Rossmann Store Sales Forecasting

Krishna Chaitanya Sanka

Problem Statement:

Rossmann operates over 3,000 drug stores in 7 European countries. Currently, Rossmann store managers are tasked with predicting their daily sales for up to six weeks in advance. Store sales are influenced by many factors, including promotions, competition, school and state holidays, seasonality, and locality. With thousands of individual managers predicting sales based on their unique circumstances, the accuracy of results can be quite varied.

Task:

The task is to forecast 6 weeks of daily sales for 1,115 stores located across Germany using machine learning approach.

Business Impact:

- Reliable sales forecasts enable store managers to create effective staff schedules that increase productivity and motivation.
- By helping Rossmann create a robust prediction model, you will help store managers stay focused on what's most important to them: their customers and their teams!

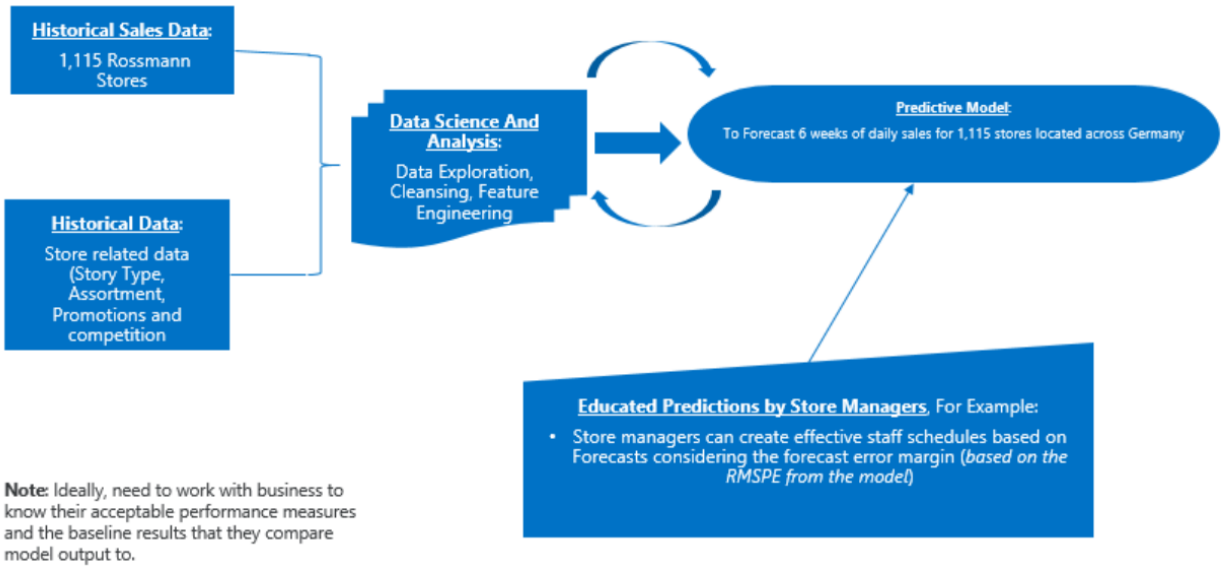
Description of the dataset:

Historical Sales data for 1,115 Rossmann stores that includes store features, promotions and data related to competition

Most of the fields are self-explanatory. The following are descriptions for those that aren't.

- **Id** - an Id that represents a (Store, Date) duple within the test set
- **Store** - a unique Id for each store
- **Sales** - the turnover for any given day (this is what you are predicting)
- **Customers** - the number of customers on a given day
- **Open** - an indicator for whether the store was open: 0 = closed, 1 = open
- **StateHoliday** - indicates a state holiday. Normally all stores, with few exceptions, are closed on state holidays. Note that all schools are closed on public holidays and weekends. a = public holiday, b = Easter holiday, c = Christmas, 0 = None
- **SchoolHoliday** - indicates if the (Store, Date) was affected by the closure of public schools
- **StoreType** - differentiates between 4 different store models: a, b, c, d
- **Assortment** - describes an assortment level: a = basic, b = extra, c = extended
- **CompetitionDistance** - distance in meters to the nearest competitor store
- **CompetitionOpenSince**[Month/Year] - gives the approximate year and month of the time the nearest competitor was opened
- **Promo** - indicates whether a store is running a promo on that day
- **Promo2** - Promo2 is a continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating
- **Promo2Since[Year/Week]** - describes the year and calendar week when the store started participating in Promo2
- **PromoInterval** - describes the consecutive intervals Promo2 is started, naming the months the promotion is started anew. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store

High Level Approach



Load the datasets

Load the dataset from CSV files:

Historical Sales Data as train :

Store data as store :

Test dataset used for forecasting as test:

```
train = pd.read_csv("train.csv")

store  = pd.read_csv("store.csv")

test = pd.read_csv("test.csv")
```

Data Exploration:

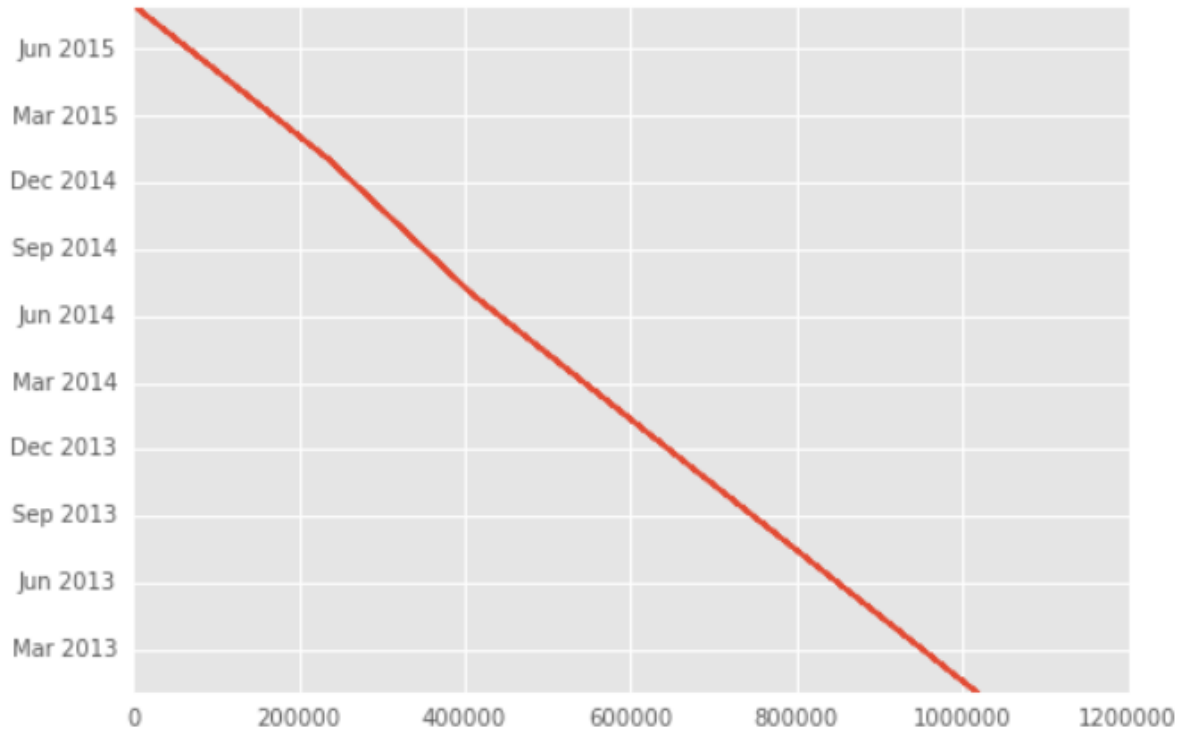
- Convert Date column to datetime object and sort train and test data based on date. This will help us to draw plots to see time series aspect and also understand the continuity of the data

```
train['Date'] = pd.to_datetime(train['Date'])

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

train = train.sort_values(by = 'Date')

test = test.sort_values(by = 'Date')
```

