

# **FORECAST THE SALES FOR A LARGE GROCERY CHAIN CORPORACIÓN FAVORITA**

By

**V N S R GANESH ATMURI (40AIML646-21/1)**

atmuriganesh17@gmail.com

## **ABSTRACT**

I have taken part in the Store Sales- Time Forecasting competition hosted on Kaggle. We presented comprehensive research and a solution to the underlying machine learning problem based on time series data, where the key issues were addressed and applicable methodologies were given. Our methodology is an ensembled Machine Learning strategy based on decision trees that use a gradient boosting framework. Using this strategy, a vast amount of time series data may eventually be handled with respectable speed and accuracy. This study, we feel, may serve as a review and guideline for the time series forecasting standard, motivating future attempts and research.

# INDEX

	PAGE NO.
<b>CHAPTER 1: INTRODUCTION</b>	
1.1 Introduction	5
1.2 Data Overview	5
1.3 Performance Metrics	10
1.4 Real-World Challenges and constraints	11
1.5 Problem Solved in Literature	12
<b>CHAPTER 2: EXPLORATORY DATA ANALYSIS (EDA)</b>	
2.1 What is EDA?	13
2.2 Analyzing Datasets	14
2.3 Handling Categorical Values	16
2.4 Handling Missing Values	18
2.5 Univariate Analysis	19
2.6 Bivariate Analysis	20
2.7 Multivariate Analysis	22
2.8 Feature Selection	24
<b>CHAPTER 3: MODELING AND ERROR ANALYSIS</b>	
3.1 Machine Learning Models	25
3.2 Time Series Forecasting	31
<b>CHAPTER 4: ADVANCED MACHINE LEARNING AND FEATURE ENGINEERING</b>	
4.1 Stacking	33
4.2 Deep Learning Hyperparameter Optimization	34
4.3 Long Short Term Memory (LSTM)	35
4.4 Error Analysis	37

## **CHAPTER 5: MODEL DEPLOYMENT**

5.1 Ways to Deploy Model	38
5.2 Libraries used to deploy model	39
5.3 Demo of Web API	40
5.4 Architecture of Heroku Deployment`	41
<b>References</b>	<b>44</b>

# CHAPTER 1: INTRODUCTION

## 1.1 Introduction:

For most supermarkets, sales forecasting is the most important factor. This provides information on future earnings, budgeting, and marketing. We had found out which model is best and produces reliable predictions as we are dealing with Corporación Favorita data which is an Ecuadorian grocery retailer located in Quito, with more than a hundred grocery stores all over the country with quite a few types of products such as Grocery I, Bread/Bakery, Produce, etc.,

Product promotions, seasonal needs, and: other factors all have an impact on sales. For example, from 2013 to 2017, sales increased towards the end of the year and on the first day of the year, i.e., January 1st the sales plummet in an instant. The competition aimed at predicting the future sales at the product level, based on historical data of 54 stores in different states and cities.

## 1.2 Data Overview:

Data was collected from the year 2013 to 2017 for 54 stores. We are tasked to predict the sales with sales for each store. We are provided with datasets: train.csv, test.csv, oil.csv, transactions.csv, holidays\_events.csv, and stores.csv

### Dataset Size

**train dataset:** 3000888 rows (477 MB)

**test dataset:** 28512 rows (4.3 MB)

**oil dataset:** 1218 rows (89.3 KB)

**transactions dataset:** 83488 rows (6.6 MB)

**holidays\_events dataset:** 350 rows (115.9 KB)

**stores dataset:** 54 rows (10.9 KB)

## Column Descriptions

### Train Dataset:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000888 entries, 0 to 3000887
Data columns (total 6 columns):
#   Column      Dtype
---  -
0   id          int64
1   date        object
2   store_nbr   int64
3   family      object
4   sales       float64
5   onpromotion float64
dtypes: float64(2), int64(2), object(2)
memory usage: 477.2 MB
```

Table 1: The data types of columns, range of train dataset.

**id:** Identifier

**date:** Sales date from 1<sup>st</sup> January 2013 to 15<sup>th</sup> August 2017.

**store\_nbr:** The store number. Range 1-54

**family:** Product / Item Classification, 33 unique categories.

**sales:** Sales that are specified at the date-store-item level

**onpromotion:** Expense of Item / Product promotion.

From Table 1, we can check the data type of each feature and also the memory that has been used for the train dataset moreover there are no null values present in the dataset. The **id**, **date**, and **family** are uniformly distributed. The **date** column is of object type, so we need to convert it into DateTime format. The **sales** column has 939130 (31.3%) and **onpromotion** 1363183 (45.4%) of zeroes. The **sales** and **onpromotion** columns are positively correlated. Since the dataset is a bit large, it is difficult to combine with other datasets and also while training the model.

## Test Dataset:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28512 entries, 0 to 28511
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   id               28512 non-null  int64
1   date             28512 non-null  object
2   store_nbr        28512 non-null  int64
3   family           28512 non-null  object
4   onpromotion      28512 non-null  float64
dtypes: float64(1), int64(2), object(2)
memory usage: 4.3 MB
```

Table 2: The data types of columns, range of test dataset.

**id:** Identifier

**date:** Sales date from 1<sup>st</sup> January 2013 to 15<sup>th</sup> August 2017.

**store\_nbr:** The store number. Range 1-54

**family:** Product / Item Classification, 33 unique categories.

**onpromotion:** Expense of Item / Product promotion.

From Table 2, we can check the data type of each feature and also the memory that has been used for the test dataset there are no null values present in the dataset. We need to predict the sales column using these features.

## Holidays & Events dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 350 entries, 0 to 349
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  -
0   date             350 non-null   object
1   type             350 non-null   object
2   locale           350 non-null   object
3   locale_name      350 non-null   object
4   description      350 non-null   object
5   transferred      350 non-null   bool
dtypes: bool(1), object(5)
memory usage: 115.9 KB
```

Table 3: The data types of columns, range of holidays & events dataset.

**date:** Holiday date

**type:** Types of holidays namely ['Holiday', 'Event', 'Additional', 'Transfer', 'Bridge']

**locale:** Types of local holiday namely ['Regional', 'Local']

**locale\_name:** Name of a local holiday in a specific location

**description:** Description of the holiday

**transferred:** whether the day is a transferred holiday.

From Table 3, we can check the data type of each feature and also the memory that has been used for the holidays & events dataset and there are no null values present in the dataset.

### Oil dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1218 entries, 0 to 1217
Data columns (total 2 columns):
#   Column      Non-Null Count  Dtype
---  -
0   date         1218 non-null   object
1   dcoilwtico   1175 non-null   float64
dtypes: float64(1), object(1)
memory usage: 89.3 KB
```

Table 4: The data types of columns, range of oil dataset.

**date:** oil price of a particular date

**dcoilwtico:** oil price

From Table 4, we can check the data type of each feature and also the memory that has been used for the oil dataset moreover there are null values present in the dataset.

### Stores dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54 entries, 0 to 53
Data columns (total 5 columns):
#   Column      Non-Null Count  Dtype
---  -
0   store_nbr   54 non-null     int64
1   city        54 non-null     object
2   state       54 non-null     object
3   type        54 non-null     object
4   cluster     54 non-null     int64
dtypes: int64(2), object(3)
memory usage: 10.9 KB
```

Table 5: The data types of columns, range of stores dataset.

**store\_nbr:** The store number. Range 1-54.



**city:** City where the store located

**state:** State where the store located

**type:** Store classification namely ['A', 'B', 'C', 'D']

**cluster:** internal store clustering. Range 1-16.

From Table 4, we can check the data type of each feature and also the memory that has been used for the store's dataset there are no null values present in the dataset.

### Transactions dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83488 entries, 0 to 83487
Data columns (total 3 columns):
#   Column          Non-Null Count  Dtype
---  -
0   date             83488 non-null  object
1   store_nbr        83488 non-null  int64
2   transactions     83488 non-null  int64
dtypes: int64(2), object(1)
memory usage: 6.6 MB
```

Table 5: The data types of columns, range of transactions dataset.

**date:** Transaction date

**store\_nbr:** The store number. Range 1-54.

**transactions:** Number of transactions

From Table 5, we can check the data type of each feature and also the memory that has been used for the transactions dataset and there are no null values present in the dataset.

It is a regression problem where we are given time series (from 01-01-2013 to 15-08-2017) data with multiple categorical and numerical features. We are tasked to predict the sales of each store from 16-08-2017 to 31-08-2017. While working with this dataset, we came across several libraries, including Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn (Sklearn), Plotly, Tensorflow, Keras-Tuner, HyperOpt, Missingno, and Warnings. NumPy and Pandas are used to process and manipulate

data, while Matplotlib, Seaborn, Plotly, and Missingno are used to visualize the data. Sklearn, Tensorflow, and Keras are used to import libraries, train the model, and determine how well the model is working (i.e., error rate), and HyperOpt and Keras-Tuner are used to tune the model for more accurate results.

### 1.3 Performance Metrics:

A business metric is a numerical value that is used to measure, track, and analyze a model's performance or failure. Because the target variable is of the regression type, we have Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), and R-squared (R2) values, as well as other extensions of the mean absolute error measure.

