cse519_hw3_KaranDipesh_Gada_113082700

October 19, 2021

0.1 Use the "Text" blocks to provide explanations wherever you find them necessary. Highlight your answers inside these text fields to ensure that we don't miss it while grading your HW.

0.2 Setup

- Code to download the data directly from the colab notebook.
- If you find it easier to download the data from the kaggle website (and uploading it to your drive), you can skip this section.

[1]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

[2]:
```
# First mount your drive before running these cells.
# Create a folder for the this HW and change to that dir
%cd drive/My\ Drive/CSE519/HW3
%pwd
```

```
/content/drive/My Drive/CSE519/HW3
```

[2]:
```
'/content/drive/My Drive/CSE519/HW3'
```

[ ]:
```
!pip install -q kaggle
```

[ ]:
```
from google.colab import files
# Create a new API token under "Account" in the kaggle webpage and download the␣
 ↪json file
# Upload the file by clicking on the browse
files.upload()
```

```
<IPython.core.display.HTML object>


Saving kaggle.json to kaggle.json
```

[ ]:
```
{'kaggle.json':
b'{"username":"karangada","key":"cd7aa263c4ab85a8083aad5f814084fa"}'}
```

[ ]:
```
%mkdir /root/.kaggle
%cp kaggle.json /root/.kaggle/
```

[ ]:
```
!kaggle competitions download -c  rossmann-store-sales
```

```
Warning: Looks like you're using an outdated API Version, please consider
updating (server 1.5.12 / client 1.5.4)
Downloading train.csv.zip to /content/drive/My Drive/CSE519/HW3
  0% 0.00/6.71M [00:00<?, ?B/s]
100% 6.71M/6.71M [00:00<00:00, 60.1MB/s]
Downloading store.csv to /content/drive/My Drive/CSE519/HW3
  0% 0.00/44.0k [00:00<?, ?B/s]
100% 44.0k/44.0k [00:00<00:00, 2.89MB/s]
Downloading sample_submission.csv to /content/drive/My Drive/CSE519/HW3
  0% 0.00/310k [00:00<?, ?B/s]
100% 310k/310k [00:00<00:00, 42.1MB/s]
Downloading test.csv.zip to /content/drive/My Drive/CSE519/HW3
  0% 0.00/192k [00:00<?, ?B/s]
100% 192k/192k [00:00<00:00, 12.7MB/s]
```

[ ]:
```
%ls
```

```
cse519_hw3_KaranDipesh_Gada_113082700.ipynb  store.csv
kaggle.json                                  test.csv.zip
sample_submission.csv                        train.csv.zip
```

0.3 Section 1: Library and Data Imports (Q1)

- Import your libraries and read the data into a dataframe. Print the head of the dataframe.

[10]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
from scipy.stats import ttest_ind
import seaborn as sns
import sklearn
from sklearn import preprocessing
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,␣
 ↪ExtraTreesRegressor, GradientBoostingRegressor
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression, SGDRegressor, Ridge
```

```
from sklearn.metrics import mean_squared_log_error
from sklearn.model_selection import train_test_split, permutation_test_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor, ExtraTreeRegressor
import sys
from xgboost import XGBRegressor
```

[11]:
```
#fill in dtypes dict for efficient memory utilization
dtypes = {
    'Id':'Int32',
    'Store':'Int32',
    'DayOfWeek':'Int32',
    'Sales':'Float64',
    'Customers':'Int64',
    'Open':'Int32',
    'StateHoliday':'category',
    'SchoolHoliday':'Int32',
    'StoreType':'category',
    'Assortment':'category',
    'CompetitionDistance':'Float64',
    'CompetitionOpenSinceMonth':'object',
    'CompetitionOpenSinceYear':'object',
    'Promo':'Int32',
    'Promo2':'Int32',
    'Promo2SinceWeek':'object',
    'Promo2SinceYear':'object',
    'PromoInterval':'object',

}

parse_date = ['Date']
```

[12]:
```
trainData = pd.read_csv('train.csv.zip', dtype=dtypes, parse_dates=parse_date)
trainData.sort_values('Date', inplace=True)
```

[13]:
```
storeData = pd.read_csv('store.csv',dtype=dtypes)
```

[14]:
```
mergedData = trainData.merge(storeData, how='left', on='Store')
```

[8]:
```
mergedData.head()
```

[8]:

	Store	DayOfWeek	...	Promo2SinceYear	PromoInterval
0	1115	2	...	2012	Mar,Jun,Sept,Dec
1	379	2	...	NaN	NaN
2	378	2	...	NaN	NaN
3	377	2	...	2010	Feb,May,Aug,Nov
4	376	2	...	NaN	NaN

```
[5 rows x 18 columns]
```

0.4 Section 2: Effect of Holidays (Q2)

As seen below only 20 of the total Stores are Open during Christmas; only 32 of the total Stores are Open during Easter and only 156 of the total Stores are Open during other state holidays.

[9]:
```
print('Stores that Open during StateHolidays are %d out of Total Stores %d'
      % (trainData.loc[(trainData['StateHoliday']=='c') & (trainData['Open'] ==␣
 ↪1)]['Store'].unique().shape[0], trainData['Store'].unique().shape[0]))
```

```
Stores that Open during StateHolidays are 20 out of Total Stores 1115
```

[10]:
```
print('Stores that Open during StateHolidays are %d out of Total Stores %d'
      % (trainData.loc[(trainData['StateHoliday']=='b') & (trainData['Open'] ==␣
 ↪1)]['Store'].unique().shape[0], trainData['Store'].unique().shape[0]))
```

```
Stores that Open during StateHolidays are 32 out of Total Stores 1115
```

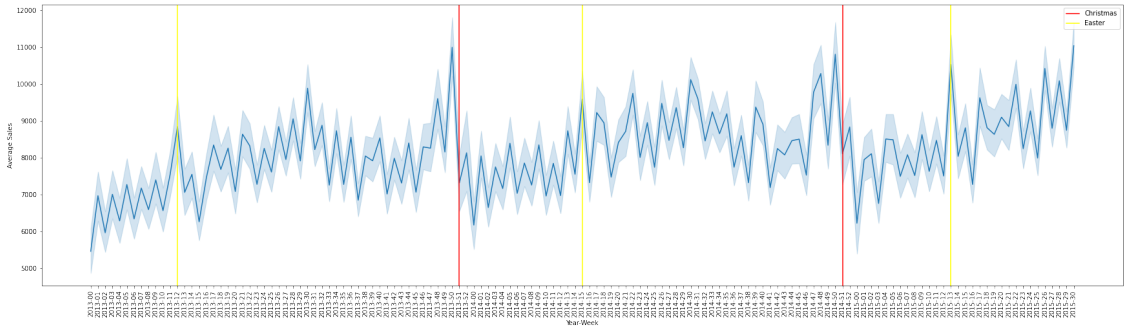
[11]:
```
print('Stores that Open during StateHolidays are %d out of Total Stores %d'
      % (trainData.loc[(trainData['StateHoliday']=='a') & (trainData['Open'] ==␣
 ↪1)]['Store'].unique().shape[0], trainData['Store'].unique().shape[0]))
```

```
Stores that Open during StateHolidays are 156 out of Total Stores 1115
```

Since the nature of `StateHoliday=a` is not known, its not possible to understand will it effect the Sales. So, I consider only `b` and `c` StateHoliday i.e. Christmas and Easter. To consider the impact on Sales here, I consider the Stores that are Open during either of the StateHoliday (Christmas or Easter). I have plotted the Average Sales per Week over time and made vertical lines to indicate the StateHoliday. It is observed that for Easter the Sales are higher during the Holiday rather than before Holiday. For Christmas the peak before the Christmas indicator suggests that people shop more before the Christmas than during Christmas hoiliday.

[12]:
```
q2Stores = trainData.loc[((trainData['StateHoliday']=='c') |␣
 ↪(trainData['StateHoliday']=='b')) & (trainData['Open'] == 1)]['Store'].
 ↪unique()
q2Data = trainData[trainData['Store'].isin(q2Stores)].copy()
q2Data['Year-Week'] = q2Data['Date'].dt.strftime('%Y-%W')
plt.figure(figsize=(30,8))
_ = sns.lineplot(data=q2Data, x='Year-Week', y='Sales')
christmas_year = set()
easter_year = set()
for value in q2Data[q2Data['StateHoliday'] == 'c']['Year-Week'].unique():
  if not value.split('-')[0] in christmas_year:
    christmas_year.add(value.split('-')[0])
    l1 = plt.axvline(value, color='red', label='Christmas')
for value in q2Data[q2Data['StateHoliday'] == 'b']['Year-Week'].unique():
```

