# Sanket_Goutam_cse519_hw3_111463594

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.

## 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.

### 0.3.1 Take a look at the training data. Combine the tables in store.csv and train.csv into a single dataframe. (5 points)

[287]:
```
import pandas as pd
import numpy as np
import random

# Display format for float values
pd.set_option('float_format', '{:f}'.format)

import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
from matplotlib.ticker import PercentFormatter
```

[337]:
```
store = 'store.csv'
train = 'train.csv'

train_data = pd.read_csv(train)
store_data = pd.read_csv(store)
```

```
/opt/conda/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3165:
DtypeWarning: Columns (7) have mixed types.Specify dtype option on import or set
low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

[107]:
```
store_data.head(5)
```

[107]:
```
   Store StoreType Assortment  CompetitionDistance  CompetitionOpenSinceMonth  \
0      1         c          a               1270.000000                  9.000000
1      2         a          a                570.000000                 11.000000
2      3         a          a              14130.000000                 12.000000
3      4         c          c                620.000000                  9.000000
4      5         a          a              29910.000000                  4.000000

   CompetitionOpenSinceYear  Promo2  Promo2SinceWeek  Promo2SinceYear  \
0               2008.000000       0              NaN              NaN
1               2007.000000       1        13.000000      2010.000000
2               2006.000000       1        14.000000      2011.000000
3               2009.000000       0              NaN              NaN
4               2015.000000       0              NaN              NaN

     PromoInterval
0              NaN
1  Jan,Apr,Jul,Oct
2  Jan,Apr,Jul,Oct
3              NaN
4              NaN
```

[108]:
```
train_data.head(5)
```

[108]:
```
   Store  DayOfWeek        Date  Sales  Customers  Open  Promo StateHoliday  \
0      1          5  2015-07-31   5263        555     1      1            0
1      2          5  2015-07-31   6064        625     1      1            0
2      3          5  2015-07-31   8314        821     1      1            0
3      4          5  2015-07-31  13995       1498     1      1            0
4      5          5  2015-07-31   4822        559     1      1            0

   SchoolHoliday
0              1
1              1
2              1
3              1
4              1
```

[338]:
```
combined_data = pd.merge(train_data, store_data, how="left", on="Store")
combined_data['Date'] = pd.to_datetime(combined_data['Date'])
combined_data.head(5)
```

[338]:
```
   Store  DayOfWeek       Date  Sales  Customers  Open  Promo StateHoliday  \
0      1          5 2015-07-31   5263        555     1      1            0
1      2          5 2015-07-31   6064        625     1      1            0
2      3          5 2015-07-31   8314        821     1      1            0
```

```
3      4           5 2015-07-31  13995        1498      1       1             0
4      5           5 2015-07-31   4822         559      1       1             0

   SchoolHoliday StoreType Assortment  CompetitionDistance  \
0              1         c          a          1270.000000
1              1         a          a           570.000000
2              1         a          a         14130.000000
3              1         c          c           620.000000
4              1         a          a         29910.000000

   CompetitionOpenSinceMonth  CompetitionOpenSinceYear  Promo2  \
0                   9.000000               2008.000000       0
1                  11.000000               2007.000000       1
2                  12.000000               2006.000000       1
3                   9.000000               2009.000000       0
4                   4.000000               2015.000000       0

   Promo2SinceWeek  Promo2SinceYear    PromoInterval
0              NaN              NaN              NaN
1        13.000000      2010.000000  Jan,Apr,Jul,Oct
2        14.000000      2011.000000  Jan,Apr,Jul,Oct
3              NaN              NaN              NaN
4              NaN              NaN              NaN
```

## 0.4 Section 2: Effect of Holidays (Q2)

**Q. Do people shop more during the holidays or before the holidays? Analyze how different types of holidays affect the sales.**

[83]:
```
subset = combined_data[["Date","Sales", "StateHoliday", "SchoolHoliday"]]

state_holidays = subset["StateHoliday"].unique()
school_holidays = subset["SchoolHoliday"].unique()
holidays_state ={}
holidays_school ={}
for holiday in state_holidays :
    if holiday != 0 and holiday != '0':
        # remove the None types
        holidays_state[holiday]=  subset[subset["StateHoliday"] ==␣
 ↪holiday]["Date"].unique()
        # gets holiday sales for unique holidays
```

[84]:
```
print(holidays_state.keys())
```

```
dict_keys(['a', 'b', 'c'])
```

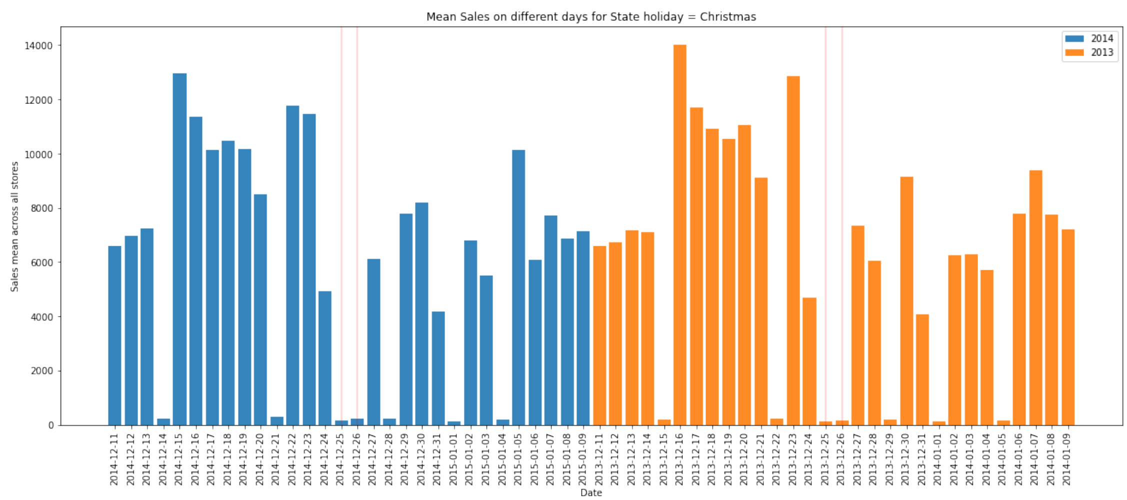
[283]:
```
subset = subset.drop(["StateHoliday", "SchoolHoliday"],1).groupby(["Date"]).
 ↪agg("mean")
subset = subset.reset_index()
```

Let's look at the sales pattern in a 14 day delta of each holiday

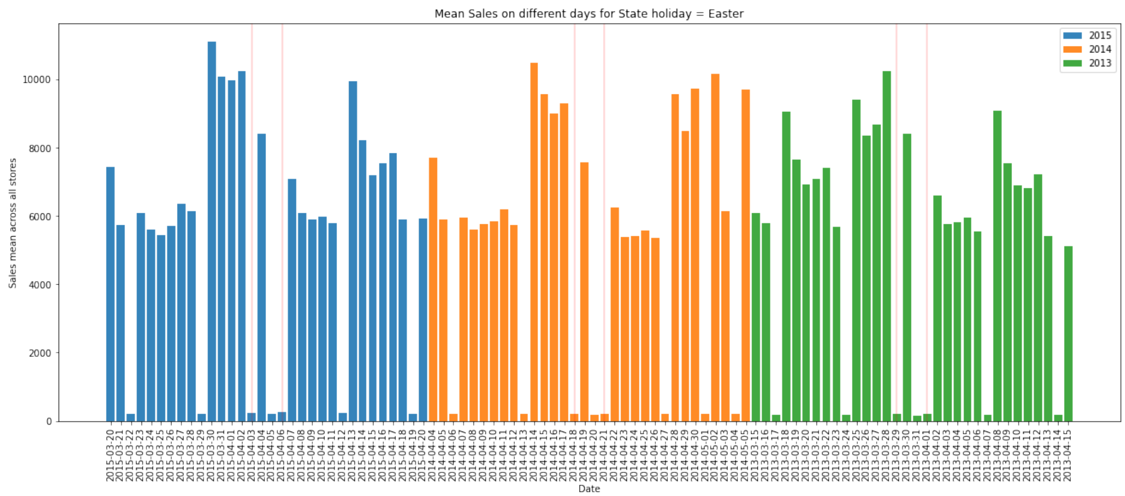
[86]:
```
# create a holiday table for each holiday type
# with aggregating dates within a 14 day period of before and after each holiday
# We want to look at how sales are affected in the month before/after each␣
↪holiday


holiday_table  = {}
for h in holidays_state:
    holiday_table[h] =[]
    if len(holidays_state[h]) != 0 :
        for date in holidays_state[h]:
            next_date = date + np.timedelta64(14,'D')
            prev_date = date - np.timedelta64(14,'D')
            holiday_table[h].append(subset[subset["Date"] >=␣
↪prev_date][subset["Date"] <= next_date ])
```

```
<ipython-input-86-9058f493ea1f>:13: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  holiday_table[h].append(subset[subset["Date"] >= prev_date][subset["Date"] <=
next_date ])
```

[87]:
```
plt.rcParams['figure.figsize'] = (16.0, 6.0)

# Look for Christmas holiday sales and 21 days before the sales
holiday_type  = "c"
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
for i in range(0,len(holiday_table[holiday_type]) , 2):
    table_holiday = pd.concat([holiday_table[holiday_type][i],␣
↪holiday_table[holiday_type][i+1]]).drop_duplicates()
    table_holiday = table_holiday.sort_values(by='Date', ascending=True,␣
↪na_position='first')
    table_holiday["Date"] = table_holiday["Date"].astype(str)
    dates = table_holiday['Date'].tolist()
    sales = table_holiday['Sales'].tolist()
    ax.bar(dates, sales, alpha=0.9, linewidth=2, label=dates[0].split("-")[0])

for date in holidays_state[holiday_type]:
    plt.axvline(x=str(date).split("T")[0], color='r' , alpha=0.2)

plt.legend()
plt.xlabel('Date');
plt.xticks(rotation=90)
plt.ylabel('Sales mean across all stores');
plt.title('Mean Sales on different days for State holiday = %s'%("Christmas"));
plt.show()
```




