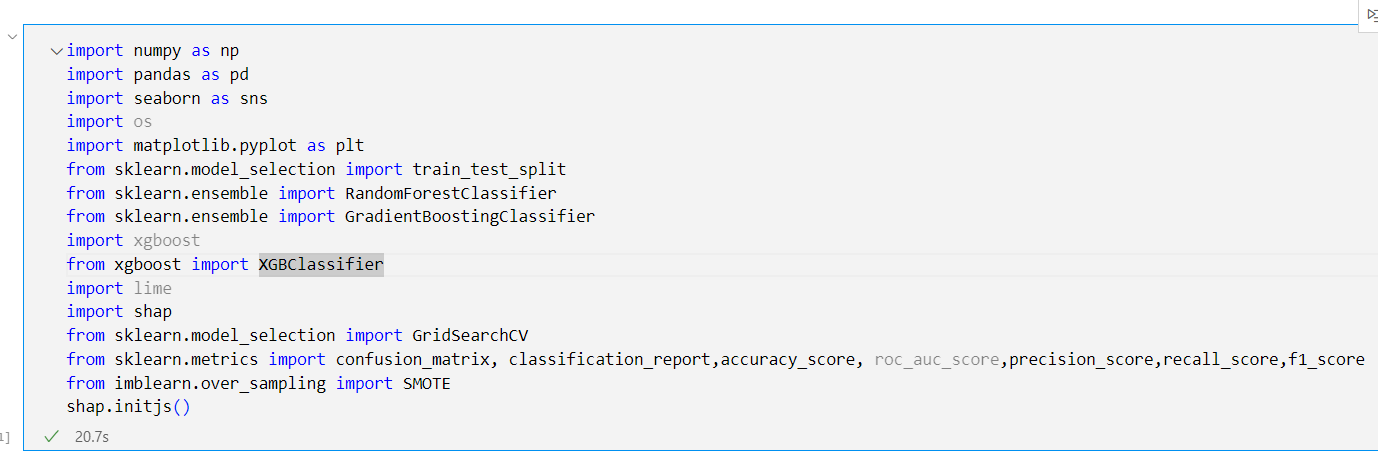
# Results

## Importing Libraries

Importation of libraries is one of the beginning steps of the data analysis process in which necessary libraries, packages, classes etc. are loaded into the python environment. These libraries provide various functionalities, modules, classes etc. using which it becomes easier to perform the credit card fraud detection and analysis.



The screenshot shown below displays the libraries, modules and classes that are being used throughout the lifecycle of the project and includes libraries such as Numpy for performing array and matrix operations, Pandas for performing cleaning and manipulation of data, scikit-learn for model building and evaluation, GridsearchCV for tuning parameter of a machine learning model, ShaP and Lime for explainability of the complex machine learning models and so on.

## Data loading

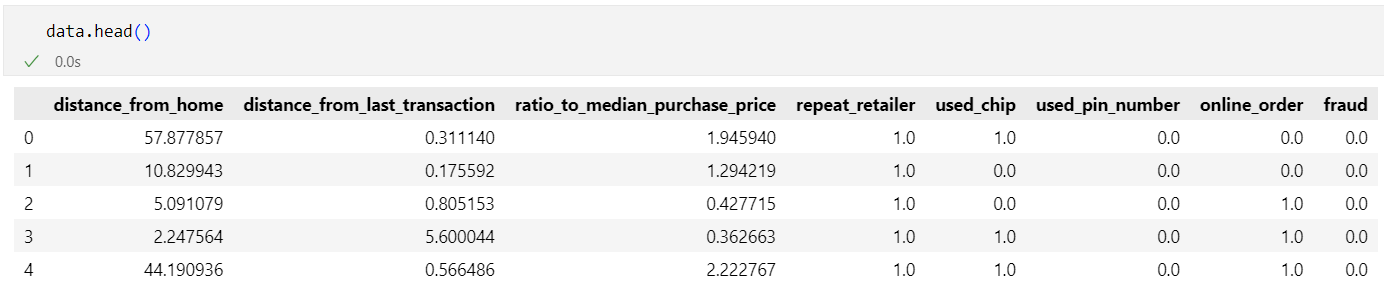
After importation of the libraries, the historical data identified and selected for the problem is loaded into the python integrated development environment (IDLE). To solve the problem of this study i.e. credit card fraud detection, a dataset of credit card transactions was downloaded from an open source platform like kaggle.



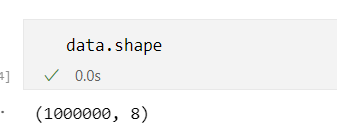
The snapshot shown above displays how historical data of credit card transactions was loaded from a csv file by making use of pandas read\_csv() function. The dataset includes various features such as credit score, distance from home, frauds etc. that are helpful for analyzing and identifying the patterns of the fraudulent transactions.

## Data Exploration

After loading the data from an open source platform like kaggle, it becomes necessary to understand the data before performing any analysis or model training. Data exploration plays a major role in identifying patterns, trends, correlations etc. present within the dataset. This further helps in identifying basic inconsistencies and unwanted information such as missing values, outliers, incorrect names, incorrect data types etc. that are lying within different features of the dataset.