From the available options, we can consider Root Mean Squared Error to check the root squared of error between the predicted and actual values, and it handles outliers well. As we already know, there are sudden drops or rises in sales that act as outliers. So, using Root Mean Squared Error overcomes this issue. Figure 6 is the mathematical formula used to calculate the Root Mean Squared Error, where  $\hat{y}_i$  represents the predicted values from the model, and  $y_i$  are the actual values. The greater the value of RMSE, the worse the model performs. The number should be closer to 0. Figure 7 explains the code snippet of Root Mean Squared Error, where the  $y\_true$  represents the  $y_i$  and  $\hat{y}_i$  represents the predicted values from the model.

$$RMSE = \sqrt{\sum_{i=1}^n \frac{(\hat{y}_i - y_i)^2}{n}}$$

$\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n$  are predicted values

$y_1, y_2, \dots, y_n$  are observed values

$n$  is the number of observations

Figure 6: Formula for RMSE

```
In [9]: def rmse(y_true,y_pred):  
        return np.sqrt(np.square(np.subtract(y_true,y_pred)).mean())
```

Figure 7: RMSE Code

The reason for not using MAE as a metric is that it is more robust to outliers than RMSE. In a regression model, the R-squared value reflects the proportion of the variation of the dependent variable that is explained by an independent variable (or variables). MAPE addresses under and over forecasting differently. If MAPE is greater than 100 percent, the mistakes are substantially bigger than the actual values, whereas MAPE is less than 100 percent for under predictions. As a result, MAPE does not appear to be the best option. Figure 8 is the mathematical formula used to calculate the R-Squared Value, where  $\hat{y}_i$  represents the predicted values from the model, and  $y_i$  are the actual values.  $SS_{RES}$  means the sum of squared residuals and  $SS_{TOT}$  means the total sum of squares.

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

Figure 8: Formula for R-Squared

The greater the R-Squared number, the better, i.e., the value should be closer to 1. If the R-Squared value is low but the independent variables are statistically significant, we may still make inferences about the connections between the variables, and the regression model may be the best fit for the given dataset. Figure 9 explains the code snippet of R Squared Value, where the  $y_{true}$  represents the  $y_i$  and  $\hat{y}_i$  represents the predicted values from the model,  $rss$  means the sum of squared residuals and  $sst$  is the total sum of squares.

```
In [10]: def rsquared(y_true, y_pred):
          rss = np.sum(np.square((y_true - y_pred)))
          mean = np.mean(y_true)
          sst = np.sum(np.square(y_true - mean))
          r_square = 1 - (rss/sst)
          return r_square
```

Figure 9: Code snippet for R Squared Value

## 1.4 Real-World Challenges and constraints:

In real-time, the datasets in the Corporación Favorita individually include irregular data points, which causes challenges when merging and worsens the training and

testing processes, reducing prediction accuracy. Together with the irregularization of data, there are many unwanted features in the data that take up memory space and affect the prediction process' run time, leading to redundancy or a low accuracy rate. To solve this difficulty, we may start by deleting irrelevant data that has nothing to do with the predation process, then standardizing or normalizing the data to make it useful.

## **1.5 Problems Solved in Literature**

We're working with Corporacion Favorita, which is a time-series sales forecasting, regression, and deep learning model. When the problem is time-series sales forecasting, we may use ARIMA, Random Forest, and MARS for regression, LSTM for deep learning, and a basic RNN model for deep learning.

## CHAPTER 2: EXPLORATORY DATA ANALYSIS (EDA)

### 2.1 What is EDA?

This is an unavoidable and critical step in fine-tuning the given dataset in a different form of analysis to understand the insights of key characteristics of various entities of the dataset, such as column, rows, and data visualization packages by using Pandas, NumPy, Statistical Methods, and data visualization packages.

#### Outcome of this phase as below

- Comprehend the provided dataset and assist in its clean up.
- It provides a clear image of the features and their relationships.
- Developing criteria for important variables while excluding or deleting unnecessary features.
- Handling missing values or human error.
- Identifying the outliers
- EDA process would maximize a dataset's insight

This method is time-consuming but extremely successful. The following tasks are completed during this phase; they will vary depending on the available data and client acceptance.

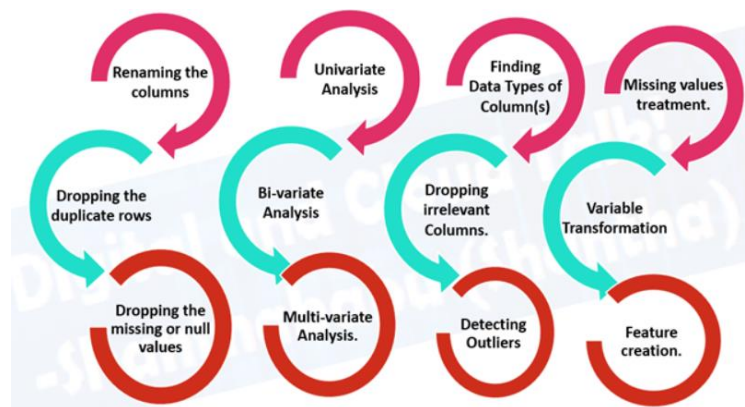


Figure 10: Steps involved in EDA

From Figure 10, we can see the steps that are involved in the process of Exploratory Data Analysis, and the process differs based on the dataset we are handling with. The examples of EDA are shown in Figure 11 and 12.

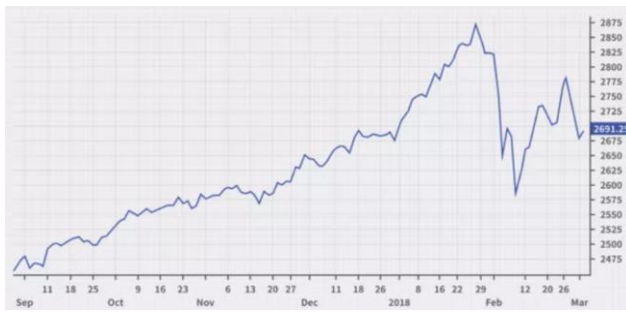


Figure 11: Line Chart

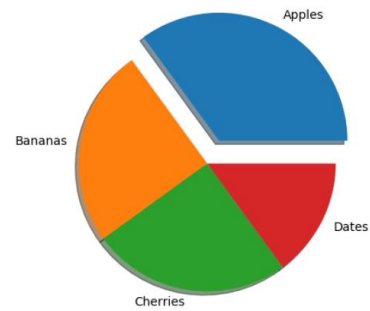


Figure 12: Pie Chart

## 2.2 Analyzing Datasets:

Here, we can obtain more detailed information of each dataset, and methods that need to apply to process data such as normalizing (or) standardizing (or) handling the missing values (or) converting the categorical features into numerical features.

Figure 13 explains the information that is extracted from the train dataset, the train dataset consists of 30 lakhs entries (rows) and 6 features (variable), where there are 4 Numerical features (**id**, **store\_nbr**, **sales**, and **onpromotion**), 1 DateTime feature (**date**), and 1 Categorical feature (**family**).

Dataset statistics		Variable types	
Number of variables	6	Numeric	4
Number of observations	3000888	DateTime	1
Missing cells	0	Categorical	1
Missing cells (%)	0.0%		
Duplicate rows	0		
Duplicate rows (%)	0.0%		
Total size in memory	308.4 MiB		
Average record size in memory	107.8 B		

Figure 13: Train Dataset Information

Figure 14 explains the information extracted from the test dataset, this dataset consists of 28 thousand entries (rows) with 5 features: 3 Numerical features (**id**, **store\_nbr**, and **onpromotion**), 1 DateTime feature (**date**), and 1 Categorical feature (**family**).

Dataset statistics		Variable types	
Number of variables	5	Numeric	3
Number of observations	28512	DateTime	1
Missing cells	0	Categorical	1
Missing cells (%)	0.0%		
Duplicate rows	0		
Duplicate rows (%)	0.0%		
Total size in memory	2.7 MiB		
Average record size in memory	99.8 B		

Figure 14: Test Dataset Information

Figure 15 explains the information extracted from the stores dataset, this dataset consists of 56 entries (rows) with 5 features: 2 Numerical features (**store\_nbr** and **cluster**) and 3 Categorical features (**city**, **state**, and **type**).

Dataset statistics		Variable types	
Number of variables	5	Numeric	2
Number of observations	54	Categorical	3
Missing cells	0		
Missing cells (%)	0.0%		
Duplicate rows	0		
Duplicate rows (%)	0.0%		
Total size in memory	10.9 KiB		
Average record size in memory	206.1 B		

Figure 15: Stores Dataset Information

Figure 16 explains the information extracted from the transaction dataset, this dataset consists of 83 thousand entries (rows) with 3 features: 1 DateTime feature (**date**) and 2 Numeric features (**store\_nbr** and **transactions**).

Dataset statistics		Variable types	
Number of variables	3	DateTime	1
Number of observations	83488	Numeric	2
Missing cells	0		
Missing cells (%)	0.0%		
Duplicate rows	0		
Duplicate rows (%)	0.0%		
Total size in memory	1.9 MiB		
Average record size in memory	24.0 B		

Figure 16: Transactions Dataset Information

Figure 17 explains the information extracted from the oil dataset, this dataset consists of 83 thousand entries (rows) with 3 features: 1 DateTime feature (**date**) and 1 Numeric

feature (dcoilwtico). There are 43 missing values i.e., 1.8% of the total dataset so we need to tackle this problem in the future before model training & prediction.