- **Check if there are any gaps in train time-period?**

```
plt.plot(train.Date)

plt.show()
```

There are no gaps in train time-period

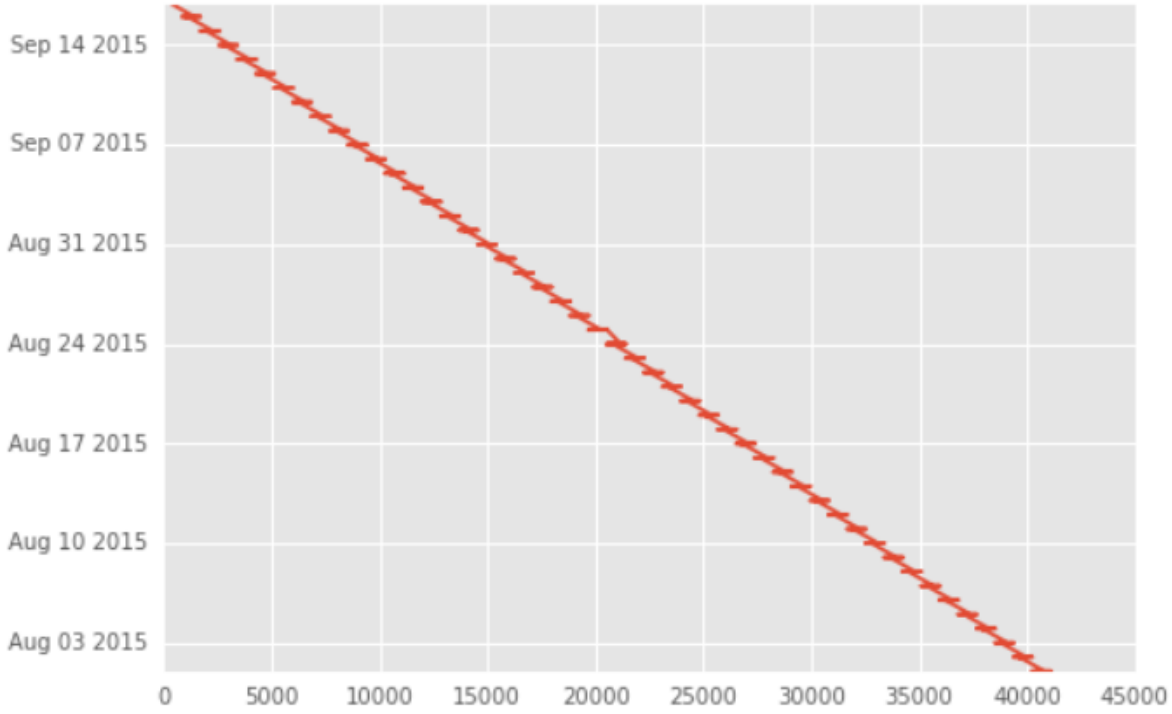


- **Check if there are any gaps in test time-period?**

```
plt.plot(test.Date)
```

```
plt.show()
```

There are no gaps in test time-period

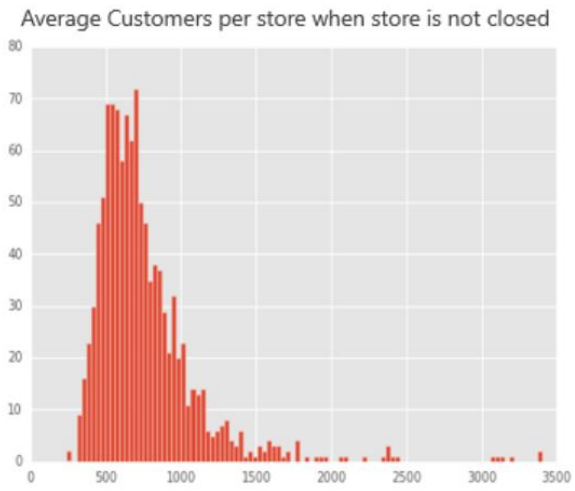


- **Average customers per store when store is not closed**

```
customers = train[train.Sales != 0]

avgcustomers = customers[['Store','Customers']].groupby('Store').mean()

plt.hist(avgcustomers.Customers, 100)

plt.show()
```

Average Sales per store when store is not closed

```
sales = train[train.Sales != 0]

meansales = sales[['Store','Sales']].groupby('Store').mean()

plt.hist(meansales.Sales, 100)

plt.show()
```