```
[88]: plt.rcParams['figure.figsize'] = (16.0, 6.0)

holiday_type = "b"
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
for i in range(0, len(holiday_table[holiday_type]), 2):
    table_holiday = pd.concat([holiday_table[holiday_type][i],␣
 ↪holiday_table[holiday_type][i+1]]).drop_duplicates()

    table_holiday = table_holiday.sort_values(by='Date', ascending=True,␣
 ↪na_position='first')
    table_holiday['Date'] = table_holiday['Date'].astype(str)
    dates = table_holiday['Date'].tolist()
    sales = table_holiday['Sales'].tolist()
    ax.bar(dates, sales, alpha=0.9, linewidth=2, label=dates[0].split("-")[0])

for date in holidays_state[holiday_type]:
    plt.axvline(x=str(date).split("T")[0], color='r', alpha=0.2)

plt.legend()
plt.xlabel('Date')
plt.xticks(rotation=90)
plt.ylabel('Sales mean across all stores')
plt.title('Mean Sales on different days for State holiday = %s'%("Easter"))
plt.show()
```



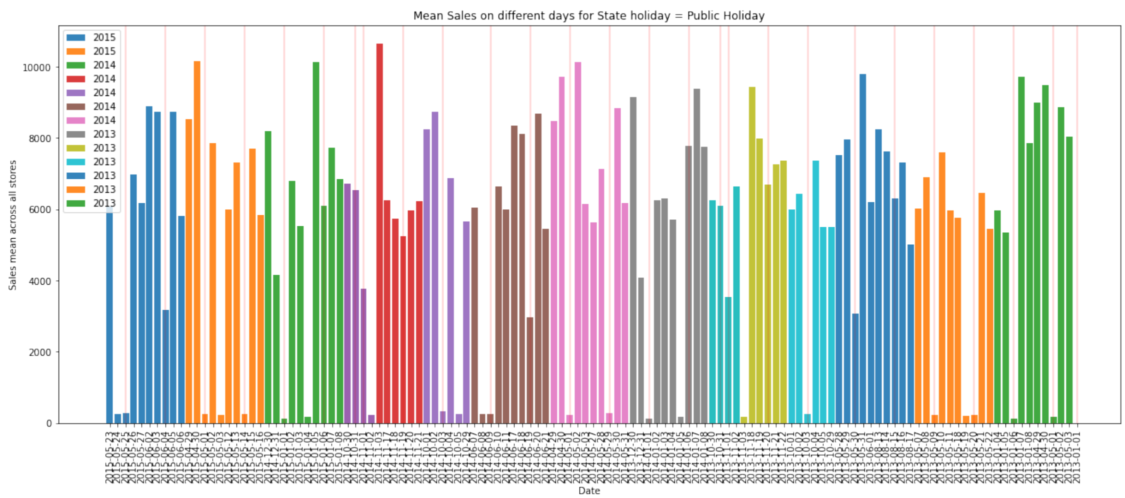

For public holidays, let's look at only 2 days before each holiday

[89]:
```python
holiday_table  = {}
for h in holidays_state:
    holiday_table[h] =[]
    if len(holidays_state[h]) != 0 :
        for date in holidays_state[h]:
            next_date = date + np.timedelta64(2,'D')
            prev_date = date - np.timedelta64(2,'D')
            holiday_table[h].append(subset[subset["Date"] >= prev_date][subset["Date"] <= next_date ])
```

```
<ipython-input-89-cca63a9edda8>:8: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  holiday_table[h].append(subset[subset["Date"] >= prev_date][subset["Date"] <=
next_date ])
```

[90]:
```python
plt.rcParams['figure.figsize'] = (16.0, 6.0)

holiday_type = "a"
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
for i in range(0, len(holiday_table[holiday_type])-1,2):
    table_holiday = pd.concat([holiday_table[holiday_type][i], holiday_table[holiday_type][i+1]]).drop_duplicates()

    table_holiday = table_holiday.sort_values(by='Date', ascending=True, na_position='first')
    table_holiday['Date'] = table_holiday['Date'].astype(str)
    dates = table_holiday['Date'].tolist()
    sales = table_holiday['Sales'].tolist()
```

```
    ax.bar(dates, sales, alpha=0.9, linewidth=2, label=dates[0].split("-")[0])

for date in holidays_state[holiday_type]:
    plt.axvline(x=str(date).split("T")[0], color='r', alpha=0.2)

plt.legend()
plt.xlabel('Date')
plt.xticks(rotation=90)
plt.ylabel('Sales mean across all stores')
plt.title('Mean Sales on different days for State holiday = %s'%("Public Holiday"))
plt.show()
```




**Observations** From the above graphs we can conclusively make the following observations

1. Mean sales in all stores are almost neglible on the day of holidays, across all holiday types there are almost no sales happening on the holidays. This can be attributed to the fact that most stores remain closed during the holidays (refer to plot charts in Q7).
2. For Christmas and Easter, we notice a similar increasing trend of sales just before the holidays and then almost no sales during and a slowly increasing sales just afterwards. This kind of trend makes sense because people tend to shop more just before the Christmas holidays and sales wouldn't return back to normal until a few weeks after the holidays.
3. For Public holidays, we don't really see much pattern across different dates so it is not conclusive how the public holidays affect sales.

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

Q. Amongst the stores with at least 6 months of sales data, list the IDs of:

- The five stores with the highest cumulative sales

- The five stores with the least cumulative sales
- Plot the sales per week over time for these two sets of stores.
- How similar are the patterns of sales each week amongst these two sets of stores? Make plots to reveal them.

[308]:
```
subset = combined_data[["Date", "Store", "Sales"]]
subset = subset.groupby("Store")
eligible_stores = subset.agg(["min", "max"])
eligible_stores["diff"] =  eligible_stores["Date"]["max"] -␣
↪eligible_stores["Date"]["min"]

# Get 6 months of sales data
eligible_stores = eligible_stores[eligible_stores["diff"] > np.
↪timedelta64(180,'D')]
eligible_stores = eligible_stores.index.tolist()
#print(eligible_stores)
```

[309]:
```
subset = subset.agg(["sum"])
subset = subset.reset_index()
boolean_series = subset.Store.isin(eligible_stores)
subset = subset[boolean_series]



subset["cumm_sum"] = subset["Sales"]["sum"]
subset = subset.sort_values(by='cumm_sum', ascending=False, na_position='first')
stores_in_descending_order = subset["Store"].tolist()

print("Stores with the highest sales : " , stores_in_descending_order[:5])



print("Stores with the least sales : " , stores_in_descending_order[-5:])
```

```
Stores with the highest sales :  [262, 817, 562, 1114, 251]
Stores with the least sales :  [263, 208, 198, 543, 307]
```

[335]:
```
subset = combined_data[["Date" , "Store", "Sales"]]
boolean_series = subset.Store.isin(stores_in_descending_order[:5] +␣
↪stores_in_descending_order[-5:])
subset = subset[boolean_series]

max = subset["Date"].max()
min = subset["Date"].min()

print("Minimum date : " , min , "Maximum date: ", max )
subset["week"] =  ((subset["Date"] - min) // 7).dt.days
subset = subset.groupby(["week" ,"Store"]).agg("sum")
subset = subset.reset_index()
```

```
Minimum date :  2013-01-01 00:00:00 Maximum date:  2015-07-31 00:00:00
```

[314]:

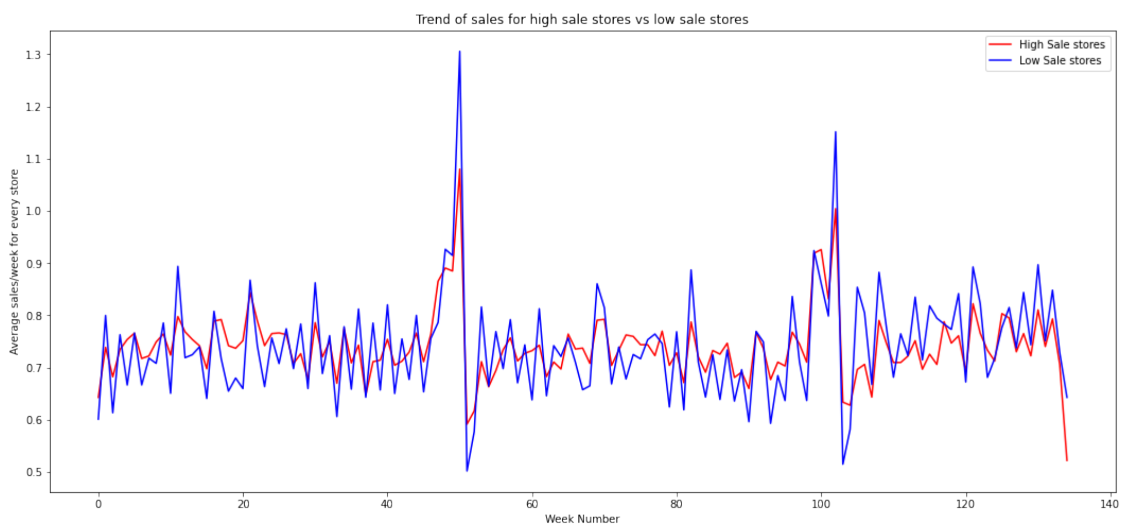
```python
plt.rcParams['figure.figsize'] = (18.0, 8.0)
fig = plt.figure()
for col in stores_in_descending_order[:5]:
    plot_data = subset[subset["Store"] == col]
    x = plot_data.week
    y = plot_data["Sales"]
    plt.plot(x, y, label = str(col));

for col in stores_in_descending_order[-5:]:
    plot_data = subset[subset["Store"] == col]
    x = plot_data.week
    y = plot_data["Sales"]
    plt.plot(x, y, linestyle='--', label = str(col));

plt.legend()
plt.xlabel('Week Number');
plt.ylabel('Cummulative sales/week');
plt.title('Cumulative sale for above stores vs week number');
```


**Observations** Plotting the average sales of stores per week shows us some trends in the data but it is not conclusive enough. For instance, we do see a strong dip in sales in all these stores roughly at the same time, i.e., around week 45 and week 105. The high grossing stores as well as the least grossing stores face a similar dip in sales.

However, other than the similar dip in sales there isn't much information that we can extract from this trend.

[340]:
```
high_sale_stores = subset.Store.isin( [262, 817, 562, 1114, 251] )

top_sellers = subset[high_sale_stores]
top_sellers = top_sellers.groupby(["week"]).agg("mean").drop("Store", 1)
top_sellers["average"] = top_sellers["Sales"] / top_sellers["Sales"].sum()  *␣
↪100

low_sale_stores = subset.Store.isin( [263, 208, 198, 543, 307] )

bottom_sellers = subset[low_sale_stores]
bottom_sellers = bottom_sellers.groupby(["week"]).agg("mean").drop("Store", 1)
bottom_sellers["average"] = bottom_sellers["Sales"] / bottom_sellers["Sales"].
↪sum() * 100
```

[341]:
```
fig = plt.figure()
plt.plot(top_sellers["average"], 'r-', label="High Sale stores")
plt.plot(bottom_sellers["average"], 'b-', label="Low Sale stores")
plt.legend()
plt.xlabel('Week Number');
plt.ylabel('Average sales/week for every store');
plt.title('Trend of sales for high sale stores vs low sale stores');
plt.show();
```




This plot compares the average sales of the high sale stores and the low sale stores. From the trend comparison it seems like the sale patterns of all stores are fairly similar. We notice that the sale peaks and dips around the same time in both the high sale stores and the low sale stores.

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

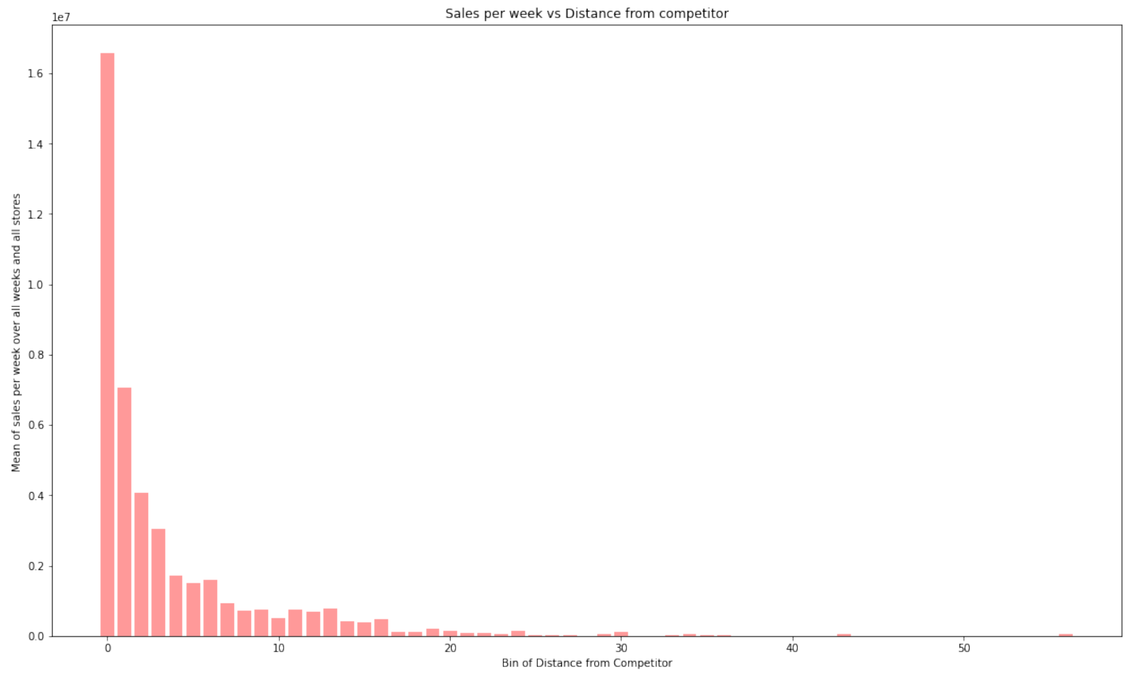
Q. Plot the sales per week against Distance of the closest competitor. Do the stores farther to competitors have a better sale per week than the closer ones? (5 points)

[231]:
```python
subset = combined_data[["Date", "Sales", "CompetitionDistance"]]
max = combined_data["Date"].max()
min = combined_data["Date"].min()

