Retail Sales Time-series Forecasting

Project By: GANESH KASTURI

Introduction:

In the increasingly competitive fashion retail industry, companies are constantly adopting strategies focused on adjusting the products characteristics to closely satisfy customers' requirements and preferences. Although the lifecycles of fashion products are very short, the definition of inventory and purchasing strategies can be supported by the large amounts of historical data which are collected and stored in companies' databases. This study explores the use of a deep learning approach to forecast sales in fashion industry, predicting the sales of new individual products in future seasons. This study aims to support a fashion retail company in its purchasing operations and consequently the dataset under analysis is a real dataset provided by this company.

Data Preparation:

In order to make sure that the experiments are conducted in a consistent way, before applying any method, it is necessary to prepare the data. Formally data preparation involves data discretization, data cleaning, data integration, data transformation and data reduction.

- ### Data Gathering:

  Data Set 1: 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.

  Data Set 2: Walmart runs several promotional markdown events throughout the year. These markdowns precede prominent holidays, the four largest of which are the Super Bowl, Labour Day, Thanksgiving, and Christmas. The weeks including these holidays are weighted five times higher in the evaluation than non-holiday weeks. Part of the challenge presented by this competition is modelling the effects of markdowns on these holiday weeks in the absence of complete/ideal historical data.

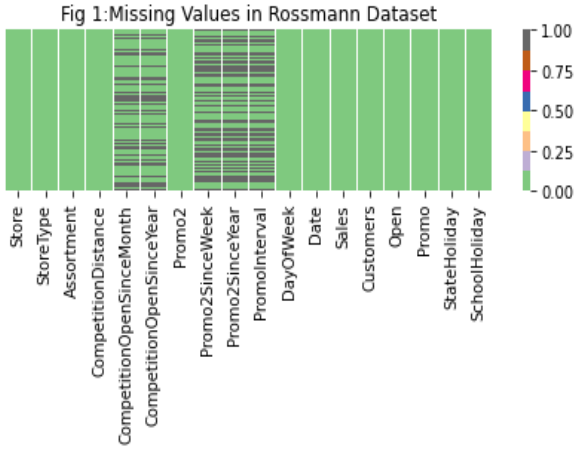
  This file contains anonymized information about the 45 stores, indicating the type and size of store.

Missing Data Handling:

Missing data is a problem because it adds ambiguity to your analysis. Consider the data in the table above. What if you want to find out the average number of employees? With the second observation missing a data value, it would be impossible to accurately work it out. You could work around this by computing the average based on the available data, but your results will always be flawed. Furthermore, an observation that has missing data for a variable indicates that it is atypical. Therefore, any analysis that assumes the missing value fits neatly into the rest of the data, is unsound.

You have three options when dealing with missing data. The most obvious and by far the easiest option, is to simply ignore any observations that have missing values. This is often called *complete case* analysis or *listwise deletion* of missing values.
Another approach is to *impute* the missing values. This involves using statistical or machine learning models to make educated guesses based about the values of the missing data. For example, you could create a model that predicts the number of employees based on the other variables and then use this model to predict the number of employees. A variant of this approach, known as *multiple imputation,* is usually considered best practice when building regression-type models (e.g., linear regression, logistic regression).

- From the Dataset we have came up that there a lot of missing values present in the dataset.



- **Fill NaN values**

  Filling up 'Promo2SinceWeek','Promo2SinceYear','PromoInterval' columns

```
[ ]  df['Promo2SinceWeek'].fillna('0',inplace=True)
     df['Promo2SinceYear'].fillna('0',inplace=True)
     df['PromoInterval'].fillna('0',inplace=True)

     print(df.isnull().sum())
```

```
Store                         0
StoreType                     0
Assortment                    0
CompetitionDistance        2642
CompetitionOpenSinceMonth  323348
CompetitionOpenSinceYear   323348
Promo2                        0
Promo2SinceWeek               0
Promo2SinceYear               0
PromoInterval                 0
DayOfWeek                     0
Date                          0
Sales                         0
Customers                     0
Open                          0
Promo                         0
StateHoliday                  0
SchoolHoliday                 0
dtype: int64
```

- Filling up 'CompetitionOpenSinceMonth','CompetitionOpenSinceYear'

```
[ ]  imp_mean = SimpleImputer( strategy='median')
     imp_mean.fit(df[['CompetitionOpenSinceYear']])
     df[['CompetitionOpenSinceYear']] = imp_mean.transform(df[['CompetitionOpenSinceYear']])
```

```
[ ]  print(df.isnull().sum())
```

```
Store                      0
StoreType                  0
Assortment                 0
CompetitionDistance        0
CompetitionOpenSinceMonth  0
CompetitionOpenSinceYear   0
Promo2                     0
Promo2SinceWeek            0
Promo2SinceYear            0
PromoInterval              0
DayOfWeek                  0
Date                       0
Sales                      0
Customers                  0
Open                       0
Promo                      0
StateHoliday               0
SchoolHoliday              0
dtype: int64
```