Dataset statistics		Variable types	
Number of variables	2	DateTime	1
Number of observations	1218	Numeric	1
Missing cells	43		
Missing cells (%)	1.8%		
Duplicate rows	0		
Duplicate rows (%)	0.0%		
Total size in memory	19.2 KiB		
Average record size in memory	16.1 B		

Figure 17: Oil Dataset Information

Figure 18 explains the information extracted from the holiday and events dataset, this dataset consists of 350 entries (rows) with 6 features: 1 DateTime feature (date), 4 Categorical features (type, locale, locale\_name, and description) and 1 Boolean feature (transferred).

Dataset statistics		Variable types	
Number of variables	6	DateTime	1
Number of observations	350	Categorical	4
Missing cells	0	Boolean	1
Missing cells (%)	0.0%		
Duplicate rows	0		
Duplicate rows (%)	0.0%		
Total size in memory	95.7 KiB		
Average record size in memory	280.0 B		

Figure 18: Holiday & Events Dataset Information

This statistical information is obtained from “**Pandas-profiling**” library.

## 2.3 Handling Categorical Values:

Machine learning models require all the input and output variables to be numeric. This means that if our data contains categorical data or boolean values (i.e., **True** or **False**), we must encode it to numbers before we fit and evaluate a



model. There are multiple ways to handle this categorical data. We first need to identify whether the categorical encoding is a nominal or ordinal encoding.

- **Nominal:** These are variables that are not related to each other in any order such as Color (**Black, Blue, Green**), Gender (**Male** or **Female**)
- **Orinal:** These are variables where a certain order can be found between them such as student grades (**A, B, C, D, Fail**), Educational Qualification (**High School, Bachelors, Graduation, and PhD**)

Figure 19 explains the Encoding Techniques that are available to convert the categorical features to numerical features.

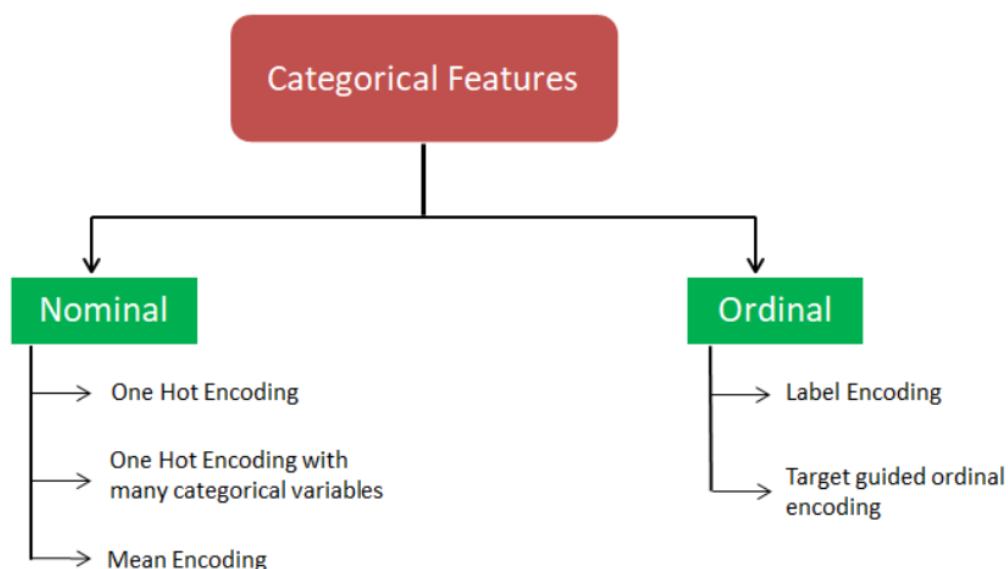


Figure 19: Encoding Techniques

Since our dataset consists of both nominal and ordinal categorical features so we use both Label Encoder and One-Hot Encoding. Each category in the One-Hot Encoding is assigned a value of 1 or 0. In this situation, 0 denotes the absence of the category, but 1 denotes its presence. In this case, dummy variables equal to the number of categories included in the feature are produced, but one column is eliminated to escape the dummy variable trap. i.e., number of columns = number of categories – 1. The advantage of using One-Hot-Encoding is that the output is binary rather than ordinal, and everything is in an orthogonal vector space. The disadvantage of large cardinality is that the feature space can quickly inflate, resulting in the "**curse of dimensionality**". To overcome the problem of the curse of

dimensionality of one-hot encoding we need to check which values are occurring the most and consider a set of values. But the dataset created after merging all the dataset work has equally distributed categorical values. So, we have considered the group average of sales and the categorical variables and considered the top 10 values.

```
In [26]: #considering top 10 highest sold state types since family type is equally distributed in df dataset
state_10=[x for x in df.groupby(['state'])['sales'].mean().sort_values(ascending=False).head(10).index]

In [27]: for i in state_10:
df["s_"+i]=np.where(df['state']==i,1,0)
```

Figure 20: Sample code of One Hot Encoding

Label Encoding is a type of ordinal encoding. Here, it converts each categorical variable to a numerical value of 0 to n-1, where n denotes the number of category types, where numbers are assigned based on alphabetical or based on the number. The advantage of using this method is it can be easily. But if we use this category to a nominal encoding, it gives higher importance to the higher values. To use this method we need to use a library “**from sklearn.preprocessing import LabelEncoder**”. Figure 21 is a sample code of how to use the LabelEncoder library and convert the ordinal values into numerical values.

```
In [30]: labelencoder=LabelEncoder()

In [31]: df['store_type']=labelencoder.fit_transform(df.type)
```

Figure 21: Sample code for Label Encoding

## 2.4 Handling Missing Values:

Missing values are common occurrences in data. Most predictive modeling techniques cannot handle any missing values. Therefore, this problem must be addressed before modeling. Missing/null values in the dataset arise the problem of “**Value Error**” while modeling. While checking the statistics of each dataset, we have seen null values are present in the oil dataset. To overcome this problem we have 5 methods they are:

1. **Deleting Rows:** If the dataset is large, we can drop the feature values or rows from the dataset which leads to loss of information which might affect the model's accuracy.
2. **Replacing with Mean/Median/Mode:** If the dataset is small, for the null values in the dataset, mean/median/mode of the particular feature. Imputing the approximations add variance and bias.
3. **Assigning a unique category:** If the null values are present for the categorical features we can replace the null values with a unique category such as "U" which indicates unknown. It adds less variance and causes poor performance.
4. **Predicting the missing values:** With the help of machine learning, we can forecast the null values by considering that particular feature as a target feature. Unless a missing number is predicted to have a very high variance, this strategy may result in superior accuracy. Bias can also occur when an inadequate conditioning collection is used to condition a category variable.
5. **Replacing with Previous (or) Next Value(Forward (or) Backward Fill):** We impute the null values with the previous (or) next values.

Because we need to deal with missing values in the oil dataset and oil prices are time-dependent, we can only use the last method, Replacing with Previous (or) Next Value, because the oil prices do not differ much within a day, but if we replace them with mean/median/mode values, the values may be too small (or) high. Figure 22 is a code snippet of filling the missing/null values in the `dcoilwtico` feature in the oil dataset.

```
In [6]: oil.dcoilwtico.fillna(method="bfill",inplace=True)
```

Figure 22: Using `bfill` method to fill the null values in `dcoilwtico` feature in oil dataset

## 2.5 Univariate Analysis:

“**Uni**” means one and “**Variate**” means a feature, this method is used once we load a dataset, the main aim of this feature is to derive the data, characteristics of the feature. This quickly identifies the patterns such as central tendency (mean, median, and mode), dispersion (range, variance), quartiles (interquartile range), outliers, and

standard deviation. “**Categorical**” and “**Numerical**” are two types of features that this analysis can handle. This can be described through the following plots:

1. **Distribution plot:** It is a mix of “**histogram**”, “**kernel density**” estimation, or normal curve, and “**rug**” plot that is used to illustrate the parametric distribution of a dataset. This helps to check which type of distribution does the feature is following. Figure 23 explains the distribution of **dcoilwtico** feature in the oil dataset, and it is following **Bi-Modal** Distribution.
2. **Count plot:** It is used for categorical features to check the count of the number of observations. Figure 24 explains the count plot of city feature in the stores dataset, and **Quito** is the most frequently occurred value in the **city** feature moreover we can conclude that most of the stores are present in the **Quito** store.
3. **Boxplot:** It can easily handle large amounts of data and is used for numerical data. It does not keep the exact values and details of the distribution results, which is an issue with dealing with such large amounts of data in this graph type, and it also shows the outliers that are present in the diamond shape near the tail.

