_dict[i] = sale_dict[i].drop('store_nbr', axis=1)
        sale_dict[i] = sale_dict[i].groupby('date').sales.sum()

    return transaction_dic, sale_dict

def  series_merge_inner_index(dic1, dic2):
    merged_dic = {}
    for key in dic1.keys():
        merged_dic[key] = dic1[key].to_frame().merge(dic2[key].to_frame(), how='inner',
                                                     left_index=True, right_index=True)
    return merged_dic
#计算不同商店每个商品每日销量
daily_sale_dict = {}
for i in train_data.store_nbr.unique():
    daily_sale = train_data[train_data['store_nbr']==i]
    daily_sale_dict[i] = daily_sale
print('日销：\n',daily_sale)
transaction_dic, sale_dic = transaction_sales_dic(transaction_data,daily_sale_dict)
merged_sales_transaction = series_merge_inner_index(transaction_dic, sale_dic)
#绘制交易量、销量散点图像
fig, ax = plt.subplots(figsize=(15,7))
for key in merged_sales_transaction.keys():
    plt.scatter(merged_sales_transaction[key].transactions,
                merged_sales_transaction[key].sales,cmap='plasma')
print('合并后表格：\n',merged_sales_transaction[1])
plt.title('交易量与销量关系')
plt.xlabel('交易量')
plt.ylabel('销量')
plt.show()
#以上图示表明，销量的时间趋势较明显，且受商品种类、商店序号、油价、地震、促销、交易量等的影响
```

(四) 构建训练数据集并建立模型进行训练

1. 建立训练数据集

```
"""
构建训练数据集
"""
#将商店地点转化为onehot编码
store_location = store_data.drop(['state','type','cluster'],axis=1)
 # 丢掉州名、类型及商店聚类
store_location = store_location.set_index('store_nbr')
store_location = pd.get_dummies(store_location,prefix='store_loc_')
print(store_location.head(10))
#地点添加到训练数据中去
train=train_data.reset_index().merge(store_location,how='outer',
                                     left_on='store_nbr',
                                     right_on=store_location.index)
print('缺失值情况：\n',train.isna().sum())
oil_data['interpolated_price']=oil_data['interpolated_price'].astype('float')
print('oil_data缺失值情况：\n',oil_data.isna().sum())
oil_data=oil_data.dropna(axis=0)
#将油价添加进去
train=train.merge(oil_data,how='inner',left_on='date',right_on='date')
print('缺失值情况：\n',train.isna().sum())
print(oil_data)
#将交易量添加进去
train.reset_index().merge(transaction_data.reset_index(),how='left',
                                      left_on=['date','store_nbr'],
                                      right_on=['date','store_nbr']
                                     )

new_cols = [col for col in train.columns if col != 'sales'] + ['sales']
train=train.reindex(columns=new_cols)

train.dropna(inplace = True)
train = train.set_index('date')
train=train.drop(['index','id'],axis=1)

train = train.reset_index()
train = train.set_index(['date','store_nbr','family'])
print('缺失值情况：\n',train.isna().sum())
print('训练数据示例：\n',train.head(10))

"""
```

在了解数据样本属性与标签的统计特征、相关关系后，发现，销量与时间、油价、交易量、商店编号（商店所在地）、商品类别等均存在明显的相关性，将需要的标签添加到训练数据中去。

2. 训练模型

(1) 线性回归模型与随机森林模型

```
"""
训练模型
"""
X=train.iloc[:,:-1]
y=train.iloc[:,-1]

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
lr=LinearRegression()
lr.fit(X_train,y_train)
y_pred=abs(lr.predict(X_test))
print('线性回归RMSLE：',mean_squared_log_error(y_test, y_pred))

R=RandomForestRegressor()
R.fit(X_train,y_train)
yr_pred=(X_test)
print('随机森林RMSLE：',mean_squared_log_error(y_test, yr_pred))
```

(2) ISLM 神经网络模型

```
'''
尝试ISLM网络
'''
#得到行列值
row = train.shape[0]
col = train.shape[1]
#将数据变成数组形式
data_value = train.values
#定义训练集和测试集大小
train_start = 0
train_end = int(np.floor(0.8*row))
test_start = train_end-1
test_end =row
#划分训练集和测试集
data_train = data_value[np.arange(train_start,train_end),:]
data_test = data_value[np.arange(test_start,test_end),:]
#数据归一化，减小噪音样本的影响
scaler = MinMaxScaler()
scaler.fit(data_train)
data_train = scaler.transform(data_train)
data_test = scaler.transform(data_test)
'''生成训练数据和测试数据
'''
time_step = 4     #时间步
#划分训练集
X_train,y_train,batch_index=[],[],[]
for i in range(len(data_train)-time_step):
    batch_index.append(i)
    x=data_train[i:i+time_step,:]
    y=data_train[i+time_step,:]
    X_train.append(x.tolist())
    y_train.append(y.tolist())
batch_index.append((len(data_train)-time_step))
#划分测试集
X_test,y_test=[],[]
for i in range(len(data_test)-time_step):
    x=data_test[i:i+time_step,:]
    y=data_test[i+time_step,:]
    X_test.append(x.tolist())
    y_test.append(y.tolist())
X_train, y_train = np.array(X_train), np.array(y_train)
X_test, y_test = np.array(X_test), np.array(y_test)
X_train = np.reshape(X_train,(X_train.shape[0], time_step, col))
X_test = np.reshape(X_test,(X_test.shape[0], time_step, col))
```