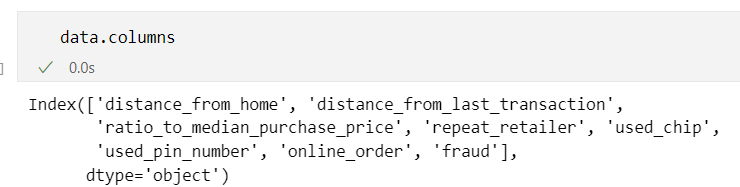
The snapshot shown above displays some of the rows and columns present inside the credit card transactions dataset. By making use of the pandas **head()** function, basic exploration of the dataset is performed. The **head()** functions display the first 5 rows of all the features of the dataset. After analyzing the above snapshot it can be seen that there are inconsistencies present inside various columns in the form of data values. For example, the online\_order column should contain data of type integer but it is containing data of float type.



The snapshot shown above displays the number of rows and columns present inside the dataset. In python, “**shape**” attribute of the data frame is used to find the dimensions of the dataset. After analyzing the result of the “shape” attribute, it can be seen that there are 1000000 rows and 8 columns present inside the loaded dataset.



The snapshot shown above displays the summarized view of the dataset by making use of various statistical techniques that are helpful in identifying the variability inside the distributions, quantiles etc. In python the describe() function of the pandas is used to generate a statistical summary of various features, by making use of statistical techniques such as mean, standard deviation, minimum, maximum etc. From such a summarized view of data, various conclusions can be drawn about central tendency, dispersion etc. of the data. After analyzing the results from various statistical techniques it can be seen that standard deviation of columns is much higher than mean of the columns, showcasing data is highly dispersed.



The snapshot shown above displays the column names present inside the dataset. In python, “columns” attribute of the data frame is used to display the feature names of the dataset. This step is further helpful in identifying the inconsistencies present in the form of column names. After analyzing the outcome of the **“columns”** attribute it can be concluded that, there are not any inconsistencies present in the dataset in the form of column names.

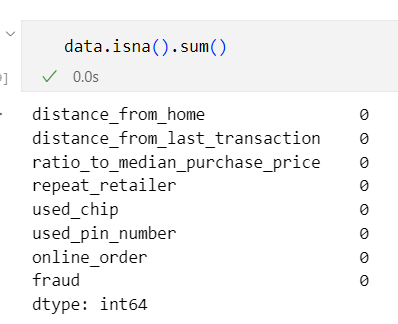
## Data preprocessing:

After analyzing the data using the data exploration step, it becomes important to identify, correct or remove any inconsistencies if found in the form of missing values, outliers, duplicates etc. inside the features of the dataset. This helps in improving the quality of the dataset , which in turn improves the outcomes of the machine learning models. The various step performed in the process of data cleaning are as follows:

### Detection and removal of missing values and duplicates:

#### Detection of missing values:

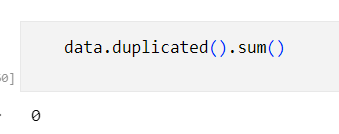
Presence of missing/Null values inside the dataset degrades the performance of machine learning models in the form of unreliable statistical results, poor decisions etc. Hence, it becomes important to detect and remove these missing/Null values before training machine learning models on the dataset.



The snapshot shown above displays the number of missing/null values present inside each of the columns of the dataset. After analyzing the outcome of the above snapshot, it can be seen that there are not any missing values present inside the dataset. In python isna() and sum() functions of pandas are used to identify and count the number of missing values inside each feature of the dataset, no need to perform any missing value removal step.

#### Detection of duplicates

Presence of duplicate rows present inside the dataset could have serious consequences on machine learning models. They affect the generalizability, and lead to bias, overfitting etc of machine learning models. Hence, it becomes important to detect and remove these duplicate rows before training a machine learning model on the dataset.

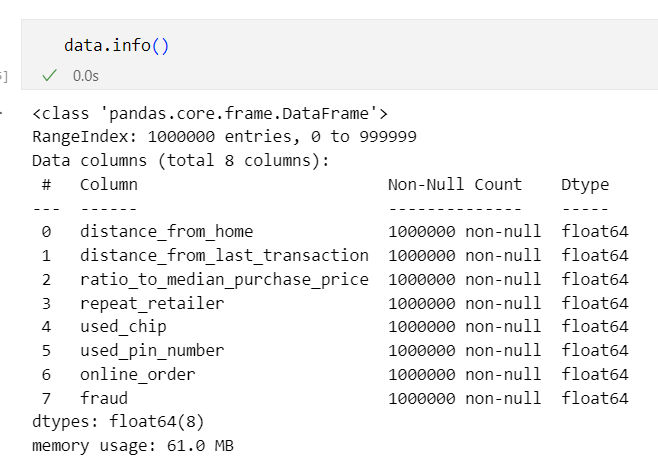


In the above snapshot the duplicated() and sum() function from pandas are used for identifying and counting the duplicate records present inside the dataset. After analyzing the outcomes of the step, it can be seen that there are not any duplicate records present inside the dataset, no need to perform the duplicates removal step.