```
  if not value.split('-')[0] in easter_year:
    easter_year.add(value.split('-')[0])
    l2 = plt.axvline(value, color='yellow', label='Easter')
_, _ = plt.xticks(rotation='vertical')
_ = plt.ylabel('Average Sales')
_ = plt.legend((l1,l2), ('Christmas', 'Easter'))
```



0.5 Section 3: Most and Least selling stores (Q3a & Q3b)

0.5.1 List For Highest and Lowest Cumulative Sales

The following code finds stores with atleast 6 months of data and finds the stores with 5 highest and 5 lowest total sales. Highest Sales Stores (from highest to lowest) - [262, 817, 562, 1114, 251] Lowest Sales Stores (from highest to lowest) - [263, 208, 198, 543, 307]

[13]:
```
def store_sales_6monthsplus(df_):
  # df_.sort_values('Date', inplace=True)
  if (df_.iloc[-1]['Date'] - df_.iloc[0]['Date'] > pd.to_timedelta(180,␣
 ↪unit='D')):
    return np.sum(df_['Sales'])
  else: return
```

[14]:
```
q3Data = trainData.groupby('Store').apply(store_sales_6monthsplus).
 ↪sort_values(ascending=False)
```

[15]:
```
highest_sales = (q3Data.index[:5].to_list())
lowest_sales = (q3Data.index[-5:].to_list())
print(highest_sales)
print(lowest_sales)
```

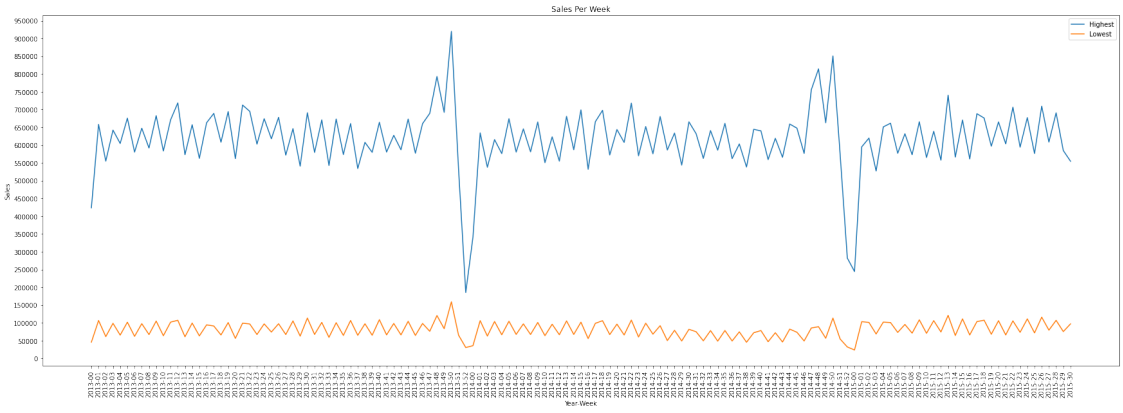
```
[262, 817, 562, 1114, 251]
[263, 208, 198, 543, 307]
```

0.5.2 Sales per Week for both Sets of Stores

Here I have taken total sales per week of highest set of stores and total sales per week of lowest set of stores over the time and plotted them on a single plot.

[16]:
```
def total_sales_per_week(df_, group_store=False):
  # df_.sort_values('Date', inplace=True)
  df_['Year-Week'] = df_['Date'].dt.strftime('%Y-%W')
  group_by_ = ['Year-Week']
  if group_store: group_by_ = ['Store'] + group_by_
  return df_.groupby(group_by_)['Sales'].agg('sum')
```

[17]:
```
q3ADataHighest = total_sales_per_week(trainData.loc[trainData['Store'].
 ↪isin(highest_sales)].copy())
q3ADataLowest = total_sales_per_week(trainData.loc[trainData['Store'].
 ↪isin(lowest_sales)].copy())
q3AData = pd.concat([q3ADataHighest.rename('Highest'), q3ADataLowest.
 ↪rename('Lowest')], axis=1)
q3AData.plot(figsize=(30,10), title='Sales Per Week', ylabel='Sales')
_,_ = plt.xticks(range(0,len(q3ADataHighest.index.to_list())),q3ADataHighest.
 ↪index.to_list(), rotation='vertical')
plt.locator_params(axis="y", nbins=20)
```

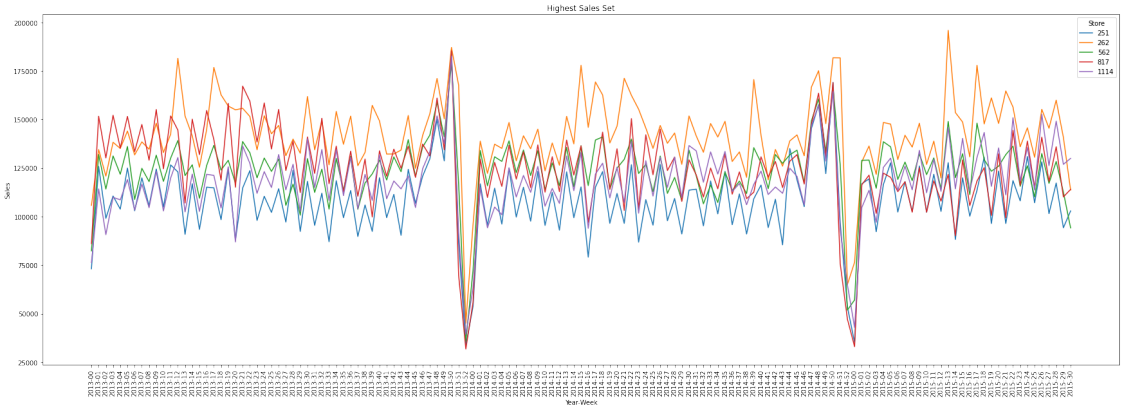


0.5.3 Pattern of Sales among both Sets of Stores

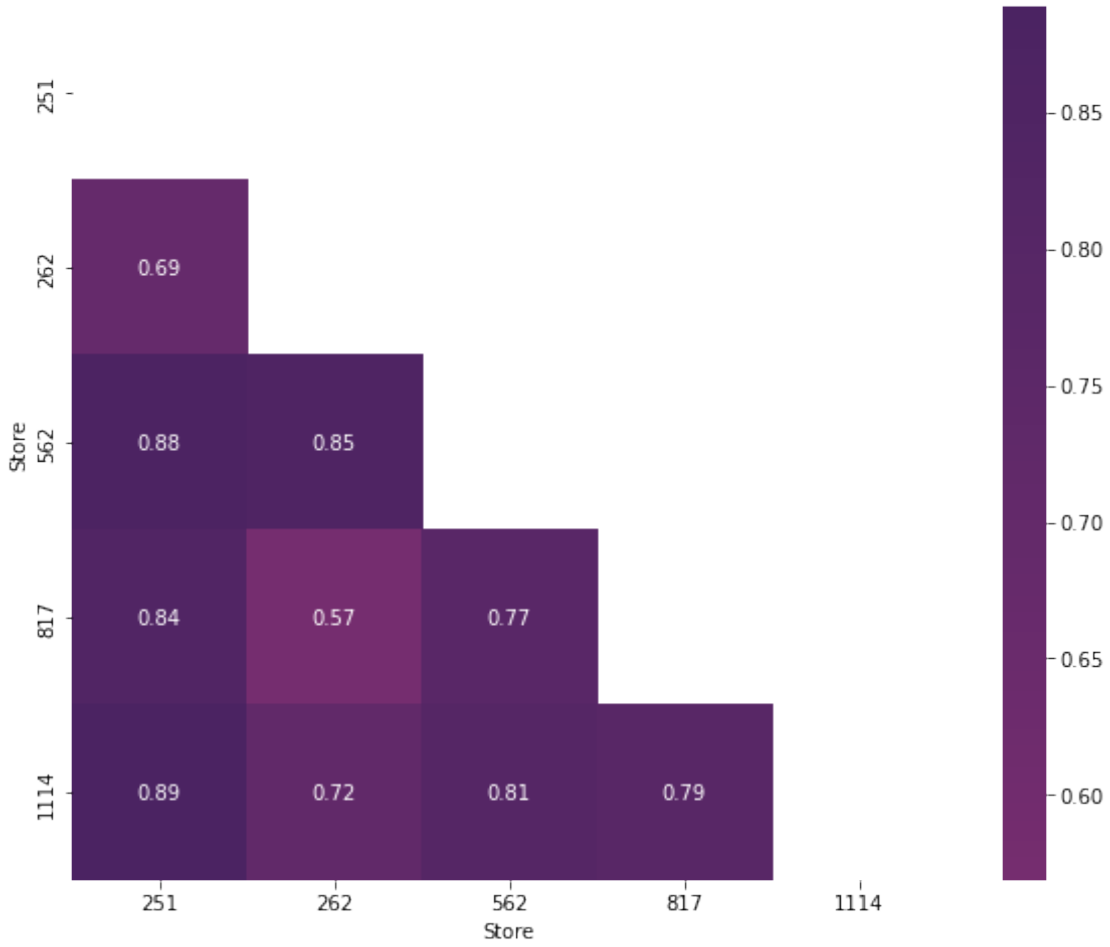