```
'''生成网络
'''
model = tf.keras.Sequential([
    LSTM(20, input_shape=(X_train.shape[1:]), return_sequences=True),
    Dropout(0.2),
    LSTM(20),
    Dropout(0.2),
    Dense(y_train.shape[1])
])
model.compile(optimizer='adam', loss='mse')
'''模型训练与预测
'''
# 训练模型
model.fit(X_train, y_train, epochs=200, validation_data=[X_test, y_test])

#预测
test_predict = model.predict(X_test)

y_predict = np.ravel(test_predict)
y_target = np.ravel(y_test)
print('ISLM网络RMSLE: ',mean_squared_log_error(y_target, y_predict))
```

四、实验结果及分析

(一) 数据信息及缺失值情况

图表 2：程序结果

```
有 33 种不同商品类别
有 54 个不同的商店
有 22 个不同的地点
有 103 个不同的假日
```

```
交易数据缺失值:
 date            0
store_nbr        0
transactions     0
dtype: int64
```

```
油价数据缺失值:
 date            0
dcoilwtico      43
dtype: int64
```

```
训练数据缺失值:
 id              0
date             0
store_nbr        0
family           0
sales            0
onpromotion      0
dtype: int64
```

```
假日数据缺失值:
 date            0
type             0
locale           0
locale_name      0
description      0
transferred      0
dtype: int64
```

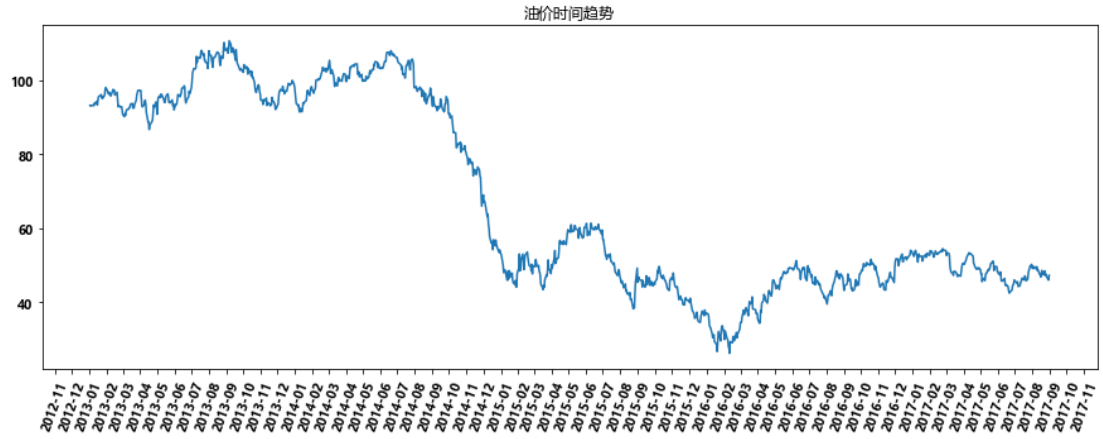
```
商店数据缺失值:
 store_nbr       0
city             0
state            0
type             0
cluster          0
dtype: int64
```

程序结果显示，只有油价存在缺失值，且相对于油价数据量而言缺失值极少，可以直接对其进行插值处理。且节假日数量较多，考虑到转换为 onehot 编码后数据维度太高，在训练模型时，直接将其舍弃，不作为输入参数。

(二) 数据标签值（销量）与各属性的统计特征、相关性

1. 油价时间趋势

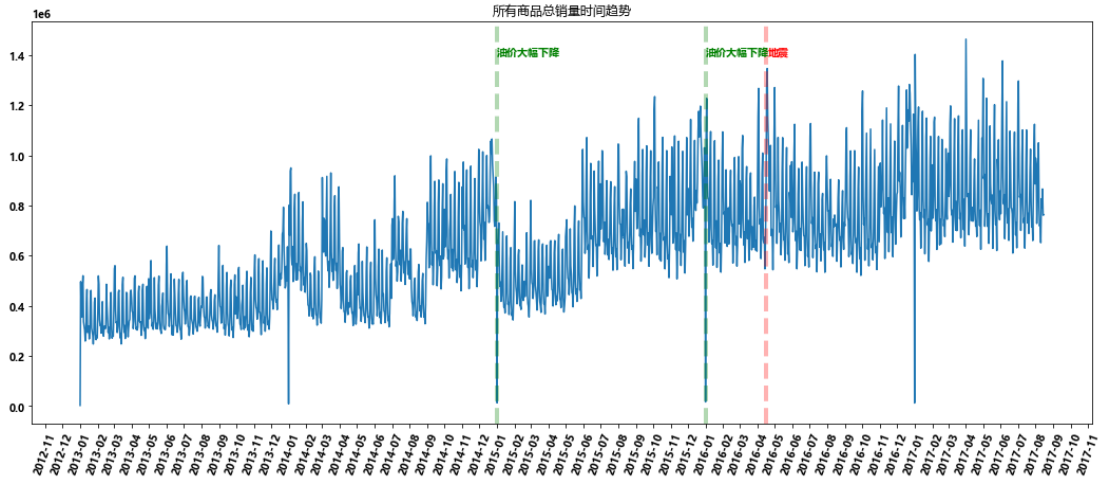
图表 3：油价时间趋势图



如图可知，油价呈现下行趋势，且分别在 2015 年 1 月，2016 年 1 月有极其大的下跌幅度，在查看商品时间销量时可以更加注意。

2. 商品总销量时间趋势

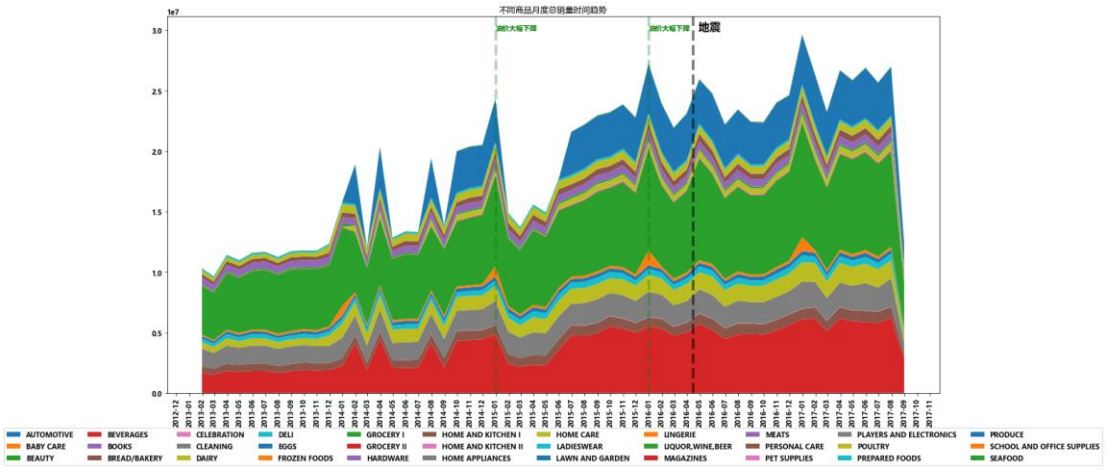
图表 4：所有商品总销量时间趋势图



可以看到，两次油价下降及一次地震，均造成了商品销量的瞬时增加，且下降幅度越大，增加效果也越显著。

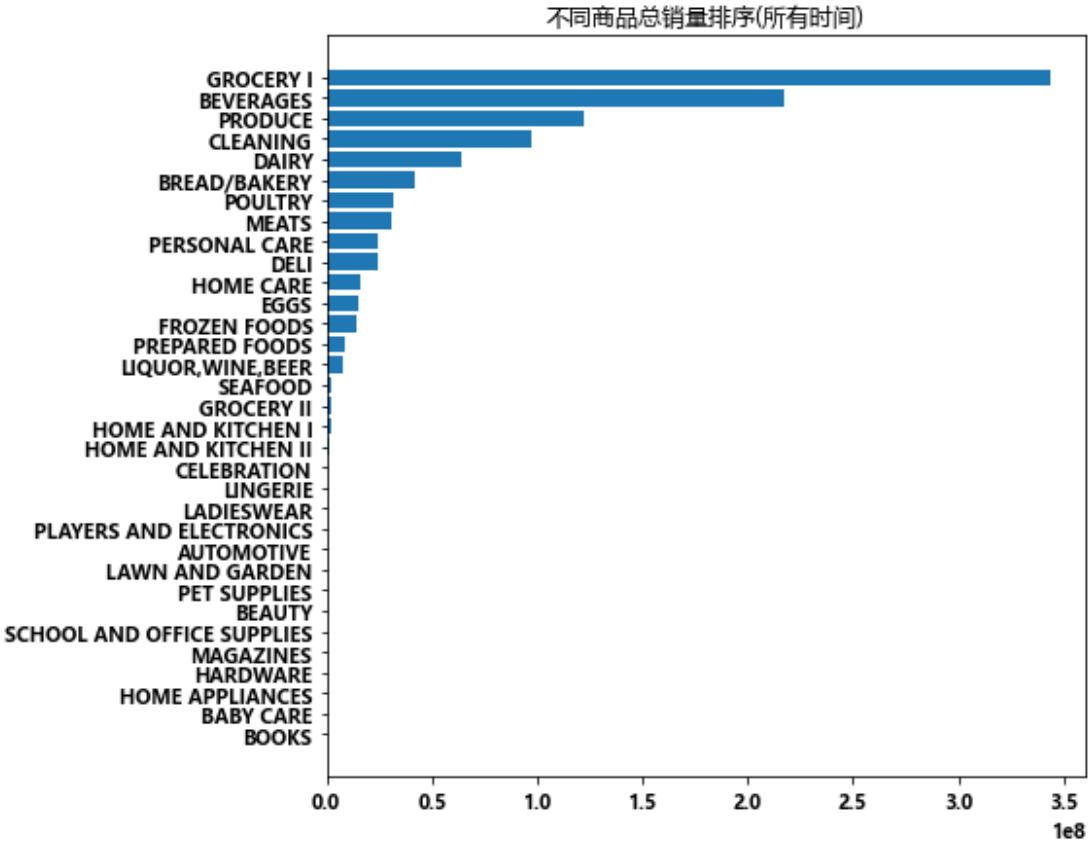
3. 不同商品月度销量时间趋势及商品销量排序

图表 5：不同商品月度销量时间趋势



如图，可以看到，随着时间变化，尽管商品销量不断变化，但不同类商品销量排序始终保持不变，进一步，对不同类商品总销量进行排序。

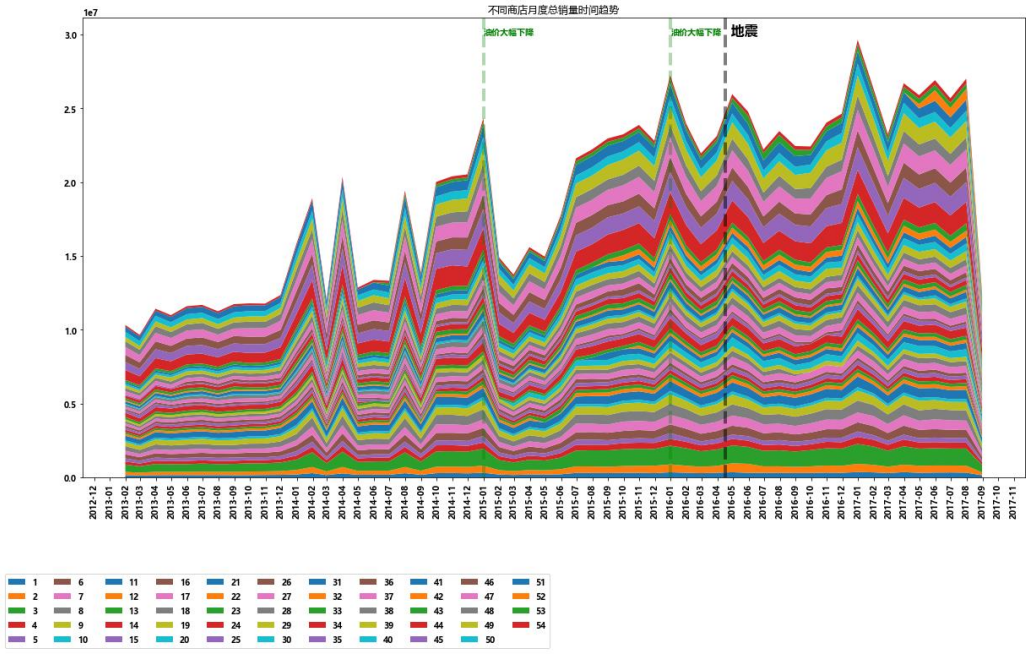
图表 6：商品销量排序



如图，可以看到，商品销量间大小差异巨大，销量最大的商品时 grocery 果蔬，符合常理，且 33 种商品中，销量前三大的商品占据总销量超 50%。

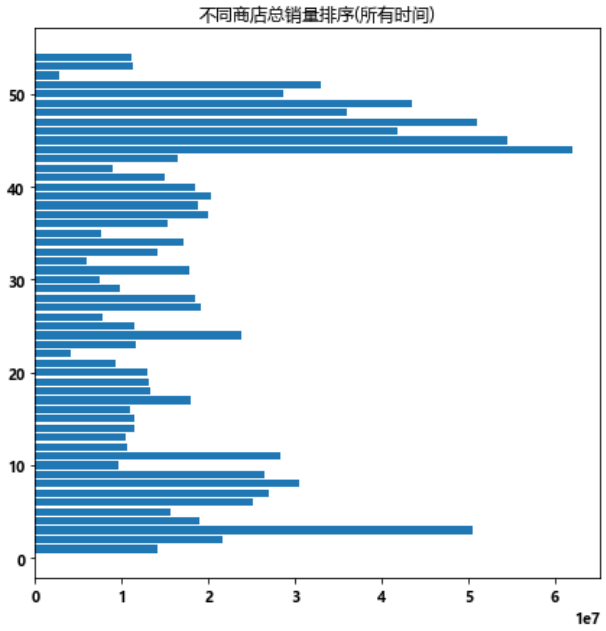
4. 不同商店月度销量时间趋势及商品销量排序

图表 7：不同商店月度销量时间趋势



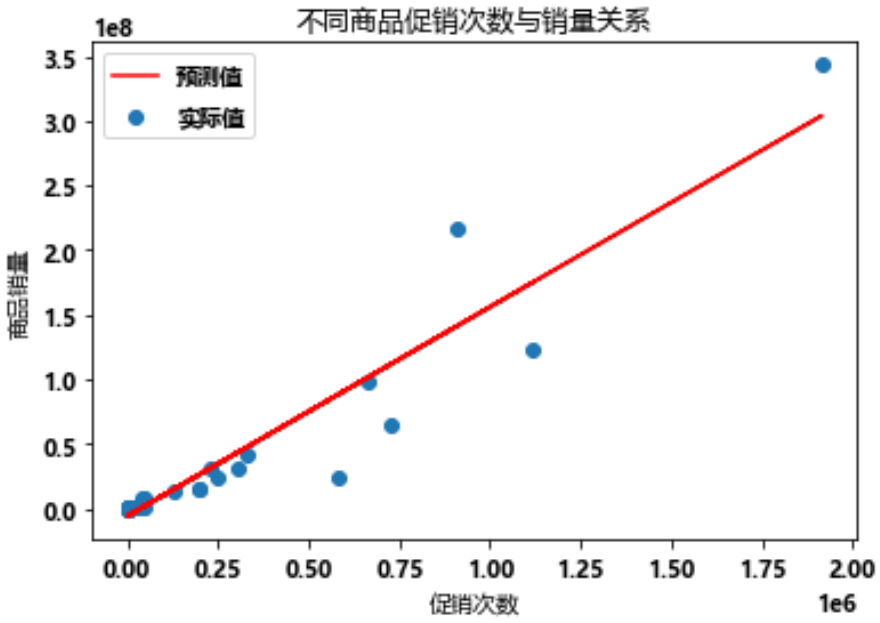
如图 7，与不同商品间的销量比较类似，不同商店销量之间排序也基本保持不变。进一步画出商店销量条形图。如下图，不同商店间销量存在差异，但差异不像商品间那么大，销量分布相对均匀。

图表 8：不通商店销量条形图



5. 不同商品促销次数与销量关系

图表 9：促销次数与销量关系

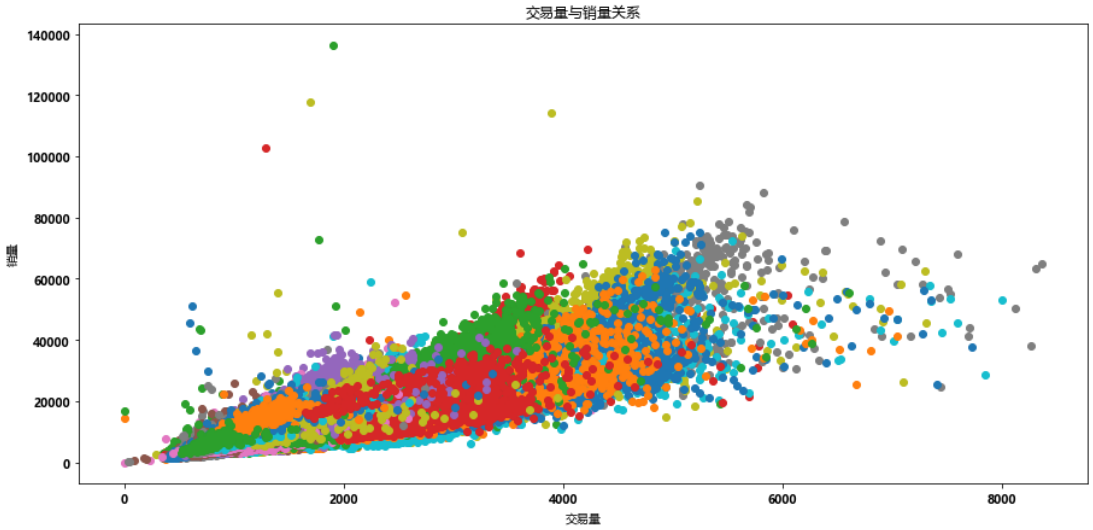