### Identification and Removal of inconsistent data type

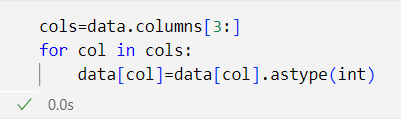
#### Identification of inconsistent data types

Identification of the data type inside each column of the dataset, plays an important role in identifying the inconsistencies present in the form of data types before training a machine learning model on the dataset. Inconsistent data types could result in inaccurate/misguided results or decisions from the model. Hence, affecting the consistency of the dataset.



The snapshot shown above displays the data type, non-null count, column names and size of the dataset. In python, Pandas info() function is used in identifying inconsistencies present inside column names and data types of the dataset. After analyzing the outcome of the function it can be seen that all of the columns present inside the dataset are of float type. Further, there are data type inconsistencies found inside the various columns of the dataset such as repeat\_retailer, used\_chip, used\_pin\_number, online\_order and fraud. Highlighting the need for its correction.

#### Removal of data type inconsistency

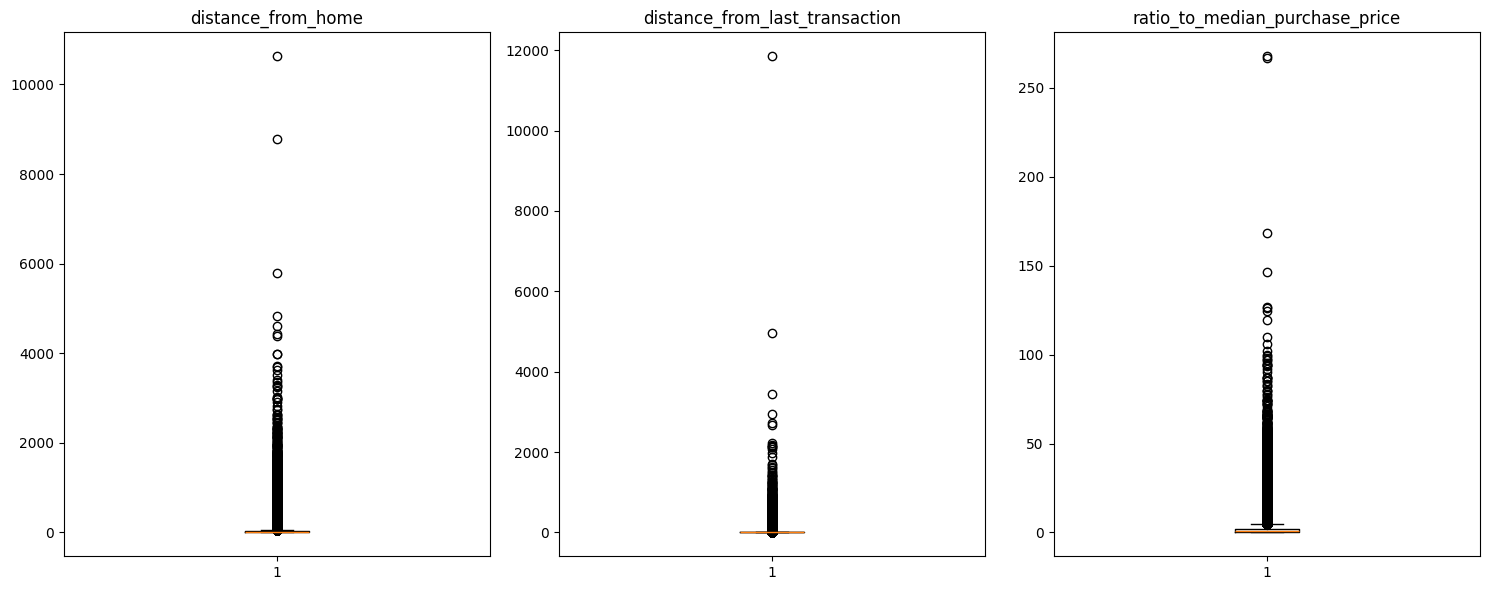


The snapshot above is displaying the use of “columns” attribute and “astype()” function of pandas in correction of data type inconsistencies. With the help of columns attribute and integer 3 within the square brackets only the columns with incorrect data type got selected. After iterating through each of the columns, the **astype()** function with “int” as data type was used for correcting the data type.

## Outlier Detection and Capping

Dataset may further contain irregular values in the form of outliers which should be removed before training the machine learning models. This helps to improve the quality of the dataset and prevents the machine learning model from producing skewed results.