I have plotted the sales per week for each store in both the Set of Stores. For Highest Set, the pattern for each store are quite similar with a certain deviation for Store 262. This can be confirmed with the lower heatmap values that depicts the correlation of Sales Per Week of Store 262 against rest of the stores in the Highest Set. For Lowest Set, the pattern for each store are quite similar aswell with a certain deviation for Store 263. This can be confirmed with the lower heatmap values that depicts the correlation of Sales Per Week of Store 263 against rest of the stores in the Lowest Set.

[18]:
```
q3BDataHighest = total_sales_per_week(trainData.loc[trainData['Store'].
 ↪isin(highest_sales)].copy(), True).reset_index()
q3BDataHighest = q3BDataHighest.pivot(index=['Year-Week'], columns=['Store'],␣
 ↪values=['Sales'])
q3BDataHighest.columns = q3BDataHighest.columns.droplevel()
q3BDataHighest.plot(figsize=(30,10), title='Highest Sales Set', ylabel='Sales')
```

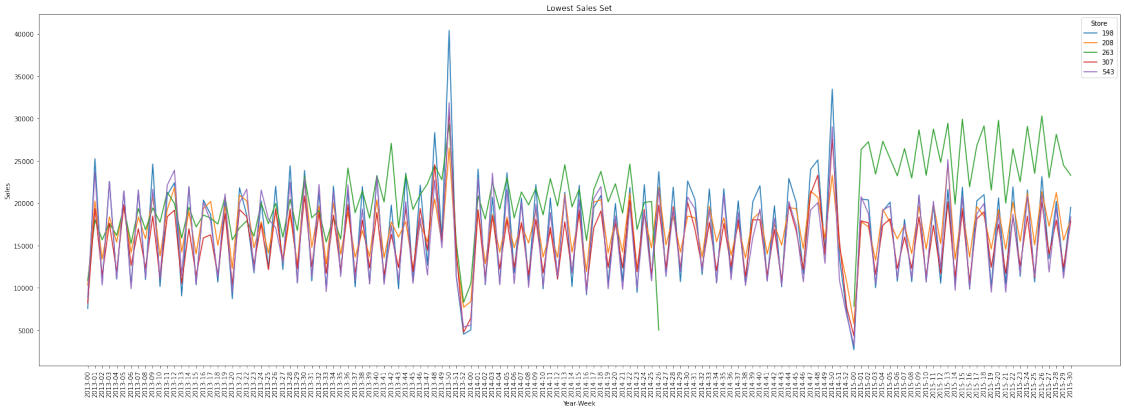
```
_,_ = plt.xticks(range(0,len(q3BDataHighest.index.to_list())),q3BDataHighest.
↪index.to_list(), rotation='vertical')
```



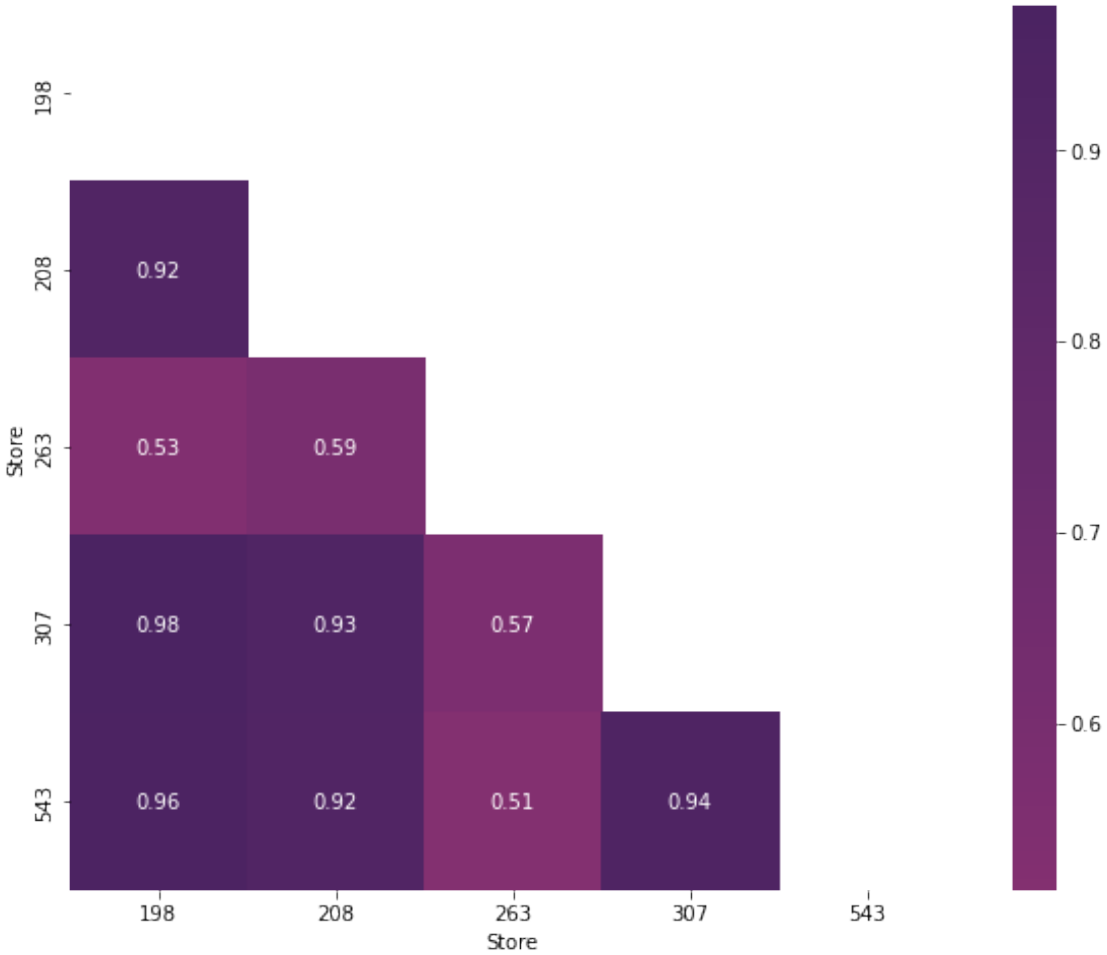
[19]:
```
q3BDataHighestCorr = q3BDataHighest.corr()
mask = np.triu(np.ones_like(q3BDataHighestCorr, dtype=bool))
plt.subplots(figsize=(10,8))
_ = sns.heatmap(q3BDataHighestCorr, mask=mask, cmap='flare', center=0,
            square=True, annot=True)
```



[20]:
```
q3BDataLowest = total_sales_per_week(trainData.loc[trainData['Store'].
 ↪isin(lowest_sales)].copy(), True).reset_index()
q3BDataLowest = q3BDataLowest.pivot(index=['Year-Week'], columns=['Store'],␣
 ↪values=['Sales'])
q3BDataLowest.columns = q3BDataLowest.columns.droplevel()
q3BDataLowest.plot(figsize=(30,10), title='Lowest Sales Set', ylabel='Sales')
_,_ = plt.xticks(range(0,len(q3BDataLowest.index.to_list())),q3BDataLowest.
 ↪index.to_list(), rotation='vertical')
```



[21]:
```
q3BDataLowestCorr = q3BDataLowest.corr()
mask = np.triu(np.ones_like(q3BDataLowestCorr, dtype=bool))
plt.subplots(figsize=(10,8))
_ = sns.heatmap(q3BDataLowestCorr, mask=mask, cmap='flare', center=0,
            square=True, annot=True)
```



0.6 Section 4: Closest Competitor: Distance and Age (Q4a & Q4b)

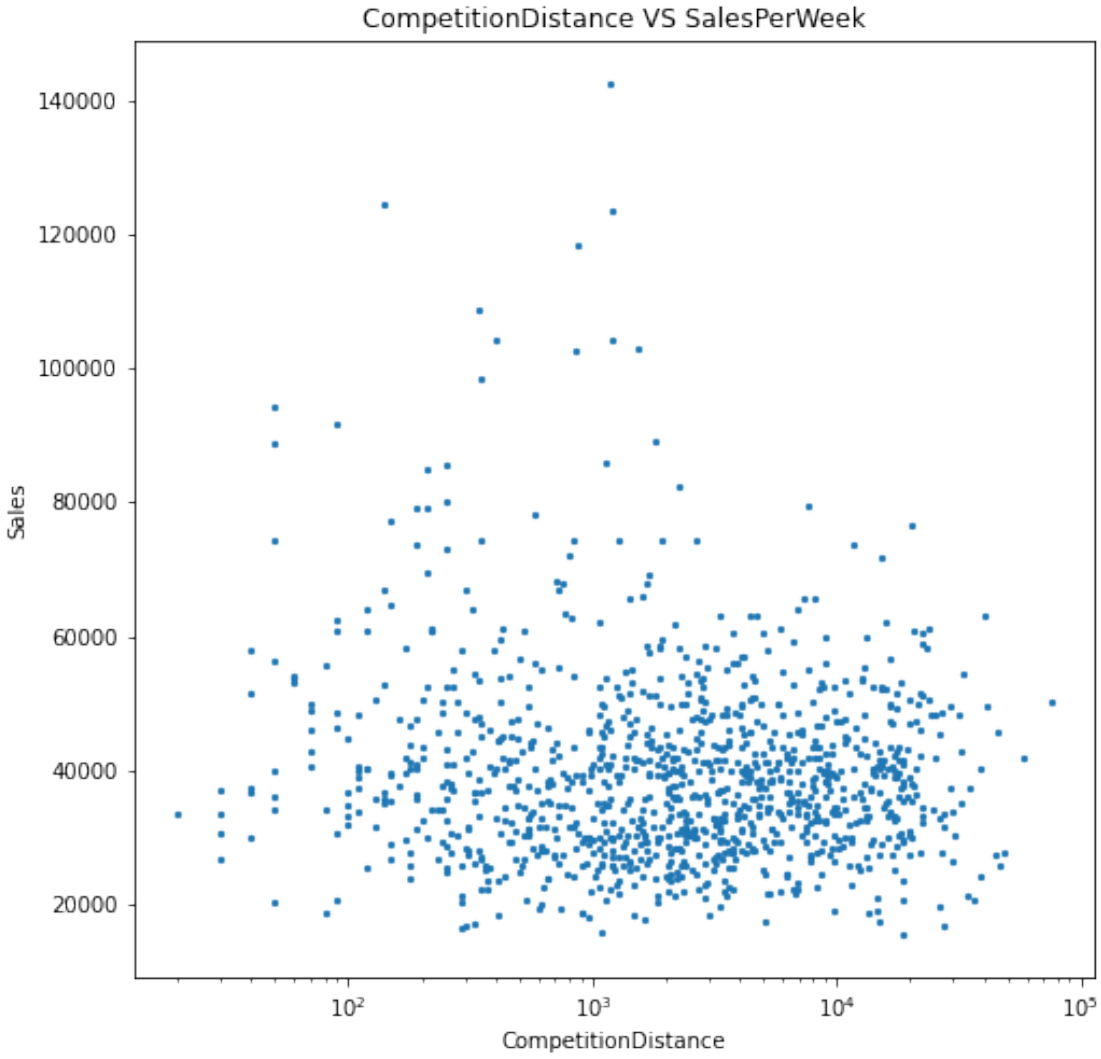
0.6.1 Sales per Week VS Competitor

There were 3 stores that didn't have competitor distance and hence were removed from the data in cleaning stage as it's very difficult to predict if they didn't have competition or was the data missing. I have plotted Average `Sales per Week` for every store vs the `Distance` of it from its Competitor. As the plot suggests the Distribution is quite scattered. One would expect the `Sales` should increase as the `Distance` decrease. However, this is not the case. This can be confirmed with the correlation of `Salesvs CompetitorDistance` which turns out to be very insignificant.