如图，每个点由一类商品的促销次数与总销量组成，散点图表明，促销越多

的商品，其销量也越大，且大多数点满足线性关系。

6. 交易量与销量关系

图表 10：交易量与销量散点图



如图，交易量越多，销量也越多。

(三) 模型训练结果

根据以上图表 1-9，可知，**商品销量与油价、时间、商品类别、促销次数、商店编号（商店所在地点）等均有较明显关系**，均加入训练数据作为样本的属性。

以 RMSLE 为评价指标，最终结果如下：

图表 11：模型 RMSLE

模型类别	RMSLE
线性回归模型	10.651001
随机森林模型	7.1527024
ISLM 神经网络	13.4598207

五、总结与反思

- 尽管做了许多统计分析，绘制了许多图，对样本属性特征进行了仔细地选取，但并未考虑到不同属性之间是否相关性很大有可能造成属性特征间存在一定的共线性、同方差性等，使其在线性回归模型中表现不佳，在随机森林模型中也表现欠佳。
- 数据为时间序列数据，波动性极强，在使用 ISLM 模型时，设置的时间步为 4 步，也就是只考虑销量的前四天对他的影响，但是根据销量时间趋势图可知，油价下降幅度极大的那两次，销量在一个月甚至两个月内的关联性都很强，或许调整时间步使其更长，ISLM 模型的预测效果会好些。

六、代码

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 10 07:37:22 2022

@author: 李任
"""


"""
导入相关包
"""
import pandas as pd#线性代数运算
import numpy as np

from sklearn.model_selection import train_test_split #划分数据
import tensorflow as tf#机器学习、神经网络
from pylab import mpl
from sklearn.linear_model import LinearRegression#线性回归
from sklearn.ensemble import RandomForestRegressor#随机森林
from tensorflow.keras.layers import LSTM, Dense, Dropout#LSTM 长短周期记忆网
络
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.metrics import mean_squared_log_error#模型评价指标
import matplotlib.pyplot as plt#绘图
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib
import seaborn as sns
import statsmodels.api as sm#统计分析

matplotlib.rc("font",family='MicroSoft YaHei',weight="bold")#画图时显示中文

"""
读取数据
"""
oil_data = pd.read_csv(r"D:\My_coding\kaggle_dataset\store-sales-time-series-forecasting\oil.csv")
holiday_data = pd.read_csv(r"D:\My_coding\kaggle_dataset\store-sales-time-series-forecasting\holidays_events.csv")
store_data = pd.read_csv(r"D:\My_coding\kaggle_dataset\store-sales-time-series-forecasting\stores.csv")
train_data = pd.read_csv(r"D:\My_coding\kaggle_dataset\store-sales-time-series-forecasting\train.csv")
transaction_data = pd.read_csv(r"D:\My_coding\kaggle_dataset\store-sales-time-series-forecasting\transactions.csv")
sample = pd.read_csv(r'D:\My_coding\kaggle_dataset\store-sales-time-series-forecasting\sample_submission.csv')
#查看训练数据情况
print('训练数据信息:\n',train_data.info())
print(train_data.head())
print('油价信息 n',oil_data.info())
print(oil_data.head())
print('假日信息:\n',holiday_data.info())
print(holiday_data.head())
print('有 {} 种不同商品类别'.format(train_data.family.nunique()))