Feature engineering

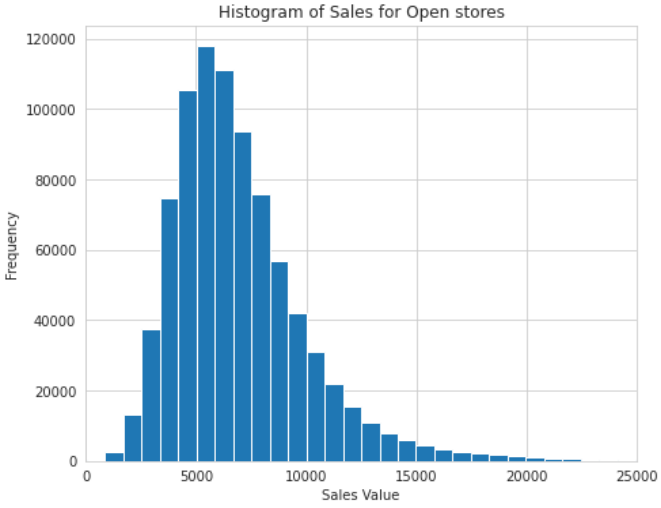
Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data. Feature engineering turn your inputs into things the algorithm can understand.

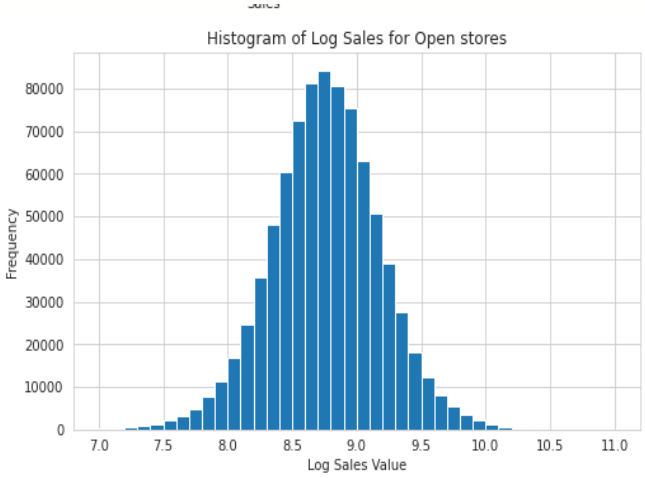
- We have used **One hot encoding.**
- **Encoding for Binary data.**
  Binary encoding is a combination of Hash encoding and one-hot encoding. In this encoding scheme, the categorical feature is first converted into numerical using an ordinal encoder. Then the numbers are transformed in the binary number. After that binary value is split into different columns.
- **Encoding for nominal data**
  Machine learning models require all input and output variables to be numeric. This means that if your data contains categorical data, you must encode it to numbers before you can fit and evaluate a model. The two most popular techniques are an Ordinal Encoding and a One-Hot Encoding.
- **Convert/Manage Date column**
  Dealing with Date and parse each element of a date is crucial in order to analyses event.
  First things with we need to convert our Date column (often considered as a string column with pandas). One of the most important field is to know how to use the format parameters.