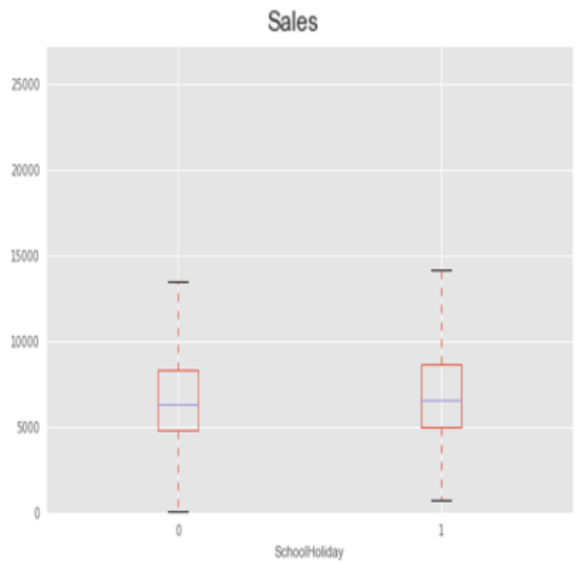


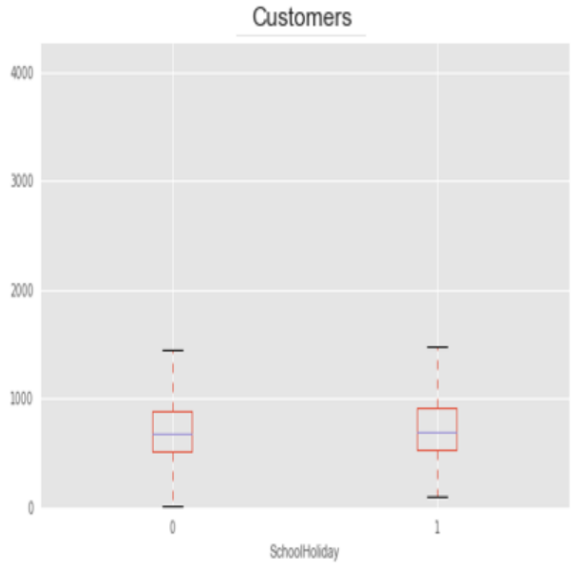
Distributions of Average Sales per store and Average Customers per Store are close to normal with few outliers. We can try creating additional features related to Average Sales per day per store, Average Customers per day per store, Average Sales per customer per day per store later in the feature engineering part.

- **Is School Holiday affecting Sales? Are we seeing more customers on a School Holiday?**

```
df = train[train.Sales != 0]

df = df[['SchoolHoliday', 'Sales']].sort_index()

fig, ax = plt.subplots(figsize=(10,  10))

df.boxplot('Sales','SchoolHoliday',ax)


df1 = df[['SchoolHoliday', 'Customers']].sort_index()

fig, ax = plt.subplots(figsize=(10,  10))

df.boxplot('Customers','SchoolHoliday',ax)
```

Yes, there is a slight increase in the Sales and the number of customers that visited stores on a school holiday.



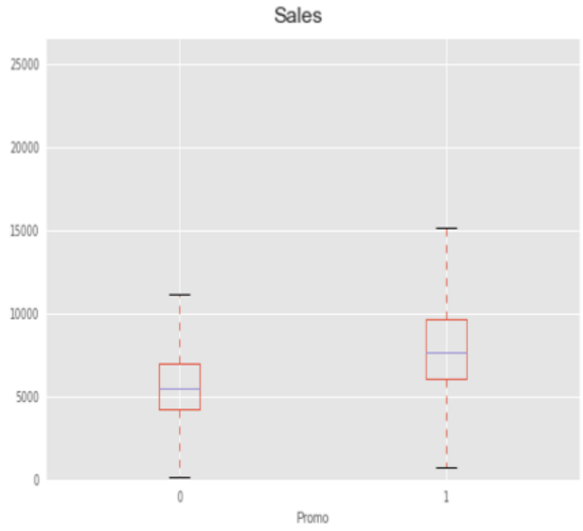


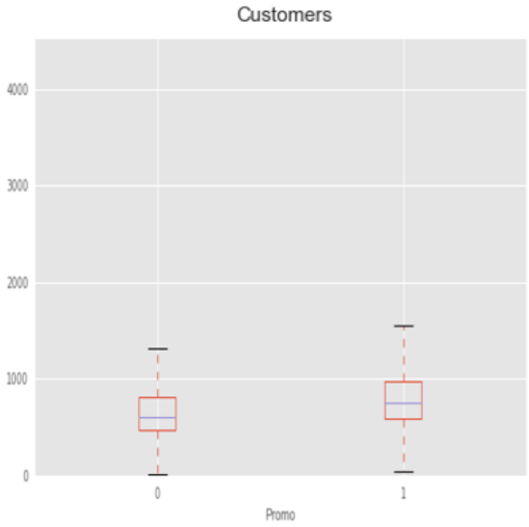
- **Are Promotions affecting Sales? Are we seeing more customers to stores when there are promotions?**

```
df = train[train.Sales != 0]

df = df[['Promo', 'Sales']].sort_index()

fig, ax = plt.subplots(figsize=(10,  10))

df.boxplot('Sales','Promo',ax)


df = train[train.Sales != 0]

df = df[['Promo', 'Customers']].sort_index()

fig, ax = plt.subplots(figsize=(10,  10))

df.boxplot('Customers','Promo',ax)
```

Yes, promotions has a huge impact on the sales. The key insight is we don't see the same spike in customers though. It might mean, the same customers are buying more than normal when there are

promotions.

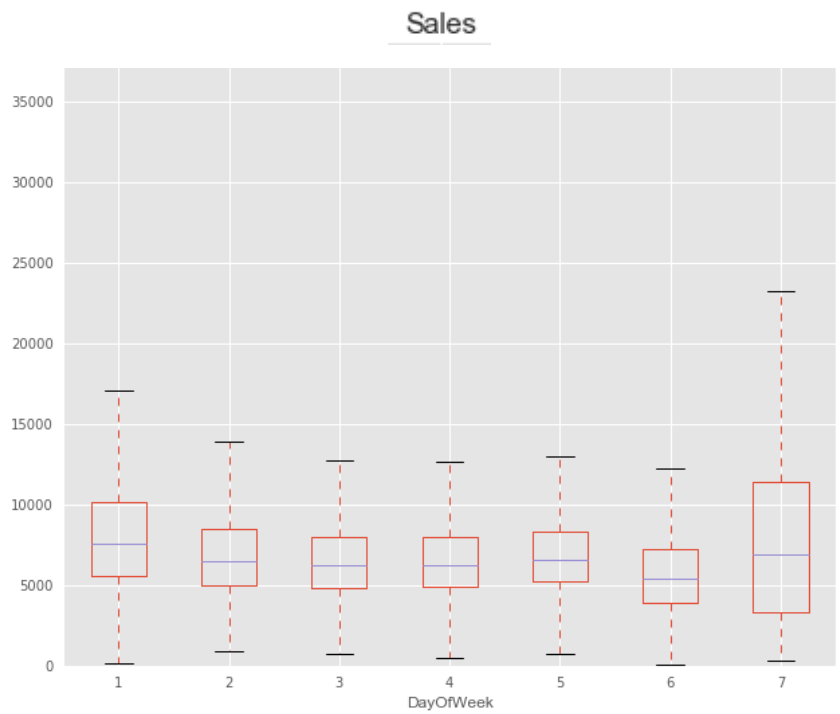




- **Are Sales high in any specific day of week?**

  Sunday (day 7) has more sales than the rest of the days. DayofWeek could be an important feature for the model.

```
df = train[train.Sales != 0]

df = df[['DayOfWeek', 'Sales']].sort_index()

fig, ax = plt.subplots(figsize=(10,  10))

df.boxplot('Sales','DayOfWeek',ax)
```