print('有 {} 个不同的商店'.format(train_data.store_nbr.nunique()))
print('有 {} 个不同的地点'.format(store_data.city.nunique()))
print('有 {} 个不同的假日'.format(holiday_data.description.nunique()))
#103 个假日，实在太多，转化为 onehot 编码数据量过大，
#而且一般而言假日与促销与否关系很大，此处直接舍去假日，不作为训练参数
#检查缺失值
print('训练数据缺失值:\n',train_data.isna().sum())
print('油价数据缺失值:\n',oil_data.isna().sum())
print('假日数据缺失值:\n',holiday_data.isna().sum())
print('商店数据缺失值:\n',store_data.isna().sum())
```

```
print('交易数据缺失值:\n',transaction_data.isna().sum())


'''
数据预处理
'''

#只有油价有缺失值,缺失值数量不多，直接插值

oil_data['date']=pd.to_datetime(oil_data['date'])
oil_data = oil_data.set_index('date')
oil_data=oil_data.resample('1D').sum()
oil_data.reset_index()
oil_data['dcoilwtico'] = np.where(oil_data['dcoilwtico']==0, np.nan,
oil_data['dcoilwtico'])
oil_data['interpolated_price'] = oil_data.dcoilwtico.interpolate()
oil_data = oil_data.drop('dcoilwtico',axis=1)
#撇除延迟的节日
holiday_data=holiday_data.drop(holiday_data[holiday_data['transferred']==True].in
dex)
'''
查看数据标签值与各个属性的统计特征、相关性
'''
#油价时间趋势
fig,ax = plt.subplots(figsize=(15,5))
plt.plot(oil_data['interpolated_price'])
ax.xaxis.set_major_locator(mdates.MonthLocator(interval=1))
plt.xticks(rotation=70)
plt.title("油价时间趋势")
plt.show()
#看到 2015 年 01 月后油价大幅下降，2016 年 02 月油价下降幅度也较大

#计算不同商品每月销量
train_data['date']=pd.to_datetime(train_data['date'])

print('训练数据按日期排列:
\n',train_data.groupby('date').sales.sum().sort_values().head(10))


#查看所有商品总销量的时间趋势
fig, ax = plt.subplots(figsize=(18, 7))
ax.set(title="所有商品总销量时间趋势")
total_sales = train_data.groupby('date').sales.sum()
```

```
        plt.plot(total_sales)

        ax.xaxis.set_major_locator(mdates.MonthLocator(interval=1))
        plt.xticks(rotation=70)
        plt.axvline(x=pd.Timestamp('2016-04-16'),color='r',linestyle='--
',linewidth=4,alpha=0.3)
        plt.text(pd.Timestamp('2016-04-20'),1400000,'地震',rotation=360,c='r')
        plt.axvline(x=pd.Timestamp('2015-01'),color='g',linestyle='--
',linewidth=4,alpha=0.3)
        plt.text(pd.Timestamp('2015-01'),1400000,'油价大幅下降',rotation=360,c='g')
        plt.axvline(x=pd.Timestamp('2016-01'),color='g',linestyle='--
',linewidth=4,alpha=0.3)
        plt.text(pd.Timestamp('2016-01'),1400000,'油价大幅下降',rotation=360,c='g')