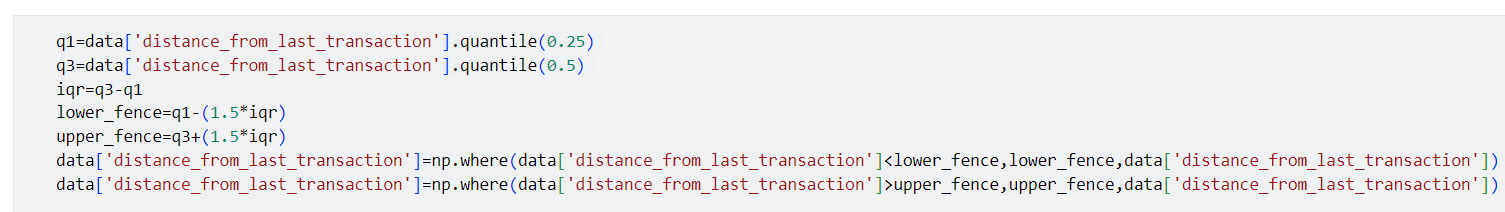
### Detection of Outliers before Capping



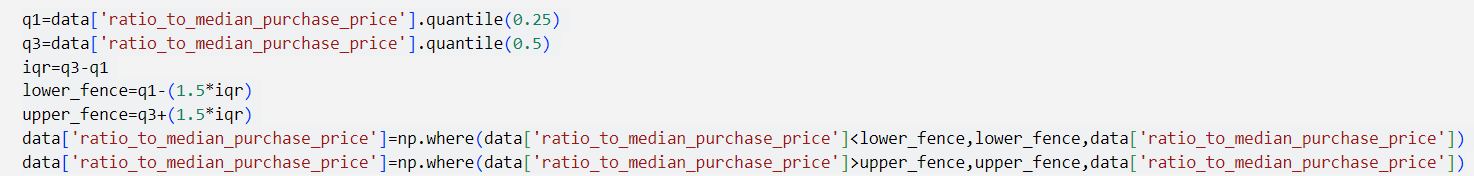
The boxplots shown above are helpful in identifying the outliers present inside various features of the dataset. In Python, the boxplot() function from the matplotlib library is used for visually identifying the outliers present inside various columns of the dataset such as distance\_from\_home, distance\_from\_last\_transaction and ratio\_to\_median\_purchase \_price. After visualizing the results from the boxplots of the columns it can be seen that there are lots of outliers present, which should be taken care of before training the machine learning models.

### Capping of outliers

Instead of removing the outliers which can result in loss of information, outliers can be replaced with minimum and maximum capped values. In order to derive the minimum and maximum capped values quantiles values are used, especially interquartile range (IQR). In python **quantile()** function of pandas is used to determine the percentile values of the columns. Then with help of where() function of numpy, outliers are capped by replacing them with minimum and maximum percentile value.

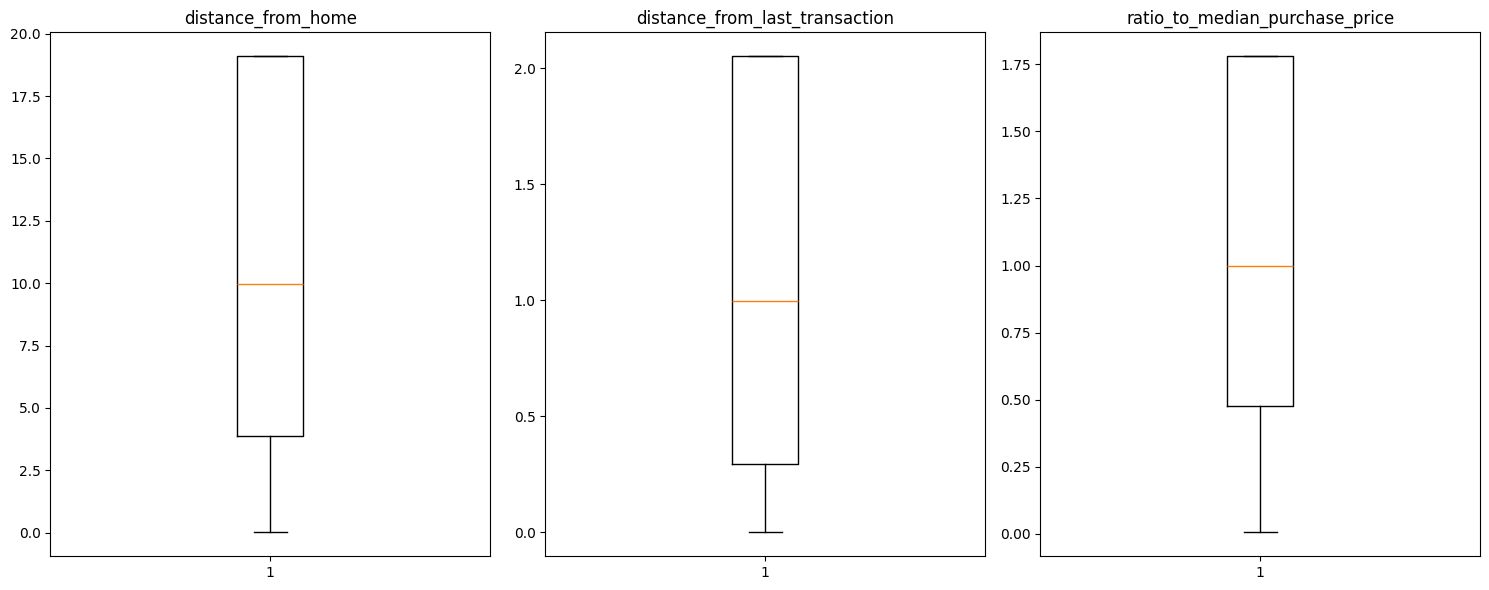






The snapshots displayed above displays how capping of outliers is done on the columns of the dataset such as distance\_from\_home, rato\_to\_mdeian\_purchase\_price and ratio\_to\_median\_purchase\_price. The snapshots further display the use of quantile() function in calculating the quantile values of the column. The results after performing capping on these columns are shown below.