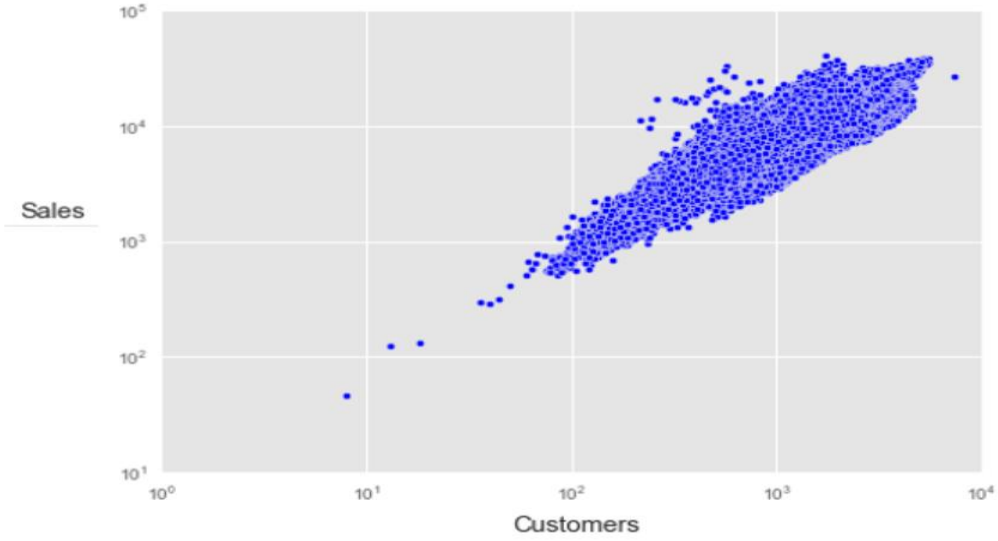


Time series based features such as Day, Day of Week, Day of Year, Month Number will be important features given the sales are different on each day.

- **What is the relationship between Sales and Customers?** The key assumption is more number of customers ~ more number of Sales

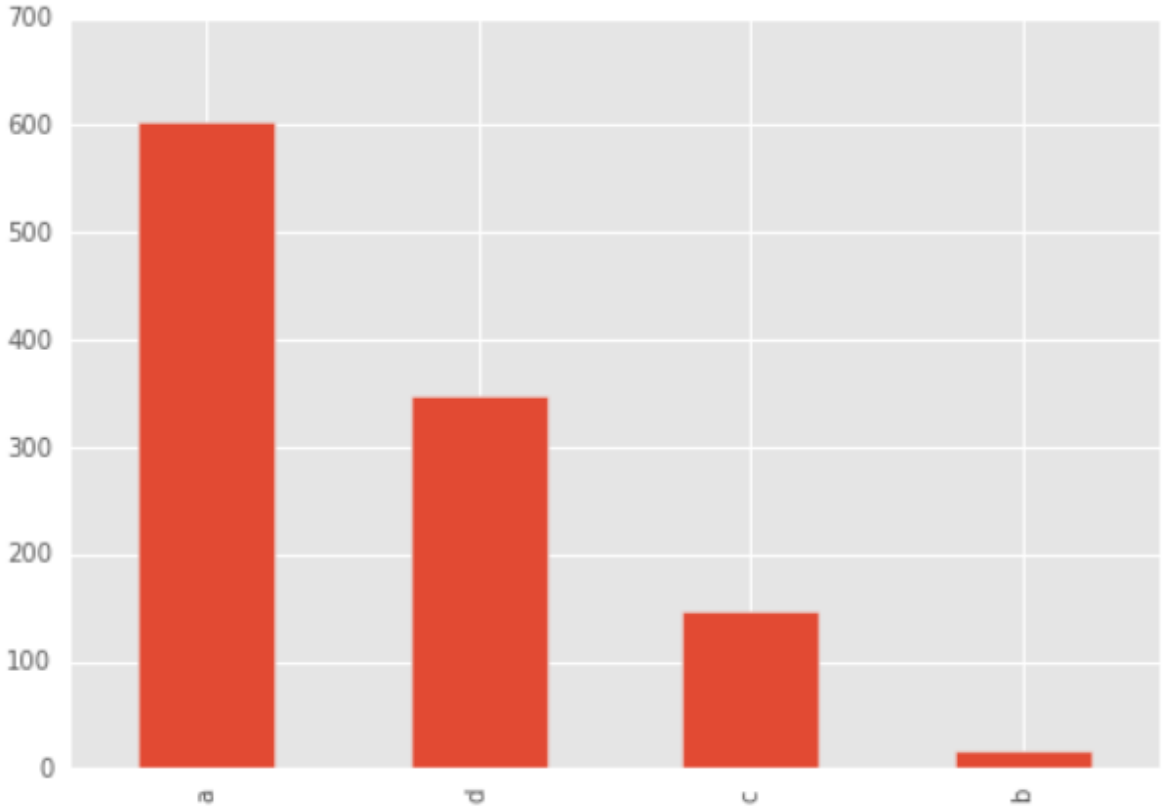
```
df = train[train.Sales != 0]

df = df[['Customers', 'Sales']].sort_index()

plt.scatter(df.Customers, df.Sales)

plt.xscale('log')

plt.yscale('log')

plt.show()
```