        plt.show()


        #不同商品月度销量的时间趋势
        date_fam_sale =train_data.groupby(['date','family']).sum().sales
        unstack = date_fam_sale.unstack()
        unstack = unstack.resample('M').sum()
        fig, ax = plt.subplots(figsize=(20, 10))
        ax.set(title="不同商品月度总销量时间趋势")
        plt.stackplot(unstack.index,unstack.T,labels=unstack.T.index)
        ax.xaxis.set_major_locator(mdates.MonthLocator(interval=1))
        plt.axvline(x=pd.Timestamp('2016-04-16'),color='black',linestyle='--
',linewidth=4,alpha=0.5)
        plt.text(pd.Timestamp('2016-04-20'),30000000,'  地震
',rotation=360,c='black',size=16)
        plt.xticks(rotation=90)
        plt.axvline(x=pd.Timestamp('2015-01'),color='g',linestyle='--
',linewidth=4,alpha=0.3)
        plt.text(pd.Timestamp('2015-01'),30000000,'油价大幅下降',rotation=360,c='g')
        plt.axvline(x=pd.Timestamp('2016-01'),color='g',linestyle='--
',linewidth=4,alpha=0.3)
        plt.text(pd.Timestamp('2016-01'),30000000,'油价大幅下降',rotation=360,c='g')
        plt.legend(loc='lower center',bbox_to_anchor=(0.5,-0.2),ncol=11)
        plt.show()


    #月度数据显示，明显，虽然各个商品的销量随时间变化而变化，但不同商品销
量的排序一直没有改变
```

```
#对不同商品销量进行排序
train_data[['store_nbr','family']].astype('category')
month_family = train_data.groupby('family').agg('sum')#不同商品总销量
fig, ax = plt.subplots(figsize=(7,7))
plt.barh(month_family.sales.sort_values().index,month_family.sales.sort_values())
ax.set(title='不同商品总销量排序(所有时间)')
plt.show()


#不同商店月度销量的时间趋势
date_fam_sale =train_data.groupby(['date','store_nbr']).sum().sales
unstack = date_fam_sale.unstack()
unstack = unstack.resample('M').sum()
fig, ax = plt.subplots(figsize=(20, 10))
ax.set(title="不同商店月度总销量时间趋势")
plt.stackplot(unstack.index,unstack.T,labels=unstack.T.index)
ax.xaxis.set_major_locator(mdates.MonthLocator(interval=1))
plt.axvline(x=pd.Timestamp('2016-04-16'),color='black',linestyle='--
',linewidth=4,alpha=0.5)
plt.text(pd.Timestamp('2016-04-20'),30000000,' 地震
',rotation=360,c='black',size=16)
plt.axvline(x=pd.Timestamp('2015-01'),color='g',linestyle='--
',linewidth=4,alpha=0.3)
plt.text(pd.Timestamp('2015-01'),30000000,'油价大幅下降',rotation=360,c='g')
plt.axvline(x=pd.Timestamp('2016-01'),color='g',linestyle='--
',linewidth=4,alpha=0.3)