subset["week"] = ((subset["Date"] - min) // 7).dt.days

nbins = 57
max = subset["CompetitionDistance"].max()
min = subset["CompetitionDistance"].min()
bin_size = (max - min)/ nbins

subset["CompetitionDistance_bins"] =  (subset["CompetitionDistance"] - min) //␣
↪bin_size
subset = subset.drop(columns=["Date" , "CompetitionDistance"])
subset = subset.groupby(["CompetitionDistance_bins", "week"]).agg(["sum",␣
↪"count"])
subset = subset.reset_index()
#subset = subset.drop(columns=["week"])
subset = subset.groupby(["CompetitionDistance_bins"]).agg(["mean", "sum"])

Weekly_sale_sum = subset["Sales"]["sum"]
Weekly_sale_count = subset["Sales"]["count"]
```

```
<ipython-input-231-0c64a33b99d3>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  subset["week"] = ((subset["Date"] - min) // 7).dt.days
<ipython-input-231-0c64a33b99d3>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  subset["CompetitionDistance_bins"] =  (subset["CompetitionDistance"] - min) //
bin_size
/opt/conda/lib/python3.8/site-packages/pandas/core/generic.py:4153:
PerformanceWarning: dropping on a non-lexsorted multi-index without a level
parameter may impact performance.
  obj = obj._drop_axis(labels, axis, level=level, errors=errors)
```

[238]:

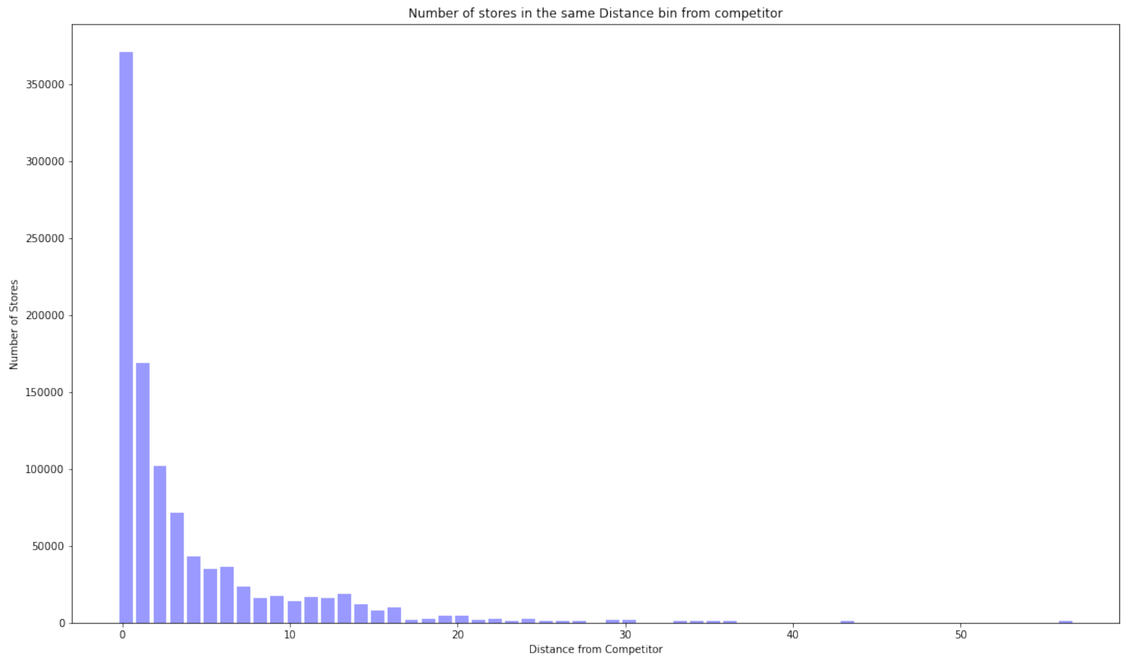
```
plt.rcParams['figure.figsize'] = (14.0, 8.0)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(Weekly_sale_sum.index,Weekly_sale_sum["mean"], alpha=0.4, color='r')
plt.xlabel('Bin of Distance from Competitor');
plt.ylabel('Mean of sales per week over all weeks and all stores ');
plt.title('Sales per week vs Distance from competitor');
plt.show()
```




We notice that the mean sales of all stores closest to their competitors usually show high sales per week. Although this cannot be concluded because the dataset does have a lot of exceptionally high-grossing stores (as we noticed in the earlier question). So it is very evident that the dataset is skewed to perform this analysis.

[241]:

```
plt.rcParams['figure.figsize'] = (14.0, 8.0)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(Weekly_sale_count.index+0.25, Weekly_sale_count["sum"], alpha=0.4,␣
 ↪color='b')
plt.xlabel('Distance from Competitor');
plt.ylabel('Number of Stores');
plt.title('Number of stores in the same Distance bin from competitor');
plt.show()
```



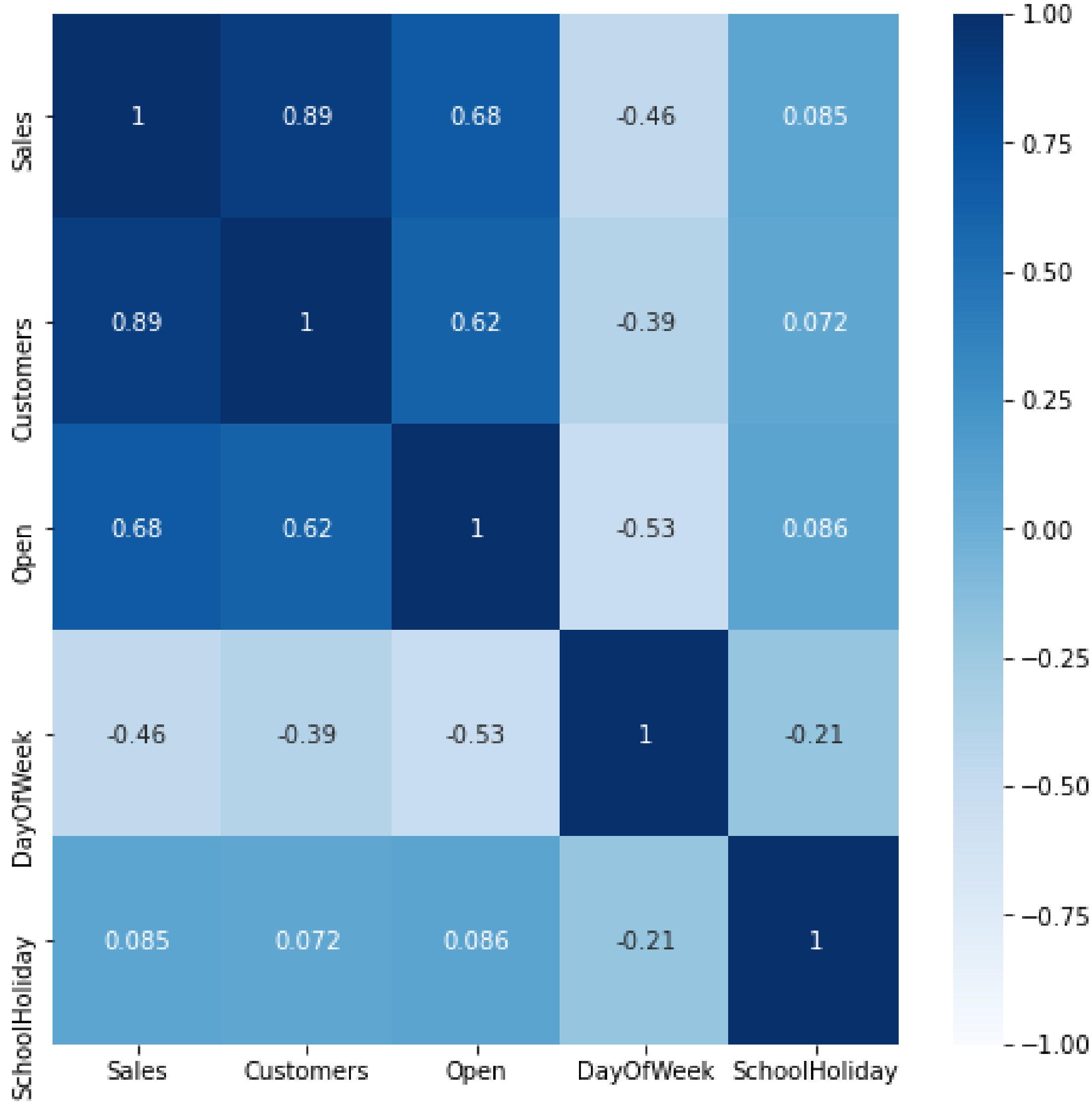

From the above plot we notice that the same bin (distance bin 0 from competitor) has exceptionally highly skewed number of stores present. Because of this the mean sales of all stores in Bin 0 is getting skewed in the previous graph.

### 0.7 Section 5: Pearson Correlation of Features (Q5)

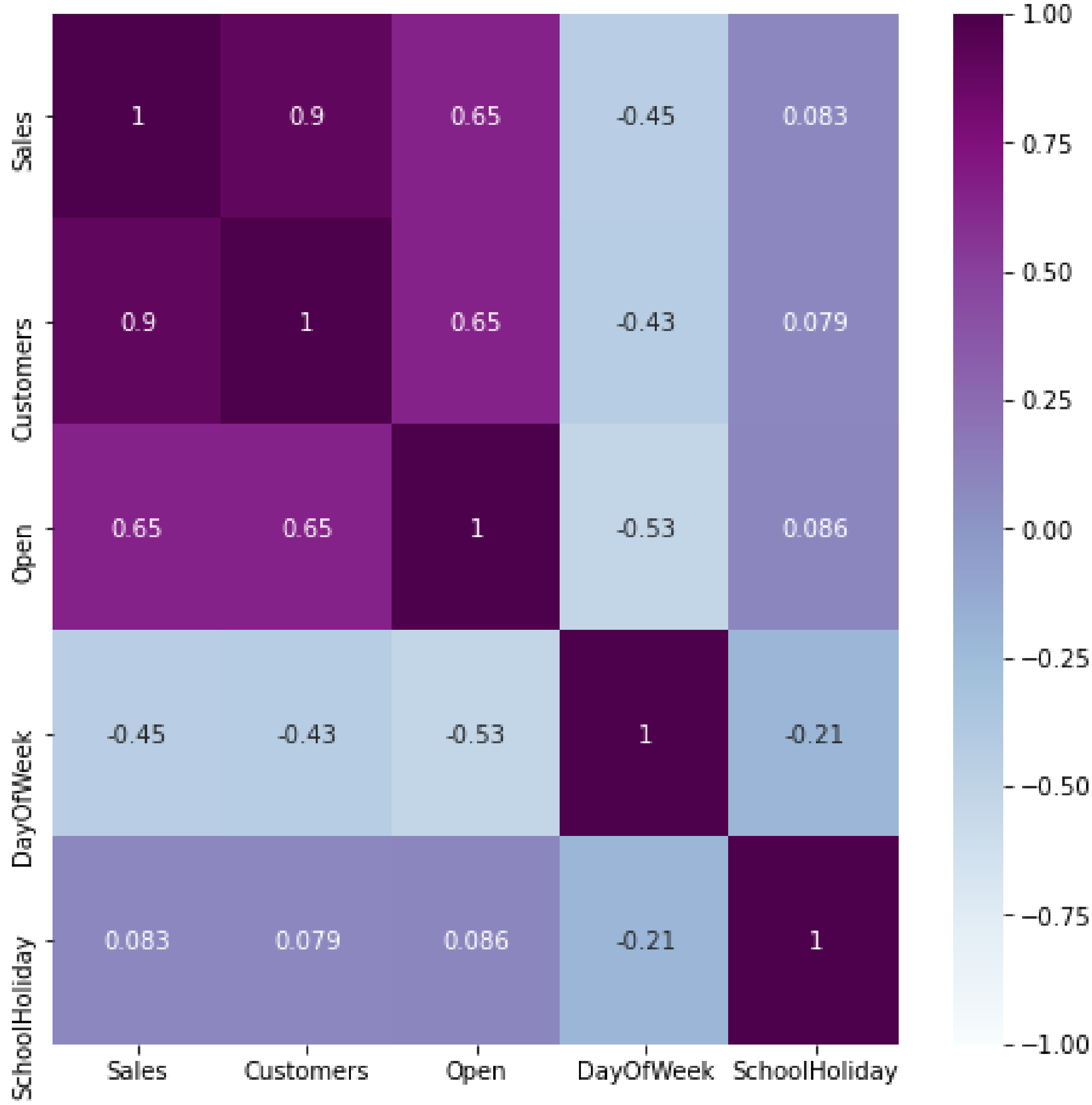
Q. Select a set of the five most interesting features (includings sales). Compute the Pearson correlation between all pairs of these variables. Show the result using a heat map, and list the feature-pairs with the strongest correlations. Which feature correlates the best with sales? How does this change with Spearman correlation? You can use the seaborn library to plot the heatmap, with instructions found here. (10 points)

[160]:
```
del subset
interesting_columns = ['Sales' , 'Customers', 'Open', "DayOfWeek",␣
↪"SchoolHoliday", "StateHoliday"]

subset = combined_data[interesting_columns]
subset = subset.fillna(-1)
```

[167]:
```
plt.rcParams['figure.figsize'] = (8.0, 8.0)
pearson = subset.corr(method='pearson')
ax = sns.heatmap(pearson, cmap='Blues', vmin=-1, annot=True)
#pearson
```




[168]:
```
plt.rcParams['figure.figsize'] = (8.0, 8.0)
spearman = subset.corr(method="spearman")
ax = sns.heatmap(spearman, cmap='BuPu', vmin=-1, annot=True)
```




The features with strongest correlation are Sales with Customers, with Pearson correlation value of 0.89 and Spearman correlation value of 0.9. This is kind of obvious as Sales is a direct outcome of number of customers.

Another feature that best correlates with Sales is the Open feature, which has a Pearson correlation of 0.62 and Spearman correlation of 0.65. This is also a pretty straightforward outcome, cause if the store is not open then the sales would tend to be 0. However, I expected this correlation value to be higher but I guess the dataset has many stores which are open on different dates (some might be closed on a Public holiday but some might not).

Among the other interesting observations is the correlation of SchoolHoliday vs Sales which is only 0.083. This means that SchoolHoliday has almost not impact on the Sales in a store.

## 0.8 Section 6: Permutation Testing (Q6)

Q. For each of three different variables (one likely good, one presumably meaningless, one at random), build single-variable regression models, and do a permutation test to determine a p-value on whether the predictions of the sales are better than chance. Use root-mean-squared error of the log(sale) as the statistic to score your model. In other words, compare how your model ranks by this metric on the real data compared to 100 (or more) random permutations of the sales assigned to the real data records. (15 points)

[207]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import permutation_test_score
```

[197]:
```
def freq_rank_enc(tdf, icol):
    mydict = {}
    len = tdf[icol].nunique()
    num = len-1
    for val, cnt in tdf[icol].value_counts().nlargest(len).iteritems():
        mydict[val] = num
        num-=1
    tdf[icol] = tdf[icol].map(mydict).astype('int16')

    return(tdf)
```

[244]:
```
X1 = combined_data.copy()
X1 = freq_rank_enc(X1, 'SchoolHoliday')[['SchoolHoliday']]
y = np.log(1 + combined_data['Sales'])
```

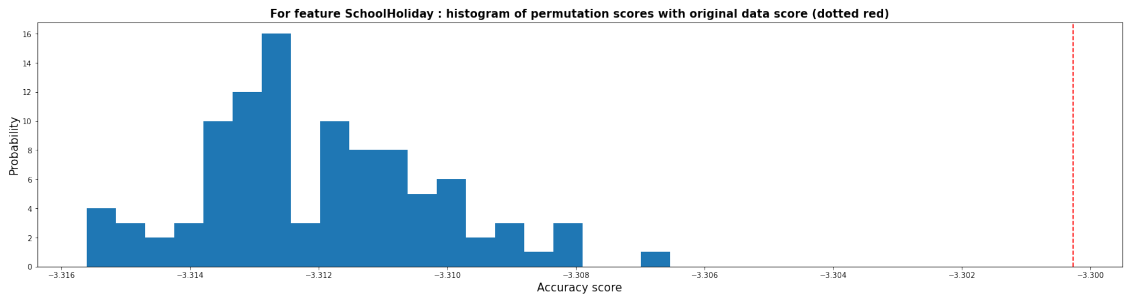
[245]:
```
X2 = combined_data[['Customers']]
y = np.log(1 + combined_data['Sales'])
```

[253]:
```
X3 = combined_data[['DayOfWeek']]
y = np.log(1 + combined_data['Sales'])
```

[247]:
```
def single_var_linreg(fX, fy):
    linreg_model = LinearRegression()
    cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
    return permutation_test_score(linreg_model, fX, fy,cv=cv,␣
 ↪scoring='neg_root_mean_squared_error', n_permutations=100, n_jobs=20)
```

[249]:
```
#X1
plt.rcParams['figure.figsize'] = (26, 6)
true_score, perm_scores, pvalue = single_var_linreg(X1, y)

fig, ax= plt.subplots()

ax.hist(perm_scores, bins=20, density=False)
ax.axvline(true_score, ls="--", color="r")
```

```
score_label = f"Score on original\ndata: {true_score:.2f}\n(p-value: {pvalue:.
↪3f})"
#ax.text(0.7, 10, score_label, fontsize=12)
ax.set_xlabel("Accuracy score", fontsize=15)
ax.set_ylabel("Probability", fontsize=15)
ax.set_title("For feature SchoolHoliday : histogram of permutation scores with␣
↪original data score (dotted red)", fontsize=15, weight='bold')
plt.show()
print("p-value: {}".format(pvalue))
```




```
p-value: 0.009900990099009901
```

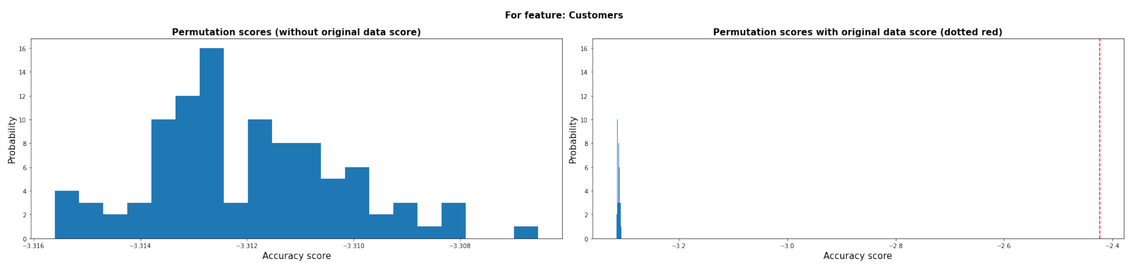
### Observations

Using 100 permutations. Using the feature 'SchoolHoliday' that is chosen at random. The pvalue is 0.009.

The red line indicates the score obtained by the classifier on the original data. The score is much better than those obtained by using permuted data and the p-value is thus very low. This indicates that there is a low likelihood that this good score would be obtained by chance alone. It provides evidence that the dataset contains real dependency between features and labels and the classifier was able to utilize this to obtain good results.

[250]:
```
#X2
plt.rcParams['figure.figsize'] = (26, 6)
true_score, perm_scores, pvalue = single_var_linreg(X2, y)

fig, axes= plt.subplots(1,2 ,figsize=(26,6))

axes[0].hist(perm_scores, bins=20, density=False)
axes[0].set_xlabel("Accuracy score", fontsize=15)
axes[0].set_ylabel("Probability", fontsize=15)
axes[0].set_title("Permutation scores (without original data score)",␣
↪fontsize=15, weight='bold')
```

```
axes[1].hist(perm_scores, bins=20, density=False)
axes[1].axvline(true_score, ls="--", color="r")
score_label = f"Score on original\ndata: {true_score:.2f}\n(p-value: {pvalue:.
↪3f})"
#axes[1].text(0.7, 10, score_label, fontsize=12)
axes[1].set_xlabel("Accuracy score", fontsize=15)
axes[1].set_ylabel("Probability", fontsize=15)
axes[1].set_title("Permutation scores with original data score (dotted red)",␣
↪fontsize=15, weight='bold')


plt.suptitle('For feature: Customers ',fontsize=15, weight='bold')
fig.tight_layout()
plt.show()
print("p-value: {}".format(pvalue))
```




```
p-value: 0.009900990099009901
```

**Observations**

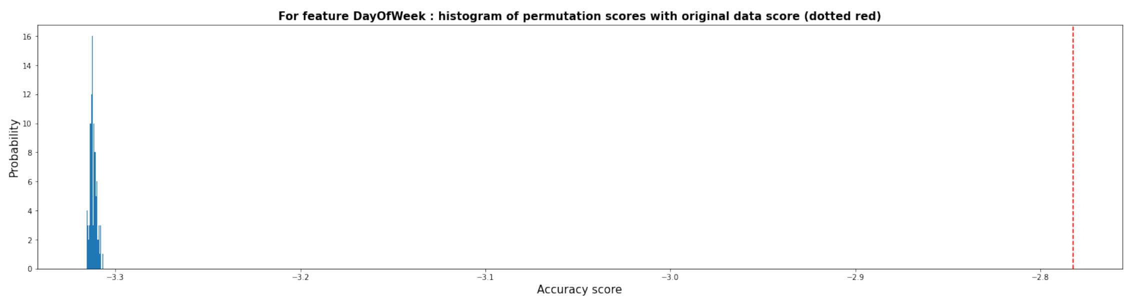
Using 100 permutations. Using the feature 'Customers' that is likely to do good. The pvalue is 0.009.