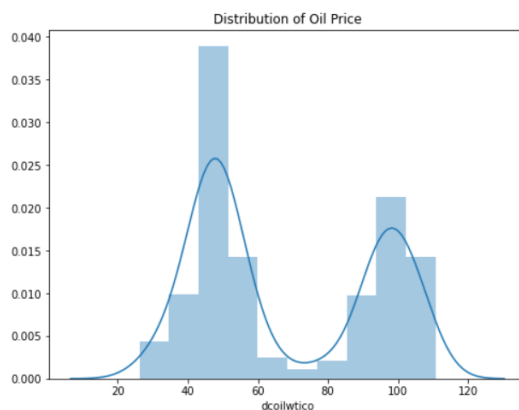


Figure 23: Distribution of oil price

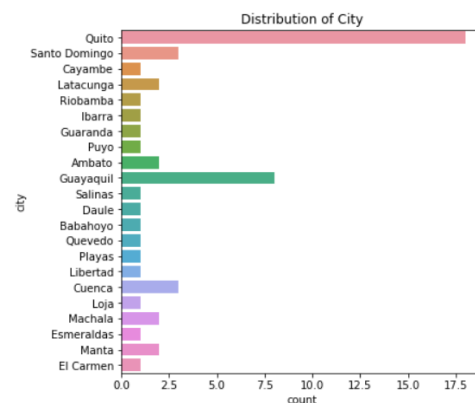


Figure 24: Distribution of City

## 2.6 Bivariate Analysis:

“**Bi**” means two and “**Variate**” means a feature, this method is used to check the relationship between two features. This can be described through the following plots:

1. **Line plot:** A line plot is a graph that uses a number line to depict data. To begin, draw a number line that encompasses all of the values in the data collection. Then, on the number line, draw an X (or a dot) above each data value. If a value appears more than once in a data collection, an X should be placed over that number each time it appears. Figure 25 explains the dcoilwtico feature in the oil dataset, As we can see the **oil** price has drastically fallen down at the end of 2014.
2. **Scatter plot:** A scatter plot is a sort of plot or mathematical diagram that uses Cartesian coordinates to represent values for a collection of data, generally two variables.
3. **Pie Chart:** It is simple and graphically depicts data as a fractional portion of a whole, but if the distribution of value is less than others, then more bits of data are utilized, and the “pie chart” becomes less effective. It also helps to view the data comparison at a glance to perform an immediate analysis or to rapidly absorb information, but basing judgments primarily on visual effects rather than data analysis causes readers to draw incorrect conclusions. Figure 26 explains the family feature distribution in the train dataset, we can conclude that **GROCERY I** feature is the most repeated feature and **AUTOMOTIVE** is the least repeated feature.

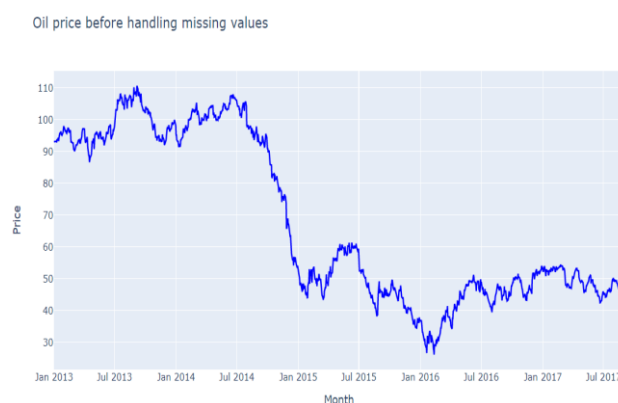


Figure 25: Oil Prices

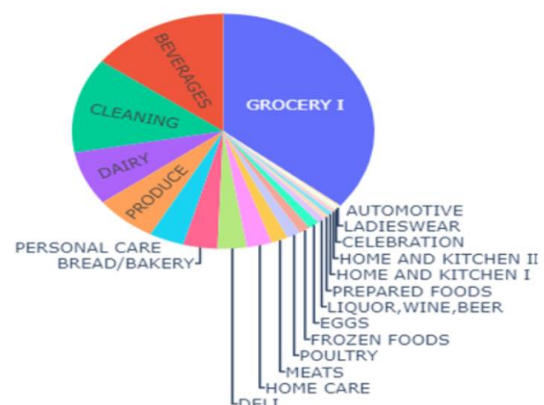


Figure 26: Distribution of Family Feature

## 2.7 Multivariate Analysis:

It is performed to understand interactions between different fields in the dataset (or) find interactions between variables more than 2. Ex:- Pair plot and 3D scatter plot. This can be described through the following plots:

1. **Heatmap:** It is used to examine the correlation between the dataset's features, with highly correlated features being discarded and less correlated features being included. A "**heat map**" depicts values by showing varying shades of the same color for each value. The deeper shades in the chart typically represent higher values than the lighter tints. A "**heat map**" might be misleading since it represents many occurrences at the same time. In terms of usefulness, heat map analysis provides a wealth of information. For those without analytics skills, "**heat map**" data analysis may not be as in-depth as necessary. Figure 27 explains the correlation between the features in the train dataset using heatmap. The darker the color means they are negatively correlated and the lighter the color means they are positively correlated. From the figure we can conclude that sales are on promotion feature are positively correlated with 0.76 out. So, by this we can remove the on promotion feature from the dataset while working training and predicting the model.
2. **Pair plot:** A "**pair plot**" allows us to view the distribution of single variables as well as the correlations between two variables. In univariate plots, diagonal charts are employed. A "**pair plot**" can also be used to identify outliers in two variables. "**Pair plots**" are a fantastic tool for identifying trends that should be further examined. This is true for both category and numerical datasets.
3. **Bar-Line Graph:** A "**bar-line graph**" is a graph that aggregates two features into a group dimension. It looks like a hybrid between a "**bar**" chart and a "**line**" chart. They are effective for depicting both quantity and changes in trends over time. This graph has a dual-axis layout, with bars on the left vertical axis and lines on the right vertical axis.
4. **Lm plot:** The "**lplot**" command requires three arguments. They are the parameters "**data**," "**x**," and "**y**." A "**scatter plot**" with a mix of linear lines throughout our plot, the best available match for the trend, is used here. When compared to huge datasets, it performs well with tiny data sets for better comprehension. In addition, we have a "**hue**" option that allows us to see this

difference in different color charts, as well as a legend with yes/no to make it easier to understand.

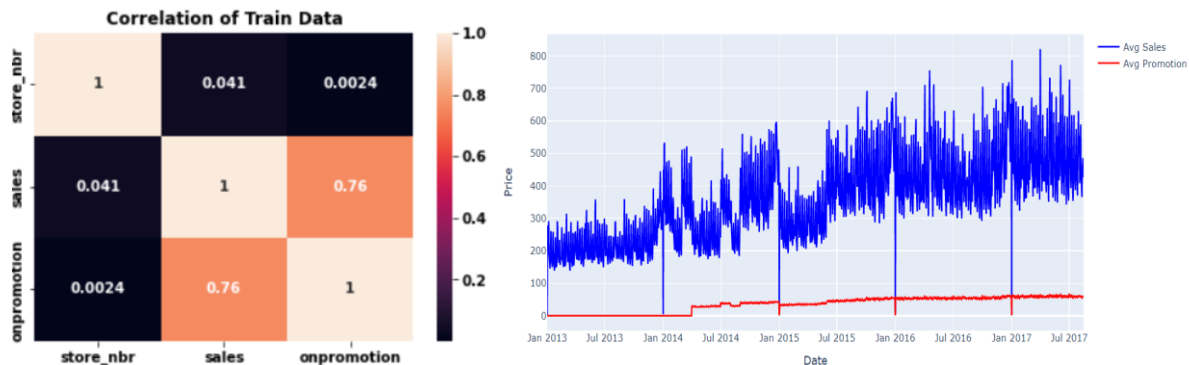


Figure 27: Correlation heatmap for train dataset      Figure: Average sales and onpromotion per day

## 2.8 Feature Selection:

Filter, Wrapper, and Embedded Method are the three feature selection techniques available. Whereas the filter technique is based on correlation with the output, the wrapper method splits the dataset into subsets and trains a model on them depending on the output, and the embedding method is a mix of the filter and wrapper methods.

We have Information Gain, Chi-Square, Variance Threshold, and Correlation in the filter approach. In the wrapper technique, we have Forward Feature Selection, Backward Feature Selection, Exhaustive Feature Selection, and Recursive Feature Elimination. We have Lasso Regularization and Random Forest Importance in the Embedded Method.

Since we used one-hot encoding for numerous features in the dataset, we have encountered the curse of dimensionality. Exhaustive Feature Selection may be used since it is a brute-force evaluation of each feature subset, but it takes a long time to select the features. When working with forward feature selection, it can predict the best features, but it cannot handle large datasets, and our dataset has 30 lakhs of data, which requires a large amount of RAM to compute. So, because the forward feature selection did not work, we may solve the problem by picking a subset of features. As a result, we are contemplating using information gain to choose the best

features from a data collection, which is quicker than others and solves the problem of forward feature selection. The reduction in entropy caused by a dataset modification is calculated as an information gain. It may be used to choose features by assessing the information gain of each variable in the context of the target variable; the main negative is that it may produce model overfitting.