### Detection of Outliers after Capping

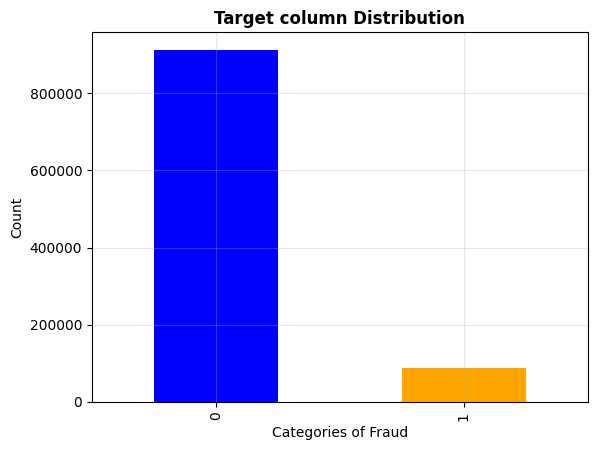


After visualizing the above boxplots it can be concluded that quantile statistical techniques are effective in capping outliers.

## Exploratory Data Analysis (EDA)

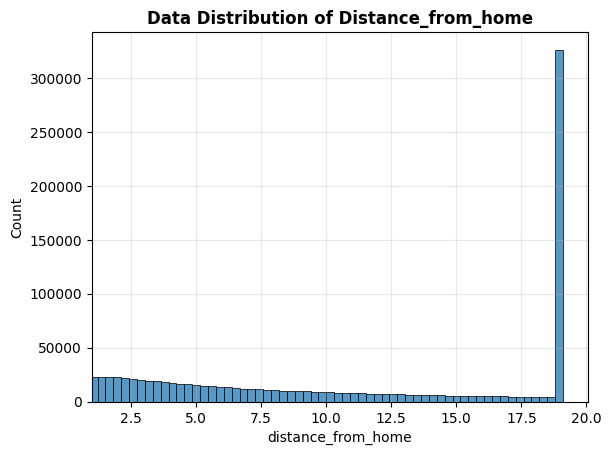
After removal of inconsistencies using the data preprocessing, exploratory data analysis (EDA) is performed. It includes various techniques that are helpful in the data analysis process to make sense of the data. Further EDA helps in summarizing and describing the key characteristics of the dataset. By performing the EDA on the credit card transactions dataset, analysts can gain a deeper understanding of the dataset patterns, which helps to make informed decisions about fraudulent transactions, and find answers/patterns within the given dataset. EDA can also help analysts to determine if the statistical techniques considered for data analysis are appropriate or not.

### Visualization 1



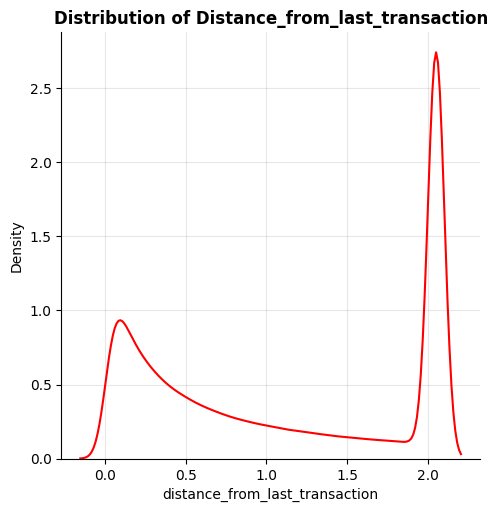
The snapshot shown above displays the distribution of the dataset based on the target column “fraud”. In python value\_counts() and plot() functions from pandas and matplotlib library are used to display the above bar chart. In the above barchart x-axis represents the type of categories while y-axis represents the count of these categories. After analyzing the distribution of the data, it can be concluded that data is highly imbalanced in nature, where out of 100000 total transactions from credit cards, 92% of the transactions are normal and only 8% of the transactions are fraudulent, specifying the relevance of the dataset with the real world transactions. Before training a machine learning model, it is important to address this imbalance in order to prevent bias and produce generalized results from the models.