[22]:
```
q4Data = pd.DataFrame(total_sales_per_week(trainData, True).groupby('Store').
↪agg('mean'))
q4Data = storeData.join(q4Data, on='Store', how='inner')
q4Data.set_index('Store', inplace=True)
```

[23]:
```
q4AData = q4Data[['CompetitionDistance', 'Sales']].dropna(how='any', axis=0)
```

[24]:
```
_ = q4AData.sort_values('CompetitionDistance').plot.
↪scatter('CompetitionDistance', 'Sales', figsize=(8,8),␣
↪ylabel='SalesPerWeek', title='CompetitionDistance VS SalesPerWeek',␣
↪logx=True, s=5)
```



[25]:
```
q4ADataa.corr()
```

[25]:
```
                      CompetitionDistance     Sales
CompetitionDistance              1.000000 -0.033615
Sales                           -0.033615  1.000000
```

0.7 Section 5: Pearson Correlation of Features (Q5)

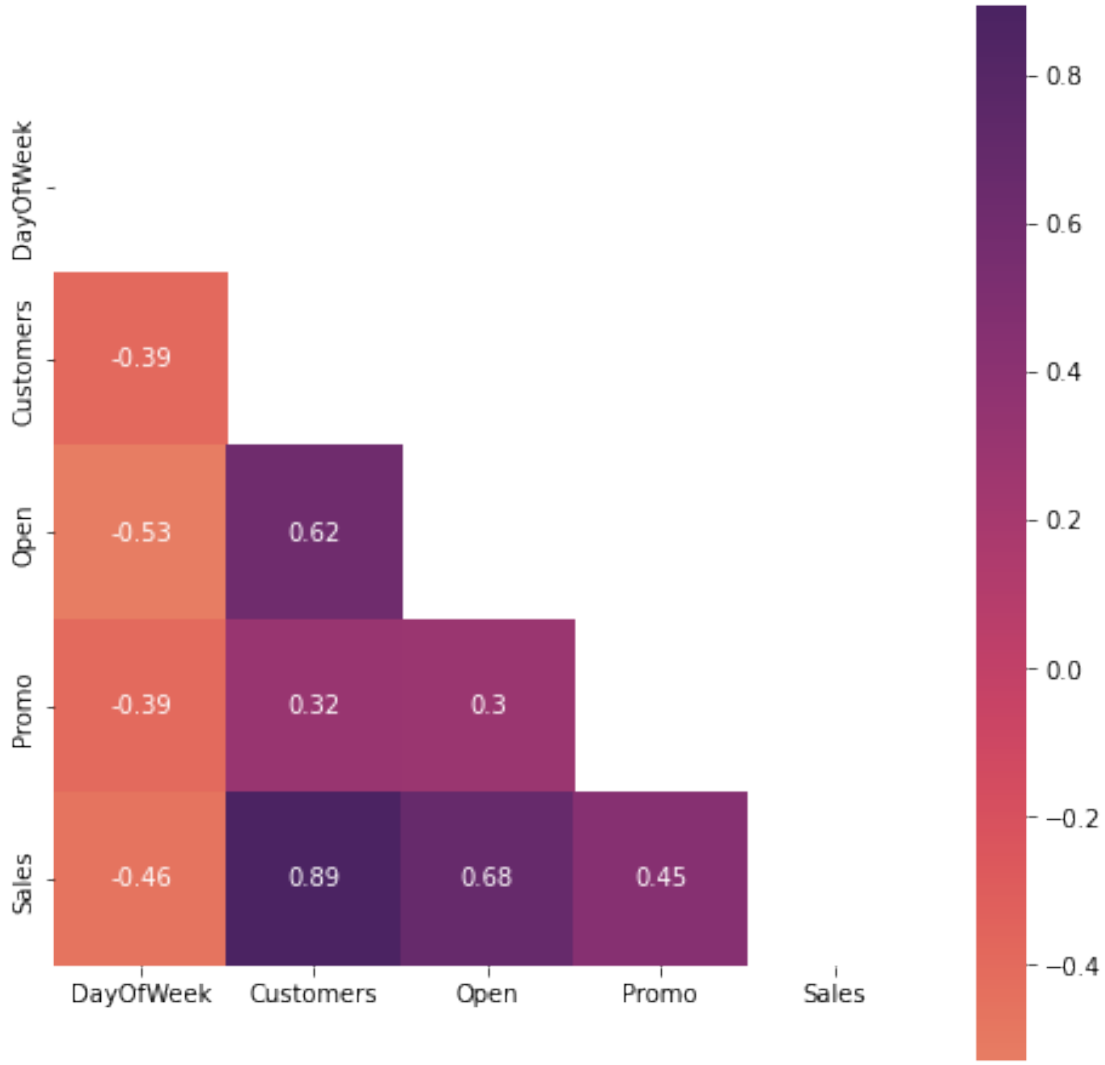
0.7.1 Pearson's Correlation

List of 5 interesting features - `'DayOfWeek'`, `'Customers'`, `'Open'`, `'Promo'`, `'Sales'` Customer-Sales have the best positive correlation. ###Spearman's Correlation The heatmap and the correlation within the features don't change significantly with Spearman's correlation.

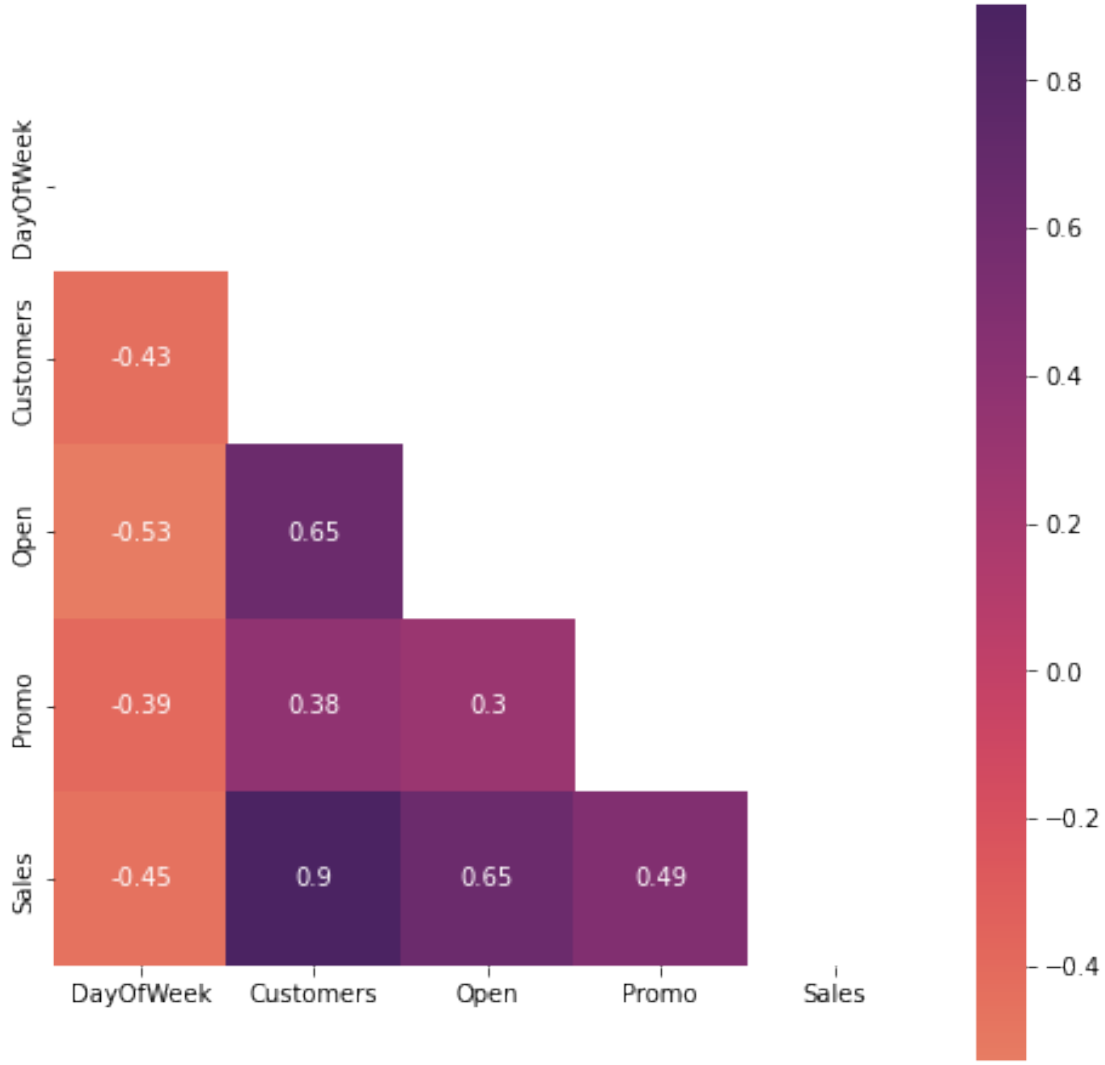
[26]:
```
correlation_features = ['DayOfWeek', 'Customers', 'Open', 'Promo', 'Sales']
```

[27]:
```
q5Data = mergedData[correlation_features]
```

[28]:
```
q5Pearson = q5Data.corr()
mask = np.triu(np.ones_like(q5Pearson, dtype=bool))
cmap = sns.color_palette('flare', as_cmap=True)
plt.subplots(figsize=(8,8))
_ = sns.heatmap(q5Pearson, mask=mask, cmap=cmap, center=0, square=True,␣
↪annot=True)
```



[29]:
```
q5Pearson = q5Data.corr('spearman')
mask = np.triu(np.ones_like(q5Pearson, dtype=bool))
cmap = sns.color_palette('flare', as_cmap=True)
plt.subplots(figsize=(8,8))
_ = sns.heatmap(q5Pearson, mask=mask, cmap=cmap, center=0, square=True,␣
↪annot=True)
```



0.8 Section 6: Permutation Testing (Q6)

Good Variable - `Customers` Meaningless - `CompetitionDistance` Random - `DayOfWeek` The pvalue suggests - 1. Prediction using `Customers` are better than chance with probability of `0.99` 1. Prediction using `CompetitionDistance` are better than chance with probability of `0` 1. Prediction using `DayOfWeek` are better than chance with probability of `0.99`

[30]:
```
q6X = mergedData[['CompetitionDistance']].fillna(0)
q6Y = mergedData['Sales']
true_rmsle, shuffle_rmsle_list, pvalue =␣
 ↪permutation_test_score(LinearRegression(), q6X, q6Y,␣
 ↪scoring='neg_mean_squared_log_error', n_permutations=100)
print('The p-value which approximates the probability that the score would be␣
 ↪obtained by chance %.5f' % pvalue)
```