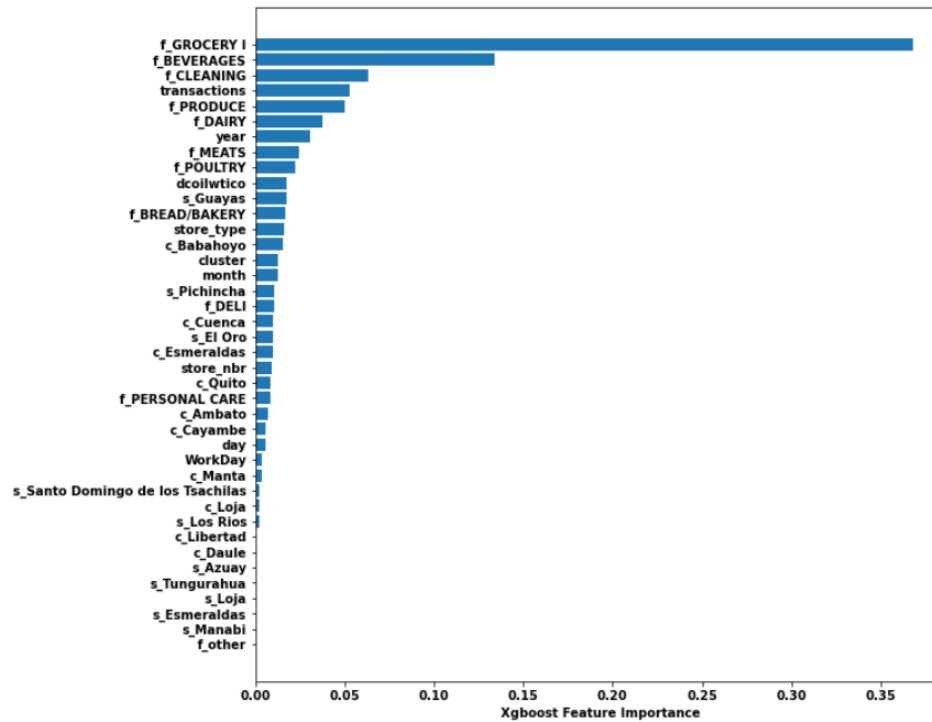


Figure 29: Feature Importance Graph



## CHAPTER 3: MODELING AND ERROR ANALYSIS

After performing EDA, Data cleaning, Data preprocessing, Feature Engineering, and Feature Selection process we have prepared our dataset which is ready to be fed into multiple models, parameter tunes them, and compare the performances.

Every Machine Learning model's development goal is to make it generalizable, trustworthy, resilient, and bias-free. As, all machine learning models rely on producing the closest version of ground truth as feasible, error analysis and appropriate action are crucial in accomplishing this aim.

In general, machine learning models have been reviewed and developed using single or aggregate metrics like accuracy, mean squared error, precision, recall, and r squared, which quantify performance across the whole dataset. This allows us to fine-tune the overall model performance by tweaking the methods more, but it does not dive into details of the problem to help with better addressing training and test failures.

As we are working with the regression-based dataset i.e., sales feature is our target feature and we have a DateTime data type i.e., date feature so our dataset can be also considered as a time-series dataset.

### 3.1 Machine Learning Models

From Figure 30, we can have multiple options like Linear Regression, Support Vector Regression, Decision Tree Regression, Lasso Regression, Ridge Regression, and Neural Networks such as LSTM, Multiple Layered Perception that is suitable by considering the dataset as regression-based, ARIMA, SARIMA, and Facebook Prophet. Once we train the model we just need to predict (or Forecast) the data that has not been trained by the model in order to determine how well the model is functioning. We have used Naïve Bayes Regression, Multiple Linear Regression, Decision Tree Regressor, Random Forest Regressor, ARIMA.

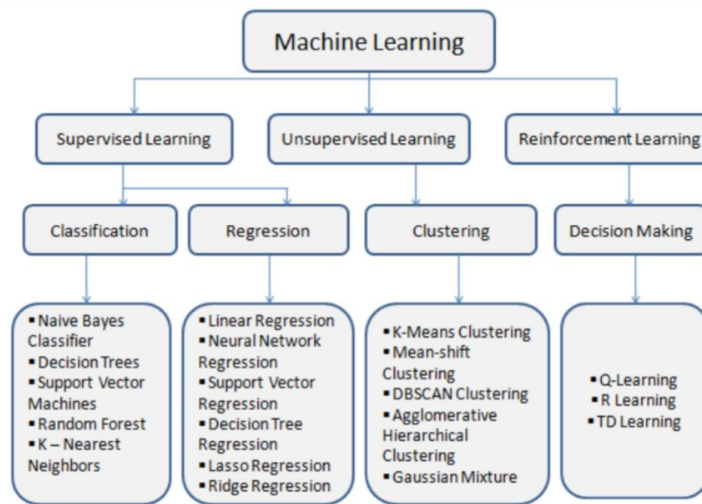


Figure 30: Machine Learning Models classified based on Data Type

### Gaussian Naïve Bayes

In this example, we will look at the challenge of forecasting a numerical goal value,  $Y$ .  $E$  is made up of  $m$  qualities:  $X_1, X_2, \dots, X_m$ . Each attribute is either numeric (which is considered as a real number) or nominal (which is a series of unordered values). If we know the probability density function  $p(Y | E)$  of the target value for all possible occurrences of  $E$ , we could choose  $Y$  to minimise the predicted prediction error. However,  $p(Y | E)$  is rarely known and must be calculated from data. Using Bayes Theorem and assuming the independence of qualities  $X_1, X_2, \dots$  given the goal value  $Y$ , Naïve Bayes achieves this.

$$p(Y|E) = \frac{p(E, Y)}{\int p(E, Y)dY} = \frac{p(E|Y)p(Y)}{\int p(E|Y)p(Y)dY},$$

Figure 31: Bayes Theorem Formula

The probability density function(pdf) of example  $E$  for a given target value  $Y$  is  $p(E|Y)$ , and the pdf of the target value before any examples are seen in  $p(Y)$ . Because, Naïve Bayes believes that given the goal value, the qualities are independent.

Figure 32 shows a code snippet for using the “**sklearn.naive\_bayes**” library. Here the model is trained on  $(X_{train}, y_{train})$  data and the predicted on new dataset i.e.,  $(X_{test})$ . This algorithm is ideal for huge datasets, and there is no need to spend a lot of effort on training data.

```

[27]: from sklearn.naive_bayes import GaussianNB
[28]: nb=GaussianNB()
[38]: nb.fit(X_train,y_train)
[38]: GaussianNB()
[39]: y_predict=[]
[40]: bsize=4000
      for batch in range(int(X_test.shape[0]/bsize)):
          y_predict.append(nb.predict(X_test[batch*bsize:(batch+1)*bsize]))

```

Figure 32: Naïve Bayes Regression Code Snippet

Another key advantage of Naïve Bayes is that it is resilient to missing data. However, the estimations can be incorrect in some circumstances, so we should not take its probability outputs so seriously, and it also does not perform well on numerical datasets, i.e., regression rather than classification.

### Multiple Linear Regression

By fitting a linear equation to observed data, multiple linear regression seeks to predict the connection between two or more explanatory factors and a response variable.

Each independent variable  $x$  value corresponds to a value of the dependent variable  $y$ . For  $p$  explanatory variables  $x_1, x_2, \dots, x_p$  is defined to be  $\mu_y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$ . This line describes how the mean response  $y$  changes with the explanatory variables. The observed values for  $\mu_y$  vary about their means  $\mu_y$  and are assumed to have the same standard deviation  $\sigma$ . The fitted values  $b_0, b_1, \dots, b_p$  estimate the parameters  $\beta_0, \beta_1, \beta_2, \dots, \beta_p$  of the population regression line.

Because the observed values for  $\mu_y$  differ from their mean  $y$ , the multiple regression model contains a term for this variation. The model is written as **DATA = FIT + RESIDUAL**, where the "FIT" term represents the equation  $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$ . The word "RESIDUAL" denotes the deviations of the observed values  $y$  from their means,  $\mu_y$ , which are normally distributed with a mean of 0 and variance. The model deviations are denoted by the symbol  $\epsilon$ .

The figure 33 shows a code snippet for using the multiple linear regression model from scratch.

```
In [63]: class MLR():
def __init__(self):
    self.beta=None

def fit(self, X_train, y_train):
    #fit the xtrain and ytrain to the model
    #inserting 1 value at the starting of each row
    X_train = np.c_[np.ones(len(X_train)), X_train]
    #calculating coefficients
    #pinv means Compute the (Moore-Penrose) pseudo-inverse of a matrix.
    #https://numpy.org/doc/stable/reference/generated/numpy.linalg.pinv.html
    self.beta = np.linalg.pinv(X_train.T.dot(X_train)).dot(X_train.T).dot(y_train)

def predict(self, X_test):
    #predicting the y value from X_test features
    X_test = np.c_[np.ones((len(X_test), 1)), X_test]
    #dot product of X_test and beta
    y_pred = np.dot(X_test, self.beta)
    return y_pred
```

Figure 33: Multiple Linear Regression Code Snippet from Scratch

From figure 33, the fit definition explains to train the model on the known data (values) where as predict definition explains to predict the values on the unknown data (values).

### Decision Tree Regressor

Decision Tree builds regression or classification model in the form of a tree structure. It gradually divides a dataset into smaller and smaller sections while also developing an associated decision tree. The end result is a tree containing leaf nodes and decision nodes. A decision node has two or more branches, each of which represents a value for attribute under consideration. A leaf node reflects a numerical target choice. The root node is the top most decision node in a tree that corresponds, to the best predictor. Both category and numerical data can be handled by using decision trees.

```
from sklearn.tree import DecisionTreeRegressor

X=df.drop("sales",1)

y=df.sales

X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=0.8,random_state=0)

rmse_depth=[]
for i in range(1,30):
    dt=DecisionTreeRegressor(criterion='mse',max_depth=i)
    dt.fit(X_train,y_train)
    y_pred=dt.predict(X_test)
    rmse=np.sqrt(np.square(np.subtract(y_test,y_pred)).mean())
    rmse_depth.append(rmse)

rmse_depth=np.array(rmse_depth)
```

Figure 34: Decision Tree Regressor Code Snippet

Figure 34 explains the code snippet of Decision Tree Regressor, here we are using “**from sklearn.tree import DecisionTreeRegressor**” to use the Decision Tree Library. We are checking error rate of different values of **max\_depth** parameter. We select a value where the error is least. Figure 35 is a graph of error change for different

**max\_depth** values. Overfitting occurs as the parameter **max\_depth** is increased. Decision Trees requires less work for data preparation during pre-processing as compared to other methods. Missing values in the data have no significant impact on the pre-process of developing a decision tree. The approach is highly natural and simple to communicate with technical teams and stake holders alike. A small change in the data can cause a large change in the structure of the decision tree, causing instability.