Exploratory Data analysis

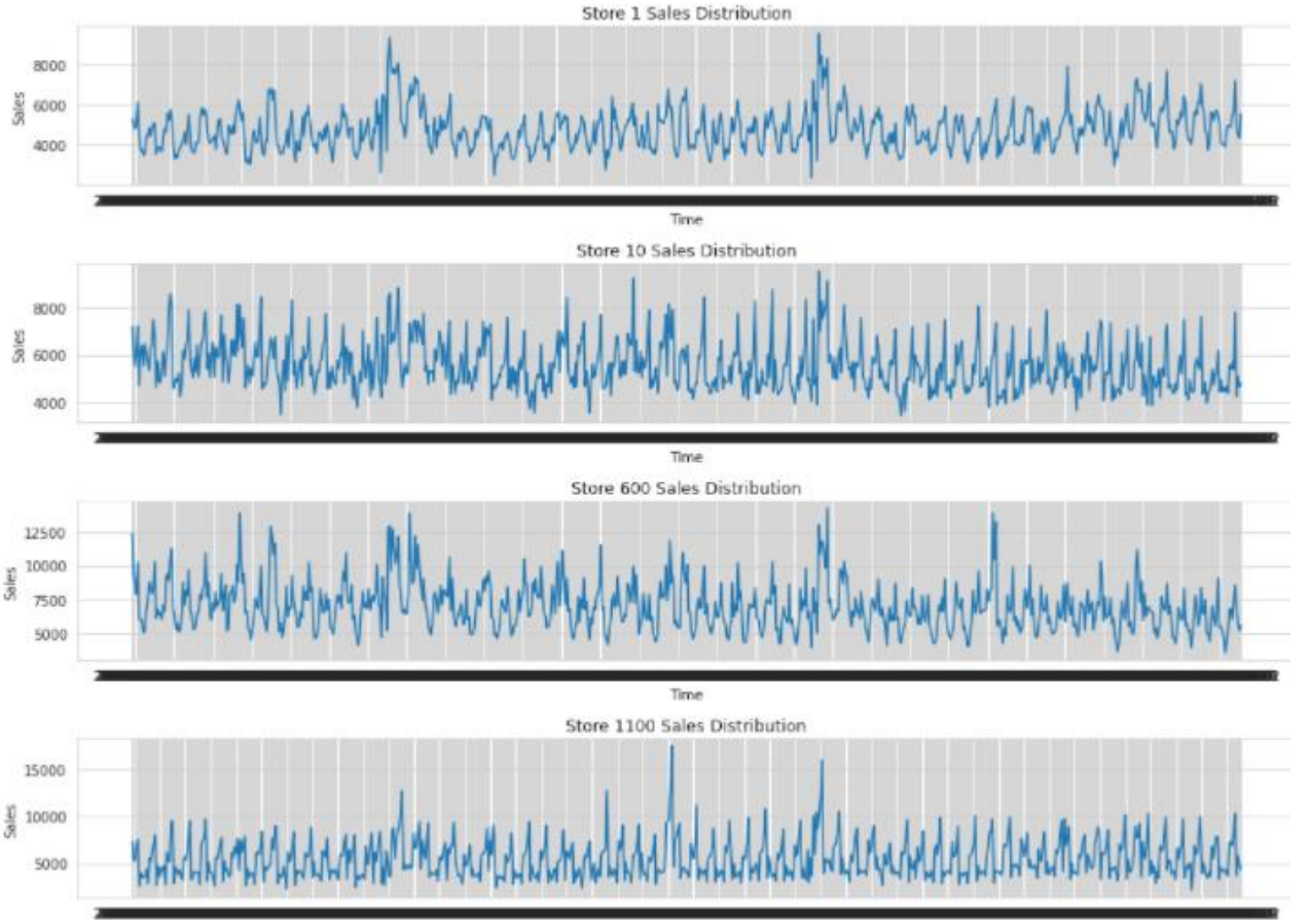
Beyond focusing on a single variable, it is quite desirable to analyse whole population, in which statistical variables are jointly considered. To this end, exploratory data analysis serves approaches to cover these information. Essentially underlying the exploratory data analysis is possible to measure the dependency between statistical variables, which makes the analysis more understandable. The key difference between the descriptive and the exploratory analysis is actually the way to handle the variables.
In this regard, correlation analysis, principal component analysis (PCA), box plot, vector quantization (VQ), scatter plot and matrix factorization are relevant tools in this field.