The left plot is a zoomed-in plot of the permutation scores alone (without the original data score). The right plot is the zoomed-out plot of the permutation scores with the original data score.

The red line in the right plot indicates the score obtained by the classifier on the original data. The score is much better than those obtained by using permuted data and the p-value is thus very low (0.0099). This indicates that there is a low likelihood that this good score would be obtained by chance alone. It provides evidence that the dataset contains real dependency between 'Customers' and 'Sales' and the classifier was able to utilize this to obtain good results.

[255]:
```
#X3
plt.rcParams['figure.figsize'] = (26, 6)
true_score, perm_scores, pvalue = single_var_linreg(X3, y)

fig, ax= plt.subplots()
```

```
ax.hist(perm_scores, bins=20, density=False)
ax.axvline(true_score, ls="--", color="r")
score_label = f"Score on original\ndata: {true_score:.2f}\n(p-value: {pvalue:.
 ↪3f})"
#ax.text(0.7, 10, score_label, fontsize=12)
ax.set_xlabel("Accuracy score", fontsize=15)
ax.set_ylabel("Probability", fontsize=15)
ax.set_title("For feature DayOfWeek : histogram of permutation scores with␣
 ↪original data score (dotted red)", fontsize=15, weight='bold')
plt.show()
print("p-value: {}".format(pvalue))
```




```
p-value: 0.009900990099009901
```

**Observations**

Using 100 permutations. Using the feature 'DayOfWeek' that is chosen at random. The pvalue is 0.009.

The red line in the right plot indicates the score obtained by the classifier on the original data. The score is much better than those obtained by using permuted data and the p-value is thus very low (0.0099). This indicates that there is a low likelihood that this good score would be obtained by chance alone. It provides evidence that the dataset contains real dependency between 'DayOfWeek' and 'Sales' and the classifier was able to utilize this to obtain good results.

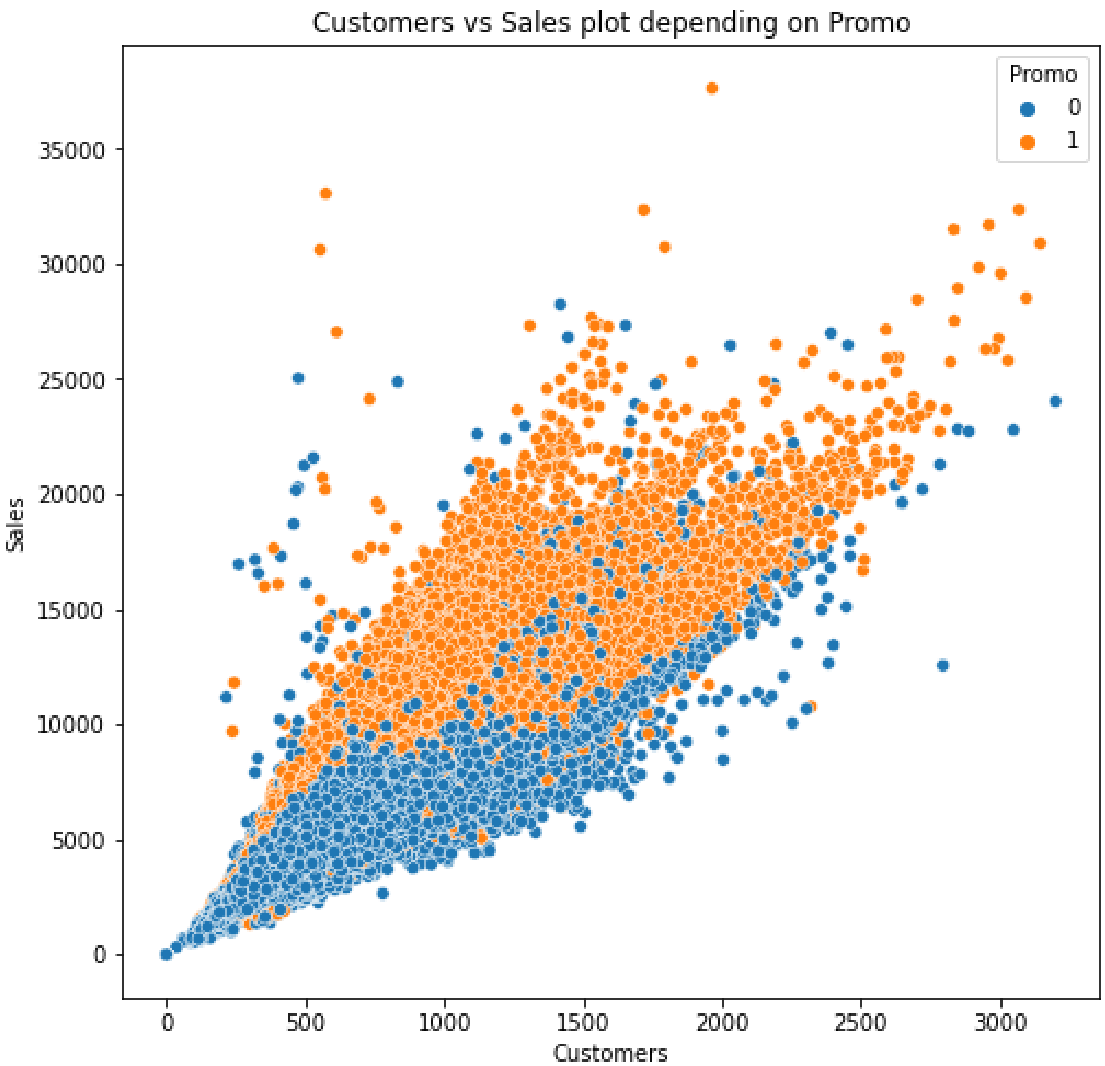
## 0.9 Section 7: Interesting findings (Q7)

Q. Produce five informative plots revealing aspects of the combined data. For each plot, describe interesting properties your visualization reveals. These must include: at least one line chart at least one scatter plot at least one histogram or bar chart

[258]:
```
subset = combined_data
subset.dropna(inplace=True)
```

### 0.9.1 Customers vs Sales (Scatter Plot)

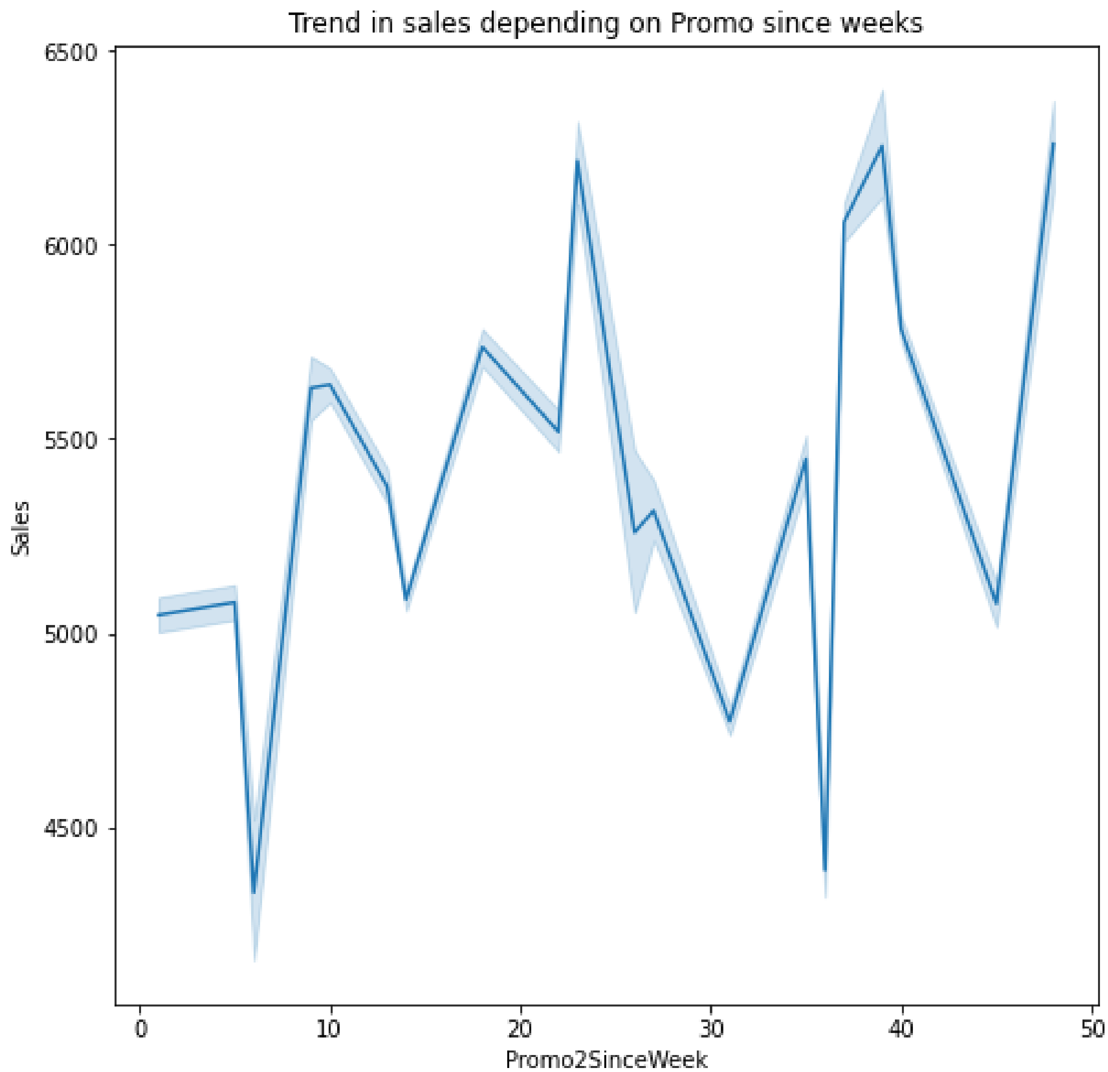
[82]:
```
plt.rcParams['figure.figsize'] = (8.0, 8.0)
sns.scatterplot(data=subset, x='Customers', y='Sales', hue='Promo')
plt.title("Customers vs Sales plot depending on Promo");
```




In the above plot, we notice the sale pattern against customers, while being dependent on whether a Store has an ongoing Promo. From the scatter plot, it seems evident that when Promo is going on the Sale of a store goes higher. We even notice a strong increase in number of customers at a store when the store has a promo going on.