plt.text(pd.Timestamp('2016-01'),30000000,'油价大幅下降',rotation=360,c='g')
plt.xticks(rotation=90)
plt.legend(loc='best',bbox_to_anchor=(0.5,-0.2),ncol=11)
plt.show()

#月度数据显示，明显，虽然各个商店的销量随时间变化而变化，但不同商品销
量的排序改变幅度较小
#对不同商店销量排序
month_store = train_data.groupby('store_nbr').agg('sum')#不同商品总销量
fig, ax = plt.subplots(figsize=(7,7))
plt.barh(month_store.sales.sort_values().index,month_store.sales.sort_values())
ax.set(title='不同商店总销量排序(所有时间)')
plt.show()
#不同商店销量差别较大，考虑可能是商店所在城市有所影响


#考虑促销活动对销量的影响
```

```
import matplotlib.pyplot as plt


y = month_family['sales'] # 因变量
x = month_family['onpromotion'] # 自变量
x = sm.add_constant(x) # 截距项
model = sm.OLS(y, x).fit() # 构建最小二乘模型并拟合

predicts = model.predict() # 模型的预测值

plt.scatter(x.onpromotion, y, cmap='plasma',label='实际值') # 散点图
plt.plot(x.onpromotion, predicts, color = 'red', label='预测值')
plt.legend() # 显示图例，即每条线对应 label 中的内容
plt.xlabel('促销次数')
plt.ylabel('商品销量')
plt.title('不同商品促销次数与销量关系')
plt.show()


#考虑销量与交易量关系，将两个 dataframe 合并，撇除无用属性
transaction_data = transaction_data.set_index('date')
transaction_data.index = pd.to_datetime(transaction_data.index)
def transaction_sales_dic(transaction_df,sale_dic):
    transaction_dic = {}
    sale_dict = sale_dic.copy()

    for i in transaction_df['store_nbr'].unique():
        store_transacion = transaction_df.loc[transaction_df['store_nbr'] == i]
        transaction_dic[i] = store_transacion['transactions']

    for i in sale_dict.keys():
        sale_dict[i] = sale_dict[i].groupby(['date','store_nbr']).sales.sum()
        sale_dict[i] = sale_dict[i].reset_index()
        sale_dict[i] = sale_dict[i].drop('store_nbr', axis=1)
        sale_dict[i] = sale_dict[i].groupby('date').sales.sum()

    return transaction_dic, sale_dict

def   series_merge_inner_index(dic1, dic2):
    merged_dic = {}
    for key in dic1.keys():
        merged_dic[key] = dic1[key].to_frame().merge(dic2[key].to_frame(),
```

```
how='inner',
                                                    left_index=True,
right_index=True)
        return merged_dic
    #计算不同商店每个商品每日销量
    daily_sale_dict = {}
    for i in train_data.store_nbr.unique():
        daily_sale = train_data[train_data['store_nbr']==i]
        daily_sale_dict[i] = daily_sale