```
The p-value which approximates the probability that the score would be obtained
by chance 1.00000
```

[31]:
```
q6X = mergedData[['Customers']]
q6Y = mergedData['Sales']
true_rmsle, shuffle_rmsle_list, pvalue =␣
 ↪permutation_test_score(LinearRegression(), q6X, q6Y,␣
 ↪scoring='neg_mean_squared_log_error', n_permutations=100)
print('The p-value which approximates the probability that the score would be␣
 ↪obtained by chance %.5f' % pvalue)
```

```
The p-value which approximates the probability that the score would be obtained
by chance 0.00990
```

[32]:
```
q6X = mergedData[['DayOfWeek']]
q6Y = mergedData['Sales']
true_rmsle, shuffle_rmsle_list, pvalue =␣
 ↪permutation_test_score(LinearRegression(), q6X, q6Y,␣
 ↪scoring='neg_mean_squared_log_error', n_permutations=100)
print('The p-value which approximates the probability that the score would be␣
 ↪obtained by chance %.5f' % pvalue)
```

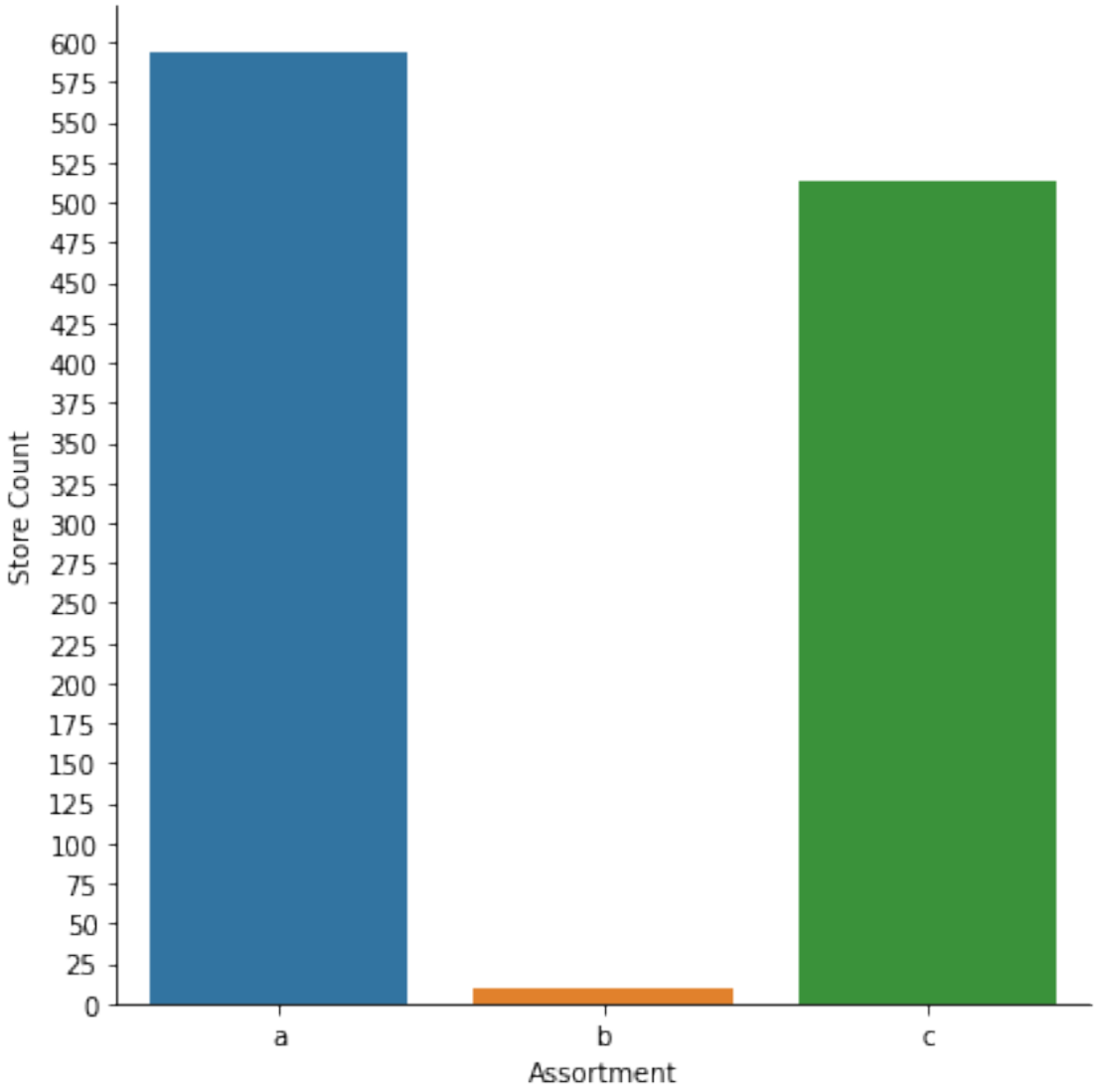
```
The p-value which approximates the probability that the score would be obtained
by chance 0.00990
```

0.9 Section 7: Interesting findings (Q7)

0.9.1 Plot 1

This is a bar chart for count of Stores for different `Assortment`. We realize very less Stores follow `type b` Assortment. It arouses curiosity why are the Stores not adopting `type b` Assortment?

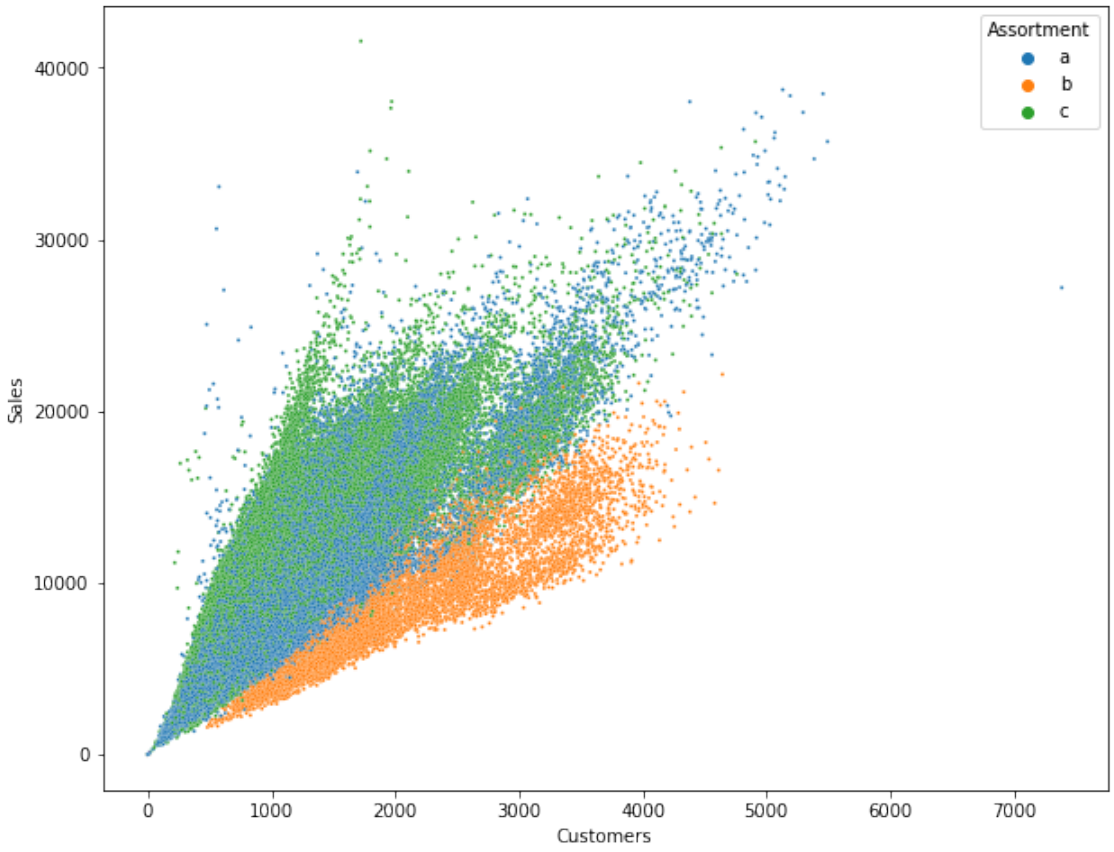
[33]:
```
_ = sns.catplot(data=storeData, x='Assortment', kind='count', height=6,␣
 ↪aspect=1)
_ = plt.ylabel('Store Count')
_ = plt.locator_params(axis='y', nbins=30)
```



0.9.2 Plot 2

To answer the question arisen by Plot 1, I plotted Scatter Plot for `Customers` vs `Sales` for all types of Assortment (differentiated by hue/color). Its observed that for `type b` Assortment Stores the Sales-to-Customers ratio is significantly lower than for `type a` and `type c`. This explain why most of the Stores avoid adopting `type b` Assortment.

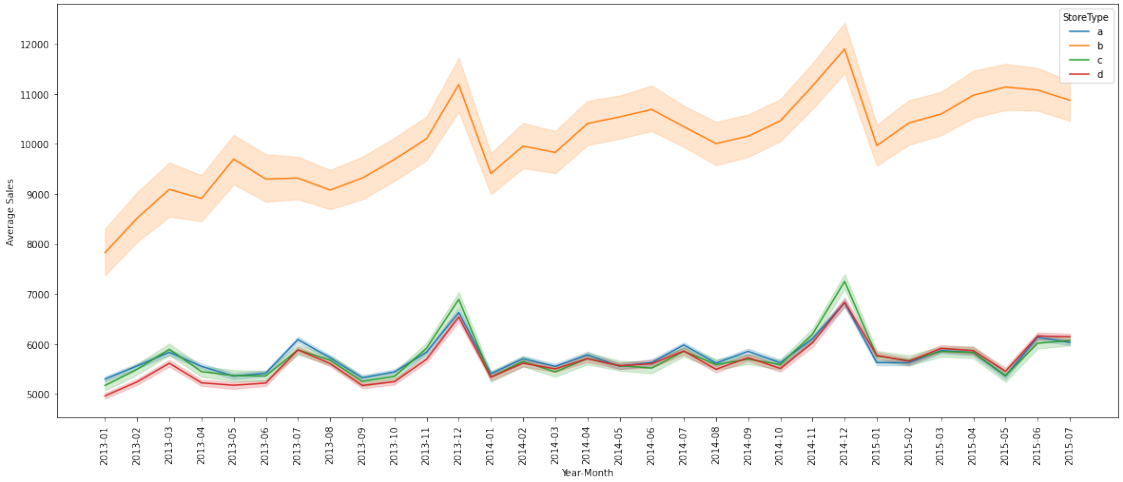
[34]:
```
plt.figure(figsize=(10,8))
_ = sns.scatterplot(data=mergedData, x='Customers', y='Sales', s=4,␣
↪hue='Assortment')
```



0.9.3 Plot 3

I plotted a Lineplot to understand the Average Sales per month over time of the Stores with different `StoreType`. The plot reveals that `type b` has a humongous Average Sales per month over time in comparison to other `StoreType`. So, it must be good to have a `StoreType` as `type b`. So how many of the Stores follow `type b` StoreType?

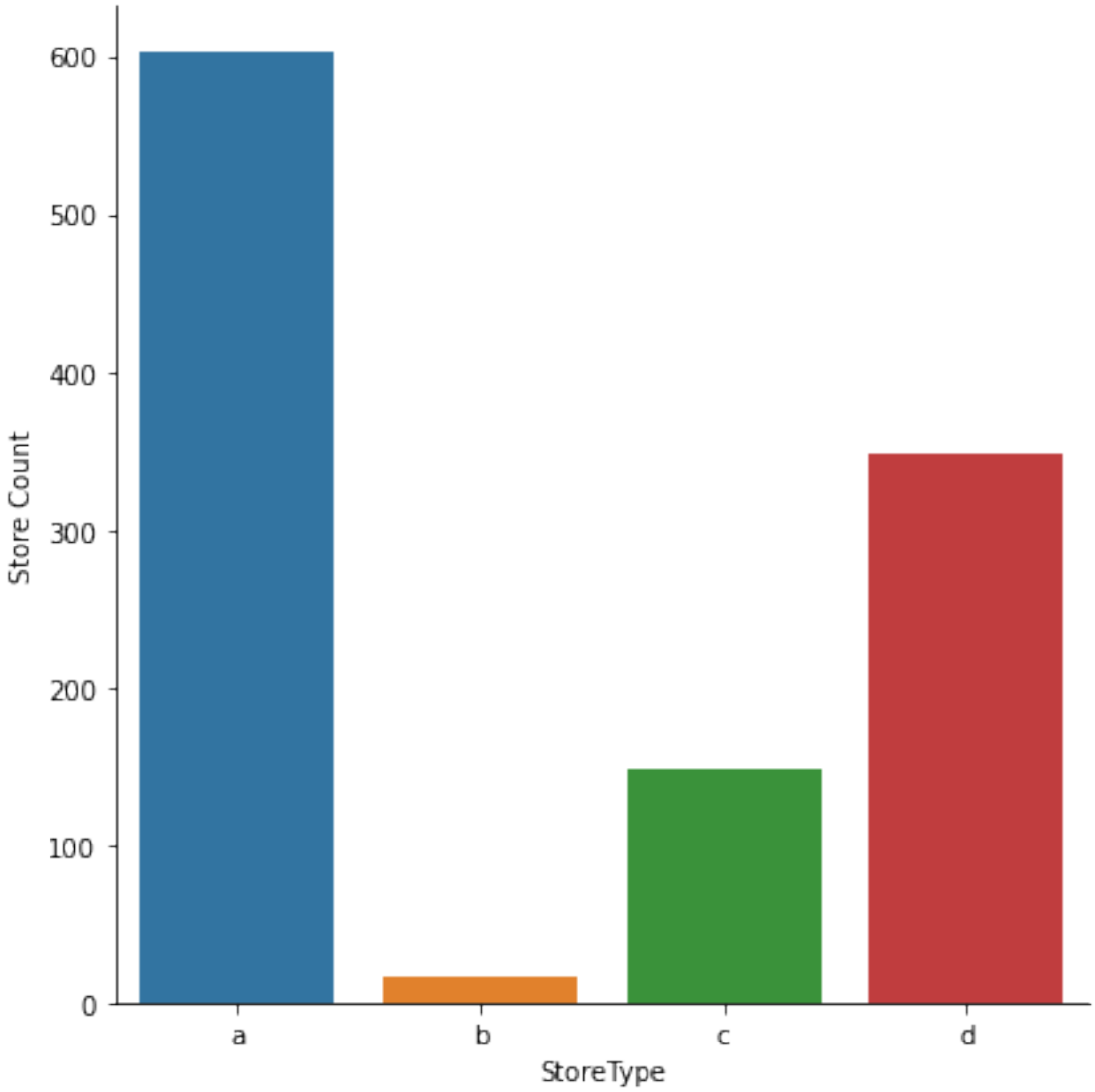
[35]:
```