Yes, we can see sort of linear relationship between Sales and Customers



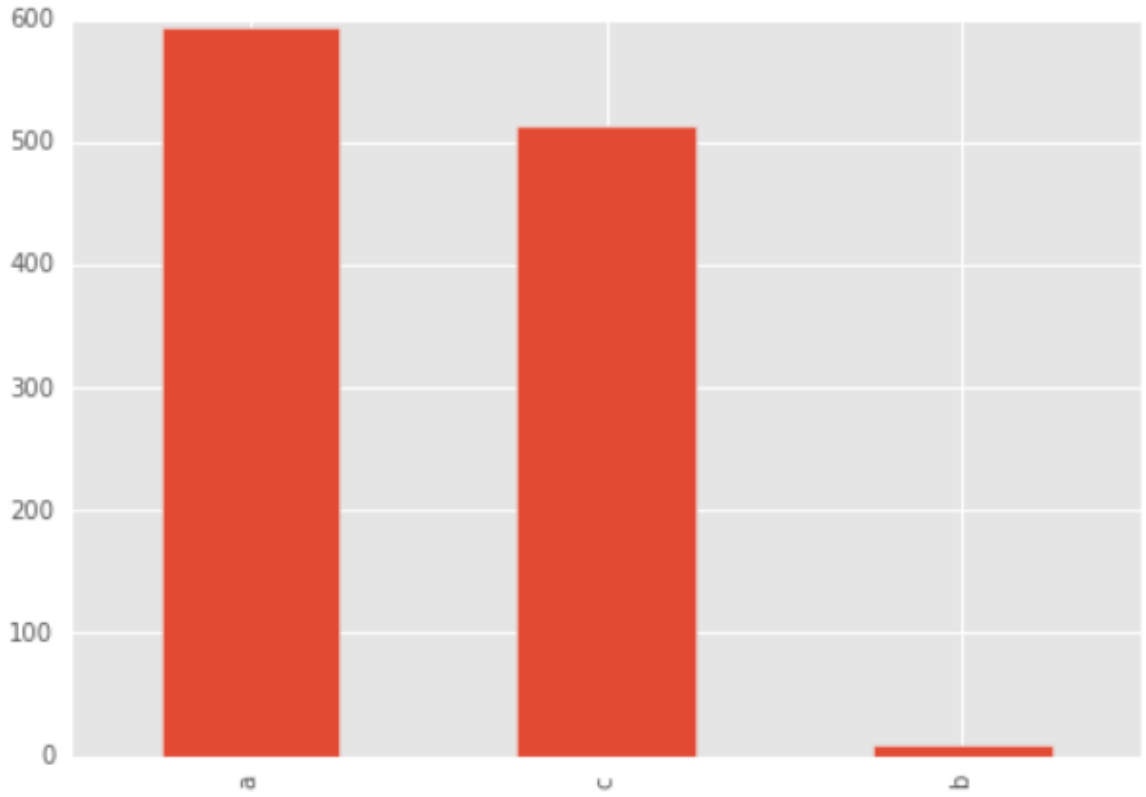
- **How many Store Types are in Store data?**

```
store.StoreType.value_counts().plot(kind = 'bar')
```



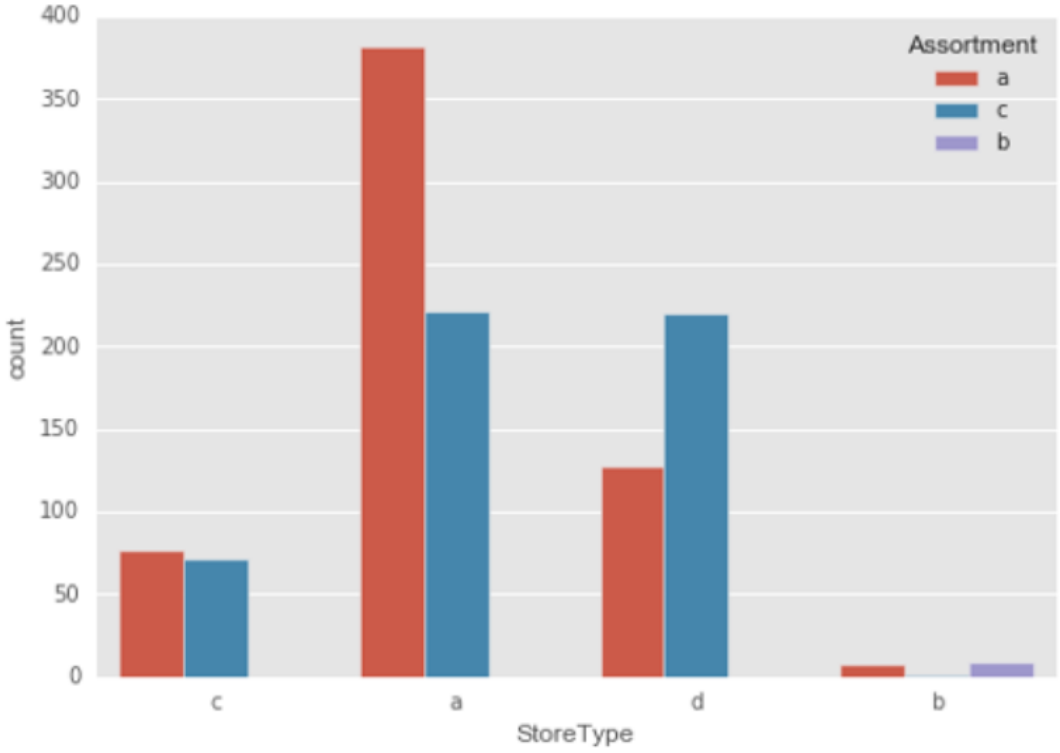
- **How many different Assortments in Store Data?**

```
store.Assortment.value_counts().plot(kind = 'bar')
```



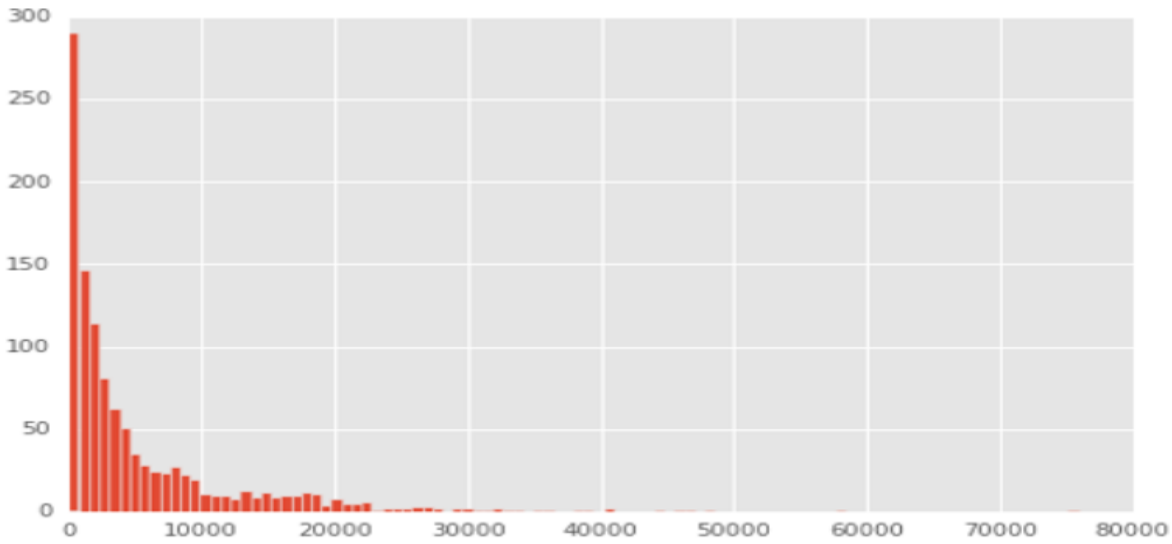
- **Relationship between Store Type and Assortment Type categories?**

```
sns.countplot(x="StoreType", hue="Assortment", data=store)
```