### Visualization 2



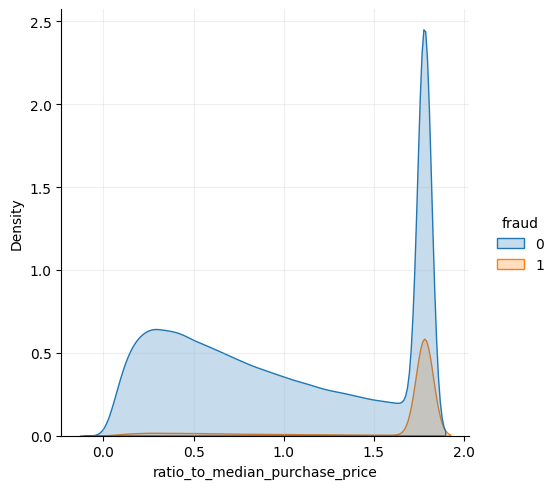
The screenshot shown above displays the distribution of the “distance\_from\_home column”. In python histplot() function from seaborn library is used to display the above histogram. In the above histogram x\_axis represents the distance from home whereas y\_axis represents frequency of the quantity. After analyzing the distribution of the distance\_from\_home column it can be concluded that most of the credit/debit card transactions have been performed within the 5km distance from home. Further the distribution concludes that more than 300000 transactions have been performed when distance from home is more than 18 km.

### Visualization 3



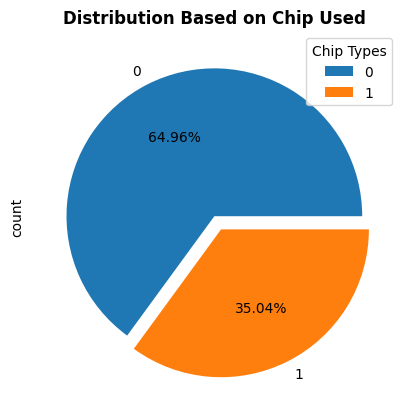
The plot shown displays the distribution of the "distance\_from\_last\_transaction" variable of the dataset. In Python distplot() function from the seaborn library with a “kind” argument is used to display the above KDE (kernel density estimation) plot. In the above KDE plot, x-axis represents the values of the distance\_from\_last\_transaction, and the y-axis represents the density of the data at each value of that column. After visualizing the distribution of the column it can be seen that data is heavily skewed towards the right, with a peak at around 2.0. This means that after 2 days most of the transactions have occurred after that last transaction and there are fewer transactions that have occurred after/within the one day. Overall, the plot provides a detailed visualization of the distribution of the "distance\_from\_last\_transaction" variable, which can be useful for understanding the frequency of transactions of credit/debit cards over time.

### Visualization 4



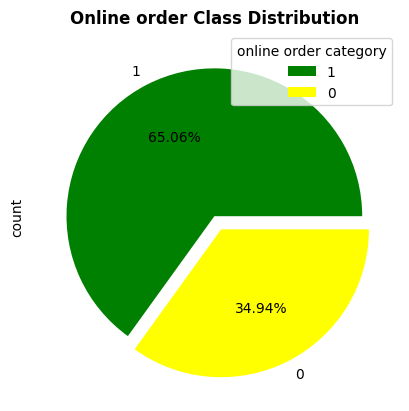
The plot shows the distribution of the "ratio\_to\_median\_purchase\_price" feature for both fraudulent and non-fraudulent transactions. In Python distplot() function from the seaborn library with “kind” and “hue” arguments are used to display the above KDE (kernel density estimation) plot. The “hue” parameter helps to analyze the behavior of the target column. In the above plot x-axis represents the values of the ratio\_to\_median\_purchase\_price, and the y-axis represents the density of the data at each value of ratio\_to\_median\_purchase\_price. After analyzing the patterns of the ratio\_to\_median\_purchase\_price distributions via “hue” as target column, it can be concluded that the distribution for non-fraudulent transactions is skewed towards the left, with a peak around 0.3 and a long tail extending to the right at around 1.7. It further indicates that most non-fraudulent transactions have a ratio\_to\_median\_purchase\_price close to 0.3, with a few transactions having much higher ratios. The distribution for fraudulent transactions is highly skewed to the right, with a sharp peak near 1.8. The outcomes from these distributions suggest that fraudulent transactions tend to have significantly higher ratios as compared to non-fraudulent transactions. The high ratio\_to\_median\_purchase\_price for fraudulent transactions could indicate that fraudulent transactions are more likely to involve purchases that are significantly different from the average purchase price.

### Visualization 5



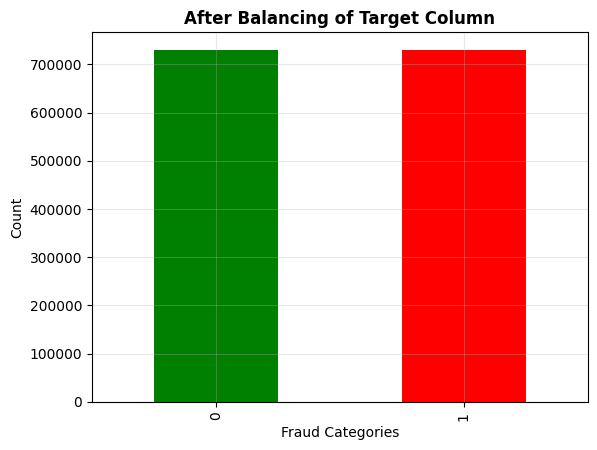
The pie chart shown above displays the distribution of transactions based on whether the chip is used or not. In Python value\_counts() and plot() functions from pandas and matplotlib library are used to display the above pie chart. After visualizing the distributions of the above pie chart it can be concluded that among all the transactions only 35.04% of them are using the chip whereas in 64.96% transactions chip is not used. This Chip column will be helpful in identifying the relationship between the chip used and the fraudulent transactions.