q7E = mergedData.copy()
q7E['Year-Month'] = q7E['Date'].dt.strftime('%Y-%m')
plt.figure(figsize=(20,8))
_ = sns.lineplot(data=q7E, x='Year-Month', y='Sales', hue='StoreType')
_, _ = plt.xticks(rotation='vertical')
_ = plt.ylabel('Average Sales')
```



0.9.4 Plot 4

As seen in below Barplot, only a handful of the Stores actually have `type b` as the `StoreType`. Why is that even with good Average Sales per month, very few Stores follow `type b` StoreType? Is the above Lineplot skewed because of less data point (just lucky) or are there more reasons?

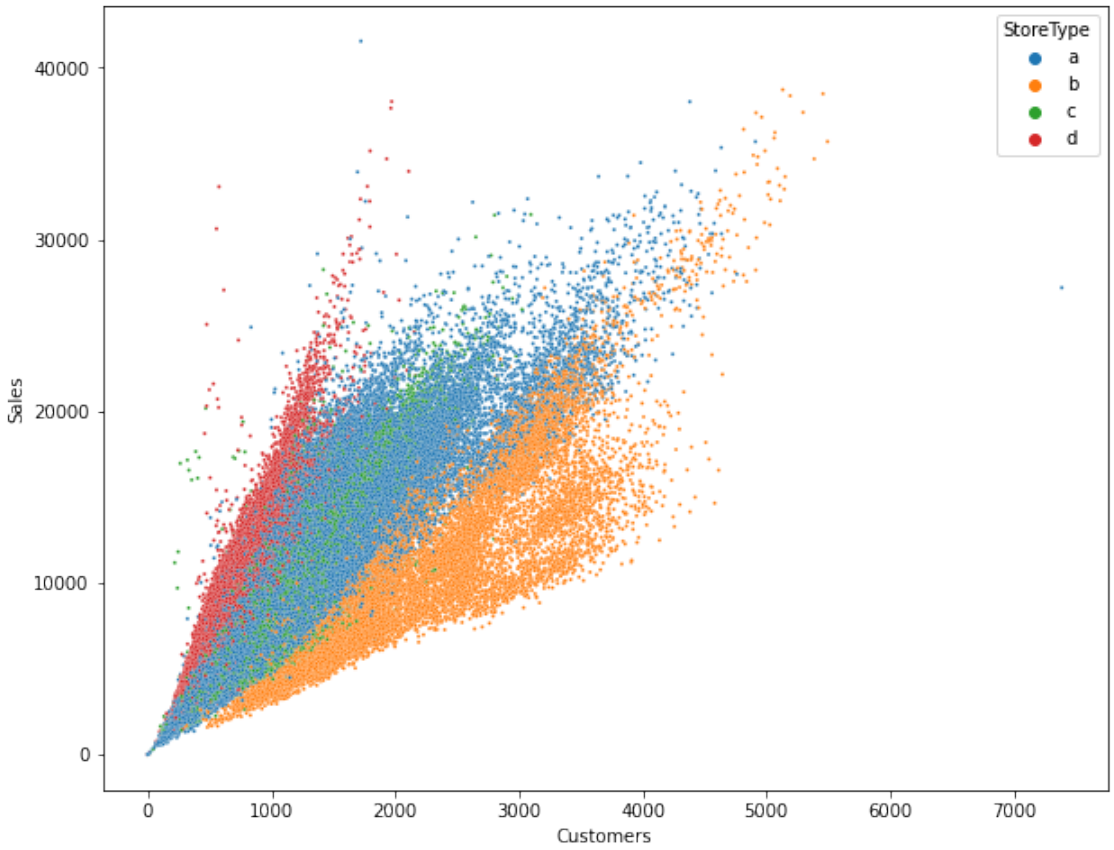
[36]:
```
_ = sns.catplot(data=storeData, x='StoreType', kind='count', height=6, aspect=1)
_ = plt.ylabel('Store Count')
```



0.9.5 Plot 5

The below ScatterPlot for `Customers` vs `Sales` reveals that although the Sales are high for `type b` StoreType the Sales-to-Customers ratio is less than `type a`, `type b` and `type c` StoreType. This would make stakeholders inclined towards type `a`, `b`, `c` StoreType.

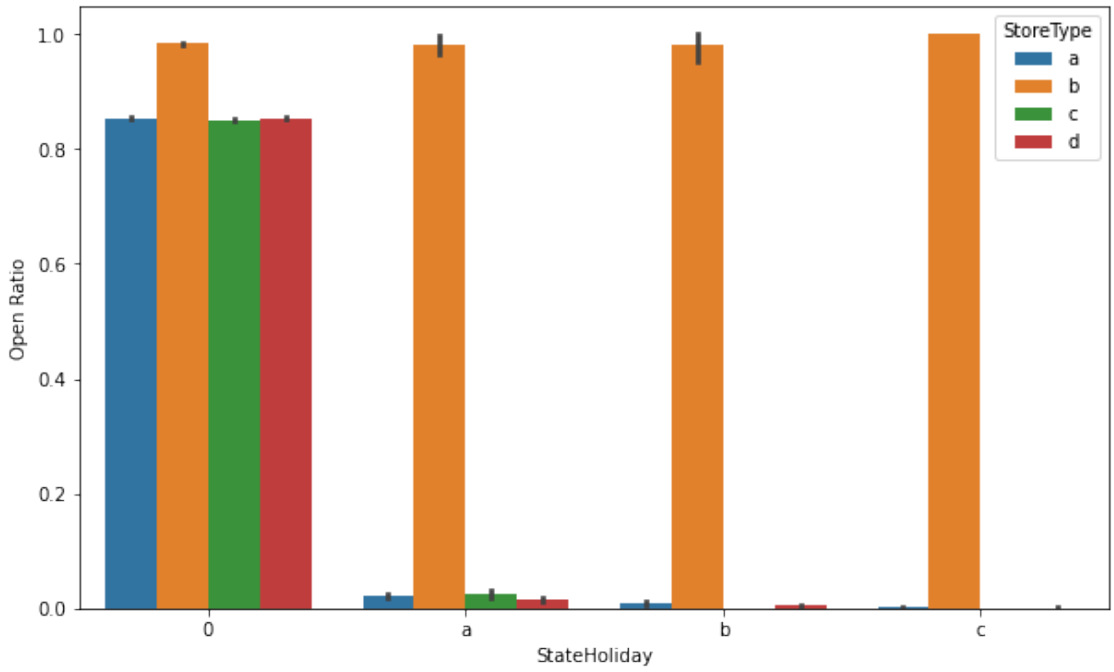
[37]:
```
plt.figure(figsize=(10,8))
_ = sns.scatterplot(data=mergedData, x='Customers', y='Sales', s=4,␣
↪hue='StoreType')
```



0.9.6 Plot 6

Additionally the below Barplot also reveals that `type b` StoreType is most probable to stay Open throughout the `StateHolidays` making it very unlikely to open up more stores with this StoreType. This can be due to logistical constraints.

[38]:
```
plt.figure(figsize=(10,6))
_ = sns.barplot(data=mergedData, x='StateHoliday', y='Open', hue='StoreType')
_ = plt.ylabel('Open Ratio')
```



0.10 Section 8: Train Test Split and Modelling (Q8)

0.10.1 Model 1 - Decision Tree

`splitter` is the strategy used to split at each node. I have used `best` and `random` as values in different models. `criterion` is the measure of the quality of a split. I have used `mse` which is mean squred error. `max_depth` is the maximum level (depth) of the DecisionTree. If unspecified it expands till all leaves are pure. If a value is provided it expands tree only till the given depth. 1. HyperParameter 1 - `splitter=random`, `criterion=mse` 2. HyperParameter 2 - `splitter=random`, `criterion=mse`, `max_depth=25` 3. HyperParameter 3 - `splitter=best`, `criterion=mse`, `max_depth=100`

0.10.2 Model 2 - RandomForest