- **Competition Distance:** Univariate Analysis

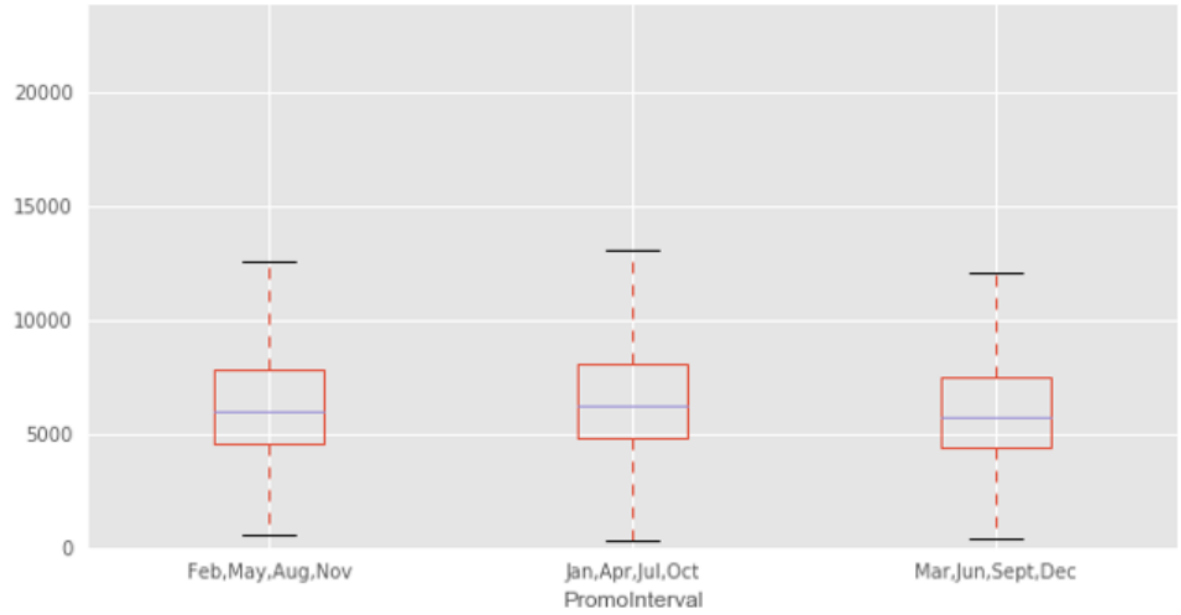
  We might need to do transformation of this feature due to the skewed distribution. Log transform would be the best approach.



- **Sales bucketized by promotion intervals. Is there any specific promotion interval dominant?**

```
fulltrain = pd.merge(train, store, on='Store')

df = fulltrain[fulltrain.Sales != 0]

df = df[['PromoInterval', 'Sales']].sort_index()

fig, ax = plt.subplots(figsize=(10,  10))

df.boxplot('Sales','PromoInterval',ax)
```

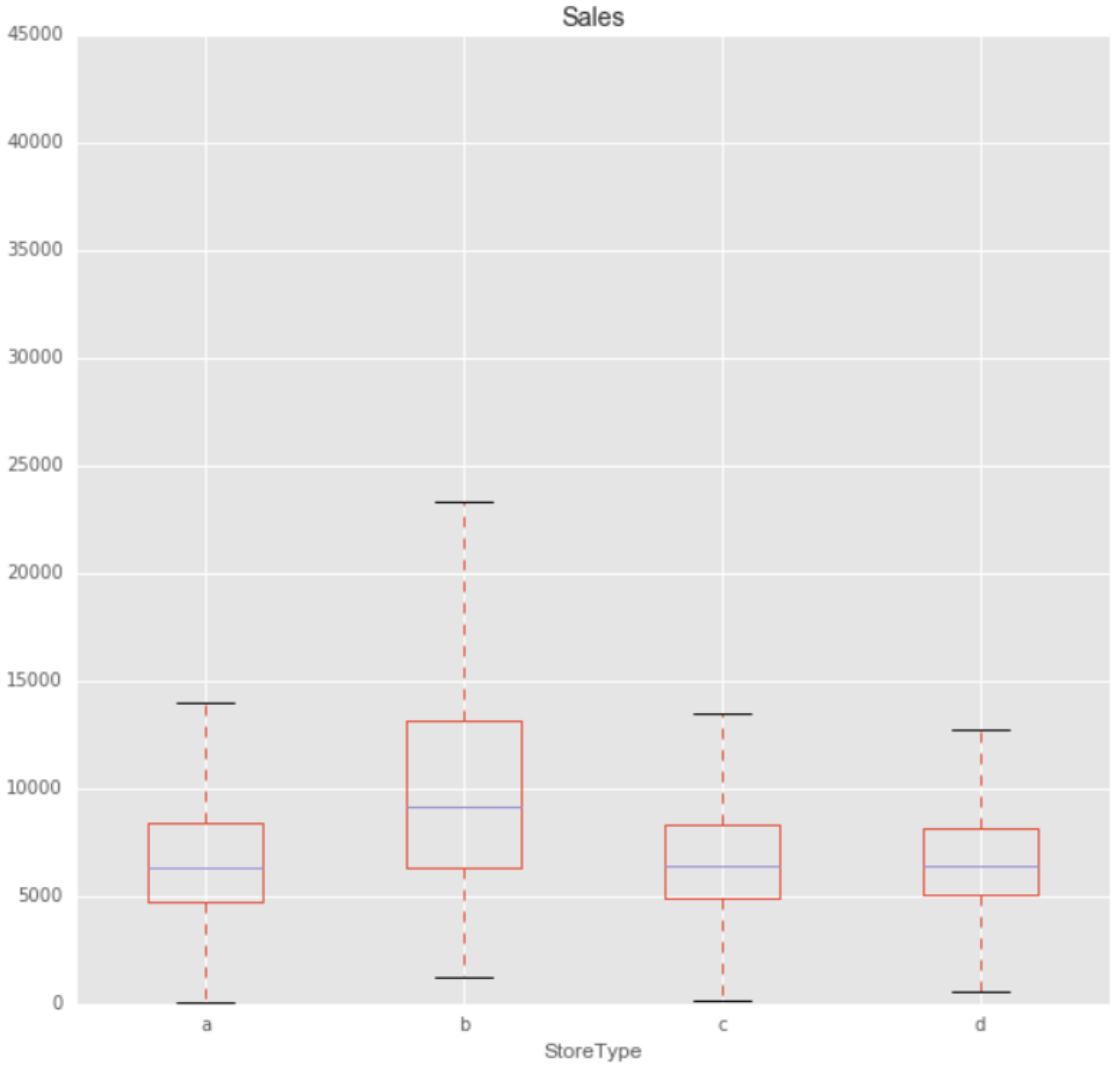
There is a slight difference in the number of sales for each promotion intervals but there is no dominance.



- **Are there higher sales for a specific Store Type?**

```
df = fulltrain[fulltrain.Sales != 0]

df = df[['StoreType', 'Sales']].sort_index()

fig, ax = plt.subplots(figsize=(10,  10))

df.boxplot('Sales','StoreType',ax)
```

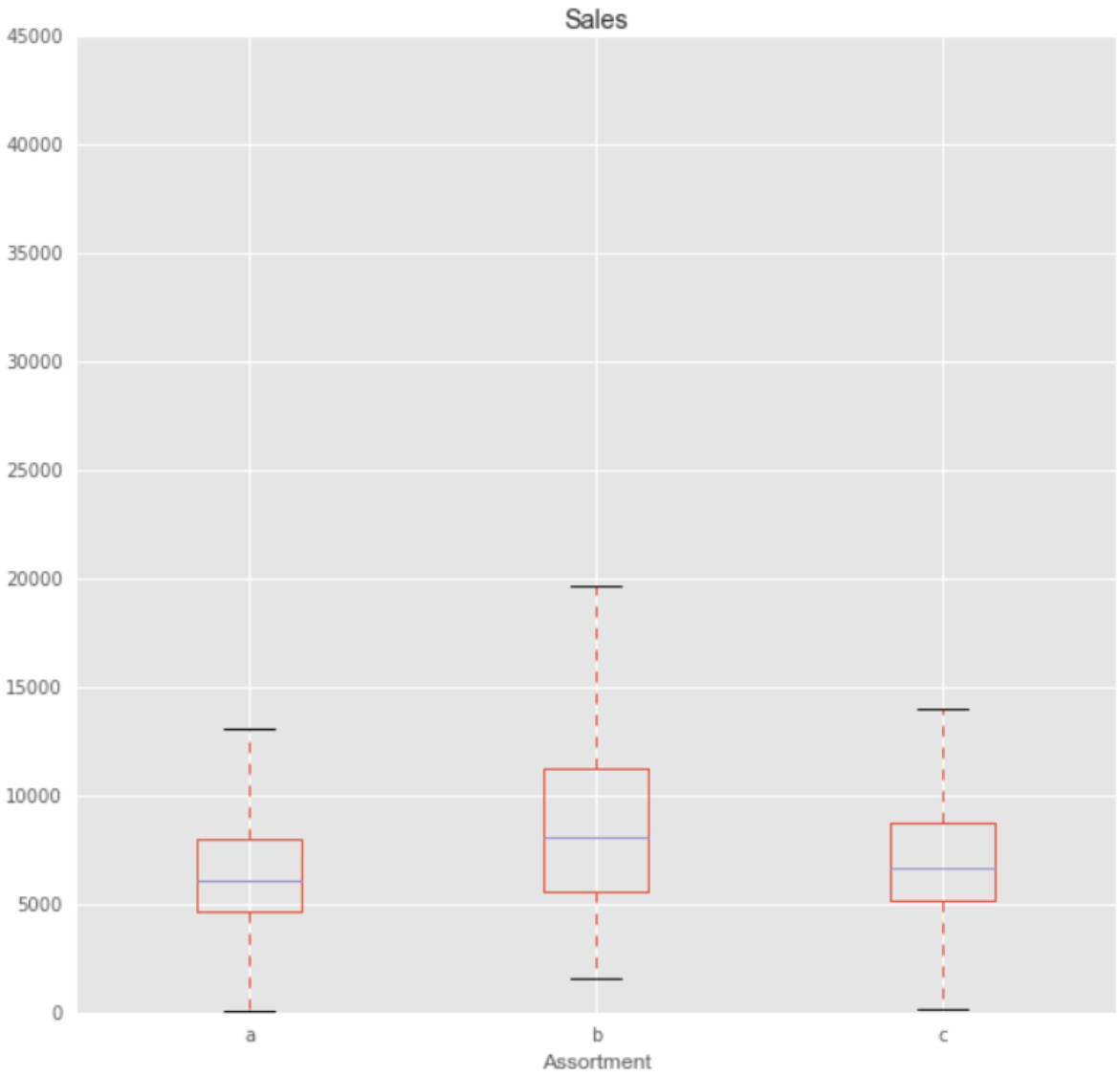
Store Type (b) has higher sales compared to other store types.



- **Are there higher sales for a specific Store Assortment Type?**

```
df = fulltrain[fulltrain.Sales != 0]

df = df[['Assortment', 'Sales']].sort_index()

fig, ax = plt.subplots(figsize=(10,  10))

df.boxplot('Sales','Assortment',ax)
```

Assortment Type (b) has higher sales compared to other Assortment types.



Data Cleansing/Pre-Processing

- Combined the train and test datasets for pre-processing. The same pre-processing needs to happen on train and test datasets. Created a feature with name “Set” that differentiates train vs test data in combined dataset (df)
- Transformed the Sales label column into to log(sales). This helps in the modelling part to not sensitive to outliers

  ```
  df.loc[df['Set'] == 1, 'SalesLog'] = np.log1p(df.loc[df['Set'] == 1]['Sales'])
  ```

- Label encoding the State Holiday (sales data), Store Type and Assortment features (store data)

```
df['StateHoliday'] = df['StateHoliday'].astype('category').cat.codes

store['StoreType'] = store['StoreType'].astype('category').cat.codes

store['Assortment'] = store['Assortment'].astype('category').cat.codes
```

- Removed rows where store is open but no sales

  df = df.loc[~((df['Open'] == 1) & (df['Sales'] == 0))]

Feature Engineering

Created the following Time Series based Features from Date Column (datetime object)

- Day
- Week
- Month
- Year
- DayofYear

```
df['Day'] = pd.Index(df['Date']).day

df['Week'] = pd.Index(df['Date']).week

df['Month'] = pd.Index(df['Date']).month

df['Year'] = pd.Index(df['Date']).year

df['DayOfYear'] = pd.Index(df['Date']).dayofyear
```

Created Derived columns – Individual columns do not make sense. Hence combining them for interpretability