    print('日销：\n',daily_sale)
    transaction_dic, sale_dic = transaction_sales_dic(transaction_data,daily_sale_dict)
    merged_sales_transaction = series_merge_inner_index(transaction_dic, sale_dic)
    #绘制交易量、销量散点图像
    fig, ax = plt.subplots(figsize=(15,7))
    for key in merged_sales_transaction.keys():
        plt.scatter(merged_sales_transaction[key].transactions,
                    merged_sales_transaction[key].sales,cmap='plasma')
    print('合并后表格：\n',merged_sales_transaction[1])
    plt.title('交易量与销量关系')
    plt.xlabel('交易量')
    plt.ylabel('销量')
    plt.show()
    #以上图示表明，销量的时间趋势较明显，且受商品种类、商店序号、油价、地
震、促销、交易量等的影响


    """
    构建训练数据集
    """
    #将商店地点转化为 onehot 编码
    store_location = store_data.drop(['state','type','cluster'],axis=1)
     # 丢掉州名、类型及商店聚类
    store_location = store_location.set_index('store_nbr')
    store_location = pd.get_dummies(store_location,prefix='store_loc_')
    print(store_location.head(10))
    #地点添加到训练数据中去
    train=train_data.reset_index().merge(store_location,how='outer',
                                         left_on='store_nbr',
                                         right_on=store_location.index)
    print('缺失值情况：\n',train.isna().sum())
    oil_data['interpolated_price']=oil_data['interpolated_price'].astype('float')
    print('oil_data 缺失值情况：\n',oil_data.isna().sum())
    oil_data=oil_data.dropna(axis=0)
```

```
#将油价添加进去
train=train.merge(oil_data,how='inner',left_on='date',right_on='date')
print('缺失值情况：\n',train.isna().sum())
print(oil_data)
#将交易量添加进去
train.reset_index().merge(transaction_data.reset_index(),how='left',
                                                    left_on=['date','store_nbr'],
                                                    right_on=['date','store_nbr']
                                                   )

new_cols = [col for col in train.columns if col != 'sales'] + ['sales']
train=train.reindex(columns=new_cols)

train.dropna(inplace = True)
train = train.set_index('date')
train=train.drop(['index','id'],axis=1)

train = train.reset_index()
train = train.set_index(['date','store_nbr','family'])
print('缺失值情况：\n',train.isna().sum())
print('训练数据示例：\n',train.head(10))

"""
训练模型
"""
X=train.iloc[:,:-1]
y=train.iloc[:,-1]