RandomForest is an ensemble models that predicts using the prediction of many weak models (DecisionTree). `n_estimators` specifies the number of weak models to use. I have used `5`, `10`, `200` as the estimators. `bootstrap` is the parameter to specify bootstrapping in ensemble. In bootstrapping, samples are drawn with replacement to train each estimator, at times same samples are used multiple times for the same estimator. In this way although each estimator might have high variance wrt particular set of training data byt predictions are made by averaging the predictions of all estimators called as Bagging. If False, all train data is used for each estimator. 1. HyperParameter 1 - `n_estimators=5`, `bootstrap=False` 2. HyperParameter 2 - `n_estimators=10`, `bootstrap=False` 3. HyperParameter 3 - `n_estimators=200`, `bootstrap=True`

[15]:
```
q7X = mergedData.copy()
q7X['Month'] = q7X['Date'].dt.month
```

[16]:
```
def RMSPE(YTrue, YPred):
  return np.sqrt(np.mean(np.square((YTrue[~(YTrue==0)] - YPred[~(YTrue==0)])/
 ↪(YTrue[~(YTrue==0)]))))
```

[17]:
```
ordinal_feat = ['DayOfWeek', 'Assortment', 'Month']
nominal_feat = ['StateHoliday', 'StoreType']
passthrough_feat = ['Store', 'Open', 'Promo', 'SchoolHoliday']
target = 'Sales'
```

[18]:
```
q7X = q7X[['Date'] + ordinal_feat + nominal_feat + passthrough_feat + [target]]
```

[19]:
```
split_sdate = pd.to_datetime('2015-05-01', format='%Y-%m-%d')
split_edate = pd.to_datetime('2015-07-31', format='%Y-%m-%d')
q7XTrain, q7XTest = q7X[(q7X['Date']<split_sdate) | (q7X['Date']>split_edate)],␣
 ↪q7X[(q7X['Date']>=split_sdate) & (q7X['Date']<=split_edate)]
q7YTrain = q7XTrain.pop('Sales')
q7YTest = q7XTest.pop('Sales')
```

[20]:
```
column_transformer = ColumnTransformer(transformers=[
                                                      ('ordinal_data',␣
 ↪OrdinalEncoder(), ordinal_feat),
                                                      ('nominal_data',␣
 ↪OneHotEncoder(), nominal_feat),
                                                     ␣
 ↪('leave_as_it_is','passthrough', passthrough_feat),
])
```

The time taken for every model and the performance (RMSPE score) is printed for both models and all hyperparameterized variation below. To avoid prediction to be negative I am using `log(Sales)` as my target variable. After prediction I convert the predicted values using `np.exp` to get the actual Sales values.

[21]:
```
model1_h1 = Pipeline(steps=[
                            ('column_transform', column_transformer),
                            ('regressor',␣
 ↪DecisionTreeRegressor(splitter='random',criterion='mse')),
], verbose=True)