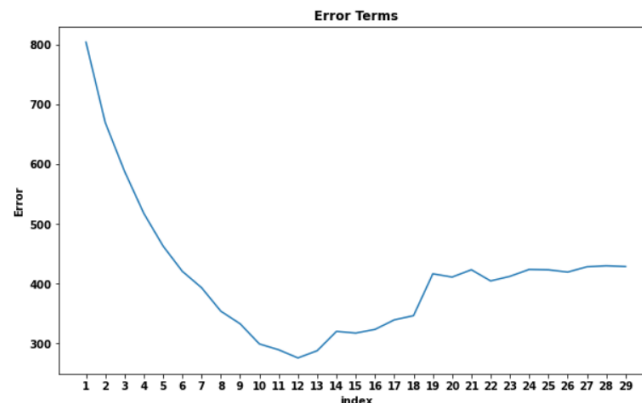


Figure 35: Elbow method to find the best **max\_depth** parameter

### Random Forest

Random Forest is a bagging method, not a boosting method. Random Forests have trees that run in parallel. While creating the trees, there is no interaction between them. At the training time, it constructs a large number of decision trees and outputs a class that is the mode of classes (classification) or mean prediction (regression) of the individual trees. A random forest is a meta-estimator (that is, it aggregates the results of numerous predictions) that aggregates several decision trees. A Random Forest, works similar to decision trees, but it avoids overfitting and helps enhance accuracy. It needs a significant amount of computer power as well as resources because it constructs multiple trees and then combines their outputs.

### **Hyperparameters in Random Forest Regressor**

- *n\_estimators* : Number of trees in the forest
- *criterion* : Function to measure the quality of a split
- *max\_depth* : Maximum depth of the tree

- *min\_samples\_split* : Minimum number of samples required to split an internal node
- *min\_samples\_leaf* : Minimum number of samples required to be at leaf node
- *max\_features* : Number of features to consider when looking for the best split

## Types of Hyperparameter Optimization:

1. Grid Search CV
2. Randomized Search CV
3. Bayesian Optimizataion – Automate Hyperparameter Tuning (Hyperopt)
4. Sequential Model Based Optimization (Tuning a scikit-learn estimator with skopt)
5. Optuna – Automate Hypparameter Tuning
6. Genetic Algorithms (TPOT Classifier)

We have chosen Bayesian Optimization from the above list because it employs probability to determine the minimum of a function. The ultimate goal is to identify the input value to a function that produces the smallest feasible output value. It often outperforms random, grid, and manual search methods in terms of testing performance and optimization time.

```
In [125]: def objective(params):
            est = int(params['n_estimators'])
            md = int(params['max_depth'])
            msl = int(params['min_samples_leaf'])
            mss = int(params['min_samples_split'])
            model = RandomForestRegressor(n_estimators=est,max_depth=md,min_samples_leaf=msl,min_samples_split=mss)
            model.fit(X_train,y_train)
            y_pred_hyperopt = model.predict(X_test)
            y_pred=[]
            for i in y_pred_hyperopt:
                if i<0:
                    y_pred.append(0)
                else:
                    y_pred.append(i)
            score = np.sqrt(np.square(np.subtract(y_test,y_pred)).mean())
            return score

            def optimize(trial):
                params={'n_estimators':hp.uniform('n_estimators',100,500),
                        'max_depth':hp.uniform('max_depth',5,20),
                        'min_samples_leaf':hp.uniform('min_samples_leaf',1,5),
                        'min_samples_split':hp.uniform('min_samples_split',2,6)}
                best=fmin(fn=objective,space=params,algo=tpe.suggest,trials=trial,max_evals=100)
                return best

            trial=Trials()
            best=optimize(trial)

100%|██████████| 100/100 [15:53<00:00, 9.54s/trial, best loss: 423.06906926971163]
```

Figure 36: Code snippet of Bayesian Optimization Hyperparameter tuning

Bayesian Optimization may be accomplished in Hyperopt bypassing three primary arguments to the function fmin.

*Objective Function:* Defines the loss function to minimize

*Domain Space:* Defines the range of input values.

*Optimization Algorithm:* Defines the search algorithm to use to select the best input values to use in each new iteration

We need to run **pip install hyper opt** in the terminal before using Bayesian Optimization. Figure 36 explains the objective and optimize functions.

## 3.2 Time Series Forecasting

### ARIMA

ARIMA, or Autoregressive Integrated Moving Average, is a statistical analysis that uses time-series data to better understand the dataset or anticipate future trends. Autoregressive statistical models anticipate future values based on previous values. An Autoregressive Integrated Moving Average model is a type of regression analysis that measures the strength of one dependent variable with other variables that change. Each of the components of an ARIMA model may be understood by outlining them as follows:

**Autoregression (AR):** a model in which a changing variable regresses on its lag, or previous values.

**Integrated (I):** the differencing of raw observations that allows the time series to become stationary (i.e., data values are replaced by the difference between the data values and the previous values).

**Moving Average (MA):** a moving average model applied to lagged observations that considers the dependence between observations and residual error.

For ARIMA models, a standard notation would be ARIMA with p, d, and q where integer values substitute for the parameters to indicate the type of ARIMA model used. The parameters can be defined as:

p: the number of lag observations in the model; also known as the lag order.

d: the number of times that the raw observations are different, also known as the degree of difference.

q: the size of the moving average window; also known as the order of the moving average.

From figure 37, the parameter **order** has value (4,1,3) represents the ARIMA (p, q, d) values. It is suitable for non-stationary time series and has high accuracy for forecasts. But it is unable to respond immediately, with a certain lag.

```
In [195]: from statsmodels.tsa.arima_model import ARIMA
model=ARIMA(train['sales'],order=(4,1,3))
model=model.fit()
model.summary()
```

Out[195]: ARIMA Model Results

Dep. Variable:	D.sales	No. Observations:	1171
Model:	ARIMA(4, 1, 3)	Log Likelihood	-6118.298
Method:	css-mle	S.D. of innovations	44.902
Date:	Tue, 08 Feb 2022	AIC	12254.596
Time:	14:09:55	BIC	12300.187
Sample:	1	HQIC	12271.791

	coef	std err	z	P> z	[0.025	0.975]
const	0.2250	0.381	0.591	0.555	-0.522	0.972
ar.L1.D.sales	-0.9522	0.055	-17.470	0.000	-1.059	-0.845
ar.L2.D.sales	-0.0917	0.108	-0.848	0.397	-0.304	0.120
ar.L3.D.sales	0.3971	0.100	3.989	0.000	0.202	0.592
ar.L4.D.sales	-0.1598	0.043	-3.742	0.000	-0.243	-0.076
ma.L1.D.sales	0.7071	0.051	13.900	0.000	0.607	0.807
ma.L2.D.sales	-0.3991	0.083	-4.821	0.000	-0.561	-0.237
ma.L3.D.sales	-0.7858	0.052	-15.208	0.000	-0.887	-0.685

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-0.8104	-0.5905j	1.0028	-0.3998
AR.2	-0.8104	+0.5905j	1.0028	0.3998
AR.3	2.0529	-1.4174j	2.4947	-0.0962
AR.4	2.0529	+1.4174j	2.4947	0.0962
MA.1	1.1765	-0.0000j	1.1765	-0.0000
MA.2	-0.8422	-0.6103j	1.0400	-0.4002

Figure 37: Code snippet of using ARIMA

Our primary goal from the start has been to have a model with the lowest Root Mean Squared Error and a high R squared value. We deployed the above-mentioned models, the performance of each model is mentioned in the below screenshot.

	Model	RMSE Error	R squared value
0	Naive Bayes Regressor	5943.463227	-22.606927
1	Multiple Linear Regression	606.708370	0.629940
2	Decision Tree Regressor	277.110316	0.922800
3	Random Forest Regressor	428.490039	0.827306
4	ARIMA	41.285019	0.161704

Figure 38: RMSE and R Squared Values for different Machine Learning Model



## CHAPTER 4: ADVANCED MACHINE LEARNING AND FEATURE ENGINEERING

### 4.1 Stacking

Stacking is a technique that takes several regression or classification models and uses their output as the input for the meta-classifier/regressor. Stacking is an ensemble learning technique much like Random Forests, where the quality of prediction is improved by combining. The types of stacking methods that can be deployed are:

- Vecstack
- Sklearn Stacking
- Mlxtend

We have chosen sklearn stacking method to perform stacking. Figure 39 explains the code snippet of Sklearn Stacking. We use “**from sklearn.ensemble import StackingRegressor**”. We append the models for estimator parameter in a tuple format, **xgb** denotes Extreme Gradient Boosting Regressor, **dt** denotes Decision Tree Regressor, and **lgbr** denotes Light Gradient Boosting Regressor.

```
In [42]: from sklearn.ensemble import StackingRegressor

In [43]: xgb=XGBRegressor(n_estimators=80)

In [44]: lgbr=LGBMRegressor(n_estimators=90)

In [45]: estimators = [("Xg Boost",xgb),
                      ("Decision Tree",dt),
                      ("LightGBM",lgbr)]

In [46]: stacking_regressor = StackingRegressor(estimators=estimators)
```

Figure 39: Code Snippet of Sklearn Stacking

The benefit of stacking is that it can harness the capabilities of a range of well-performing models on classification or regression tasks and make predictions that have better performance than any single model in the ensemble. Once we obtain the

best model, we can perform feature selection depending on the model. The features that are obtained are model-dependent.