### 0.9.2 Promo2SinceWeek vs Sales (Line Chart)

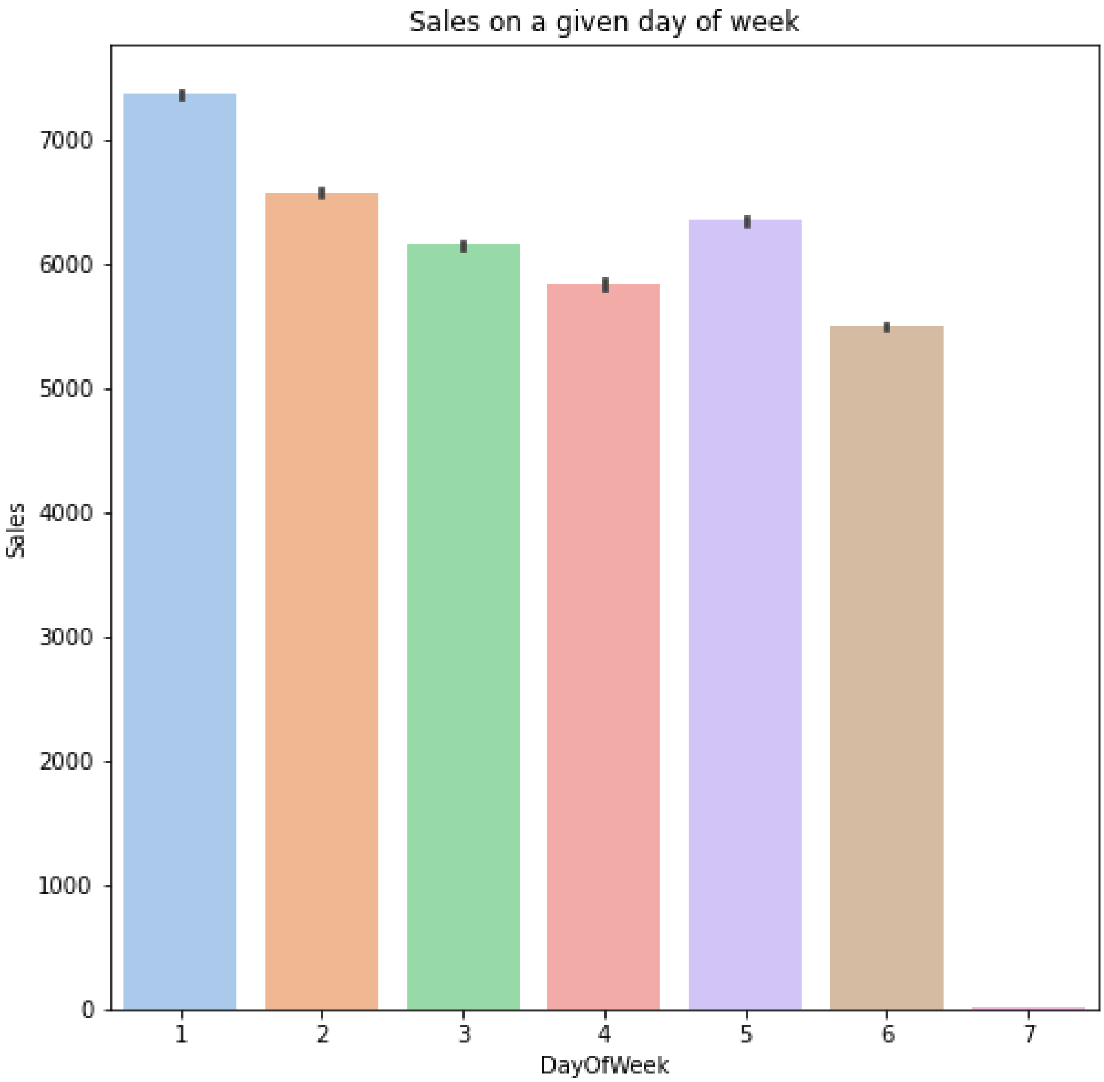
[83]:
```
plt.rcParams['figure.figsize'] = (8.0, 8.0)
sns.lineplot(data=subset, x='Promo2SinceWeek', y='Sales')
plt.title("Trend in sales depending on Promo since weeks");
```




The above plot shows how the trend of sale changes in the stores depending on how long they have had a promo ongoing. It would be expected that the sales would typically be higher somewhere in between (roughly 10-20 weeks) of the ongoing sale period. However, the trend does not seem to provide a concrete relation between sales and the length of an ongoing sale.

### 0.9.3 Sales depending on Day of Week

[279]:
```
plt.rcParams['figure.figsize'] = (8.0, 8.0)
sns.barplot(data=subset, x='DayOfWeek', y='Sales', palette='pastel')
plt.title("Sales on a given day of week");
```



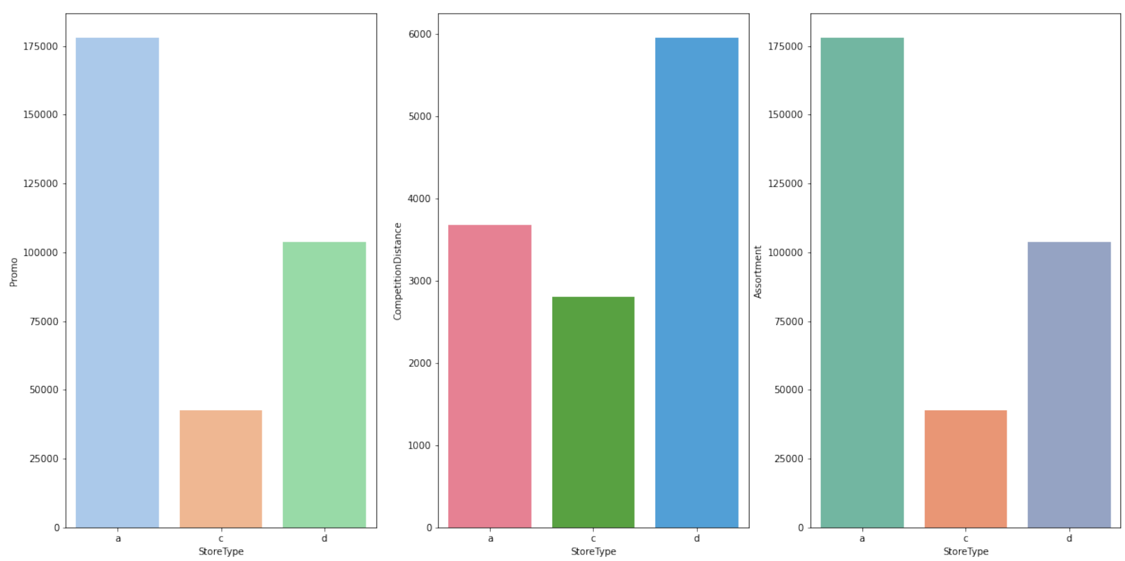

Day of the week mapping: 1 - Monday 2 - Tuesday 3 - Wednesday 4 - Thursday 5 - Friday 6 - Saturday 7 - Sunday

The above plot show how sales are affected by the day of the week. We notice maximum sales are seen on Monday and negligible (almost 0) sales on Sunday. This can be attributed to the fact that most stores tend to remain closed on Sunday, which in the given dataset is considered as a public holiday.

### 0.9.4 Performance of each store types and possible reasons

[282]:
```
fig, (ax1, ax2,ax3) = plt.subplots(nrows=1, ncols=3, figsize=(20,10))



tempDf = subset.groupby(subset.StoreType).count()
sns.barplot(tempDf.index, tempDf['Promo'], ax=ax1, palette='pastel')

tempDf = subset.groupby(subset.StoreType).mean()
sns.barplot(tempDf.index, tempDf['CompetitionDistance'], ax=ax2, palette='husl')

tempDf = subset.groupby(subset.StoreType).count()
sns.barplot(tempDf.index, tempDf['Assortment'], ax=ax3, palette='Set2')
plt.show()
```

```
/opt/conda/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
/opt/conda/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
/opt/conda/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



An interesting observation from the sales patterns could be to look at what kind of factors contribute towards the sales of a particular store type. We notice that maximum stores of type 'a' have a promo going on and also have the highest number of assortments.

On the contrary, maximum stores of StoreType 'd' have higher CompetitionDistance. These features may influence the sales/customer pattern of each store type.

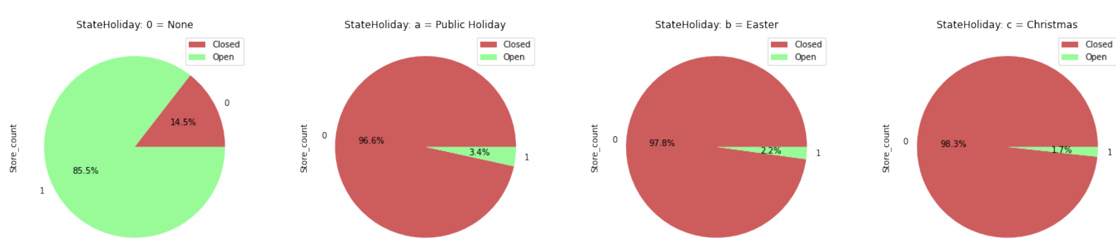
### 0.9.5 Stores closed/open on holidays

[161]:
```
subset = combined_data

subset['StateHoliday'].where(~(subset['StateHoliday'] == 0), other='0', inplace=True)
```

[164]:
```
subset['StateHoliday'].unique()
```

[164]:
```
array(['0', 'a', 'b', 'c'], dtype=object)
```

[167]:
```
fig, axes= plt.subplots(1, 4,figsize=(24,5))
df_hols = subset.groupby('StateHoliday')



i = 0
lmap = ['Closed', 'Open']
hmap = {'0':'None', 'a':'Public Holiday', 'b':'Easter', 'c':'Christmas'}
for key, grp in df_hols:
    subgrp = grp.groupby('Open').agg({'Store':['count']})
    subgrp.columns = subgrp.columns.map('_'.join)
    subgrp.plot(y='Store_count', kind='pie', ax=axes[i], colors = ['indianred', 'palegreen'], autopct='%1.1f%%', startangle=0)
    axes[i].set_title("StateHoliday: {} = {}".format(key, hmap[key]))
    axes[i].legend(lmap)
    i+=1
```




A very interesting and quite important analysis from the dataset is the number of stores that are open during a given Holiday. We find out that a large majority of stores are closed on Public

Holidays, Easter, and Christmas holidays. In the earlier analysis done in Q2, we noticed almost 0 sales on these holiday dates. This analysis validates the outcome as most stores remain closed during the holidays.

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

Q. Create a training set and a validation set using the data given in train.csv. The validation set must contain all the data from May, June, and July of 2015. The training set will consist of the rest of the data. Build two different prediction models to solve the task. Evaluate your model on the validation set using Root Mean Square Percentage Error ( ). You are free to do any kind of preprocessing, and use any algorithm for training. Explain the hyperparameters of your model. Report how the performance of the model and the time taken for training changes for different hyperparameter settings. You should try at least three different hyperparameter settings for each model. (15 points)

**Cleaning and Preprocessing of the data**

[12]:
```python
subset = combined_data

subset['year'] = subset['Date'].dt.year
subset['month'] = subset['Date'].dt.month
subset['day'] = subset['Date'].dt.day

# drop months 10, 11, 12 because they are not in test.csv
subset = subset.loc[subset['month']<10]
subset[['Promo','StateHoliday','SchoolHoliday']] =␣
 ↪subset[['Promo','StateHoliday','SchoolHoliday']].fillna(0)

subset.head(10)
```

```
/opt/conda/lib/python3.8/site-packages/pandas/core/frame.py:3191:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[k1] = value[k2]
```

[12]:

| | Store | DayOfWeek | Date | Sales | Customers | Open | Promo | StateHoliday \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 5 | 2015-07-31 | 5263 | 555 | 1 | 1 | 0 |
| 1 | 2 | 5 | 2015-07-31 | 6064 | 625 | 1 | 1 | 0 |
| 2 | 3 | 5 | 2015-07-31 | 8314 | 821 | 1 | 1 | 0 |
| 3 | 4 | 5 | 2015-07-31 | 13995 | 1498 | 1 | 1 | 0 |
| 4 | 5 | 5 | 2015-07-31 | 4822 | 559 | 1 | 1 | 0 |
| 5 | 6 | 5 | 2015-07-31 | 5651 | 589 | 1 | 1 | 0 |
| 6 | 7 | 5 | 2015-07-31 | 15344 | 1414 | 1 | 1 | 0 |
| 7 | 8 | 5 | 2015-07-31 | 8492 | 833 | 1 | 1 | 0 |
| 8 | 9 | 5 | 2015-07-31 | 8565 | 687 | 1 | 1 | 0 |