Sales

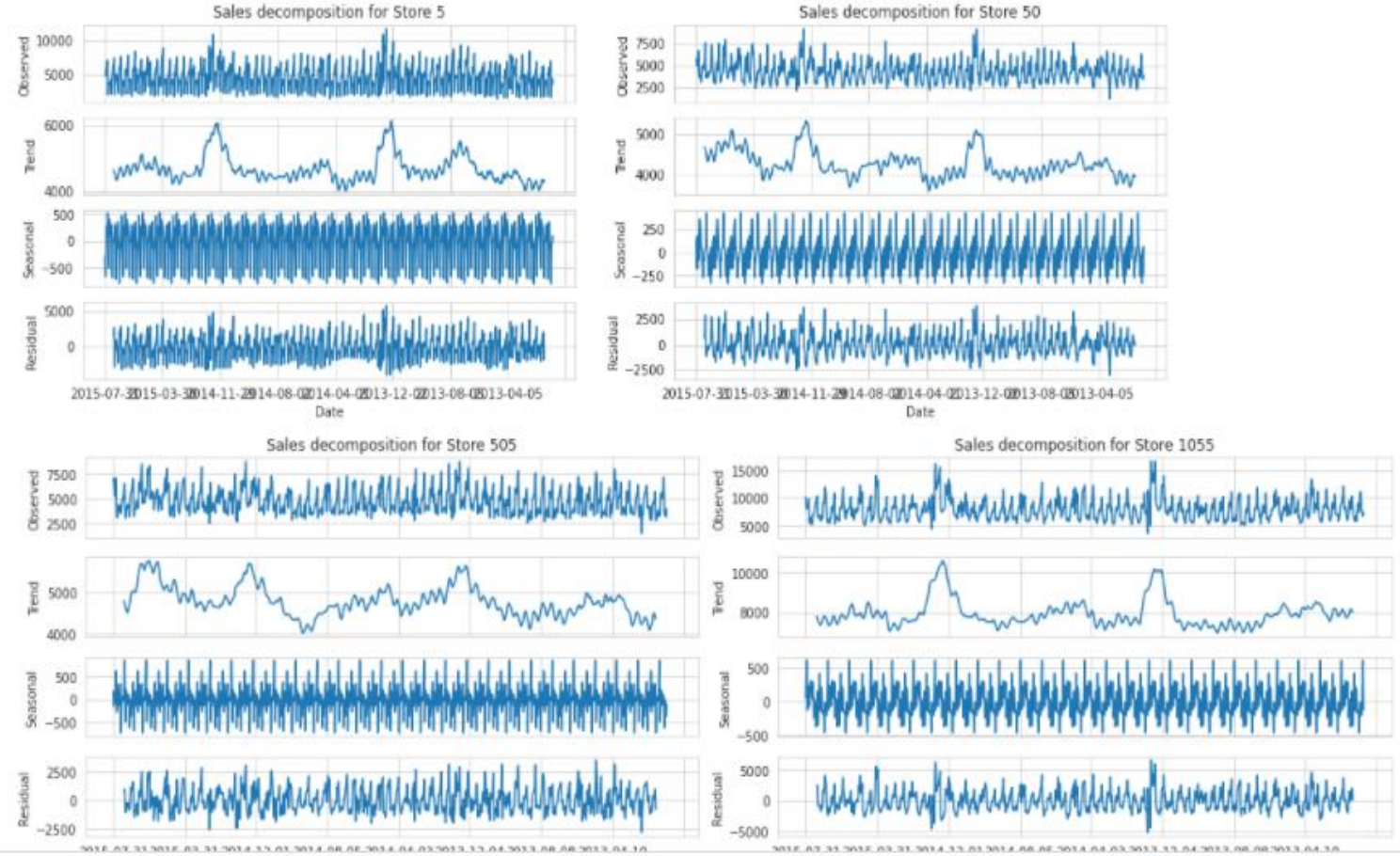




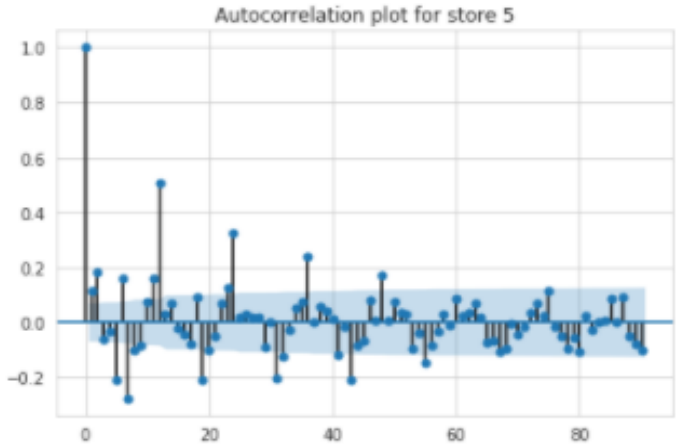
Plotting sales distribution plot for different stores.

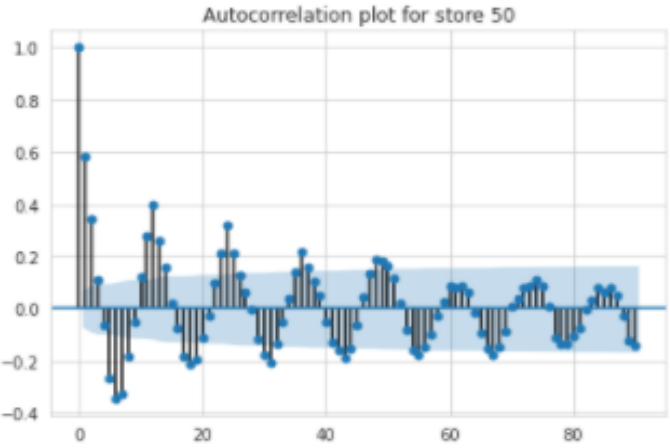


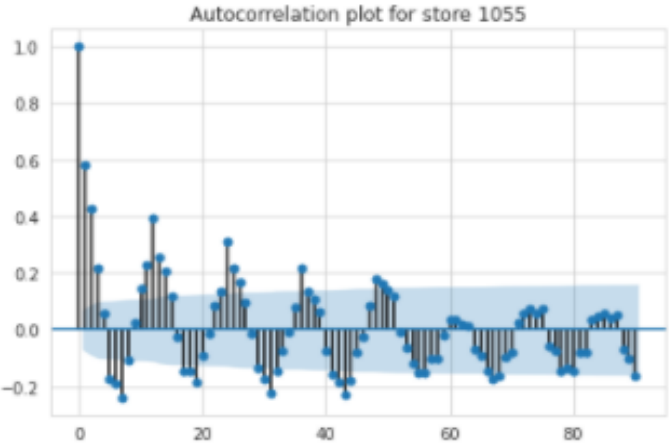
For different stores:



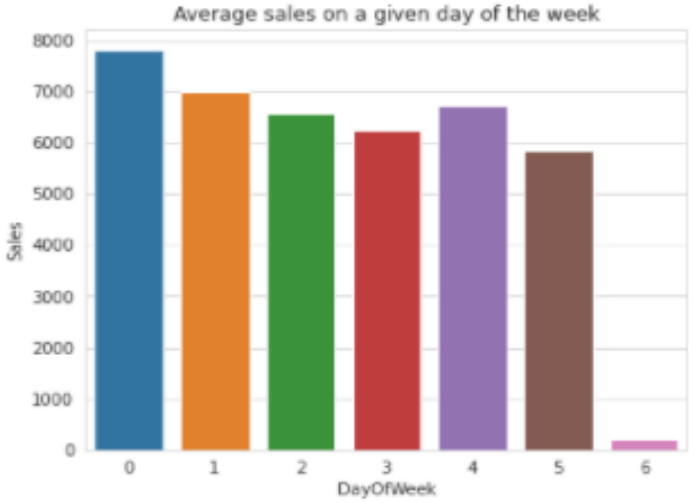
Autocorrelation plot for store

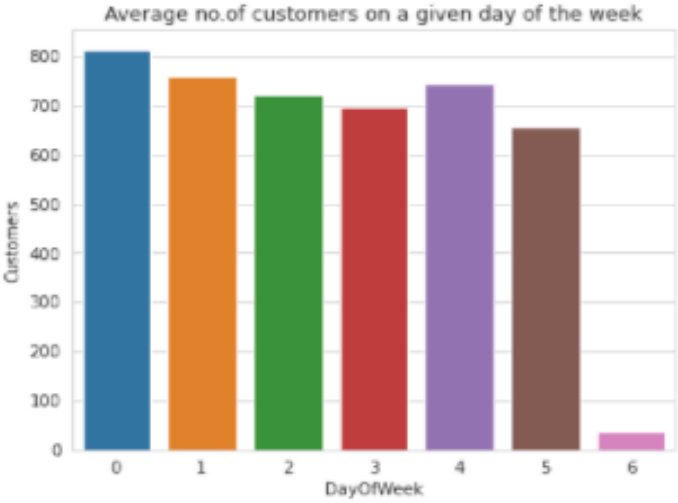




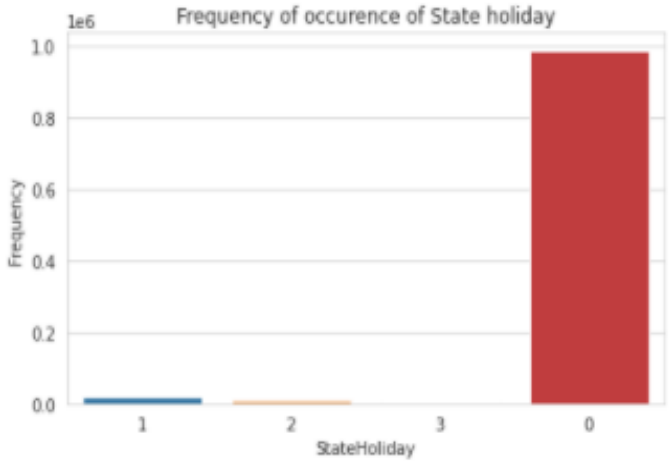


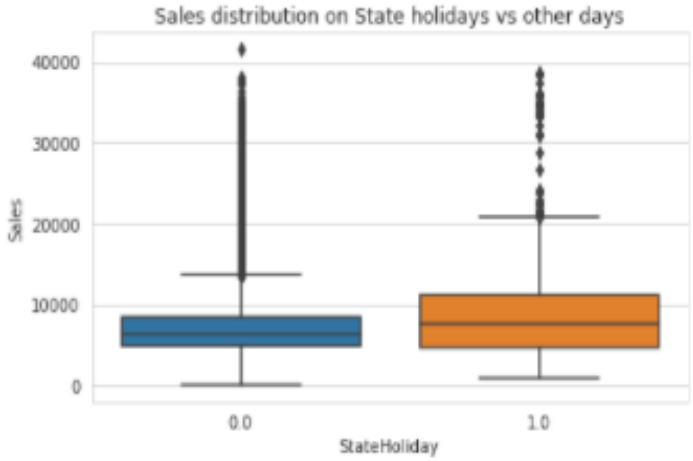
Day of Week:

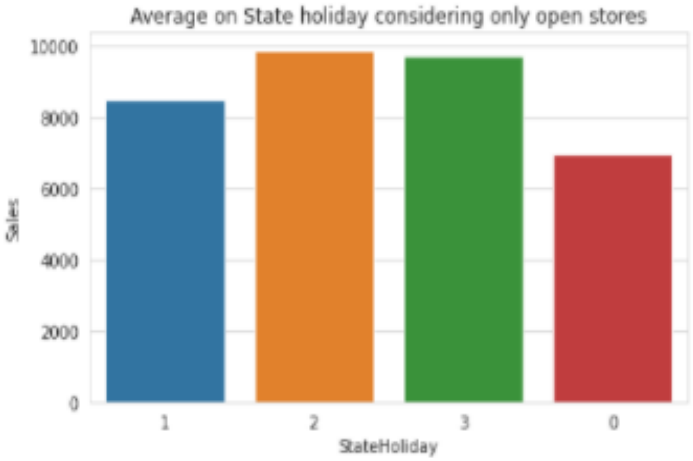




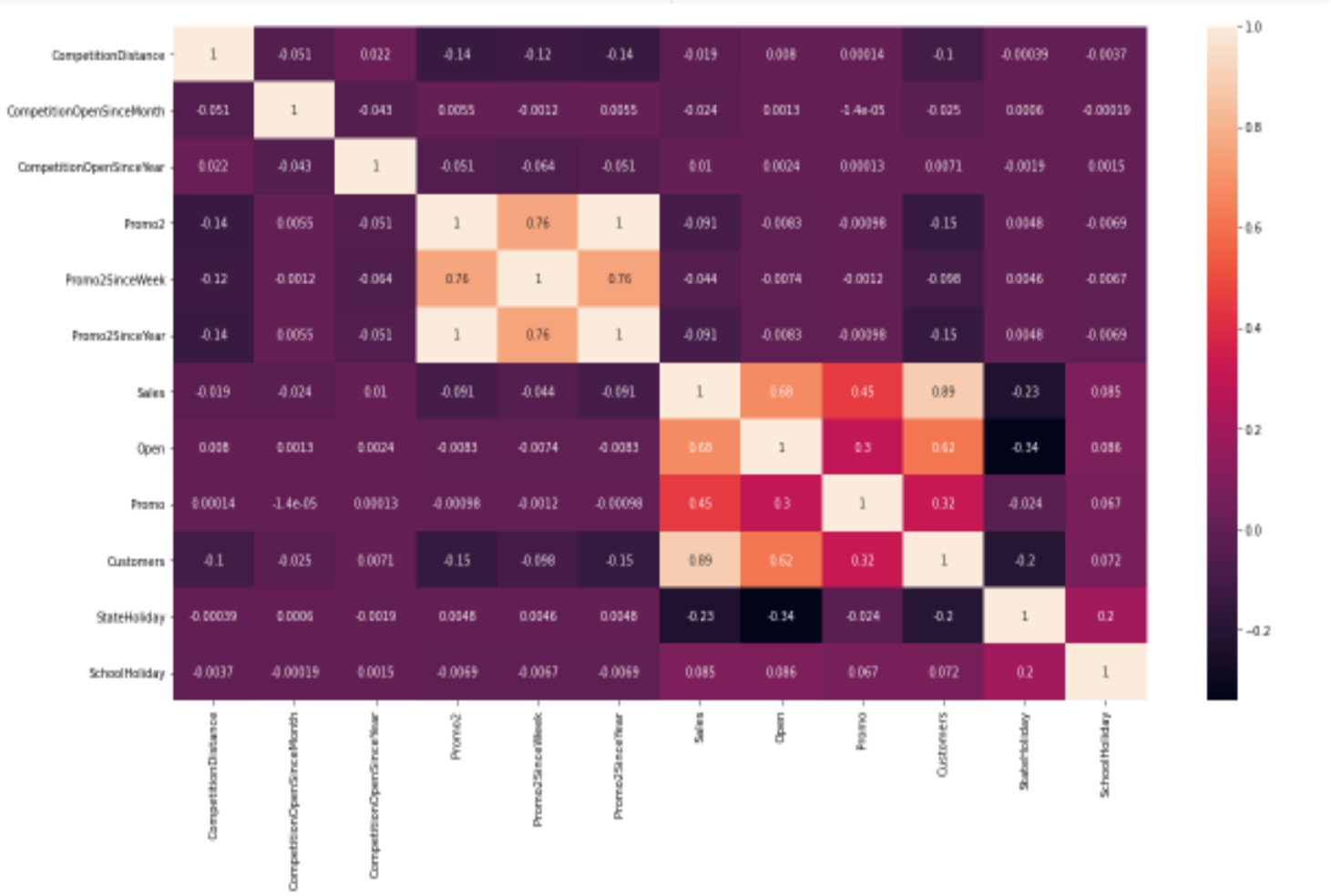
State Holiday







Heatmap of Correlation in numerical values



Model Building(TIME SERIES MODELS)

```
# ARIMA
arima= ARIMA(training_data,order=(1,1,1))
model=arima.fit()
model.summary()
```

