Introduction to Business Problem

Project: Travel and Tourism

Business problem: Forecasting sales for next 30 days

Tools: Spyder and Tableau

Techniques: Time-Series Analysis

- Pre-processing data (aggregating sales day wise)
- Exploring time-series, Level, trend and seasonality
- Trying and testing different forecasting methods and selecting the best one to forecast sales of the next 30 days

Description about Data

Columns: Order date, Sales, Order Type,

Type, State

Order date: Date on which order is placed. Data are given in different different formats we need to convert it into Sales

Sales: Sales on that particular day on that particular order type and State

Order Type: Categorical data types. Types of Order it takes for sale

```
data.iloc[:,2].value_counts()
```

```
Out[10]:
Air                              151995
payment                           55394
Charge                            32775
Other Product                     18692
Hotel                              7904
refund                             5495
Air Cancellation                   4600
Other Product Cancellation          547
Air Debit Note                      535
Hotel Cancellation                  289
Air Loss                            210
Other Product Debit Note             21
Hotel Loss                            4
Hotel Debit Note                      3
Name: Order Type, dtype: int64

In [11]:
```

Order Date	Sales	Order Type	Type	State
02/12/18	269	Other Product	Domestic	Delhi
02/12/18	269	Other Product	Domestic	Delhi
02/12/18	269	Other Product	Domestic	Delhi
02/12/18	269	Other Product	Domestic	Delhi
02/12/18	0	Charge	nan	Rajasthan
02/12/18	0	Charge	nan	Rajasthan
02/12/18	0	Charge	nan	Rajasthan
02/12/18	2218	Hotel	Domestic	Rajasthan
02/12/18	0	Charge	nan	Rajasthan
02/12/18	2218	Hotel	Domestic	Rajasthan
02/12/18	0	Charge	nan	Rajasthan
08/12/18	0	Charge	nan	Rajasthan
02/12/18	0	Charge	nan	Rajasthan
08/12/18	0	Charge	nan	Rajasthan
02/12/18	5743	Air	Domestic	Maharashtra
02/12/18	10	Other Product	Domestic	Maharashtra
02/12/18	2688	Air	Domestic	Kerala
02/12/18	10	Other Product	Domestic	Kerala
03/12/18	3650	Air	Domestic	Tamil Nadu
03/12/18	2413	Air	Domestic	Tamil Nadu
03/12/18	10	Other Product	Domestic	Tamil Nadu

Type: Domestic and International

State: State name on where purchase has been done.

EDA Observations and steps along with code

```
# Importing Dataset
import pandas as pd

data = pd.read_csv("C:\\Users\\dell\\Desktop\\project7\\data\\sales_project7_modified.csv")

data.iloc[:,2].value_counts()
```

Order Date is not in one format. Separating Order Day, Month and Year in different columns, then again combine them in a single column in one single format.

Order Date	Sales
12/12/18	150
12/12/18	150
12/12/18	8098
12/12/18	8098
12/12/18	3600
12/12/18	0
12/12/18	3349
12/12/18	0
13-12-2018 00:01	10
13-12-2018 13:54	6664
13-12-2018 13:54	0
13-12-2018 15:07	7850
13-12-2018 10:45	269
13-12-2018 12:06	5305
13-12-2018 13:11	2131
13-12-2018 13:11	2131

```
data['day'] = data['Order Date'].str[0:2]

data['month'] = data['Order Date'].str[3:5]

data['year'] = data['Order Date'].str[6:]

data.loc[data['year'].str.len()==2, 'year'] = '20'+ data['year']

data['year'] = data['year'].str[0:5]

# dd-mm-yyyy string format in one column

data['odatestring'] =  data['day'] +'/'+ data['month'] + '/' + data['year']
```

Index	Order Date	Sales	Order Type	Type	State	day	month	year	odatestring
8909	12/01/19	4127.04	Air	International	Haryana	12	01	2019	12/01/2019
8910	12/01/19	499	Other Product	International	Haryana	12	01	2019	12/01/2019
8911	12/01/19	499	Other Product	International	Haryana	12	01	2019	12/01/2019
8912	12/01/19	3626	Air	Domestic	Haryana	12	01	2019	12/01/2019
8913	12/01/19	0	Charge	nan	Haryana	12	01	2019	12/01/2019
8914	13-01-2019 00:01	7497	Air	Domestic	Uttar Pradesh	13	01	2019	13/01/2019
8915	13-01-2019 00:01	0	Charge	nan	Uttar Pradesh	13	01	2019	13/01/2019
8916	13-01-2019 00:06	10620	Hotel	Domestic	Uttar Pradesh	13	01	2019	13/01/2019

```
data.dtypes
```

```
Order Date       object
Sales           float64
Order Type       object
Type             object
State            object
day              object
month            object
year             object
odatestring      object
dtype: object
```

```
# filling NaN values by zero
data.dropna(inplace = True)
```

Order Date	Sales	Order Type
31-12-2018 15:53	nan	payment
31-12-2018 15:58	nan	payment
31-12-2018 16:00	nan	payment
31-12-2018 21:14	nan	refund
31-12-2018 14:52	nan	payment
31-12-2018 16:05	nan	payment
31-12-2018 16:06	nan	payment
31-12-2018 14:56	nan	payment
31-12-2018 15:14	nan	payment
31-12-2018 15:14	nan	payment
31-12-2018 15:26	nan	payment
31-12-2018 15:26	nan	payment
31-12-2018 17:05	nan	payment

```
# Dropping extra columns
sales_dates = data.drop(['Order Date','Order
Type','Type','State','day','month','year'], axis=1)
```

```
# check missing values
sales_dates.isnull().sum()
```

```
In [16]: sales_dates.isnull().sum()
Out[16]:
Sales          0
odatestring    0
dtype: int64
```

```
#converting string to Date and time object; sales to integer
sales_dates['odatestring'] = pd.to_datetime(sales_dates['odatestring'], dayfirst =
True)
sales_dates['Sales'] = sales_dates['Sales'].astype(float)
sales_dates.dtypes
```

```
Out[17]:
Sales                 float64
odatestring    datetime64[ns]
dtype: object

In [18]:
```

```
#renaming column name
sales_dates =
sales_dates.rename(columns={'odatestring':'order_date'})
```

creating index on date column to work resample

```
sales_dates = sales_dates.set_index('order_date')
```

aggregating sales Day wise using resample method

```
sales_day_wise = sales_dates.resample('D').sum()
```

Index	Sales
2018-04-01 00:00:00	200746
2018-04-02 00:00:00	510258
2018-04-03 00:00:00	372377
2018-04-04 00:00:00	608408
2018-04-05 00:00:00	1.06378e+06
2018-04-06 00:00:00	806566
2018-04-07 00:00:00	647209
2018-04-08 00:00:00	294807
2018-04-09 00:00:00	717486

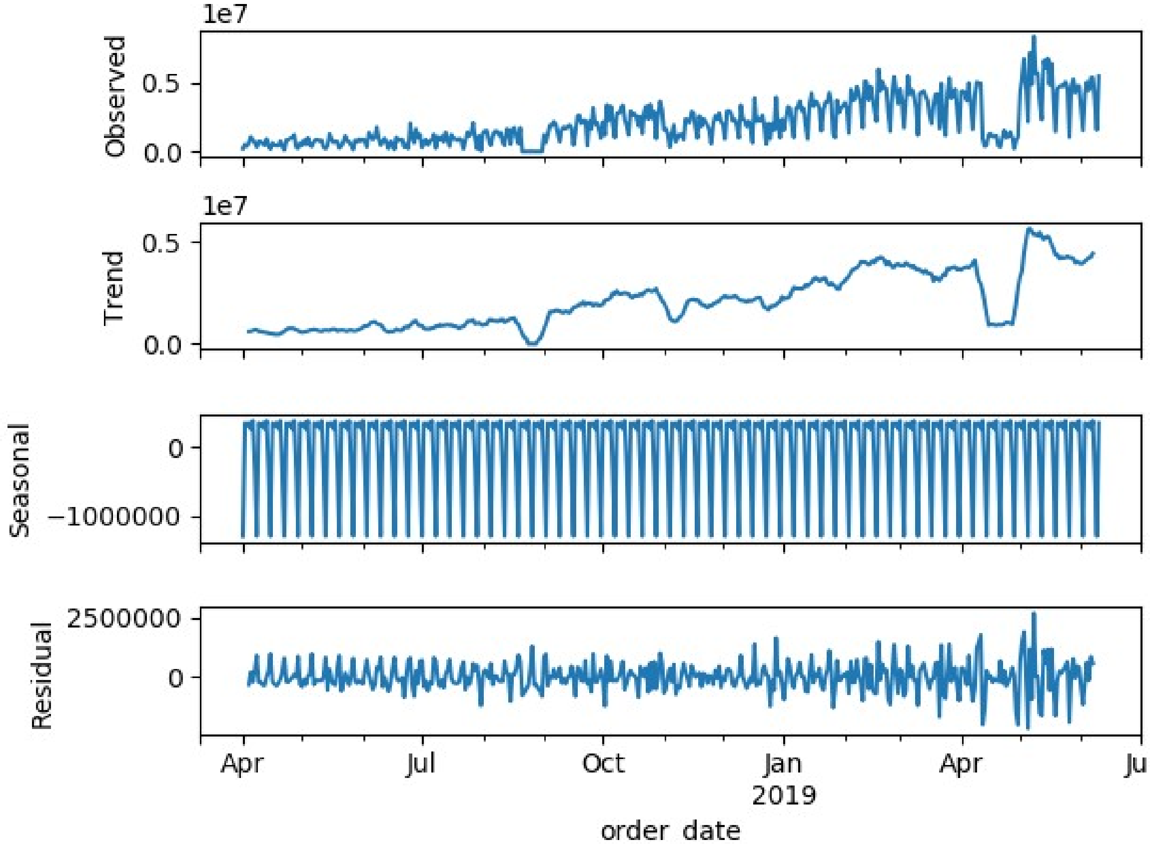
21-Aug-2018 to 31-Aug-2018 sales are 0. Replacing them by 1. B'coz taking log of 0 is infinite

```
sales_day_wise.loc[sales_day_wise['Sales']==0, 'Sales'] = 1

sales_day_wise.dtypes
```

Visualizing Time-Series

seasonal_decompose expects timeseries column as index

```
from statsmodels.tsa.seasonal import seasonal_decompose

decompose_data = seasonal_decompose(sales_day_wise['Sales'], model = 'additive')

decompose_data.plot()

decompose_data = seasonal_decompose(sales_day_wise['Sales'], model = 
'multiplicative')

decompose_data.plot()
```

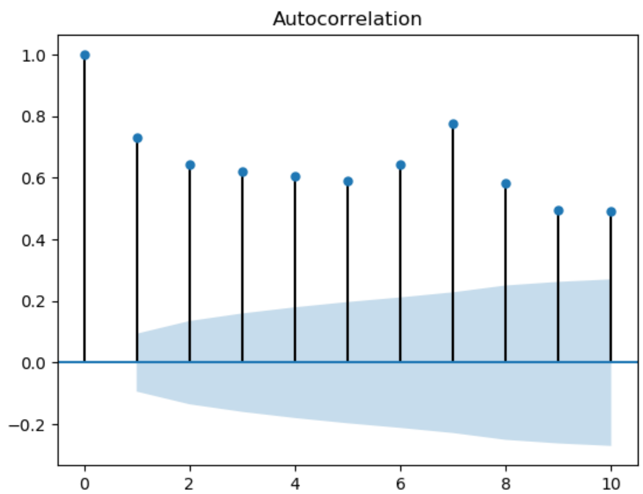


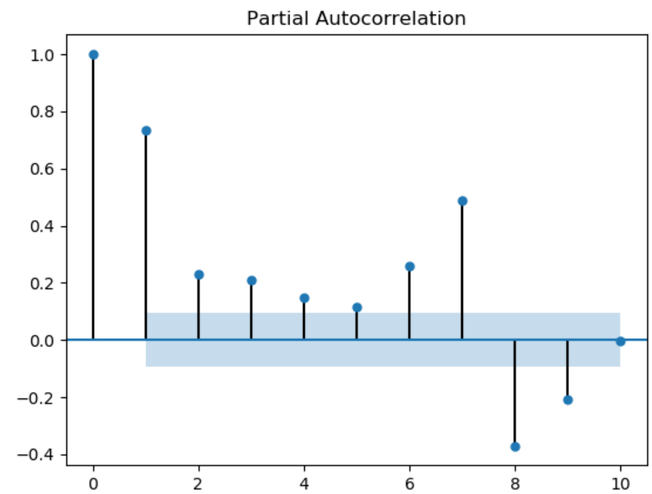
Inferences from above plot:

Trend: Linear

Seasonality: Additive (M=6 or M=7)

```
#ACF Plots and PACF Plots on orignal Data
import statsmodels.graphics.tsaplots as tsa_plts
tsa_plts.plot_acf(sales_day_wise['Sales'], lags=10)
tsa_plts.plot_pacf(sales_day_wise['Sales'], lags=10)
```





Inferences from above plots:

ACF lag = 1

Moving average window = 5

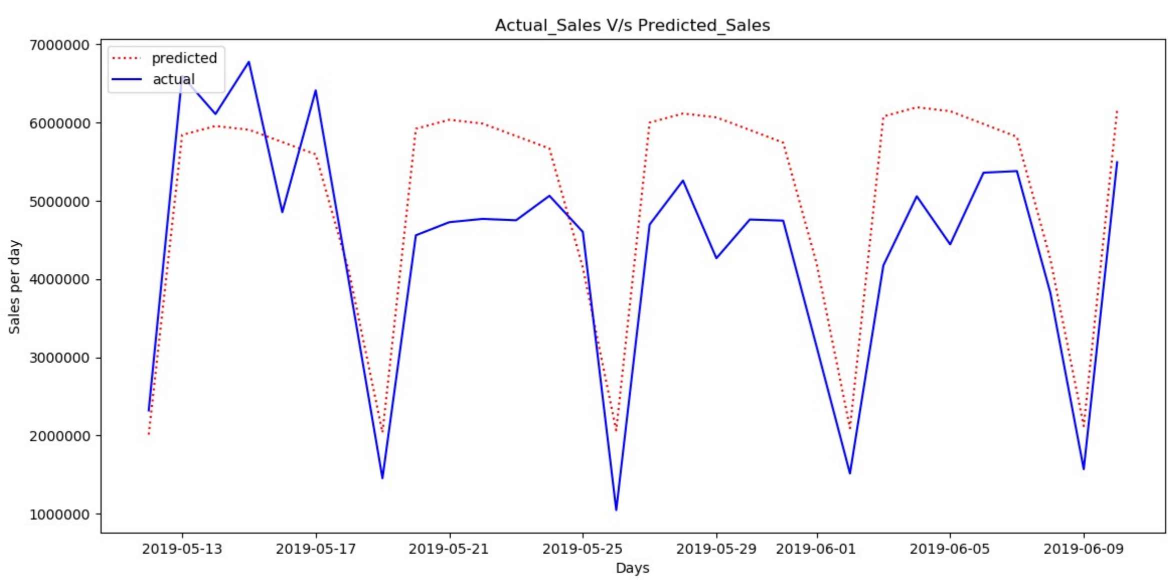
Splitting Train and Test Data

```
# Splitting Train and Test Data taking last 30 days into our test dataset
Train = sales_day_wise.head(406)
Test = sales_day_wise.tail(30)


# Creating a MAPE function
def MAPE(pred, orig):
    temp = np.abs((pred-orig)/orig)*100
    return np.mean(temp)
```

Building model

```
#########################

## Smoothing Techniques ##

#########################


# Simple Exponential Smoothing

from statsmodels.tsa.holtwinters import SimpleExpSmoothing

ses_model = SimpleExpSmoothing(Train['Sales']).fit()

pred_ses = ses_model.predict(start = Test.index[0], end = Test.index[-1])

mape_ses = MAPE(pred_ses, Test['Sales']) #52.92 %


# Holt method only trend

from statsmodels.tsa.holtwinters import Holt

holt_model = Holt(Train['Sales']).fit()

pred_holt = holt_model.predict(start = Test.index[0], end = Test.index[-1])

mape_holt = MAPE(pred_holt, Test['Sales']) #57.83%


# Winter method with both trend and Seasonality

from statsmodels.tsa.holtwinters import ExponentialSmoothing


#winter model with additive seasonality = 7

winter_model_7 = ExponentialSmoothing(Train['Sales'], seasonal = 'add', trend =
'add', seasonal_periods = 7).fit()

pred_winter_model_7 = winter_model_7.predict(start = Test.index[0], end =
Test.index[-1])

mape_winter_model_7 = MAPE(pred_winter_model_7, Test['Sales']) #29.71%


#winter model with additive seasonality = 6

winter_model_6 = ExponentialSmoothing(Train['Sales'], seasonal = 'add', trend =
'add', seasonal_periods = 6).fit()

pred_winter_model_6 = winter_model_6.predict(start = Test.index[0], end =
Test.index[-1])

mape_winter_model_6 = MAPE(pred_winter_model_6, Test['Sales']) #54.24%
```

```
#winter model with additive seasonality = 5

winter_model_5 = ExponentialSmoothing(Train['Sales'], seasonal = 'add', trend =
'add', seasonal_periods = 5).fit()

pred_winter_model_5 = winter_model_5.predict(start = Test.index[0], end =
Test.index[-1])

mape_winter_model_5 = MAPE(pred_winter_model_5, Test['Sales']) #60.52%


#winter model with multiplicative seasonality = 7

winter_model_7_mul = ExponentialSmoothing(Train['Sales'], seasonal = 'mul', trend =
'add', seasonal_periods = 7).fit()

pred_winter_model_7_mul = winter_model_7_mul.predict(start = Test.index[0], end =
Test.index[-1])

mape_winter_model_7_mul = MAPE(pred_winter_model_7_mul, Test['Sales']) #24.28%
(best)

import pylab

x = pd.date_range('2019-05-12', periods=30, freq='D')

pylab.plot(x, pred_winter_model_7_mul,':r', label = 'predicted')

pylab.plot(x, Test['Sales'],'-b', label = 'actual')

pylab.xlabel('Days')

pylab.ylabel('Sales per day')

pylab.title('Actual_Sales V/s Predicted_Sales')

pylab.legend(loc='upper left')

pylab.show()
```



```
winter_model_7_mul_resi = pd.DataFrame(winter_model_7_mul.resid)

tsa_plts.plot_acf(winter_model_7_mul_resi, lags=12) # 2 is showing significance


#Using ARIMA forecasting errors for Lag =2

winter_model_7_mul_resi = winter_model_7_mul_resi.reset_index(drop=True)

resi_winter_model_7_mul = ARIMA(winter_model_7_mul_resi[0],
order=(2,0,0)).fit(transparams=True)

tsa_plts.plot_acf(resi_winter_model_7_mul.resid, lags=12) # No significant lags

winter_forecast_errors = resi_winter_model_7_mul.forecast(steps = 30)[0]


winters_predictions = pd.DataFrame(columns =
['forecast_sales','forecast_errors','improved'])

winters_predictions['forecast_sales'] = pd.Series(pred_winter_model_7_mul)

winters_predictions = winters_predictions.reset_index(drop=True)

winters_predictions['forecast_errors'] = pd.Series(winter_forecast_errors)

winters_predictions['improved'] = winters_predictions['forecast_sales']
+winters_predictions['forecast_errors']

Test = Test.reset_index(drop=True)

mape_winter_model_7_mul_AR = MAPE(winters_predictions['improved'], Test['Sales'])
#24.82%

###############################
######   ARIMA Model   ########
###############################


from statsmodels.tsa.arima_model import ARIMA


#ARIMA for Lag=1 and window_size = 2

arima_m2 = ARIMA(Train['Sales'], order=(1,1,2)).fit(transparams=True)


#ARIMA for Lag=1 and window_size = 5

arima_m5 = ARIMA(Train['Sales'], order=(1,1,5)).fit(transparams=True)


#residuals of residuals

arima_m2_resi = pd.DataFrame(arima_m2.resid)
```

```
arima_m5_resi = pd.DataFrame(arima_m5.resid)

# ACF Plot of Residuals of Residuals
tsa_plts.plot_acf(arima_m2_resi, lags=12)
tsa_plts.plot_acf(arima_m5_resi, lags=12)
#Showing Significance at Lag = 7

arima_m2_resi_lag7 = ARIMA(arima_m2_resi[0], order=(7,1,2)).fit(transparams=True)
arima_m5_resi_lag7 = ARIMA(arima_m5_resi[0], order=(7,1,5)).fit(transparams=True)

#residuals of residuals of residuals
arima_m2_resi_lag7_resi = pd.DataFrame(arima_m2_resi_lag7.resid)
arima_m5_resi_lag7_resi = pd.DataFrame(arima_m5_resi_lag7.resid)

# ACF Plot of Residuals of Residuals of Residuals
tsa_plts.plot_acf(arima_m2_resi_lag7_resi, lags=12)
tsa_plts.plot_acf(arima_m5_resi_lag7_resi, lags=12)


# Predicting values using ARIMA model

# Moving Average = 2
arima_ma2_forecast_values =  arima_m2.forecast(steps = 30)[0]
arima_ma2_forecast_errors = arima_m2_resi_lag7.forecast(steps = 30)[0]

arima_predictions = pd.DataFrame(columns =
['forecast_sales','forecast_errors','improved'])
arima_predictions['forecast_sales'] = pd.Series(arima_ma2_forecast_values)
arima_predictions['forecast_errors'] = pd.Series(arima_ma2_forecast_errors)
arima_predictions['improved'] = arima_predictions['forecast_sales']
+arima_predictions['forecast_errors']

Test = Test.reset_index(drop=True)
mape_arima_ma2 = MAPE(arima_predictions['improved'], Test['Sales']) #55.31
```

```
# Moving Average = 5

arima_ma5_forecast_values =  arima_m5.forecast(steps = 30)[0]

arima_ma5_forecast_errors = arima_m5_resi_lag7.forecast(steps = 30)[0]


arima_predictions_5 = pd.DataFrame(columns =
['forecast_sales','forecast_errors','improved'])

arima_predictions_5['forecast_sales'] = pd.Series(arima_ma5_forecast_values)

arima_predictions_5['forecast_errors'] = pd.Series(arima_ma5_forecast_errors)

arima_predictions_5['improved'] = arima_predictions_5['forecast_sales']
+arima_predictions_5['forecast_errors']


mape_arima_ma5 = MAPE(arima_predictions_5['improved'], Test['Sales']) #65.81
```

Final model along with code

```
#Building final model with whole Data set : winter model with multiplicative
seasonality = 7
sales_day_wise = sales_dates.resample('D').sum()
sales_day_wise.loc[sales_day_wise['Sales']==0, 'Sales'] = 1
sales_day_wise.dtypes
from statsmodels.tsa.holtwinters import ExponentialSmoothing
final_model = ExponentialSmoothing(sales_day_wise['Sales'], seasonal = 'mul', trend
= 'add', seasonal_periods = 7).fit()

# creating Dates for next 30 days to be forcasted
date_range = pd.date_range('2019-06-11', periods=30, freq='D')
next_30_days = pd.DataFrame(index=date_range)

# Forecasting Sales of next 30 days using Final model
final_predictions = final_model.predict(start = next_30_days.index[0], end =
next_30_days.index[-1])
final_predictions = final_predictions.astype(int)
final_predictions
```