```
9     10           5 2015-07-31    7185         681      1       1             0

   SchoolHoliday StoreType  … CompetitionDistance  \
0              1         c  …         1270.000000
1              1         a  …          570.000000
2              1         a  …        14130.000000
3              1         c  …          620.000000
4              1         a  …        29910.000000
5              1         a  …          310.000000
6              1         a  …        24000.000000
7              1         a  …         7520.000000
8              1         a  …         2030.000000
9              1         a  …         3160.000000

   CompetitionOpenSinceMonth  CompetitionOpenSinceYear  Promo2  \
0                   9.000000               2008.000000       0
1                  11.000000               2007.000000       1
2                  12.000000               2006.000000       1
3                   9.000000               2009.000000       0
4                   4.000000               2015.000000       0
5                  12.000000               2013.000000       0
6                   4.000000               2013.000000       0
7                  10.000000               2014.000000       0
8                   8.000000               2000.000000       0
9                   9.000000               2009.000000       0

   Promo2SinceWeek  Promo2SinceYear    PromoInterval  year  month  day
0              NaN              NaN              NaN  2015      7   31
1        13.000000      2010.000000  Jan,Apr,Jul,Oct  2015      7   31
2        14.000000      2011.000000  Jan,Apr,Jul,Oct  2015      7   31
3              NaN              NaN              NaN  2015      7   31
4              NaN              NaN              NaN  2015      7   31
5              NaN              NaN              NaN  2015      7   31
6              NaN              NaN              NaN  2015      7   31
7              NaN              NaN              NaN  2015      7   31
8              NaN              NaN              NaN  2015      7   31
9              NaN              NaN              NaN  2015      7   31

[10 rows x 21 columns]
```

[13]:
```python
subset = subset.drop(columns=['Promo2', 'Promo2SinceWeek', 'Promo2SinceYear',␣
 ↪'PromoInterval'])

subset[['CompetitionDistance']] = subset[['CompetitionDistance']].
 ↪fillna(subset['CompetitionDistance'].max())
```

```
subset.head()
```

```
[13]:    Store  DayOfWeek       Date  Sales  Customers  Open  Promo StateHoliday  \
      0      1          5 2015-07-31   5263        555     1      1            0
      1      2          5 2015-07-31   6064        625     1      1            0
      2      3          5 2015-07-31   8314        821     1      1            0
      3      4          5 2015-07-31  13995       1498     1      1            0
      4      5          5 2015-07-31   4822        559     1      1            0

         SchoolHoliday StoreType Assortment  CompetitionDistance  \
      0              1         c          a          1270.000000
      1              1         a          a           570.000000
      2              1         a          a         14130.000000
      3              1         c          c           620.000000
      4              1         a          a         29910.000000

         CompetitionOpenSinceMonth  CompetitionOpenSinceYear  year  month  day
      0                   9.000000               2008.000000  2015      7   31
      1                  11.000000               2007.000000  2015      7   31
      2                  12.000000               2006.000000  2015      7   31
      3                   9.000000               2009.000000  2015      7   31
      4                   4.000000               2015.000000  2015      7   31
```

```
[14]: subset[['CompetitionOpenSinceYear']] = subset[['CompetitionOpenSinceYear']].
       ↪fillna(2018)



      subset[['CompetitionOpenSinceMonth']] = subset[['CompetitionOpenSinceMonth']].
       ↪fillna(12)



      # Assuming Competition started on the 1st day of that month

      subset['CompetitionOpenSinceDate'] = subset['CompetitionOpenSinceYear'].
       ↪astype('int').astype('str') + '-' + subset['CompetitionOpenSinceMonth'].
       ↪astype('int').astype('str') + '-01'
      subset['CompetitionOpenSinceDate'] = pd.
       ↪to_datetime(subset['CompetitionOpenSinceDate'])

      subset['CompetitionOpenSince_TotalMonths'] = subset['Date'].dt.to_period('M').
       ↪astype(int) - subset['CompetitionOpenSinceDate'].dt.to_period('M').
       ↪astype(int)
```

```
subset['CompetitionOpenSince_TotalMonths'] =␣
↪subset['CompetitionOpenSince_TotalMonths'].map(lambda x: 0 if x < 0 else x).
↪fillna(0)
subset = subset.drop(columns=['Date','CompetitionOpenSinceYear',␣
↪'CompetitionOpenSinceMonth', 'CompetitionOpenSinceDate'])
```

[15]:
```
#### Normalize all columns

norm_cols = ['CompetitionOpenSince_TotalMonths','Store', 'DayOfWeek','month',␣
↪'day', 'CompetitionDistance', 'Sales']
encode_cols = ['StateHoliday', 'StoreType', 'Assortment']
train_cols =␣
↪['CompetitionOpenSince_TotalMonths_norm','CompetitionDistance_norm','Assortment_norm','Stor
↪'DayOfWeek_norm', 'Open', 'Promo', 'StateHoliday_norm', 'SchoolHoliday',␣
↪'month_norm']
label_cols = ['Sales_norm']
```

[16]:
```
def normalizer(new_df, scaling):
    for col in norm_cols:
        new_col_name = str(col) + "_norm"
        new_df[[new_col_name]] = scaling.fit_transform(new_df[[col]])
    for col in encode_cols:
        new_col_name = str(col) + "_norm"
        new_df[[new_col_name]] = scaling.fit_transform(new_df[[col]])
    return new_df
```

[17]:
```
save_dicts = {}
def freq_rank_encoding(new_df):
    for col in encode_cols:
        mydict = {}
        len = new_df[col].nunique()
        num = len-1
        for val, cnt in new_df[col].value_counts().nlargest(len).iteritems():
            mydict[val] = num
            num-=1
        save_dicts[col] = mydict
        new_df[col] = new_df[col].map(mydict).astype('int16')

    return(new_df)
```

[18]:
```
def recover_freq_encoding(new_df):
    for col in encode_cols:
        mydict = save_dicts[col]
        new_df[col] = new_df[col].map(mydict).astype('int16')
    return(new_df)
```

[19]:
```python
from sklearn.preprocessing import StandardScaler
# encode
subset = freq_rank_encoding(subset)
# normalize
std_scaler = StandardScaler()
subset = normalizer(subset, std_scaler)
```

[13]:
```python
subset.head(5)
```

[13]:
```
   Store  DayOfWeek  Sales  Customers  Open  Promo  StateHoliday  \
0      1          5   5263        555     1      1             3
1      2          5   6064        625     1      1             3
2      3          5   8314        821     1      1             3
3      4          5  13995       1498     1      1             3
4      5          5   4822        559     1      1             3

   SchoolHoliday  StoreType  Assortment  …  \
0              1          1           2  …
1              1          3           2  …
2              1          3           2  …
3              1          1           1  …
4              1          3           2  …

   CompetitionOpenSince_TotalMonths_norm  Store_norm  DayOfWeek_norm  \
0                               0.604503   -1.731198        0.499698
1                               0.755847   -1.728092        0.499698
2                               0.922325   -1.724985        0.499698
3                               0.422891   -1.721879        0.499698
4                              -0.591110   -1.718772        0.499698

   month_norm  day_norm  CompetitionDistance_norm  Sales_norm  \
0    0.949070  1.747326                 -0.510730   -0.114770
1    0.949070  1.747326                 -0.593185    0.098603
2    0.949070  1.747326                  1.004079    0.697964
3    0.949070  1.747326                 -0.587295    2.211285
4    0.949070  1.747326                  2.862843   -0.232245

   StateHoliday_norm  StoreType_norm  Assortment_norm
0           0.435296       -1.784566         0.928010
1           0.435296        0.808038         0.928010
2           0.435296        0.808038         0.928010
3           0.435296       -1.784566        -1.011715
4           0.435296        0.808038         0.928010

[5 rows x 25 columns]
```

**Splitting data into train and test sets**

[20]:
```
subset_train = subset[subset['month'] < 5]


## Test set with months 5 - 8
subset_test = subset[subset['month'] >= 5]

subset_test = subset_test[subset_test['month'] < 8]
```

[21]:
```
train_X = subset_train[train_cols]
train_y = subset_train[label_cols]
test_X = subset_test[train_cols]
test_y = subset_test[label_cols]
```

### 0.10.1 Random Forest with hyperparameter changes

[24]:
```
from sklearn import metrics
from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestRegressor
import joblib
```

[26]:
```
def rmspe(y_pred, y_gt):
    answer = np.sqrt(np.mean(np.square((y_gt-y_pred)/(y_gt))))
    return answer
```

### 0.10.2 Model 1.1

[37]:
```
train_X = subset_train[train_cols]
train_y = subset_train[label_cols]
test_X = subset_test[train_cols]
test_y = subset_test[label_cols]
```

[38]:
```
rf_model = RandomForestRegressor()


rf_model.fit(train_X, train_y)


filename = 'rfmodel_1-1.dat'
joblib.dump(rf_model, filename)


test_y_pred = rf_model.predict(test_X)

test_y_pred = np.reshape(np.asarray(test_y_pred),(test_y_pred.shape[0],1))
test_y = np.reshape(np.asarray(test_y),(test_y.shape[0],1))
```

```
test_rmspe = rmspe(test_y_pred, test_y)
test_mse = metrics.mean_squared_error(test_y, test_y_pred, squared=False)
print(test_rmspe, test_mse)
```

```
<ipython-input-38-7881fe97275b>:4: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  rf_model.fit(train_X, train_y)
```

```
76.36201139058976 0.3019347858980826
```

### 0.10.3 Model 1.2

[39]:
```
train_X = subset_train[train_cols]
train_y = subset_train[label_cols]
test_X = subset_test[train_cols]
test_y = subset_test[label_cols]
```

[40]:
```
rf_model = RandomForestRegressor(n_estimators=150, max_depth=10)



rf_model.fit(train_X, train_y)



filename = 'rfmodel_1-2.dat'
joblib.dump(rf_model, filename)



test_y_pred = rf_model.predict(test_X)

test_y_pred = np.reshape(np.asarray(test_y_pred),(test_y_pred.shape[0],1))
test_y = np.reshape(np.asarray(test_y),(test_y.shape[0],1))

test_rmspe = rmspe(test_y_pred, test_y)
test_mse = metrics.mean_squared_error(test_y, test_y_pred, squared=False)
print(test_rmspe, test_mse)
```

```
<ipython-input-40-25d8c2f012a2>:4: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  rf_model.fit(train_X, train_y)
```

```
123.32178096218212 0.5868480976122588
```

### 0.10.4 Model 1.3

[41]:
```
train_X = subset_train[train_cols]
train_y = subset_train[label_cols]
test_X = subset_test[train_cols]
```

```
test_y = subset_test[label_cols]
```

[42]:
```
rf_model = RandomForestRegressor(n_estimators=50, max_depth=15)


rf_model.fit(train_X, train_y)


filename = 'rfmodel_1-3.dat'
joblib.dump(rf_model, filename)

test_y_pred = rf_model.predict(test_X)

test_y_pred = np.reshape(np.asarray(test_y_pred),(test_y_pred.shape[0],1))
test_y = np.reshape(np.asarray(test_y),(test_y.shape[0],1))

test_rmspe = rmspe(test_y_pred, test_y)
test_mse = metrics.mean_squared_error(test_y, test_y_pred, squared=False)
print(test_rmspe, test_mse)
```