## 4.2 Deep Learning Hyperparameter Optimization

Usually, hyperparameter tuning is done in machine learning by using "RandomSearchCV," "GridSearchCV," "HyperOpt," and "BayesianOptimization." But, while using deep learning models, we can also perform hyperparameter optimization by using KerasTuner. We need to run **pip install keras-tuner** in the terminal before using Bayesian Optimization.

The hyperparameters tuning doesn't change over training time and remains constant and manipulates the training process of a model. The tasks we need to find in deep learning are:

- Number of layers to choose
- The number of neurons in a layer to choose
- Choice of the optimization function
- Choice of the learning rate for optimization function
- Choice of the loss function
- Choice of metrics
- Choice of activation function
- Choice of layer weight initialization

From all the above options we are selecting number of layers, number of neurons in a layer, the activation function. Our main aim is to get the Root Mean Squared Error (RMSE) value to be low, so we are considering Mean Squared Error (MSE).

Figure 40 explains the code snippet of Bayesian optimization. In the definition **build\_model**, we represent the range of each parameters that need to be chosen. As, we have used multiple machine learning models and deep learning models. Among them we are considering the model which has highest **R squared** score and least **Root Mean Squared Error (RMSE)**. We have selected the best model and checked the loss that have been obtained. The **RMSE** for Keras tuner hyper parameter optimization is 2014.34.

```
In [15]: def build_model(hp):
model = keras.Sequential()
for i in range(hp.Int('num_layers', 2, 20)):
    model.add(layers.Dense(units=hp.Int('units_' + str(i),
                                     min_value=32,
                                     max_value=512,
                                     step=32),
                           activation=hp.Choice('activation', ['sigmoid', 'relu', 'tanh'])))
model.add(layers.Dense(1, activation='linear'))
model.compile(
    optimizer=keras.optimizers.Adam(
        hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
    loss='mse',
    metrics=['mse'])
return model
```

## Bayesian Optimization

```
In [16]: tuner = BayesianOptimization(
    build_model,
    objective='mse',
    max_trials=4,
    executions_per_trial=1,
    directory='diploma_project',
    project_name='BayesianOptimization_corporación favorita')
```

Figure 40: Code snippet of Bayesian Optimization

## 4.3 Long Short Term Memory (LSTM)

Figure 41, explains the architecture of LSTM model, it is an artificial recurrent neural network architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can process not only single data points, but also entire sequences of data. The Long Short-Term Memory (LSTM) cell can process data sequentially and keep its hidden state through time.

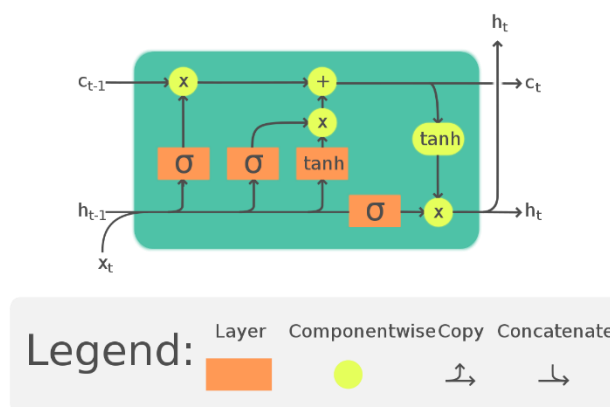


Figure 41: Architecture of LSTM

Now, we have a look how to build an LSTM model in Python. The bi-directional LSTM with dropouts works fairly well, when compared with uni-directional LSTM.

Figure 42 explains the code snippet of LSTM Model, We need to use “from tensorflow.keras.layers import LSTM, Bidirectional, Dense, Dropout” to this snippet.

```
In [68]: model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(50,return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(50,return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(50,return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(50)))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

Figure 42: Code snippet for Bidirectional LSTM

Figure 43 provides the information about:

- The layers and their order in the model.
- The output shape of each layer.
- The number of parameters (weights) in each layer.
- The total number of parameters (weights) in the model.

```
Model: "sequential_5"
-----
```

Layer (type)	Output Shape	Param #
lstm_20 (LSTM)	(None, 100, 50)	10400
dropout (Dropout)	(None, 100, 50)	0
bidirectional (Bidirectional)	(None, 100, 100)	40400
dropout_1 (Dropout)	(None, 100, 100)	0
bidirectional_1 (Bidirectional)	(None, 100, 100)	60400
dropout_2 (Dropout)	(None, 100, 100)	0
bidirectional_2 (Bidirectional)	(None, 100, 100)	60400
dropout_3 (Dropout)	(None, 100, 100)	0
bidirectional_3 (Bidirectional)	(None, 100)	60400
dense_5 (Dense)	(None, 1)	101