### Visualization 6



The pie chart shown above displays the distribution of transactions based whether performed online or offline transactions via using the credit/debit card. In Python plot() and value\_counts() functions from matplotlib and pandas library are used respectively to develop the above pie chart. After analyzing the outcomes of the pie chart it can be concluded that most of the transactions 65.06% are done via performing the online payments whereas only 34.94% of the transactions of credit/debit card are done while performing the offline payments.

### Visualization 7



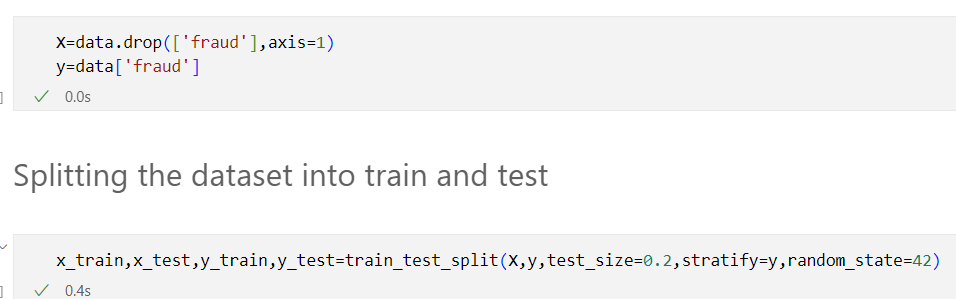
The snapshot shown above displays the distribution of the dataset after performing the balancing of the target column. In python value\_counts() and plot() functions from the pandas and matplotlib are used for displaying the above barchart. After analyzing the distribution of the categories of the target column “fraud” after data balancing, it can be concluded that the distribution of both the categories of the training data becomes equal i.e. 730078 transactions. This will improve both the generalizability of predictions and improves the quality of the credit card transactions dataset.

## Feature Engineering

After analyzing and visualizing the patterns, correlations and relationships between various columns of the dataset using various plots and charts. It becomes important to perform selection, transformation, creation and extraction of the features, before training the machine learning models on the credit card transactions dataset. Feature engineering consists of various features which are as follows:

### Data Split

Before training the machine learning model on the data, it becomes necessary to divide the loaded dataset into training and testing dataset. Training dataset helps machine learning models to identify patterns within the dataset whereas testing dataset is used for evaluating the machine learning models on the new data i.e. data which the machine learning model has n't seen. In python, the train\_test\_split() function from the scikit-learn library is used for splitting the dataset into train and test.



The snapshot shown above displays how data was splitted into the training and testing dataset. After analyzing the snapshots it can be seen that data was splitted into the split of 80-20, where the training dataset contains 80% of the records of credit card transactions and the testing dataset contains 20% of the records present inside the credit transactions dataset.

### Scaling of the dataset

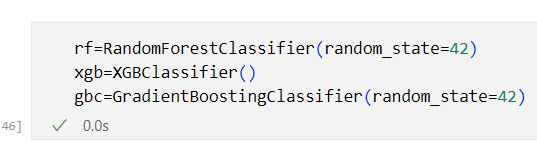
In order to bring all the features under the specified range, feature scaling substep is performed. This step helps in reducing the bias, computational time and leads to improved performance and outcomes of the machine learning model such as improved accuracy, generalizability etc.

|  |  |
| --- | --- |

After analyzing the results from the above table it can be seen that all the features are specified within the fixed range between -3 to +3. In Python StandardScaler() class from scikit-learn library is used to perform the scaling of the loaded dataset. Furthermore, the snapshot displays on the training dataset both fit() and transform() function were computed whereas on the testing dataset only the transform() function is applied.

## Model Development

After preparing the data for the machine learning models it becomes important to develop the identified models selected for the problem. During the model development phase necessary parameters are specified in order to improve the performance of the machine learning models. For the current study various machine learning models were selected and developed including Random Forest, Gradient boosting, and xgboost.



The snapshot shown above displays the development of three selected machine learning models for credit card fraud detection. In Python RandomForestClassifier(), XGBClassifier() and GradientBosstingClassifer() classes from scikit-learn and xgboost respectively are used for developing the above three specified models. The reason behind selection of these algorithms is their high-performance, less training time, scalability and flexibility to perform best even when datasets are complex and large. In the current study the dataset was very large consisting of approximately 1400000 samples of the credit cards, developing such flexible and scalable models was a better choice.

### Training Machine Learning models

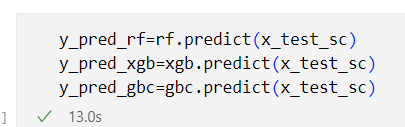
After development/creation of selected models for the problem, it becomes important to train these machine learning models on the training data. During the training phase the machine learning models learn the weights and patterns present inside various features of the training dataset.

|  |
| --- |
|  |
|  |

The screenshots shown in the above table display how training of the machine learning models such as RandomForest, Xgboost and gradient boosting was done by passing the training dataset. The python fit() function from scikit-learn is used for training the three models. After analyzing the above screenshots it can be seen that the parameters required for the model are kept at default values. While training, all the three developed models will learn the patterns of both normal and fraudulent credit/debit transactions based on the features present inside the dataset. This would help in detecting and minimizing the problem of credit card frauds.