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
lr=LinearRegression()
lr.fit(X_train,y_train)
y_pred=abs(lr.predict(X_test))
print('线性回归 RMSLE：',mean_squared_log_error(y_test, y_pred))

R=RandomForestRegressor()
R.fit(X_train,y_train)
yr_pred=(X_test)
print('随机森林 RMSLE：',mean_squared_log_error(y_test, yr_pred))


'''
尝试 ISLM 网络
'''
```

```
#得到行列值
row = train.shape[0]
col = train.shape[1]
#将数据变成数组形式
data_value = train.values
#定义训练集和测试集大小
train_start = 0
train_end = int(np.floor(0.8*row))
test_start = train_end-1
test_end =row
#划分训练集和测试集
data_train = data_value[np.arange(train_start,train_end),:]
data_test = data_value[np.arange(test_start,test_end),:]
#数据归一化，减小噪音样本的影响
scaler = MinMaxScaler()
scaler.fit(data_train)
data_train = scaler.transform(data_train)
data_test = scaler.transform(data_test)
'''生成训练数据和测试数据
'''
time_step = 4      #时间步
#划分训练集
X_train,y_train,batch_index=[],[],[]
for i in range(len(data_train)-time_step):
    batch_index.append(i)
    x=data_train[i:i+time_step,:]
    y=data_train[i+time_step,:]
    X_train.append(x.tolist())
    y_train.append(y.tolist())
batch_index.append((len(data_train)-time_step))
#划分测试集
X_test,y_test=[],[]
for i in range(len(data_test)-time_step):
    x=data_test[i:i+time_step,:]
    y=data_test[i+time_step,:]
    X_test.append(x.tolist())
    y_test.append(y.tolist())
X_train, y_train = np.array(X_train), np.array(y_train)
X_test, y_test = np.array(X_test), np.array(y_test)
X_train = np.reshape(X_train,(X_train.shape[0], time_step, col))
X_test = np.reshape(X_test,(X_test.shape[0], time_step, col))
'''生成网络
'''
```

```
        model = tf.keras.Sequential([
            LSTM(20, input_shape=(X_train.shape[1:]), return_sequences=True),
            Dropout(0.2),
            LSTM(20),
            Dropout(0.2),
            Dense(y_train.shape[1])
        ])
        model.compile(optimizer='adam', loss='mse')
        '''模型训练与预测
        '''
        # 训练模型
        model.fit(X_train, y_train, epochs=200, validation_data=[X_test, y_test])

        #预测
        test_predict = model.predict(X_test)

        y_predict = np.ravel(test_predict)
        y_target = np.ravel(y_test)
    print('ISLM 网络 RMSLE:  ',mean_squared_log_error(y_target, y_predict))
```