Dep. Variable:	D.Sales	**No. Observations:**	910
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-8347.739
Method:	css-mle	**S.D. of innovations**	2323.676
Date:	Thu, 11 Feb 2021	**AIC**	16703.478
Time:	10:53:05	**BIC**	16722.732
Sample:	01-02-2013	**HQIC**	16710.829
	- 06-30-2015		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.3166	0.368	0.859	0.390	-0.406	1.039
ar.L1.D.Sales	0.2061	0.033	6.321	0.000	0.142	0.270
ma.L1.D.Sales	-1.0000	0.003	-355.418	0.000	-1.005	-0.994

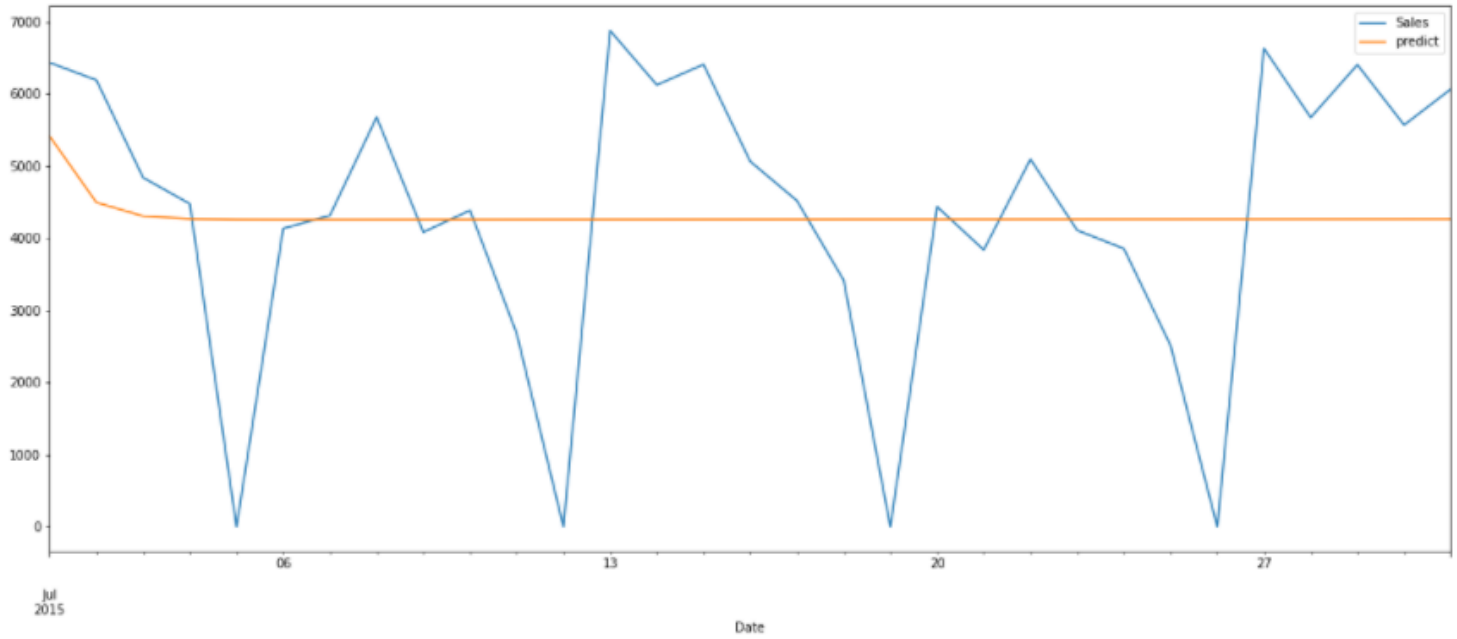
	Real	Imaginary	Modulus	Frequency
AR.1	4.8509	+0.0000j	4.8509	0.0000
MA.1	1.0000	+0.0000j	1.0000	0.0000

```
model.aic
```

```
16703.477774558305
```

```
pred= model.forecast(steps=31)[0]
test_data['predict']=pred
test_data[['Sales','predict']].plot(figsize=(20,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7a94813080>
```



```
model=sm.tsa.statespace.SARIMAX(df['Sales'],order=(1, 1, 1),seasonal_order=(1,1,1,12))
results=model.fit()
results.summary()
```

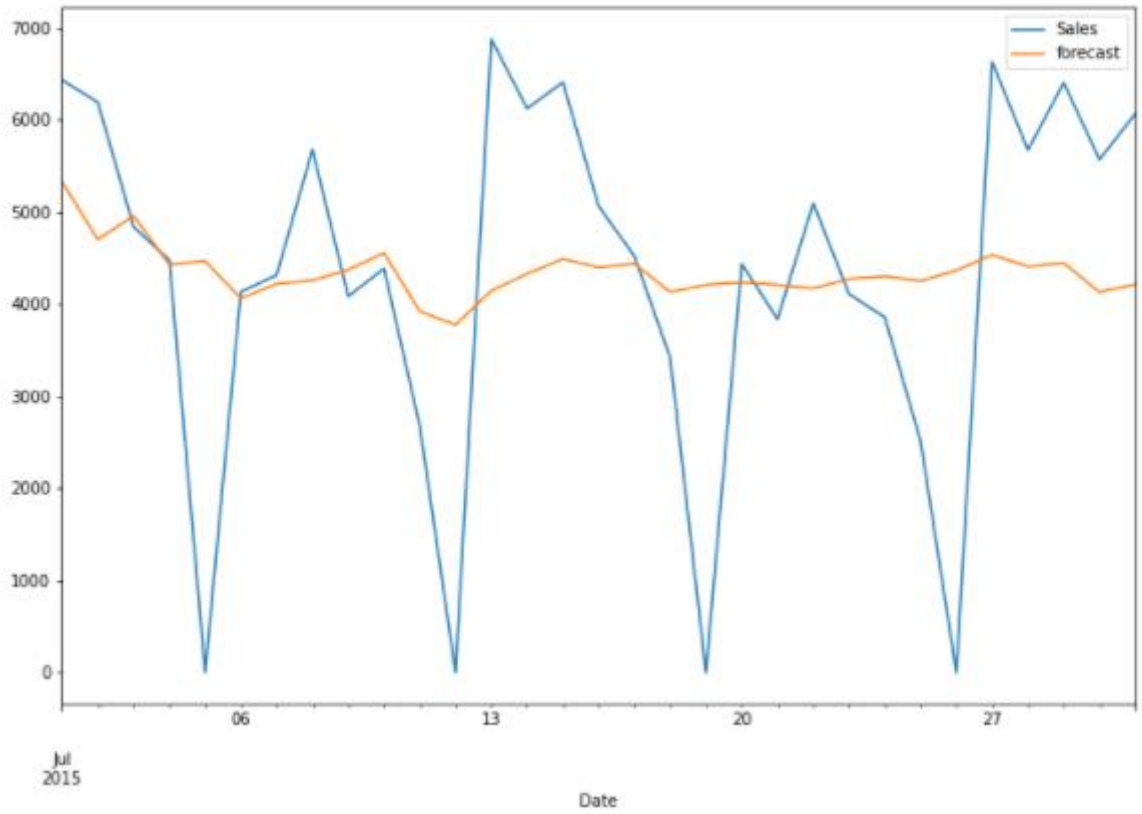
Dep. Variable:	Sales	**No. Observations:**	942
Model:	SARIMAX(1, 1, 1)x(1, 1, 1, 12)	**Log Likelihood**	-8542.386
Date:	Thu, 11 Feb 2021	**AIC**	17094.771
Time:	10:53:23	**BIC**	17118.942
Sample:	01-01-2013	**HQIC**	17103.991
	- 07-31-2015		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.2030	0.030	6.703	0.000	0.144	0.262
ma.L1	-1.0000	0.539	-1.854	0.064	-2.057	0.057
ar.S.L12	-0.0859	0.045	-1.923	0.054	-0.174	0.002
ma.S.L12	-0.9990	0.542	-1.845	0.065	-2.060	0.062
sigma2	5.336e+06	1.01e-07	5.28e+13	0.000	5.34e+06	5.34e+06

Ljung-Box (Q):	2647.72	**Jarque-Bera (JB):**	0.15
Prob(Q):	0.00	**Prob(JB):**	0.93
Heteroskedasticity (H):	1.15	**Skew:**	-0.02
Prob(H) (two-sided):	0.22	**Kurtosis:**	2.95

```
test_data['forecast']=results.predict(start=-31,dynamic=True)
test_data[['Sales','forecast']].plot(figsize=(12,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7a9d893e80>
```



Model Implementation:

```
accuracy['Train Adjusted R2']=accuracy['Train Adjusted R2']*100
print(accuracy['Train Adjusted R2'])
```

```
LassoRegression     89.55
RFRegressor         99.21
DL                  90.28
Name: Train Adjusted R2, dtype: float64
```

```
accuracy['Test Adjusted R2']=accuracy['Test Adjusted R2']*100
print(accuracy['Test Adjusted R2'])
```

```
LassoRegression     89.67
RFRegressor         98.12
DL                  91.23
Name: Test Adjusted R2, dtype: float64
```