```
-----
Total params: 232,101
Trainable params: 232,101
Non-trainable params: 0
-----
```

Figure 43: Summary for LSTM model developed using python

Figure 44, is the visualization of future 60 values by considering the previous 100 days of values. The blue line represent the 100 days of data i.e., true values where as the yellow line represents the next 60 days of data i.e., predicted values.

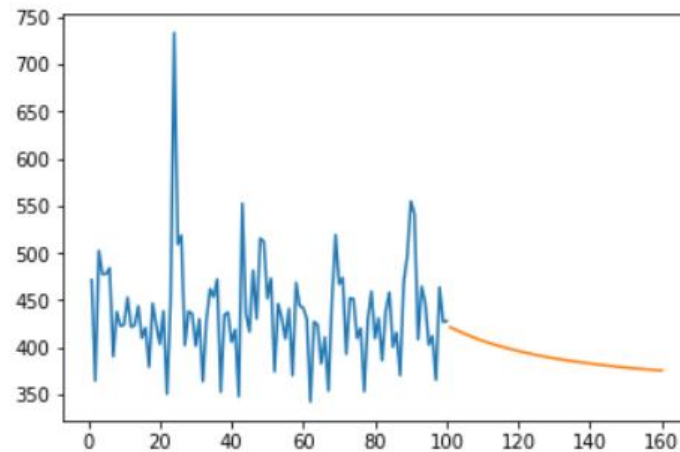


Figure 44: Predictions for next 60 days, by considering 100 days of data

## 4.4 Error Analysis

	Model	RMSE Error	R Squared Value
0	Xg Boost	256.011421	0.934108
1	Decision Tree	330.348469	0.890287
2	LightGBM	274.886096	0.924034
3	MLP	2014.349275	NaN
4	LSTM	0.079500	NaN
5	CatBoost	274.261563	NaN

Figure 45: Performance of Advanced Machine Learning and Deep Learning models

Figure 45, explains the RMSE and R squared values for different machine learning and Deep Learning models, We can conclude that LSTM, CatBoost, Extreme Gradient Boosting have given least RMSE values. To check the model interpretability we have methods like SHAP and LIME, But here we have used SHAP to know the model interpretability. There are few research papers that helped out they are:

- 1.Improving Sales Forecasting Accuracy: A Tensor Factorization Approach with Demand Awareness
2. Stock Price Prediction Using Time Series, Econometric, Machine Learning, and Deep Learning Models

## CHAPTER 5: MODEL DEPLOYMENT

### 5.1 Ways to Deploy Model

Most of these issues can be solved by deploying a machine learning model as a service, and predictions will be made in real time. However, there will be concerns such as scalability, monitoring, and service downtime. There are several cloud companies available to tackle these challenges and give 24\*7 assistance. We have the following choices:

1.     **Algorithmia:** Algorithmia is a MLOps tool. This is a subset of "algorithms as a service." It enables users to write code snippets that run the ML model and host them on Algorithmia. Then our code may be referred to as an API. The most significant advantage of doing so is that it separates the Machine Learning concerns from the rest of our application.
2.     **PythonAnywhere:** PythonAnywhere is a rapidly expanding platform-as-a-service (PaaS) based on the Python programming language. It makes it simple to execute Python programmes in the cloud and provides a simple approach to hosting web-based Python applications. The main disadvantage is that it does not support GPU. If our deep learning model is based on CUDA and the GPU, we cannot consider this and must seek alternate ways.
3.     **Heroku:** Heroku is a cloud Platform as a Service (PaaS) that enables developers to rapidly create, manage, and scale contemporary apps without worrying about infrastructure. It provides a wide range of services and tools to accelerate our development and prevent us from having to start from zero. Python, Scala, PHP, Java, Node, Go, Ruby, and Clojure are among the programming languages supported.
4.     **Google Cloud Platform:** Google Cloud Platform (GCP) is a Google platform that offers cloud computing services such as compute, storage, and database, artificial intelligence (AI) / machine learning (ML), networking, Big Data, and identity and security. It offers infrastructure as a service (IaaS), platform as a service (PaaS), and serverless environments.

5. Microsoft Azure: Functions are serverless cloud services offered by Microsoft Azure as Functions-as-a-Service (FaaS). Azure services allow developers to focus on their apps rather than infrastructure maintenance. It supports C#, F#, Node.js, Python, JavaScript, PHP, Java 8, Powershell Core, and TypeScript functions.

6. Azure Lambda: Amazon Web Services (AWS) Lambda is a serverless computing solution offered by Amazon as part of Amazon Web Services. AWS Lambda allows us to run code without having to worry about the underlying infrastructure. We can send our code in the form of a container image or a zip file. Lambda will automatically distribute compute capacity to run our code depending on incoming requests or events without requiring our configuration assistance. AWS Lambda enables us to connect our code to other AWS resources such as an Amazon DynamoDB table, an Amazon S3 bucket, an Amazon SNS notification, and an Amazon Kinesis stream. We can use Amazon API Gateway to access the model after it has been deployed on AWS Lambda. Python, Java, GO, PowerShell, Node.js, Ruby, and C# code are all supported.

Considering the aforementioned techniques, putting our machine learning model on Heroku using Flask will solve our difficulties if it is our own AI/ML project and we don't want to spend additional time handling cloud installations or DevOps duties and want a rapid deployment choice.

## 5.2 Libraries used to deploy model

Below mentioned libraries are used while deployment.

- Virtualenv: To create a virtual environment so that only particular libraries will be installed in the environment and that will be helpful after deployment. To create a virtual environment, we will need to first install virtualenv on our current Python installation. We need to run below command in our terminal.

**pip install virtualenv**

To create a virtual environment we need to pass below command. Where, **"kagglesales"** is the name of the virtual environment.

**conda create -n kagglesales python=3.7**

To activate and deactivate the virtual environment we pass the below command in the terminal.

**conda activate kagglesales**

**conda deactivate**

- Flask: This library is used to create a Web Api. We need to run below command in the terminal to install.

**pip install Flask**

- Flask-Cors: This library is an flask extension library for handling cross origin resources making cross-origin AJAX possible. We need to run below command in the terminal to install.

**pip install Flask-Cors**

- Gunicorn: Gunicorn is a WSGI HTTP Server for UNIX. It's a pre-fork worker model ported from Ruby's Unicorn project. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.

**pip install gunicorn**

- Pickle: Pickle module is used for serializing and de-serializing python object structures. The process to converts any kind of python objects into byte streams is called pickling.

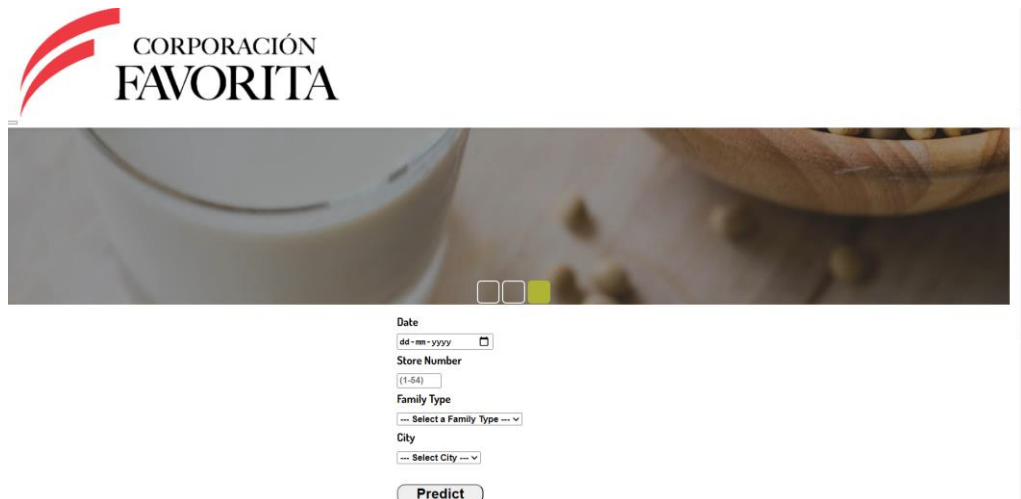
**pip install pickle**

### **5.3 Demo of Web API**

The website is shown in the attached screenshot, where clients may fill in the essential data such as **Date**, **Store Number**, **Family Type**, and **City**. These are the entries that the client can simply respond to; we have additional features accessible for more accuracy, but the consumer has difficulty providing precise numbers. As a result, we only included features that customers could simply enter. After entering the numbers, the client can click on **predict** button to acquire a price range for a



product. As a result, we have reduced the number of entries that can be easily replied to by customers.



The screenshot shows the top of a web browser with the 'CORPORACIÓN FAVORITA' logo. Below the logo is a large image of a bowl of food. Underneath the image is a form with the following fields: 'Date' (dd-mm-yyyy), 'Store Number' (1-54), 'Family Type' (dropdown menu), and 'City' (dropdown menu). At the bottom of the form is a 'Predict' button.

Figure 46: Demo of Web Api

The deployed Web Api is available in below link:

<https://storesalesecuador.herokuapp.com/>

## 5.4 Architecture of Heroku Deployment

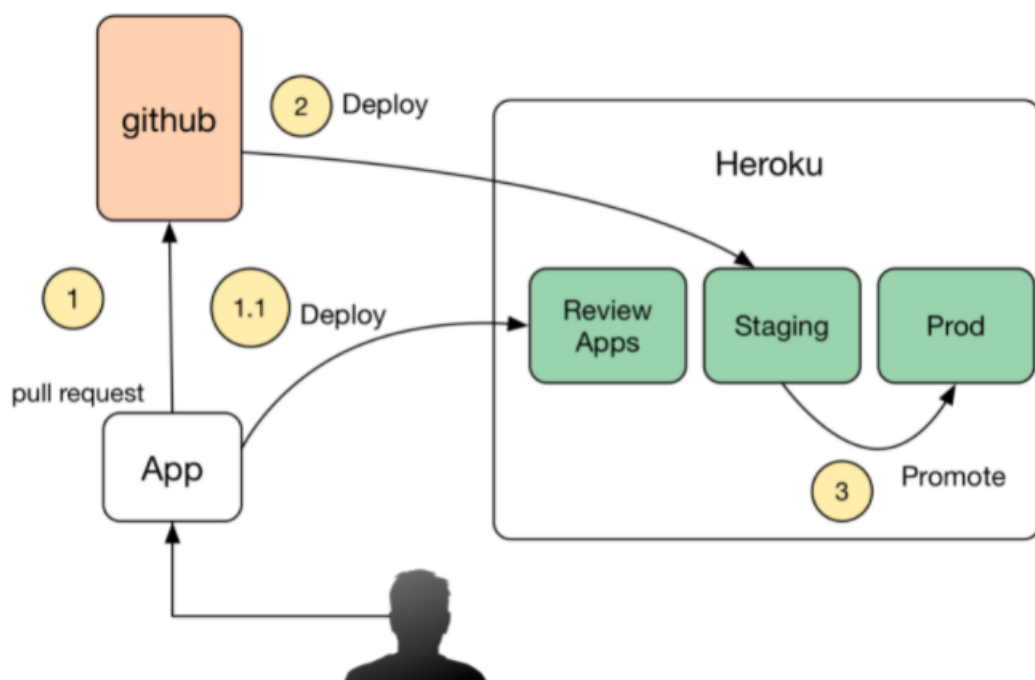


Figure 47: Architecture of Heroku Deployment

Figure 48: Output of the given input in WebAPI

We need to run the below commands to create a git remote and push the app.

Once we run the dependencies libraries, we need to run the following in the virtual environment to setup git:

```
git init
```

```
git add .
```

```
git commit -m "Initial Commit"
```

### **Setup Heroku**

You need to have heroku-cli installed on your system, and have a Heroku account configured to proceed with this step. Code to login and create a new app

```
heroku login
```

```
heroku create storesalesprediction
```

Code to push the initial commit to heroku

```
git push heroku master
```

### **App File**

Inside this folder, create an app folder which would contain all files related to your Flask application.

## Nomenclature

Ensure the following specification in your flask app, for the sake of nomenclature used in this guide:

your application is inside the app folder, i.e., it's path is flask-deployment/app/main.py

your app is named main.py

your app is named app, as in `app = Flask(name)` — default naming.

## Procfile

Procfile should contain the following sentence:

web: gunicorn wsgi:app

Here, app denotes the application that declared in flask that we have pass as `__name__` to Flask and assign it the variable app. The below mentioned application needs to be assigned in Procfile.

```
from flask import Flask
```

```
app=Flask(__name__)
```

## requirements.txt

We need to run the below command to get this file

```
pip freeze > requirements.txt
```

## References:

[How to Deploy a Machine Learning API and App using Flask on Heroku | by Niyel Hassan | Geek Culture | Medium](#)

<https://arxiv.org/pdf/2011.03452.pdf>

<https://arxiv.org/ftp/arxiv/papers/2111/2111.01137.pdf>

[https://en.wikipedia.org/wiki/Random\\_forest](https://en.wikipedia.org/wiki/Random_forest)

<https://www.khanacademy.org/math/cc-2nd-grade-math/x3184e0ec:data/cc-2nd-line-plots/v/introduction-to-line-plots>

<https://www.analyticsvidhya.com/blog/2021/04/rapid-fire-eda-process-using-python-for-ml-implementation/>

<https://medium.com/dwadda/ways-of-encoding-categorical-variables-b7a798931c8c>

<https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/>

<https://www.analyticsvidhya.com/blog/2021/04/exploratory-analysis-using-univariate-bivariate-and-multivariate-analysis-techniques/>