```
<ipython-input-42-8475c41bd4b4>:4: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  rf_model.fit(train_X, train_y)
```

```
115.32741270834569 0.4879421765480409
```

## 0.11 XGBoost with hyperparameter tuning

[43]:
```
from xgboost.sklearn import XGBRegressor
```

### 0.11.1 Model 2.1

[44]:
```
train_X = subset_train[train_cols]
train_y = subset_train[label_cols]
test_X = subset_test[train_cols]
test_y = subset_test[label_cols]
```

[46]:
```
xgb_model = XGBRegressor()
xgb_model.fit(train_X, train_y)


filename = 'xgboost_2-1.dat'
joblib.dump(xgb_model, filename)


test_y_pred = xgb_model.predict(test_X)
```

```
test_y_pred = np.reshape(np.asarray(test_y_pred),(test_y_pred.shape[0],1))
test_y = np.reshape(np.asarray(test_y),(test_y.shape[0],1))

test_rmspe = rmspe(test_y_pred, test_y)
test_mse = metrics.mean_squared_error(test_y, test_y_pred, squared=False)
print(test_rmspe, test_mse)
```

```
84.24669462302715 0.35671914205797595
```

### 0.11.2 Model 2.2

[214]:
```
train_X = subset_train[train_cols]
train_y = subset_train[label_cols]
test_X = subset_test[train_cols]
test_y = subset_test[label_cols]
```

[216]:
```
xgb_model = XGBRegressor(tree_method='hist', max_depth=15)
xgb_model.fit(train_X, train_y)


filename = 'xgboost_2-2.dat'
joblib.dump(xgb_model, filename)


test_y_pred2 = xgb_model.predict(test_X)

test_y_pred2 = np.reshape(np.asarray(test_y_pred2),(test_y_pred2.shape[0],1))
test_y = np.reshape(np.asarray(test_y),(test_y.shape[0],1))

test_rmspe = rmspe(test_y_pred, test_y)
test_mse = metrics.mean_squared_error(test_y, test_y_pred2, squared=False)
print(test_rmspe, test_mse)
```

```
69.20161086585692 0.32136793353181586
```

### 0.11.3 Model 2.3

[217]:
```
train_X = subset_train[train_cols]
train_y = subset_train[label_cols]
test_X = subset_test[train_cols]
test_y = subset_test[label_cols]
```

[218]:
```
xgb_model = XGBRegressor(tree_method='exact', max_depth=15)
xgb_model.fit(train_X, train_y)


filename = 'xgboost_2-3.dat'
joblib.dump(xgb_model, filename)
```

```
test_y_pred3 = xgb_model.predict(test_X)

test_y_pred3 = np.reshape(np.asarray(test_y_pred3),(test_y_pred3.shape[0],1))
test_y = np.reshape(np.asarray(test_y),(test_y.shape[0],1))

test_rmspe = rmspe(test_y_pred3, test_y)
test_mse = metrics.mean_squared_error(test_y, test_y_pred3, squared=False)
print(test_rmspe, test_mse)
```

```
71.08587605737577 0.30911001604978294
```

### 0.11.4 Results comparison

| Model | Description | Hyperparameter | RMSPE Score | MSE score |
|---|---|---|---|---|
| Model 1.1 | Random Forest | default | 76.362 | 0.301 |
| Model 1.2 | Random Forest | n_estimators=150, max_depth=10 | 123.321 | 0.586 |
| Model 1.3 | Random Forest | n_estimators=50, max_depth=15 | 115.327 | 0.487 |
| Model 2.1 | XGBoost | default | 84.246 | 0.356 |
| Model 2.2 | XGBoost | tree_method='hist', max_depth=15 | 69.201 | 0.321 |
| Model 2.3 | XGBoost | tree_method='exact', max_depth=15 | 71.085 | 0.309 |

The best performing models on this dataset seems to be (based on the RMSPE score), Models 2.2 and 2.3. Both of these models are based on XGBoost with a small change in the hyperparameter. Both of these models are using a max_depth of 15 and the difference is in the tree_method I used.

## 0.12 Section 9: t-test (Q9)

[219]:
```
from scipy.stats import ttest_ind


tscore,pvalue=ttest_ind(test_y_pred2,test_y_pred3)
print("T-Statistic:",tscore)
print("P-value:",pvalue)
```

```
T-Statistic: [-1.04677531]
P-value: [0.29520364]
```

**Observations** Assume a significance threshold of alpha=0.05 (=5% or 1/20 chance) for rejecting the null hypothesis that both models perform equally well on the dataset and conduct the t test.

If the pvalue is smaller than alpha, we reject the null hypothesis and accept that there is a significant difference in the two models. Since pvalue > alpha, we cannot reject the null hypothesis and may conclude that the performance of the two algorithms is not significantly different.

Assuming that we conducted this test with a significance level of alpha=0.05, we can reject the null-hypothesis that both models perform equally well on this dataset, since the p-value (p<0.001) is smaller than alpha. We accept that there is a significant difference between the two models.

Significance level is 0.05

Since p<0.05, We can reject the null-hypothesis that both models perform equally well on this dataset and thus conclude that the two algorithms are significantly different. Since p>0.05, we cannot reject the null hypothesis and may conclude that the performance of the two algorithms is not significantly different.

### 0.13 Section 10: Screenshots (Q10)

Q. Predict the sales for all the test instances in "test.csv". Write the result into a csv file following the format of the file "sample_submission.csv" and submit it to the website. Do this for both models you develop. Report the private score, the public score for your highest scoring model and the total number of submissions you have made on Kaggle. Include a snapshot of your best score from my submission page as confirmation. Be sure to provide a link to your Kaggle profile.

[264]:
```
test_data = pd.read_csv('test.csv', parse_dates=True ,low_memory=False)

test_data['Date'] = pd.to_datetime(test_data['Date'])

test_data['year'] = test_data['Date'].dt.year
test_data['month'] = test_data['Date'].dt.month
test_data['day'] = test_data['Date'].dt.day

test_data[['Promo','StateHoliday', 'SchoolHoliday']] =␣
↪test_data[['Promo','StateHoliday', 'SchoolHoliday']].fillna(0)
```

[265]:
```
test_data = test_data.merge(store_data, how='left', on='Store')
```

[266]:
```
# correcting competitionopensince
test_data[['CompetitionOpenSinceYear']] =␣
↪test_data[['CompetitionOpenSinceYear']].fillna(2018) # future year which␣
↪will get removed in calculations
test_data[['CompetitionOpenSinceMonth']] =␣
↪test_data[['CompetitionOpenSinceMonth']].fillna(12)


# Assuming Competition started on the 1st day of that month
test_data['CompetitionOpenSinceDate'] = test_data['CompetitionOpenSinceYear'].
↪astype('int').astype('str') + '-' + test_data['CompetitionOpenSinceMonth'].
↪astype('int').astype('str') + '-01'
test_data['CompetitionOpenSinceDate'] = pd.
↪to_datetime(test_data['CompetitionOpenSinceDate'])

test_data['CompetitionOpenSince_TotalMonths'] = test_data['Date'].dt.
↪to_period('M').astype(int) - test_data['CompetitionOpenSinceDate'].dt.
↪to_period('M').astype(int)
```

```python
test_data['CompetitionOpenSince_TotalMonths'] =␣
↪test_data['CompetitionOpenSince_TotalMonths'].map(lambda x: 0 if x < 0 else␣
↪x).fillna(0)
test_data = test_data.drop(columns=['Date','CompetitionOpenSinceYear',␣
↪'CompetitionOpenSinceMonth', 'CompetitionOpenSinceDate'])

test_data = test_data.fillna(0) # for Open column
```

[267]:
```python
#### normalize now
norm_cols = ['CompetitionOpenSince_TotalMonths','Store', 'DayOfWeek','month',␣
↪'day', 'CompetitionDistance']
encode_cols = ['StateHoliday', 'StoreType', 'Assortment']
train_cols =␣
↪['CompetitionOpenSince_TotalMonths_norm','CompetitionDistance_norm','Assortment_norm','Stor
↪'DayOfWeek_norm', 'Open', 'Promo', 'StateHoliday_norm', 'SchoolHoliday',␣
↪'month_norm']
label_cols = ['Sales_norm']


# encode
test_data = recover_freq_encoding(test_data)


# normalize this data
std_scaler = StandardScaler()
test_data = normalizer(test_data, std_scaler)
```

[268]:
```python
test_X = test_data[train_cols]
```

[269]:
```python
rf_model = joblib.load('rfmodel_1-1.dat')

test_y_pred = rf_model.predict(test_X)
```

[270]:
```python
meanval = train_data['Sales'].mean()
stdval = train_data['Sales'].std()
test_y_pred = (test_y_pred * stdval) + meanval
```

[185]:
```python
submission = pd.DataFrame({
    'Id': test_data['Id'],
    'Sales': test_y_pred
})
submission.to_csv('sanket_random_forest_submission.csv', index=False)
```

[186]:
```python
xgb_model = joblib.load('xgboost_2-2.dat')


test_y_pred = xgb_model.predict(test_X)
```

```
[187]: meanval = train_data['Sales'].mean()
       stdval = train_data['Sales'].std()
       test_y_pred = (test_y_pred * stdval) + meanval
```

```
[188]: submission = pd.DataFrame({
           'Id': test_data['Id'],
           'Sales': test_y_pred
       })
       submission.to_csv('sanket_xgboost_submission.csv', index=False)
```

Public Score & Highest Rank:

0.41843

Private Score & Highest Rank:

0.44061

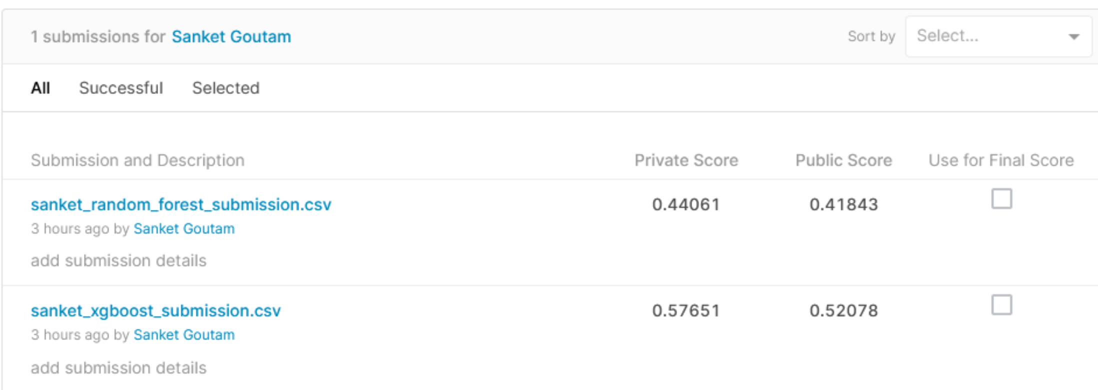
Kaggle profile link:

https://www.kaggle.com/sanketgoutam

Screenshot(s):

```
[272]: from IPython.display import Image
       Image(filename='Sanket_Goutam_HW3.PNG')
```

[272]:



1 submissions for Sanket Goutam    Sort by Select...

All    Successful    Selected

| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| sanket_random_forest_submission.csv<br>3 hours ago by Sanket Goutam<br>add submission details | 0.44061 | 0.41843 | ☐ |
| sanket_xgboost_submission.csv<br>3 hours ago by Sanket Goutam<br>add submission details | 0.57651 | 0.52078 | ☐ |