model1_h1.fit(q7XTrain, np.log(1+q7YTrain))
model1_h1_YPred = np.exp(model1_h1.predict(q7XTest))
print(RMSPE(q7YTest, model1_h1_YPred))
```

```
[Pipeline] .. (step 1 of 2) Processing column_transform, total=   1.6s
[Pipeline] ... (step 2 of 2) Processing regressor, total=   2.0s
0.15789730593147508
```

[22]:
```
model1_h2 = Pipeline(steps=[
                            ('column_transform', column_transformer),
                            ('regressor',␣
 ↪DecisionTreeRegressor(splitter='random',criterion='mse', max_depth=25)),
```

```
], verbose=True)

model1_h2.fit(q7XTrain, np.log(1+q7YTrain))
model1_h2_YPred = np.exp(model1_h2.predict(q7XTest))
print(RMSPE(q7YTest, model1_h2_YPred))
```

```
[Pipeline] .. (step 1 of 2) Processing column_transform, total=   1.5s
[Pipeline] ... (step 2 of 2) Processing regressor, total=   2.0s
0.20924852955314885
```

[23]:
```
model1_h3 = Pipeline(steps=[
                            ('column_transform', column_transformer),
                            ('regressor',␣
 ↪DecisionTreeRegressor(splitter='best',criterion='mse', max_depth=100)),
], verbose=True)

model1_h3.fit(q7XTrain, np.log(1+q7YTrain))
model1_h3_YPred = np.exp(model1_h3.predict(q7XTest))
print(RMSPE(q7YTest, model1_h3_YPred))
```

```
[Pipeline] .. (step 1 of 2) Processing column_transform, total=   1.4s
[Pipeline] ... (step 2 of 2) Processing regressor, total=   4.0s
0.15437086352497184
```

[24]:
```
model2_h1 = Pipeline(steps=[
                            ('column_transform', column_transformer),
                            ('regressor', RandomForestRegressor(n_estimators=5,␣
 ↪bootstrap=False))
], verbose=True)
model2_h1.fit(q7XTrain, np.log(1+q7YTrain))
model2_h1_YPred = np.exp(model2_h1.predict(q7XTest))
print(RMSPE(q7YTest, model2_h1_YPred))
```

```
[Pipeline] .. (step 1 of 2) Processing column_transform, total=   1.4s
[Pipeline] ... (step 2 of 2) Processing regressor, total=  19.4s
0.1543699479352234
```

[25]:
```
model2_h2 = Pipeline(steps=[
                            ('column_transform', column_transformer),
                            ('regressor',␣
 ↪RandomForestRegressor(n_estimators=10, bootstrap=False))
], verbose=True)
model2_h2.fit(q7XTrain, np.log(1+q7YTrain))
model2_h2_YPred = np.exp(model2_h2.predict(q7XTest))
print(RMSPE(q7YTest, model2_h2_YPred))
```

```
[Pipeline] .. (step 1 of 2) Processing column_transform, total=   1.5s
[Pipeline] ... (step 2 of 2) Processing regressor, total=  38.8s
0.15437115016609812
```

[26]:
```
model2_h3 = Pipeline(steps=[
                            ('column_transform', column_transformer),
                            ('regressor',␣
 ↪RandomForestRegressor(n_estimators=200, bootstrap=True))
], verbose=True)
model2_h3.fit(q7XTrain, np.log(1+q7YTrain))
model2_h3_YPred = np.exp(model2_h3.predict(q7XTest))
print(RMSPE(q7YTest, model2_h3_YPred))
```

```
[Pipeline] .. (step 1 of 2) Processing column_transform, total=   1.7s
[Pipeline] ... (step 2 of 2) Processing regressor, total= 8.5min
0.15137566728893614
```

0.11 Section 9: t-test (Q9)

I have taken the predictions from Model 1:HyperParameter 1 and Model 2:HyperParameter 3 to perform T-test. The T-statistic `-0.164` The p-value `0.86` suggests that the Null Hypothesis that both the predictions are samples of same distribution cannot be rejected. Hence the predictions are not different significantly.

[53]:
```
ttest_ind(model1_h1_YPred, model2_h3_YPred)
```

[53]:
```
Ttest_indResult(statistic=-0.1641275840481657, pvalue=0.8696308690108223)
```

0.12 Section 10: Screenshots (Q10)

[28]:
```
testData = pd.read_csv('test.csv.zip', dtype=dtypes, parse_dates=['Date'])
testData[testData['Open'].isnull()]
```

[28]:

	Id	Store	DayOfWeek	...	Promo	StateHoliday	SchoolHoliday
479	480	622	4	...	1	0	0
1335	1336	622	3	...	1	0	0
2191	2192	622	2	...	1	0	0
3047	3048	622	1	...	1	0	0
4759	4760	622	6	...	0	0	0
5615	5616	622	5	...	0	0	0
6471	6472	622	4	...	0	0	0
7327	7328	622	3	...	0	0	0
8183	8184	622	2	...	0	0	0
9039	9040	622	1	...	0	0	0
10751	10752	622	6	...	0	0	0

[11 rows x 8 columns]

[29]:
```
testData.loc[testData['Open'].isnull(), 'Open'] = 1
```

[30]:
```
testData = testData.merge(storeData, how='left', on='Store')
testData['Month'] = testData['Date'].dt.month
testData.head()
```

[30]:
```
   Id  Store  DayOfWeek  ... Promo2SinceYear    PromoInterval  Month
0   1      1          4  ...              NaN              NaN      9
1   2      3          4  ...             2011  Jan,Apr,Jul,Oct      9
2   3      7          4  ...              NaN              NaN      9
3   4      8          4  ...              NaN              NaN      9
4   5      9          4  ...              NaN              NaN      9

[5 rows x 18 columns]
```

[31]:
```
testData_transformed = testData[['Date'] + ordinal_feat + nominal_feat +␣
↪passthrough_feat]
```

With Model 2:Hyperparameter 3

[43]:
```
test_YPred = np.exp(model2_h3.predict(testData_transformed))
```

[44]:
```
test_YPred = pd.Series(test_YPred, index=testData['Id'], name='Sales').
↪reset_index()
test_YPred.head()
```

[44]:
```
   Id        Sales
0   1  4426.849557
1   2  7748.966359
2   3  8926.443493
3   4  6575.015780
4   5  5923.583692
```

[45]:
```
test_YPred.to_csv('model2_hyperparam3.csv', index=False)
```

With Model 1:Hyperparameter 3

[32]:
```
test_YPred = np.exp(model1_h3.predict(testData_transformed))
```

[33]:
```
test_YPred = pd.Series(test_YPred, index=testData['Id'], name='Sales').
↪reset_index()
test_YPred.head()
```

[33]:
```
   Id        Sales
0   1  4399.882624
1   2  7724.909284
2   3  8964.494214
3   4  6583.028626
4   5  5879.805506
```

[34]:
```
test_YPred.to_csv('model1_hyperparam3.csv', index=False)
```

Public Score & Highest Rank: 0.15342
Private Score & Highest Rank: 0.15616
Kaggle profile link: https://www.kaggle.com/karangada
Screenshot(s):

[ ]: