INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT AKURDI, PUNE

Documentation On

“Global Superstore Profit Prediction”
PG-DBDA MARCH 2022

Submitted by:

GroupNo:08
Sayali Pendharkar 223349
Sonia Dharankar 223315

Mr. Prashant Karhale
Centre Coordinator

Mr. Akshay Tilekar
Project Guide

INDEX

ABSTRACT

The task of predicting profit is an important task for every business to set an achievable goal. For predicting the profit of a company for a particular period we need to train a machine learning model with a dataset that contains historical data about the profit generated by the company**.** Nowadays, machine learning models are important tools to replace human tasks. In this case, there are several features to predict the profit but the entire features will not be relevant for better prediction. So, our thesis work is focusing on what profit features are important to get the promising result. For the purpose of regression model and evaluation of the relevant features, we used multiple algorithms.

In this study, we used Global Superstore dataset. To evaluate performance, we used metric log loss for all machine learning algorithm.

We aim to put on different machine learning techniques to construct and adjust a Profit forecasting model and perform the estimation. Prediction has been performed by considering 'Order Date' as the prominent feature and the 'Profit' as the target variable. Hence this is a time series based forecasting. The results have been depicted using some visualization tools.

1. INTRODUCTION

In today’s world, data are produced everywhere like just travelling to different location (GPS data), browsing the internet (internet history), storing pictures and many more. This information is being used to provide a personalized environment to the user. But, the challenge in here is that such data is quite large and it cannot be just processed by a single individual or even a team as this data grows tremendously due to its sources of production (if a mobile data in turned on then data is started to generate from that user). So, there comes Machine Learning that makes use of all these data and provide what the users want. This core concept of Machine Learning is used in predicting the profit of a company. So, by considering the history about the companies viz., their previous profit record, a model has been built that recognizes a pattern via the product categories affecting the profit so that it can be better predicted by the means of this model.

2. FUNCTIONAL REQUIREMENTS

2.1Python3:

- Python is a general purpose and high level programming language.
- It is used for developing desktop GUI applications, websites and web applications.
- Pythonallowstofocusoncorefunctionalityoftheapplicationbytakingcareofcommonprogramming tasks.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68,
- Small Talk, and UNIX shell and other scripting language.

2.2 Machine Learning:

The data available is increasing day by day and such a huge amount of unprocessed data is needed to be analyzed precisely, as it can give very informative and finely pure gradient results as per current standard requirements. It is not wrong to say as with the evolution of Artificial Intelligence over the past two decades, Machine Learning is also on a fast pace for its evolution. ML is an important mainstay of IT sector and with that, a rather central, albeit usually hidden, part of our life. As the technology progresses, the analysis and understanding of data to give good results will also increase as the data is very useful in current aspects. In machine learning, one deals with both supervised and unsupervised types of tasks and generally a classification type problem accounts as a resource for knowledge discovery. It generates resources and employs regression to make precise predictions about future, the main emphasis being laid on making a system self-efficient, to be able to do computations and analysis to generate much accurate and precise results. By using statistic and probabilistic tools, data can be converted into knowledge. The statistical inferencing uses sampling distributions as a conceptual key.

2.3 Machine Learning Algorithms

2.3.1. Random Forest:

Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

2.3.2. Extreme Gradient Boosting (XGBoost) :

XGBoost is an implementation of Gradient Boosted decision trees. This library was written in C++. It is a type of Software library that was designed basically to improve speed and model performance. In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and the variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

2.3.3. Long short-term memory (LSTM):

Long Short Term Memory is a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give an efficient performance. LSTM can by default retain the information for a long period of time. It is used for processing, predicting, and classifying on the basis of time-series data.

2.3.4. AutoRegressive Integrated Moving Average(ARIMA):

The ARIMA model predicts a given time series based on its own past values. It can be used for any nonseasonal series of numbers that exhibits patterns and is not a series of random events. For example, sales data from a clothing store would be a time series because it was collected over a period of time. One of the key characteristics is the data is collected over a series of constant, regular intervals. A modified version can be created to model predictions over multiple seasons. For a period of multiple seasons, the data must be corrected to account for differences between the seasons. For example, holidays fall on different days of the year, causing a seasonal effect to the data. Sales may be artificially higher or lower depending on where the holiday falls in the calendar. The data scientist must be able to seasonally adjust the data to provide an accurate prediction for future sales.

2.3.5 FBProphet:

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

3. METHODOLOGY

3.1. Overall Description

3.1.1 Workflow of Project:

The diagram below shows the workflow of this project.

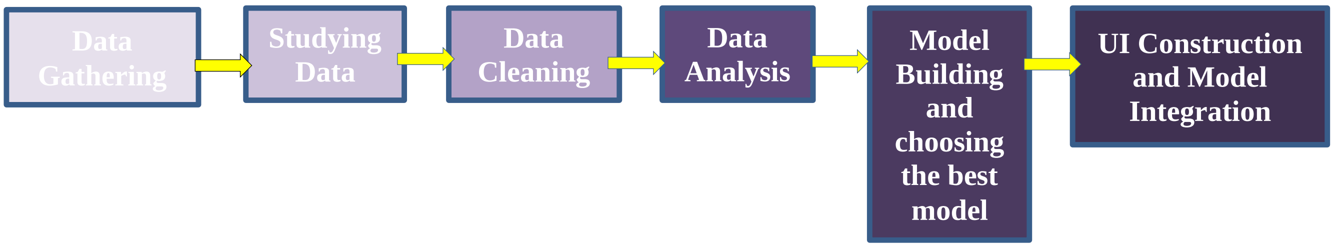

Data Gathering
Studying Data
Data Cleaning
Data Analysis
Model Building and choosing the best model
UI Construction and Model Integration

Figure 1: Workflow Diagram

3.2 Data Cleaning Process:

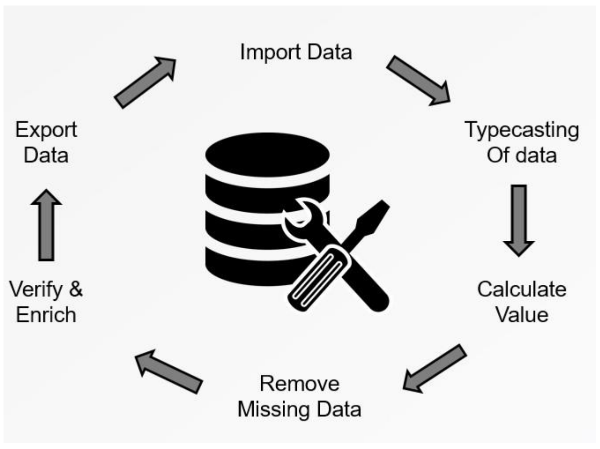
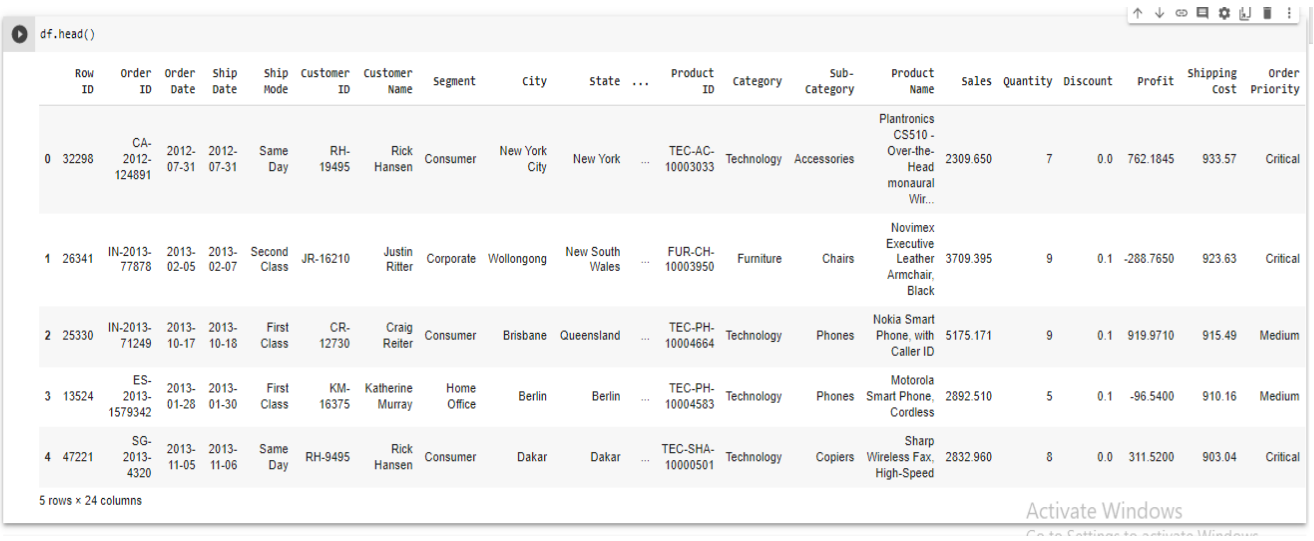


Figure 2: Data Cleaning Process

This project deals with prediction of future profit of a global super store. In this model, daily transactional data of a global super market has been used with transaction records for a period between year 2011 to the year 2014.The data contains prominent features: Order date and Profit earned behind each product. Each product comes under three major categories: Furniture, Office Supplies and Technology. From observations, the contribution of each category in the profit making of the superstore can be found. The dataset looks like shown in Figure 3 on using head() function on the dataset variable.



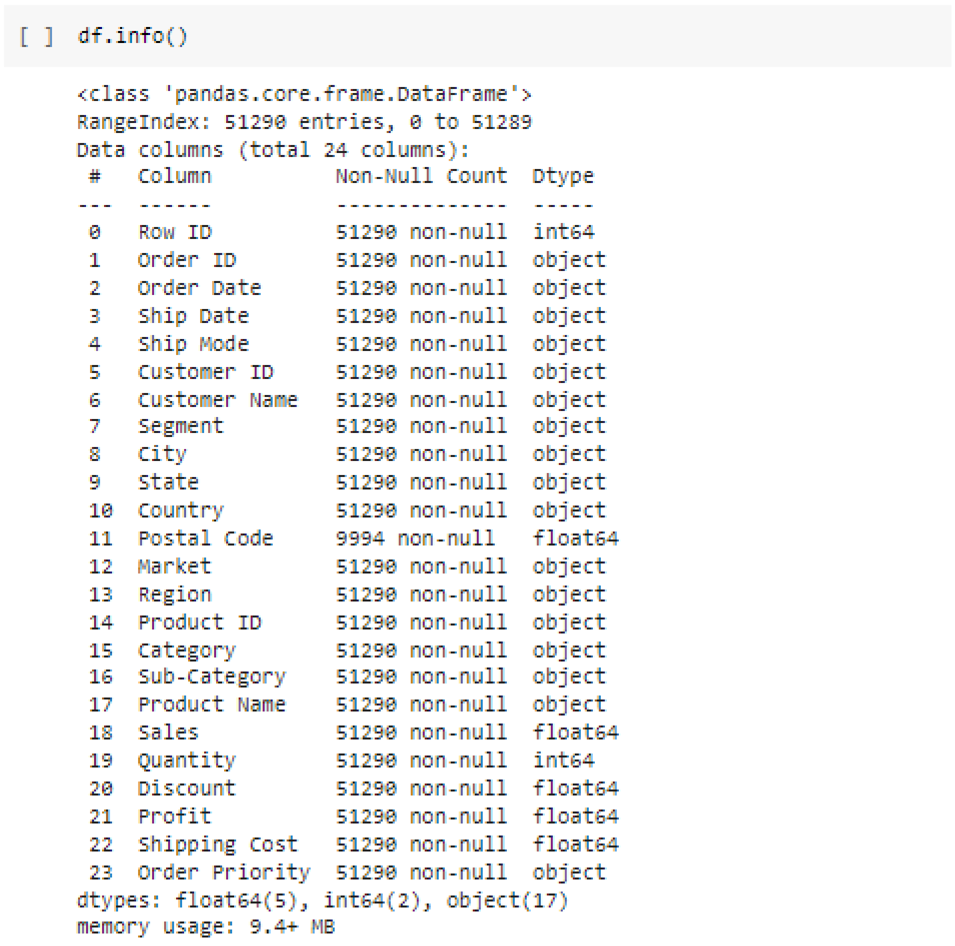
```
df.head()
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	City	State	...	Product ID	Category	Sub-Category	Product Name	Sales	Quantity	Discount	Profit	Shipping Cost	Order Priority
0	32298	CA-2012-124891	2012-07-31	2012-07-31	Same Day	RH-19495	Rick Hansen	Consumer	New York City	New York	...	TEC-AC-10003033	Technology	Accessories	Plantronics CS510 - Over-the-Head monaural Wir...	2309.650	7	0.0	762.1845	933.57	Critical
1	26341	IN-2013-77878	2013-02-05	2013-02-07	Second Class	JR-16210	Justin Ritter	Corporate	Wollongong	New South Wales	...	FUR-CH-10003950	Furniture	Chairs	Novimex Executive Leather Armchair, Black	3709.395	9	0.1	-288.7650	923.63	Critical
2	25330	IN-2013-71249	2013-10-17	2013-10-18	First Class	CR-12730	Craig Reiter	Consumer	Brisbane	Queensland	...	TEC-PH-10004664	Technology	Phones	Nokia Smart Phone, with Caller ID	5175.171	9	0.1	919.9710	915.49	Medium
3	13524	ES-2013-1579342	2013-01-28	2013-01-30	First Class	KM-16375	Katherine Murray	Home Office	Berlin	Berlin	...	TEC-PH-10004583	Technology	Phones	Motorola Smart Phone, Cordless	2892.510	5	0.1	-96.5400	910.16	Medium
4	47221	SG-2013-4320	2013-11-05	2013-11-06	Same Day	RH-9495	Rick Hansen	Consumer	Dakar	Dakar	...	TEC-SHA-10000501	Technology	Copiers	Sharp Wireless Fax, High-Speed	2832.960	8	0.0	311.5200	903.04	Critical

5 rows × 24 columns

Figure 3: Dataset first five rows

The data set consists of 3 data types: object and integer and float. as shown in Figure 4.

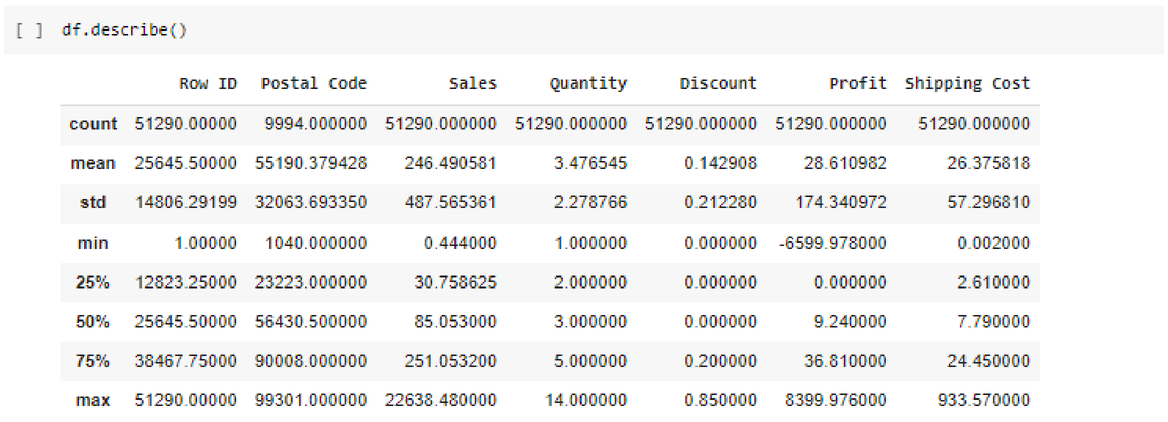


```
[ ]  df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51290 entries, 0 to 51289
Data columns (total 24 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Row ID          51290 non-null  int64  
 1   Order ID        51290 non-null  object 
 2   Order Date      51290 non-null  object 
 3   Ship Date       51290 non-null  object 
 4   Ship Mode       51290 non-null  object 
 5   Customer ID     51290 non-null  object 
 6   Customer Name   51290 non-null  object 
 7   Segment         51290 non-null  object 
 8   City            51290 non-null  object 
 9   State           51290 non-null  object 
 10  Country         51290 non-null  object 
 11  Postal Code     9994 non-null   float64
 12  Market          51290 non-null  object 
 13  Region          51290 non-null  object 
 14  Product ID      51290 non-null  object 
 15  Category        51290 non-null  object 
 16  Sub-Category    51290 non-null  object 
 17  Product Name    51290 non-null  object 
 18  Sales           51290 non-null  float64
 19  Quantity        51290 non-null  int64  
 20  Discount        51290 non-null  float64
 21  Profit          51290 non-null  float64
 22  Shipping Cost   51290 non-null  float64
 23  Order Priority  51290 non-null  object 
dtypes: float64(5), int64(2), object(17)
memory usage: 9.4+ MB
```

Figure 4: Datatypes of columns

In the raw data, there can be various types of underlying patterns which also gives an in-depth knowledge about subject of interest and provides insights about the problem. But caution should be observed with respect to data as it may contain null values, or redundant values, or various types of

ambiguity, which also demands for pre-processing of data. Dataset should therefore be explored as much as possible.
Various factors important by statistical means like mean, standard deviation, median, count of values and maximum value etc. are shown in Fig.4 for numerical variables of our dataset.

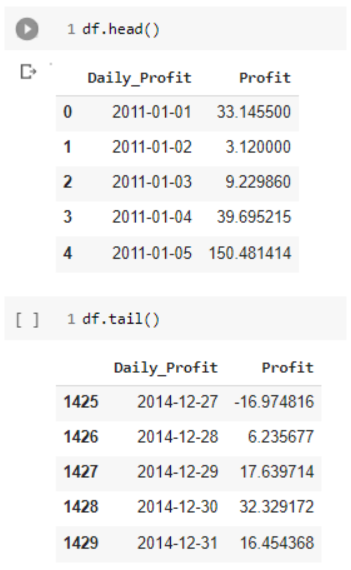


```
[ ] df.describe()
```

	Row ID	Postal Code	Sales	Quantity	Discount	Profit	Shipping Cost
count	51290.00000	9994.000000	51290.000000	51290.000000	51290.000000	51290.000000	51290.000000
mean	25645.50000	55190.379428	246.490581	3.476545	0.142908	28.610982	26.375818
std	14806.29199	32063.693350	487.565361	2.278766	0.212280	174.340972	57.296810
min	1.00000	1040.000000	0.444000	1.000000	0.000000	-6599.978000	0.002000
25%	12823.25000	23223.000000	30.758625	2.000000	0.000000	0.000000	2.610000
50%	25645.50000	56430.500000	85.053000	3.000000	0.000000	9.240000	7.790000
75%	38467.75000	90008.000000	251.053200	5.000000	0.200000	36.810000	24.450000
max	51290.00000	99301.000000	22638.480000	14.000000	0.850000	8399.976000	933.570000

Figure 5: Dataset description

Preprocessing of this dataset includes doing analysis on the independent variables like checking for null values in each column and then replacing or filling them with supported appropriate data types, so that analysis and model fitting is not hindered from its way to accuracy. In the preprocessing of the dataset, multiple date records with different products were found. In order to consolidate the multiple records into one row, mean value of Profit for each day was taken for each date. The dataset was ordered as per unique dates.



```
df.head()
```

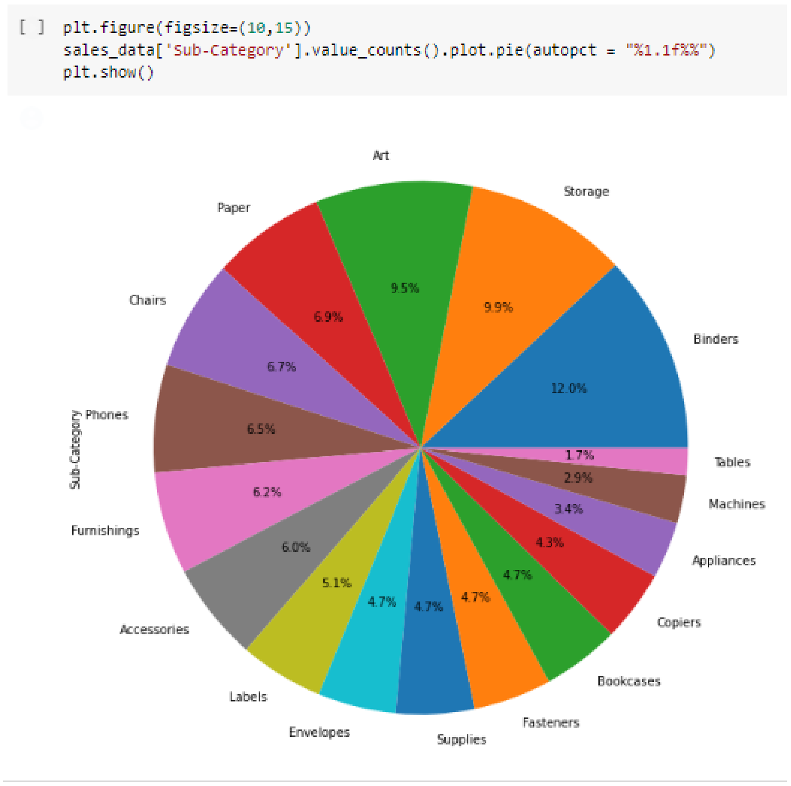
	Daily_Profit	Profit
0	2011-01-01	33.145500
1	2011-01-02	3.120000
2	2011-01-03	9.229860
3	2011-01-04	39.695215
4	2011-01-05	150.481414

```
[ ] df.tail()
```

	Daily_Profit	Profit
1425	2014-12-27	-16.974816
1426	2014-12-28	6.235677
1427	2014-12-29	17.639714
1428	2014-12-30	32.329172
1429	2014-12-31	16.454368

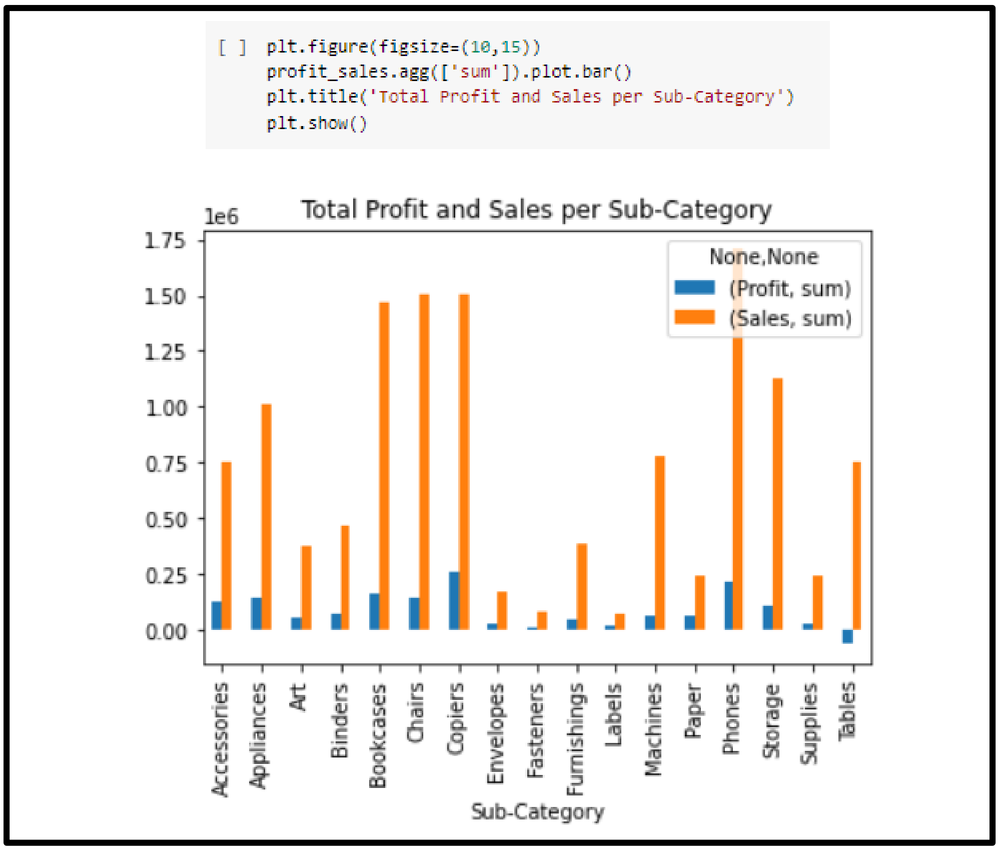
Figure 6: Data after preprocessing

4.PRIMARY DATA ANALYSIS AND VISUALIZATION



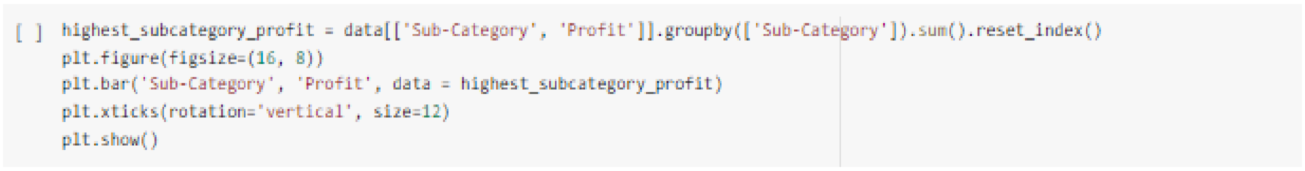
```
[ ]  plt.figure(figsize=(10,15))
     sales_data['Sub-Category'].value_counts().plot.pie(autopct = "%1.1f%%")
     plt.show()
```

Figure 7: Pie Chart of Sub-category wise products



```
[ ]  plt.figure(figsize=(10,15))
     profit_sales.agg(['sum']).plot.bar()
     plt.title('Total Profit and Sales per Sub-Category')
     plt.show()
```

Figure 8: Bar chart comparing sub category wise Sales vs Profit generated by products



```
[ ]  highest_subcategory_profit = data[['Sub-Category', 'Profit']].groupby(['Sub-Category']).sum().reset_index()
     plt.figure(figsize=(16, 8))
     plt.bar('Sub-Category', 'Profit', data = highest_subcategory_profit)
     plt.xticks(rotation='vertical', size=12)
     plt.show()
```

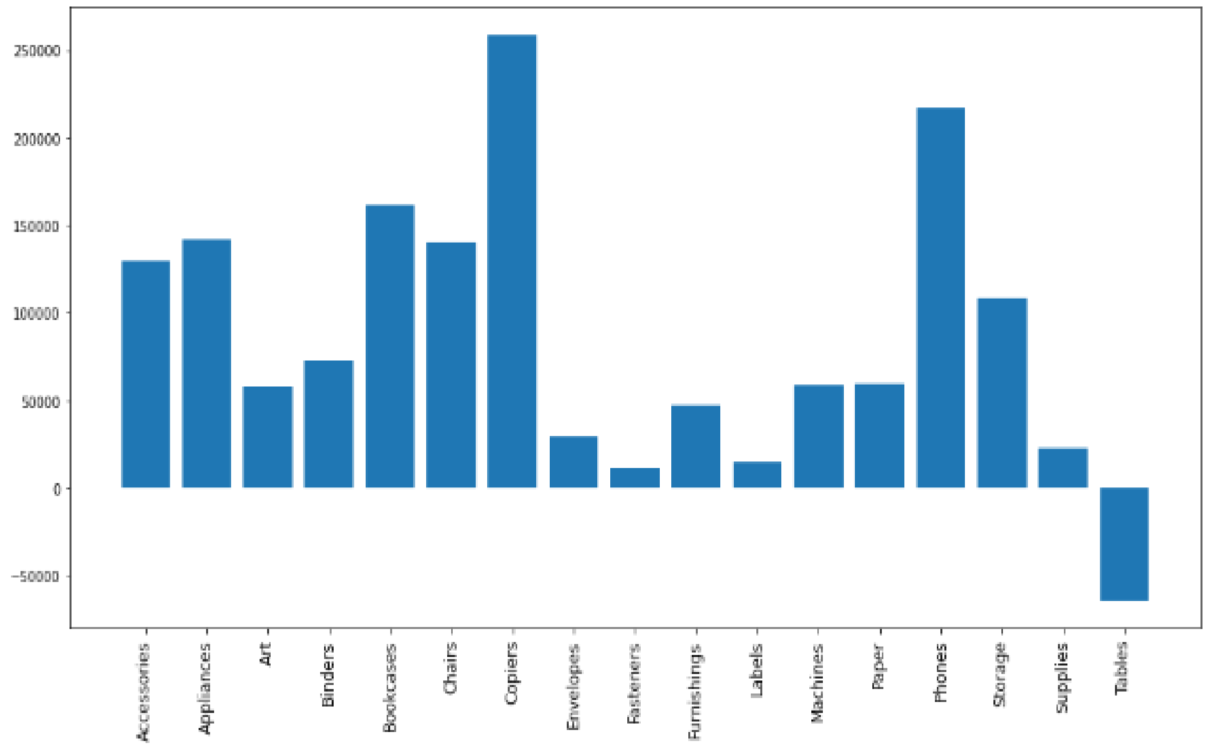


Figure 9: Total Profit generated by each Product Sub Category

4. METHODOLOGY

4.1 Data Collection

The Global Superstore Market datasets have been used in this project which is obtained from the website: kaggle which contains a large collection of datasets. The dataset was loaded using the Python library 'Pandas'. Pandas is a Python library used for working with data sets. It has functions for analysing, cleaning, exploring, and manipulating data.

4.2 Algorithms

To find out the most precise relation between features and response variable i.e. quality we applied different types of algorithms which are listed below

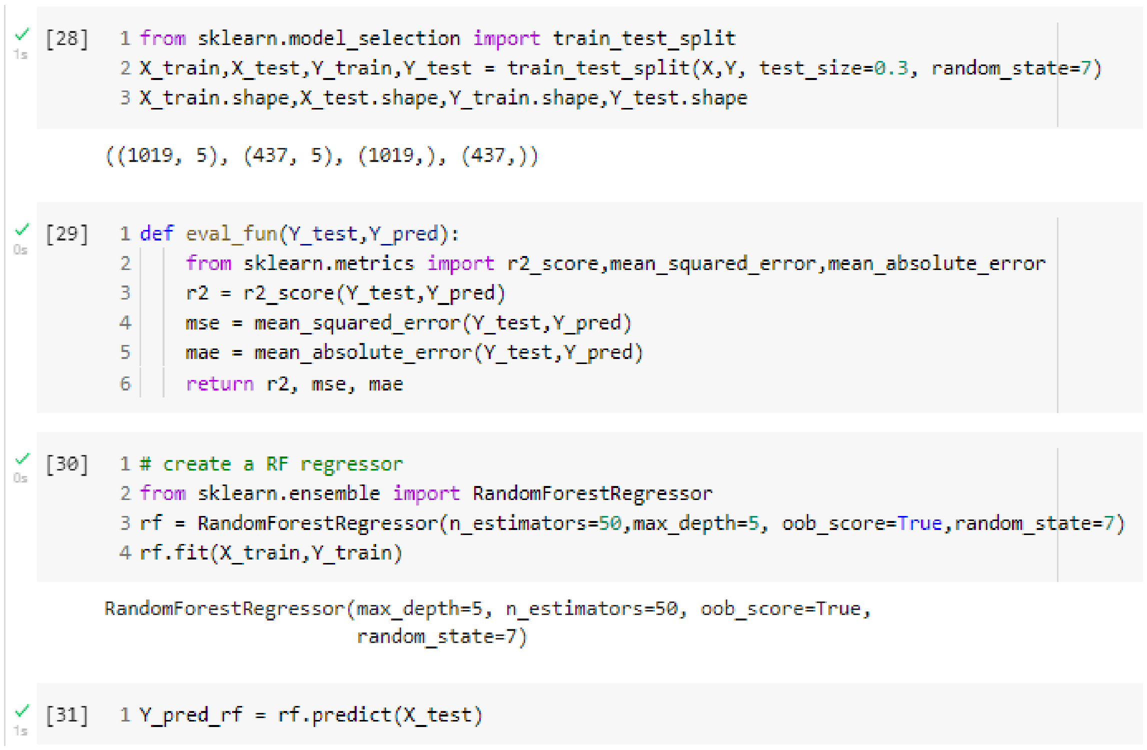
- Random Forest classifier
- XGB Classifier
- Long short-term memory (LSTM)
- AutoRegressive Integrated Moving Average(ARIMA)
- FBProphet

4.3 Algorithms employed

Scikit-Learn can be used to track machine-learning system on wholesome basis. Algorithms employed for predicting sales for this dataset are discussed as follows:

4.3.1 Random Forest Algorithm

Random forest algorithm is a very accurate algorithm to be used for predicting profit. It is easy to use and understand for the purpose of predicting results of machine learning tasks. In profit prediction, random forest classifier is used because it has decision tree like hyperparameters. The tree model is same as decision tool. To solve regression tasks of prediction by virtue of random forest, the ***sklearn.ensemble***library's **RandomForestRegressor** class is used. The key role is played by the parameter termed as ***n_estimators***which also comes under random forest regressor. Random forest can be referred to as a meta-estimator used to fit upon numerous decision trees (based on classification) by taking the dataset's different sub-samples. ***Min_samples_split***is taken as the minimum number when splitting an internal node if integer number of minimum samples are considered. A split's quality is measured using ***mse*****(mean squared error)**, which can also be termed as feature selection criterion. This also means reduction in variance ***mae*****(mean absolute error)**, which is another criterion for feature selection. Maximum tree depth, measured in integer terms, if equals one, then all leaves are pure or pruning for better model fitting is done for all leaves less than ***min_samples_split***samples.

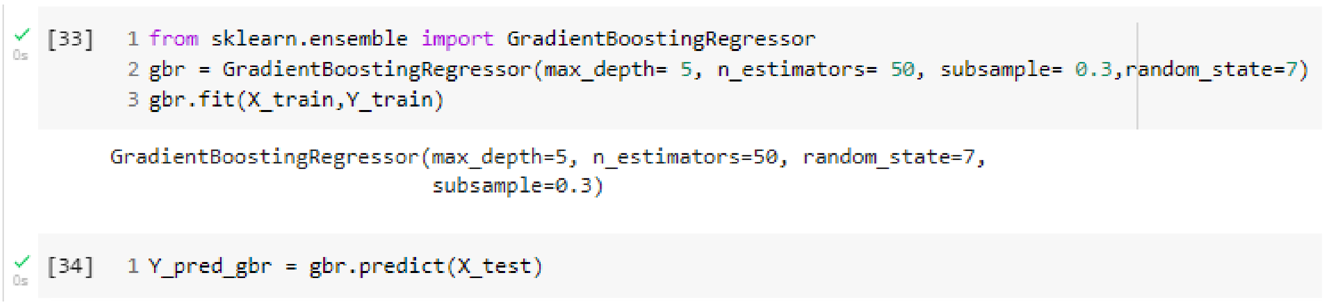


```
[28]
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size=0.3, random_state=7)
X_train.shape,X_test.shape,Y_train.shape,Y_test.shape

((1019, 5), (437, 5), (1019,), (437,))

[29]
def eval_fun(Y_test,Y_pred):
    from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
    r2 = r2_score(Y_test,Y_pred)
    mse = mean_squared_error(Y_test,Y_pred)
    mae = mean_absolute_error(Y_test,Y_pred)
    return r2, mse, mae

[30]
# create a RF regressor
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=50,max_depth=5, oob_score=True,random_state=7)
rf.fit(X_train,Y_train)

RandomForestRegressor(max_depth=5, n_estimators=50, oob_score=True,
                      random_state=7)

[31]
Y_pred_rf = rf.predict(X_test)
```

Figure 10: Application of Random Forest Regressor

4.3.2 XGBoost Algorithm

XGBoost is a powerful approach for building supervised regression models. The validity of this statement can be inferred by knowing about its (XGBoost) objective function and base learners. To solve regression tasks of prediction by virtue of Gradient Boosting algorithm, the ***sklearn.ensemble***library's **GradientBoostingRegressor** class is used. The objective function contains loss function and a regularization term. It tells about the difference between actual values and predicted values, i.e. how far the model results are from the real values. The most common loss functions in XGBoost for regression problems is reg:linear, and that for binary classification is reg:logistics. Ensemble learning involves training and combining individual models (known as base learners) to get a single prediction, and XGBoost is one of the ensemble learning methods. XGBoost expects to have the base learners which are uniformly bad at the remainder so that when all the predictions are combined, bad predictions cancel out and better one sums up to form final good predictions.



```
[33]
from sklearn.ensemble import GradientBoostingRegressor
gbr = GradientBoostingRegressor(max_depth= 5, n_estimators= 50, subsample= 0.3,random_state=7)
gbr.fit(X_train,Y_train)

GradientBoostingRegressor(max_depth=5, n_estimators=50, random_state=7,
                          subsample=0.3)

[34]
Y_pred_gbr = gbr.predict(X_test)
```

Figure 11: Application of GradientBoost Regressor

4.3.3 LSTM (Long short-term memory)

To solve the problem of Vanishing and Exploding Gradients in a Deep Recurrent Neural Network, many variations were developed. One of the most famous of them is the **Long Short Term Memory Network**(LSTM). In concept, an LSTM recurrent unit tries to "remember" all the past knowledge that the network is seen so far and to "forget" irrelevant data. This is done by introducing different activation function layers called "gates" for different purposes.
Each LSTM recurrent unit also maintains a vector called the **Internal Cell State** which conceptually describes the information that was chosen to be retained by the previous LSTM recurrent unit. A Long Short Term Memory Network consists of four different gates for different purposes as described below: -

- **Forget Gate(f):** It determines to what extent to forget the previous data.
- **Input Gate(i):** It determines the extent of information be written onto the Internal Cell State.
- **Input Modulation Gate(g):** It is often considered as a sub-part of the input gate and much literature on LSTM's does not even mention it and assume it is inside the Input gate. It is used to modulate the information that the Input gate will write onto the Internal State Cell by adding non-linearity to the information and making the information **Zero-mean**. This is done to reduce the learning time as Zero-mean input has faster convergence. Although this gate's actions are less important than the others and are often treated as a finesse-providing concept, it is good practice to include this gate in the structure of the LSTM unit.
- **Output Gate(o):** It determines what output (next Hidden State) to generate from the current Internal Cell State.

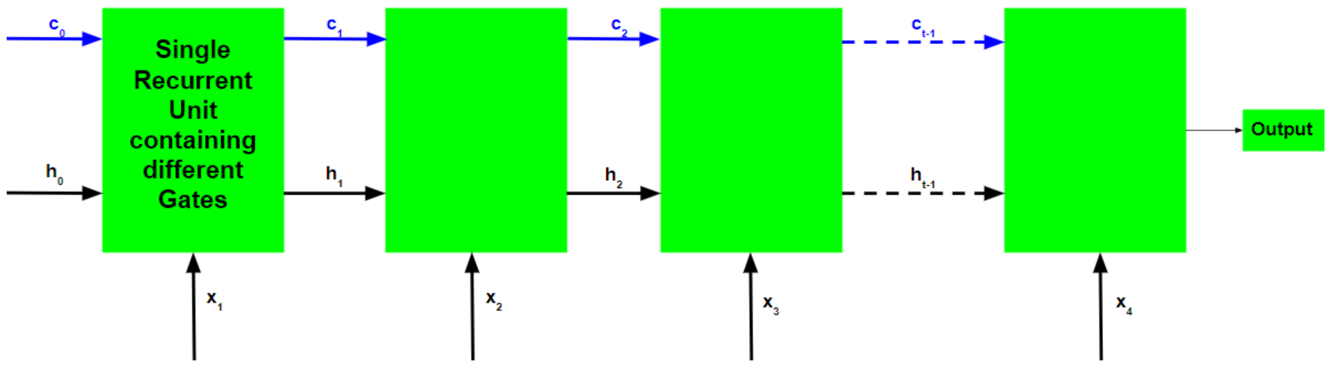
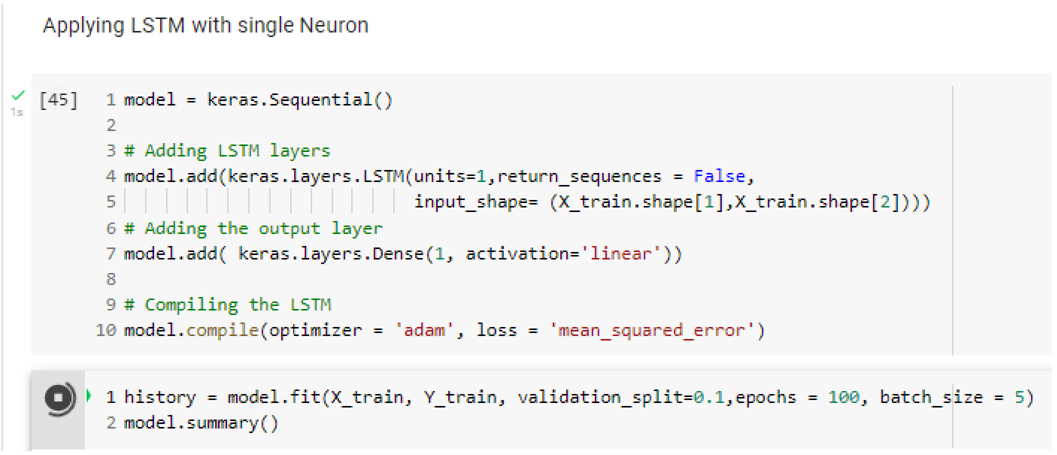


Figure 12: Long Short Term Memory Network(LSTM)

The basic workflow of a Long Short Term Memory Network is similar to the workflow of a Recurrent Neural Network with the only difference being that the Internal Cell State is also passed forward along with the Hidden State.

4.3.3.1 Application of LSTM using single Neuron



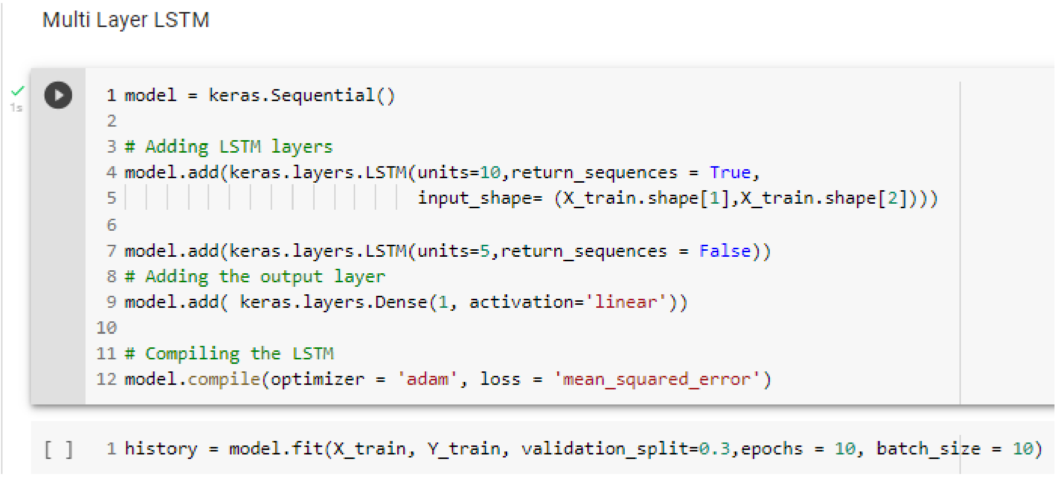
Applying LSTM with single Neuron

```
[45]
model = keras.Sequential()

# Adding LSTM layers
model.add(keras.layers.LSTM(units=1,return_sequences = False,
                            input_shape= (X_train.shape[1],X_train.shape[2])))
# Adding the output layer
model.add( keras.layers.Dense(1, activation='linear'))

# Compiling the LSTM
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

```
history = model.fit(X_train, Y_train, validation_split=0.1,epochs = 100, batch_size = 5)
model.summary()
```

Figure 13: Application of LSTM using Single Neuron

4.3.3.2 Application of Multi-Layer LSTM



Multi Layer LSTM

```
model = keras.Sequential()

# Adding LSTM layers
model.add(keras.layers.LSTM(units=10,return_sequences = True,
                            input_shape= (X_train.shape[1],X_train.shape[2])))

model.add(keras.layers.LSTM(units=5,return_sequences = False))
# Adding the output layer
model.add( keras.layers.Dense(1, activation='linear'))

# Compiling the LSTM
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

```
[ ]
history = model.fit(X_train, Y_train, validation_split=0.3,epochs = 10, batch_size = 10)
```

Figure 14: Application of Multi-Layer LSTM

4.3.4 ARIMA

A popular and widely used statistical method for time series forecasting is the ARIMA model.ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a class of model that captures a suite of different standard temporal structures in time series data.

An ARIMA model is a class of statistical models for analyzing and forecasting time series data.It explicitly caters to a suite of standard structures in time series data, and as such provides a simple yet powerful method for making skillful time series forecasts. ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average and adds the notion of integration. This acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

AR: *Auto regression*. A model that uses the dependent relationship between an observation and some number of lagged observations.

I: *Integrated*. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.

MA: *Moving Average*. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Each of these components are explicitly specified in the model as a parameter. A standard notation is used of ARIMA(p, d, q) where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used.

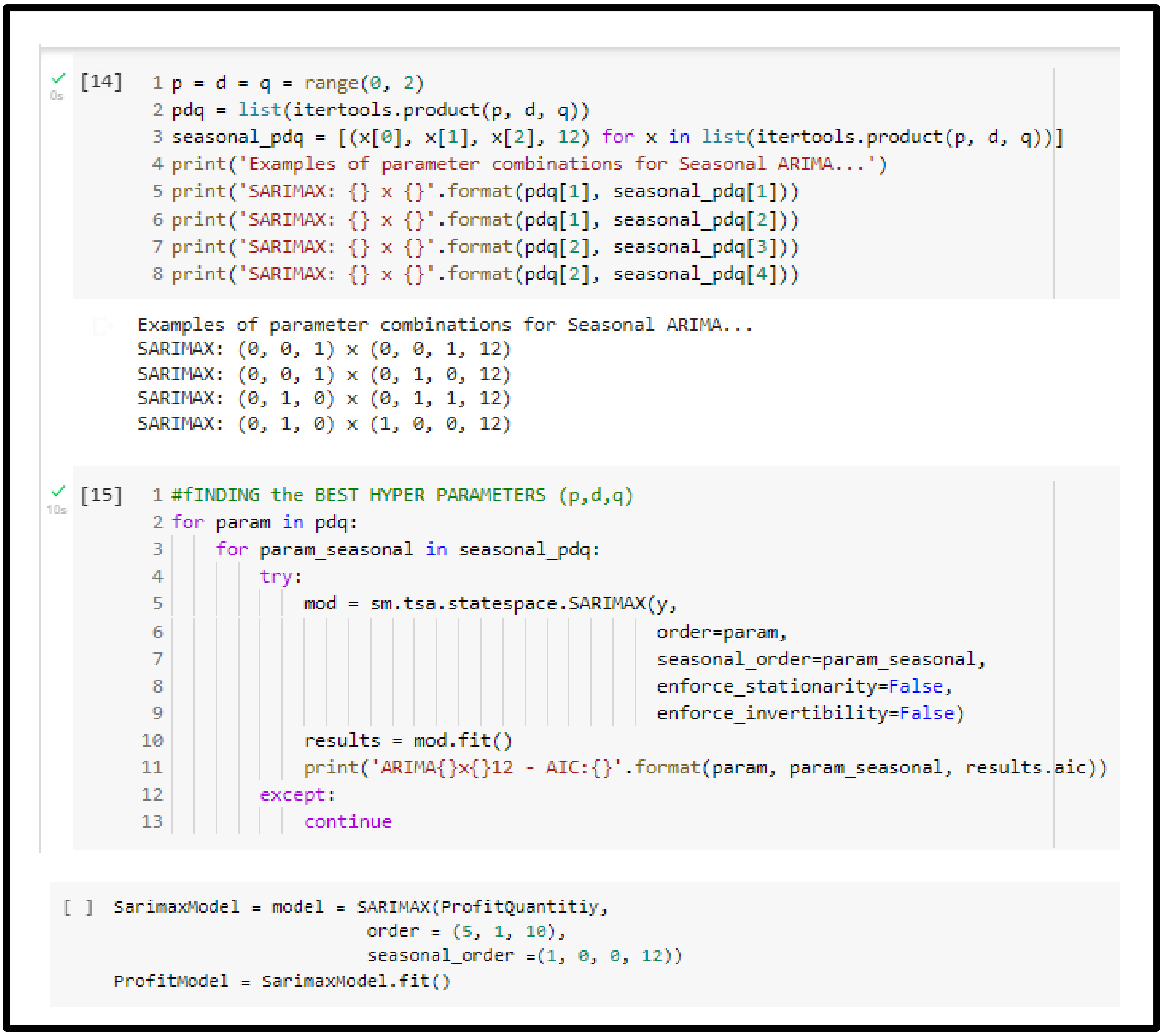
The parameters of the ARIMA model are defined as follows:

- **p**: The number of lag observations included in the model, also called the lag order.
- **d**: The number of times that the raw observations are differenced, also called the degree of differencing.
- **q**: The size of the moving average window, also called the order of moving average.

A linear regression model is constructed including the specified number and type of terms, and the data is prepared by a degree of differencing in order to make it stationary, i.e. to remove trend and seasonal structures that negatively affect the regression model.

A value of 0 can be used for a parameter, which indicates to not use that element of the model. This way, the ARIMA model can be configured to perform the function of an ARMA model, and even a simple AR, I, or MA model.

Adopting an ARIMA model for a time series assumes that the underlying process that generated the observations is an ARIMA process. This may seem obvious, but helps to motivate the need to confirm the assumptions of the model in the raw observations and in the residual errors of forecasts from the model.



```
[14]
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

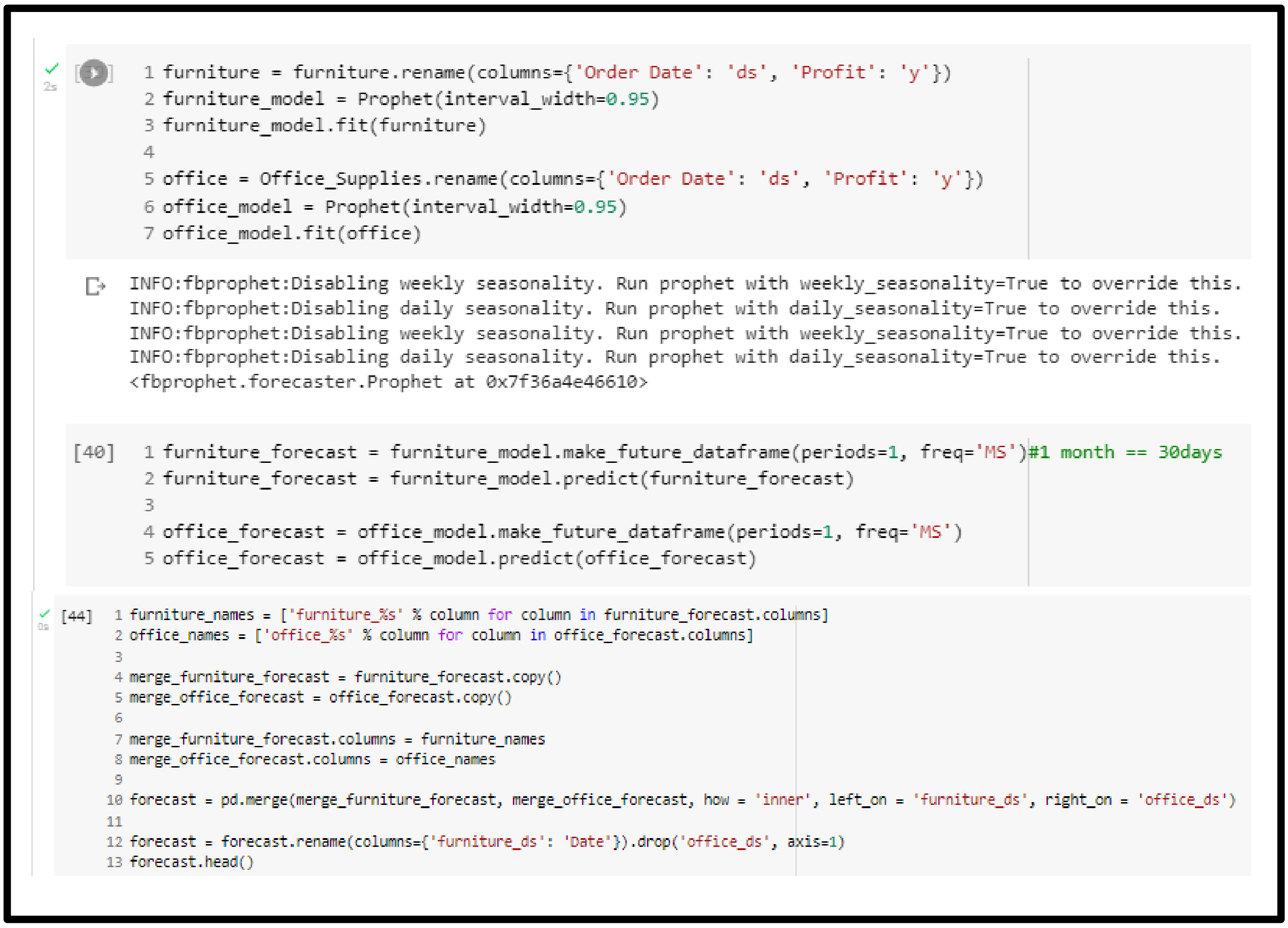
```
[15]
#fINDING the BEST HYPER PARAMETERS (p,d,q)
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)
            results = mod.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
        except:
            continue
```

```
[ ] SarimaxModel = model = SARIMAX(ProfitQuantitiy,
                                   order = (5, 1, 10),
                                   seasonal_order =(1, 0, 0, 12))
    ProfitModel = SarimaxModel.fit()
```

Figure 15: Application of ARIMA

4.3.5 FBProphet

Prophet is an open source library published by Facebook that is based on **decomposable (trend+seasonality+holidays) models**. It provides us with the ability to make time series predictions with good accuracy using simple intuitive parameters and has support for including impact of custom seasonality and holidays. When a forecasting model doesn’t run as planned, we want to be able to tune the parameters of the method with regards to the specific problem at hand. Tuning these methods requires a thorough understanding of how the underlying time series models work. The first input parameters to automated ARIMA, for instance, are the maximum orders of the differencing, the auto-regressive components, and the moving average components. A typical analyst will not know how to adjust these orders to avoid the behavior and this is the type of expertise that is hard to acquire and scale.



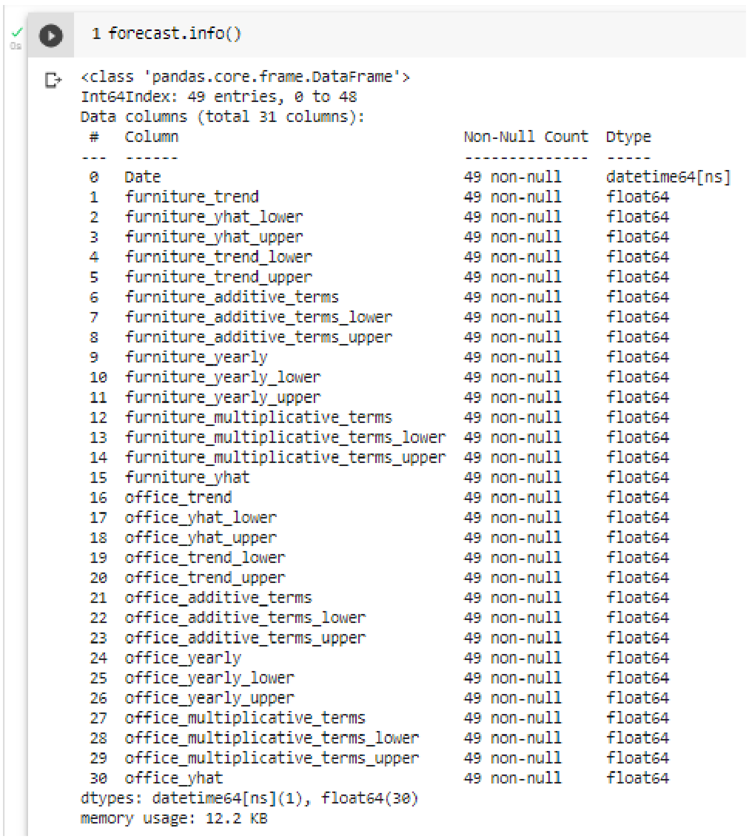
```
[ ]
furniture = furniture.rename(columns={'Order Date': 'ds', 'Profit': 'y'})
furniture_model = Prophet(interval_width=0.95)
furniture_model.fit(furniture)

office = Office_Supplies.rename(columns={'Order Date': 'ds', 'Profit': 'y'})
office_model = Prophet(interval_width=0.95)
office_model.fit(office)
```

```
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
<fbprophet.forecaster.Prophet at 0x7f36a4e46610>
```

```
[40]
furniture_forecast = furniture_model.make_future_dataframe(periods=1, freq='MS')#1 month == 30days
furniture_forecast = furniture_model.predict(furniture_forecast)

office_forecast = office_model.make_future_dataframe(periods=1, freq='MS')
office_forecast = office_model.predict(office_forecast)
```

```
[44]
furniture_names = ['furniture_%s' % column for column in furniture_forecast.columns]
office_names = ['office_%s' % column for column in office_forecast.columns]

merge_furniture_forecast = furniture_forecast.copy()
merge_office_forecast = office_forecast.copy()

merge_furniture_forecast.columns = furniture_names
merge_office_forecast.columns = office_names

forecast = pd.merge(merge_furniture_forecast, merge_office_forecast, how = 'inner', left_on = 'furniture_ds', right_on = 'office_ds')

forecast = forecast.rename(columns={'furniture_ds': 'Date'}).drop('office_ds', axis=1)
forecast.head()
```

Figure 16: Application of Facebook Prophet



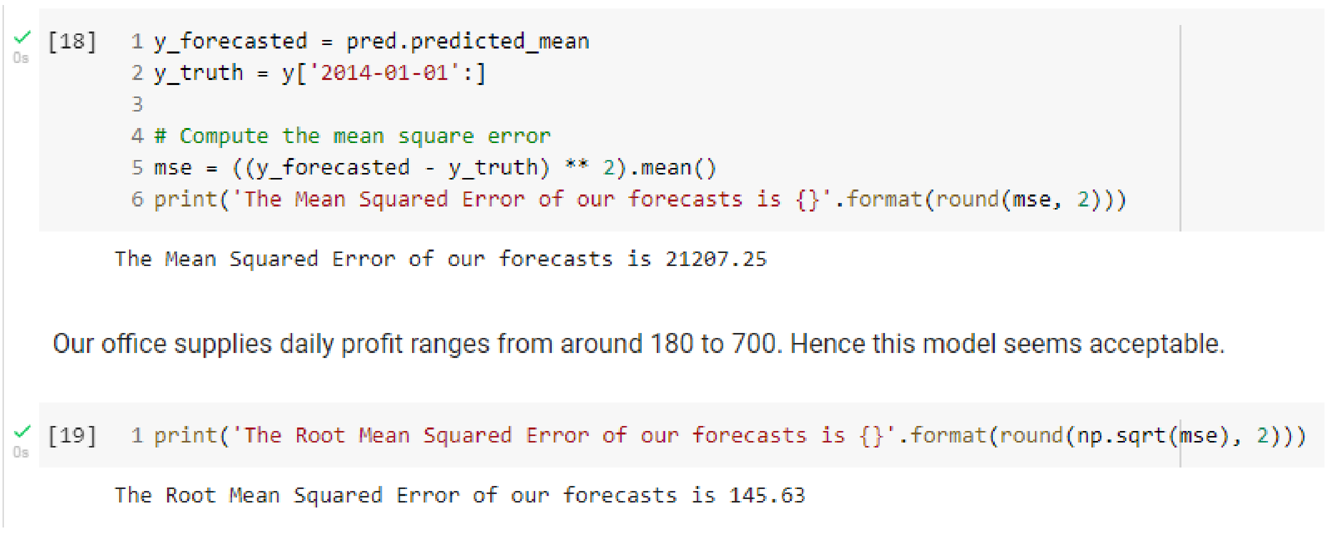
```
1 forecast.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 49 entries, 0 to 48
Data columns (total 31 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   Date                                 49 non-null     datetime64[ns]
 1   furniture_trend                      49 non-null     float64
 2   furniture_yhat_lower                 49 non-null     float64
 3   furniture_yhat_upper                 49 non-null     float64
 4   furniture_trend_lower                49 non-null     float64
 5   furniture_trend_upper                49 non-null     float64
 6   furniture_additive_terms             49 non-null     float64
 7   furniture_additive_terms_lower       49 non-null     float64
 8   furniture_additive_terms_upper       49 non-null     float64
 9   furniture_yearly                     49 non-null     float64
 10  furniture_yearly_lower               49 non-null     float64
 11  furniture_yearly_upper               49 non-null     float64
 12  furniture_multiplicative_terms       49 non-null     float64
 13  furniture_multiplicative_terms_lower 49 non-null     float64
 14  furniture_multiplicative_terms_upper 49 non-null     float64
 15  furniture_yhat                       49 non-null     float64
 16  office_trend                         49 non-null     float64
 17  office_yhat_lower                    49 non-null     float64
 18  office_yhat_upper                    49 non-null     float64
 19  office_trend_lower                   49 non-null     float64
 20  office_trend_upper                   49 non-null     float64
 21  office_additive_terms                49 non-null     float64
 22  office_additive_terms_lower          49 non-null     float64
 23  office_additive_terms_upper          49 non-null     float64
 24  office_yearly                        49 non-null     float64
 25  office_yearly_lower                  49 non-null     float64
 26  office_yearly_upper                  49 non-null     float64
 27  office_multiplicative_terms          49 non-null     float64
 28  office_multiplicative_terms_lower    49 non-null     float64
 29  office_multiplicative_terms_upper    49 non-null     float64
 30  office_yhat                          49 non-null     float64
dtypes: datetime64[ns](1), float64(30)
memory usage: 12.2 KB
```

Figure 17: New Columns generated upon application of FBProphet

4.3.6 Metrics for Data Modelling

The error measurement is an important metric in the estimation period. Root mean squared error (RMSE) and Mean Absolute Error (MAE) have been used for continuous variable's accuracy measurement. It can be said that the average model prediction error can be expressed in units of the variable of interest by using both MAE and RMSE. MAE is the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight. The square root of the average of squared differences between prediction and actual observation can be termed as RMSE. RMSE is an absolute measure of fit. RMSE helps in measuring the variable's average error and it is also a quadratic scoring rule. Low RMSE values obtained for linear or multiple regression correspond to better model fitting. As per the best fit model, **MSE = 21207.25 and RMSE = 145.63**.



```
[18]
y_forecasted = pred.predicted_mean
y_truth = y['2014-01-01':]

# Compute the mean square error
mse = ((y_forecasted - y_truth) ** 2).mean()
print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

```
The Mean Squared Error of our forecasts is 21207.25
```

Our office supplies daily profit ranges from around 180 to 700. Hence this model seems acceptable.