- CompetitionOpenSince - Appended CompetitionOpenSInceYear + CompetitionOpenSinceMonth

```
def convertCompetitionOpen(df):

    try:
```

```
        date = '{}-{}'.format(int(df['CompetitionOpenSinceYear']), int(df['Competition
OpenSinceMonth']))
        return pd.to_datetime(date)
    except:
        return np.nan

store['CompetitionOpenSince'] = store.apply(lambda df: convertCompetitionOpen(df), axi
s=1).astype(np.int64)
```

- Promo2Since – Appended Promo2SinceYear + Promo2SinceWeek columns

```
def convertPromo2(df):
    try:
        date = '{}{}1'.format(int(df['Promo2SinceYear']), int(df['Promo2SinceWeek']
))
        return pd.to_datetime(date, format='%Y%W%w')
    except:
        return np.nan
store['Promo2Since'] = store.apply(lambda df: convertPromo2(df), axis=1).astype(np.
int64)
```

- PromoInterval ~ Created 4 features – one for each promointerval

```
s = store['PromoInterval'].str.split(',').apply(pd.Series, 1)
s.columns = ['PromoInterval0', 'PromoInterval1', 'PromoInterval2', 'PromoInterval3']
store = store.join(s)
```

Store based features (at store level per day) - This helps us to use customers feature as part of the model. These metrics are store specific and might help for predicting store sales

- SalesPerDay – Average number of sales per day at store level
- CustomersPerDay – Average number of customer per day at store level

- SalesPerCustomersPerDay - Average sales per customer per day

Modelling Approach

- Divided the train dataset into two sets (90% train and 10% validation set)
- Played with multiple algorithms (randomForest, Decision Trees, XGBoost) and ended up with XGBoost because of the following reasons:
  - Designed for speed and performance.
  - Automatic handling of missing values - It is sparse aware
  - Parallel Processing: It has block structure (to support the parallelization of tree construction)

    Reference: http://zhanpengfang.github.io/418home.html
  - Built-In Cross Validation:

    XGBoost allows user to run a **cross-validation at each iteration** of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.
  - High Flexibility: It has the capability to optimize for a custom evaluation metric
  - Continue Existing model:

    User can start training an XGBoost model from its last iteration of previous run. This can be of significant advantage in certain specific applications.

If you have validation set, you can use early stopping to find the optimal number of boosting rounds.

The model will train until the validation score stops improving. Validation error needs to decrease at least every early_stopping_rounds to continue training.

num_boost_round = 5000, early_stopping_rounds=200

Used the framework mentioned in the following blog for hyperparameter tuning

https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

params = {"objective": "reg:linear", # Default option – Linear Regression

"booster" : "gbtree",

```
# this is default option – gbtree uses tree based model)
"eta": 0.1,
"max_depth": 10,
"subsample": 0.85,
"colsample_bytree": 0.4,
"min_child_weight": 6,
"silent": 1,
"thread": 4,
"seed": seed
}
```

Hyperparameters explanation

https://github.com/dmlc/xgboost/blob/master/doc/parameter.md

Predictions

Best Iteration for the model on 90% of trainset ~ 2110 (number of boosting rounds)

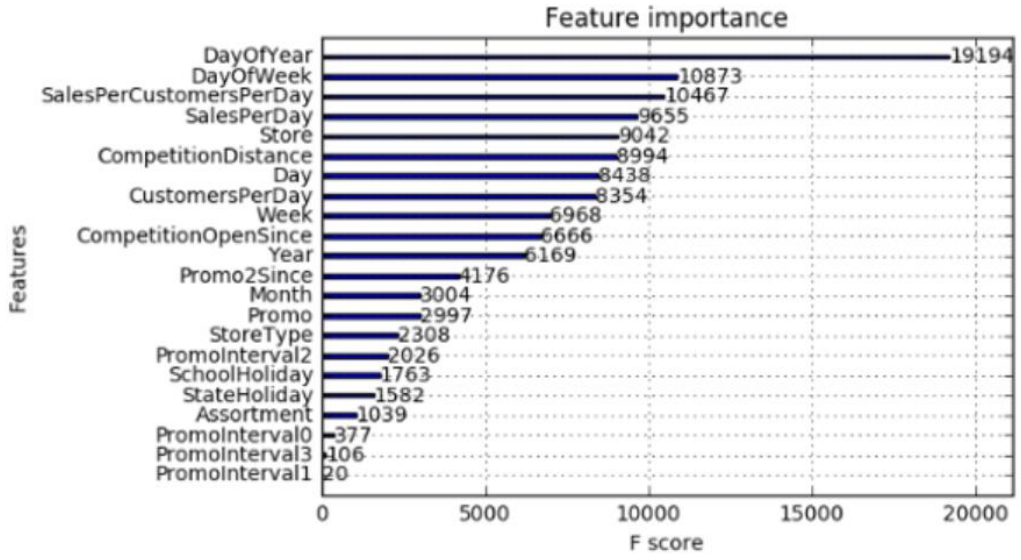
- Trainset performance (90% of train set) : RMSPE: 0.091063
- Test Set Performance (10% of train set) : RMSPE: 0.106646
- Finally ran the model on the blind test set Rossmann provided and submitted the results in Kaggle and RMSPE is ~ 0.11996
- Close to ~560 Rank on Private leaderboard

(top 20% in Kaggle for this result)

Feature Importance

- DayofYear is the most important feature of the model and has a higher relative importance than other features and this is a derived feature.
- Most of the top features that are important for sales predictions at store level are derived features

Some of the Promotional Intervals did not come in the top feature list (PromoInterval1 and PromoInterval3)



Future Work

- Do a Grid Search of Parameters and Feature Selection - It is very computational Intensive, but we can do it once to get the best parameters and features combination to get the model with lowest RMSPE.
- Check if there are any outliers in each store and check the methodologies to correct and test the above approach
- So far, the algorithms tried are Decision tree (for interpretability), randomForest and XGBoost. XGBoost is fast as well as it gave best results. I would like to try neural networks as well to see how it performs on this dataset.
- Solve this problem using time series approach instead of machine learning approach as we have enough data to capture trend and seasonality.

How this model can be implemented in production?

Ideal way to approach this is *(need to check with business to see if we need to include any of their constraints in implementing the below approach)*

- Check with business on how frequently do we need to refresh this model from production standpoint.
- Develop an end to end pipeline that takes the consolidated sales data from all 1,115 stores and do the data pre-processing, feature engineering and then train the model (using the cross validation approach) and output the predictions based on the refresh frequency
- The pipeline should enable a continuous integration of new data (every day/week) and help forecast to be as accurate as possible as and when the model is trained including new data.
- A report should be send to each store manager for his specific store forecasts for next 6 weeks