```
[19]
print('The Root Mean Squared Error of our forecasts is {}'.format(round(np.sqrt(mse), 2)))
```

```
The Root Mean Squared Error of our forecasts is 145.63
```

Figure 18: MSE and RMSE values of ARIMA model

6. Implementation and Results

In this section, the programming language, libraries, implementation platform along with the data modeling and the observations and results obtained from it have been discussed.

6.1 Implementation Platform and Language

Python is a general purpose, interpreted-high level language used extensively nowadays for solving domain problems instead of dealing with complexities of a system. It is also termed as the 'batteries included language' for programming. It has various libraries used for scientific purposes and inquiries along with number of third-party libraries for making problem solving efficient.
In this work, the Python libraries of NumPy, for scientific computation, and Matplotlib, for 2D plotting have been used. Along with this, Pandas tool of Python has been employed for carrying out data analysis. As a development platform, Google Colab Notebook, which proves to work great due to its excellence in 'literate programming', where human friendly code is punctuated within code blocks, has been used.

6.2 Data Modeling and Observations

The relation between 'Order Date' column and the 'Profit' column has been used to estimate a forecast of Profit generation in the future. The best chosen models were ARIMA and Facebook Prophet. Conclusions derived as per the best models have been brought to light under the heading 'Results and Finding'

6.3 RESULTANDFINDING

6.3.1 FINDINGS FROM ARIMA MODEL

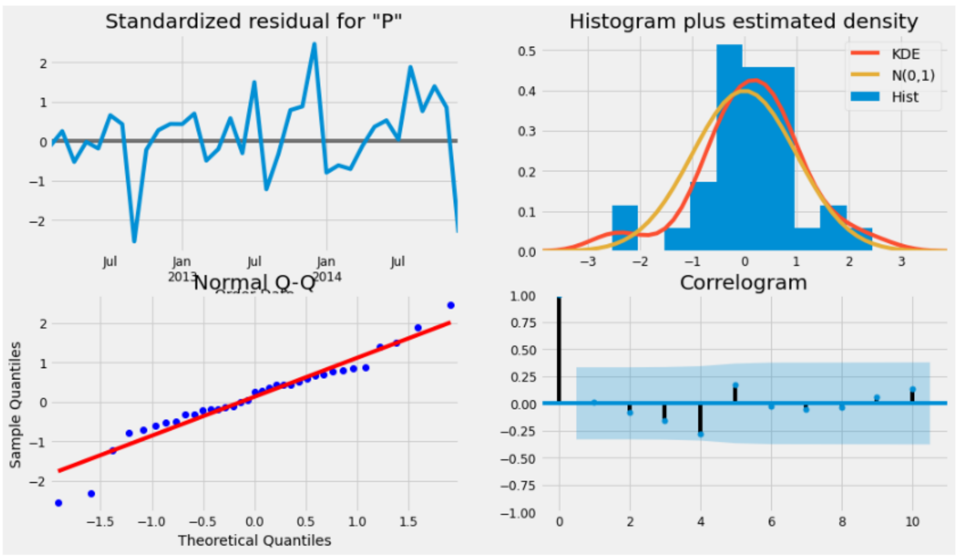


Figure 19: ARIMA model diagnostics

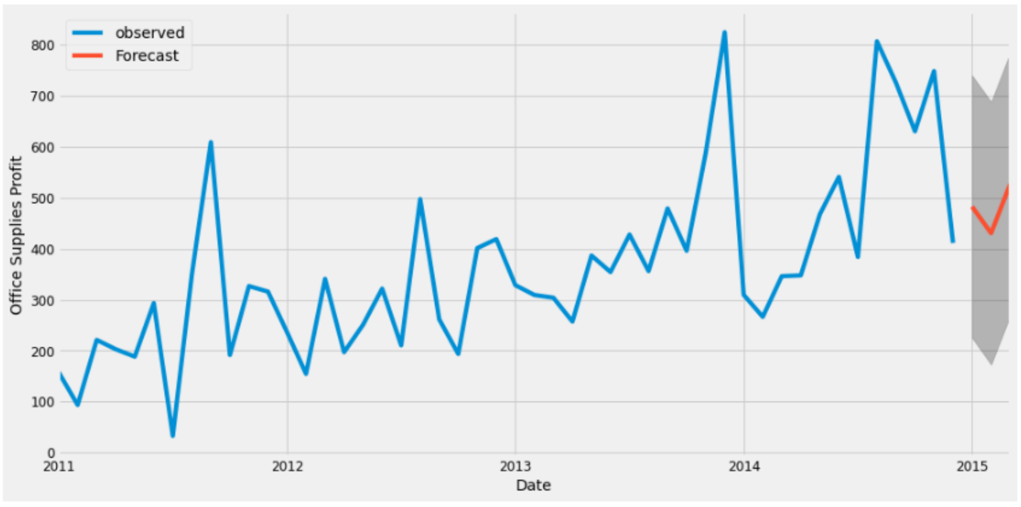


Figure 20: Forecast for next 3 months

6.3.2 Findings from the Facebook Prophet Model

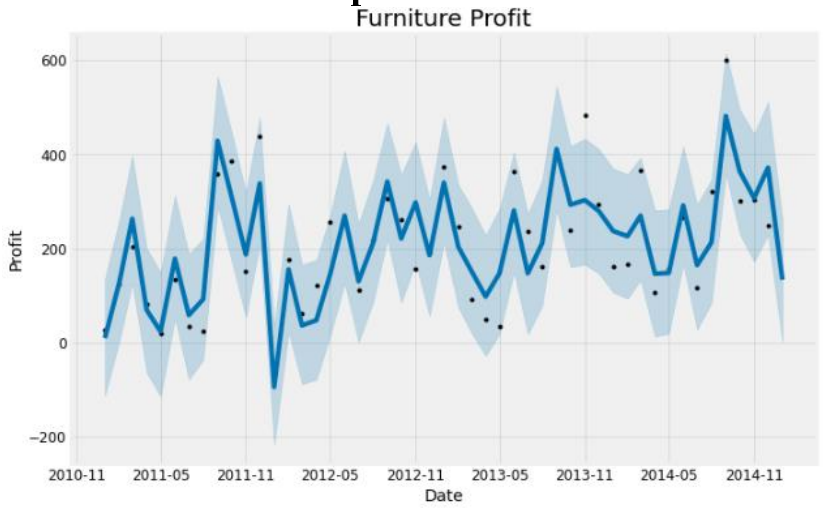


Figure 21: Forecast for the product category 'Furniture' for one month after 31 Dec 2014 predicted after the model training from year 2011 to year 2014.

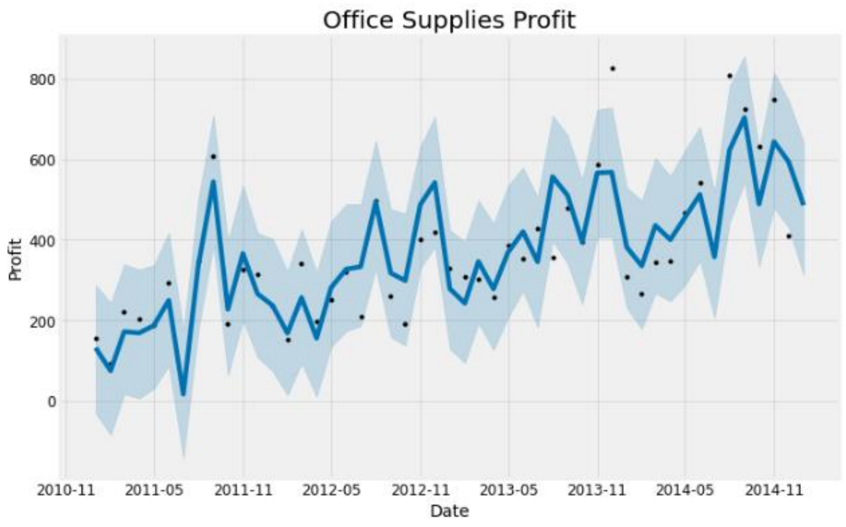


Figure 22: Forecast for the product category 'Office Supplies' for one month

after 31 Dec 2014 predicted after the model training from year 2011 to year 2014.

6. System Requirement Specification

7.1 Hardware Requirement:

- 500 GB hard drive (minimum requirement)
- 5 GB RAM (Minimum requirement)
- PC x64-bit CPU

7.2 Software Requirement:

- Windows/Mac/Linux
- Python-3.9.12
- VS Code/Anaconda/Jupyter
- **Libraries:**
  a) Pandas
  b) Numpy
  c) Prophet
  d) Statsmodel
  e) Matplotlib
  f) Sklearn
  g) Tensorflow
  h) Keras
  i) TBATS

- Any Modern Web Browser like Google Chrome
  To access the web application (Google Colab)

8. CONCLUSION AND FUTURE SCOPE

The project titled ‘Global Superstore Profit Prediction’ contains to prominent columns: ‘Order Date’ and ‘Profit’. The objective of the project was to predict the Profit generated by the Superstore for a certain future time period. Clearly, this is a time series based forecasting. Since multiple rows of same date with profit being generated for different products were found, a decision was taken to take mean of the Profit for each day and consolidate the multiple similar date columns into a single row. These newly generated rows were grouped as per the product category, namely, ‘Office Supplies’, ‘Furniture’ and ‘Technology’. The category, ‘Office Supplies’ having the highest number of records among the three categories, was chosen for making future Profit prediction. Though Random Forest, Extreme Gradient boost and LSTM models were implemented upon the dataset, the models best chosen for prediction were ARIMA and Facebook Prophet. Four years’ data from year 2011 to year 2014 was used as historical data for training of the models and the prediction period was set to 1 month and 3 months. The results were plotted. These models have not only helped in visualizing the yearly trends of three different product Categories sold by the superstore but have also given a valuable insight into the future.

In the future, the accuracy of the models can be improved by applying more performance measurements and other machine learning algorithms for more accurate results.

This study can help the manufacturing industries to predict profit that can be earned from different categories of products, making investment decisions, setting target customers, conducting lookahead planning, and estimating expenditure for manufacturing, transportation and labor.

REFERENCES

- https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-arima-in-python-3
- https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-prophet-in-python-3
- https://www.kaggle.com/datasets/apoorvaappz/global-super-store-dataset
- https://www.kaggle.com/code/tanmaygangurde/global-superstore-ml/notebook