## Evaluation of the Models

Evaluation of the machine learning models is done after training the models on the training dataset. In the evaluation phase, performance of the machine learning models is evaluated by using various metrics such as accuracy, precision, f1\_score, recall etc. To evaluate the performance of the machine learning models, predictions are generated from the models by passing the testing dataset to the machine learning models.



The snapshot shown above displays how predictions are generated from the specified machine learning models by passing the testing datasets to the models. In python, the predict() function from the machine learning models is used to generate the predictions from the machine learning models. These predictions are helpful in determining the performance of the models by passing them to the metrics such as accuracy, f1\_score, precision etc.

| **Evaluating the Random Forest Model** |
| --- |
| **Evaluating the Gradient Bossing Mode** |
| **Evaluating the Xgboost model** |

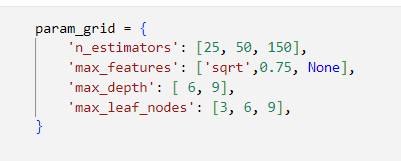
The snapshots shown above display how evaluation of machine learning models such as Random Forest, Gradient Boosting and Xgboost is done by using various metrics such as accuracy score, precision, recall, f1\_score, etc. These results obtained from the specified metrics are displayed below:

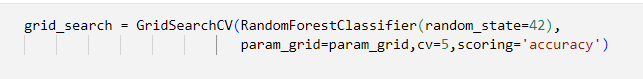
|  |  |
| --- | --- |
|  |  |
|  |  |

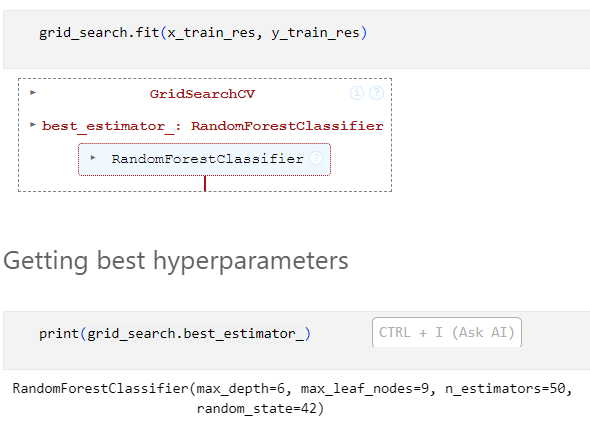
The snapshots shown above display the results obtained from above specified metrics such as accuracy\_score, precision , recall, f1-score etc. After analyzing the results from the screenshots it can be seen that out of all the three machine learning models **RandomForest** outperformed the Xgboost and Gradient boosting model both in terms of the performance and the accuracy of the model. To generate the generalized results and optimize the parameters values of the high performing model i.e. Random Forest model, hyper-parameter tuning is performed.

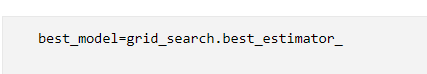
## Hyperparameter tuning

Hyperparameter tuning is the process of finding the best values for the hyperparameters of the model. These parameters act like settings that control the training/learning of the model such as number of decision trees (n\_estimators), max\_features, max-depth, maximum leaf nodes etc. For the current study, hyperparameter tuning was performed on the best performing model i.e. **Random Forest.** In python, the GridSearchCV algorithm from the scikit-learn library is used for performing the hyperparameter tuning of the model. The algorithms helped in finding the optimal parameter values that optimized the performance of the RandomForest model.



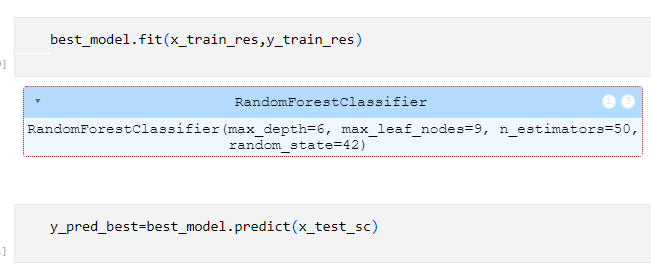




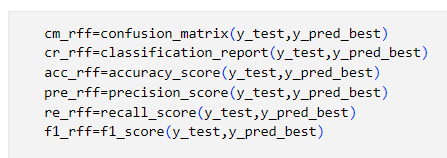


The screenshots shown above display how hyperparameter tuning algorithms like GridsearchCV can be performed on the machine learning model like Random forest. The snapshot1 displays the range and parameters that will be tuned using the GridSearchCV algorithm. After training the tuning algorithm GridsearchCV on the training data, the tuned parameters obtained are **max\_depth as 6**, **max\_leaf\_nodes as 9**, **n\_estimators as 50** and **max\_features as None**. Furthermore the algorithm helped to obtain the **best hypertuned Random Forest model**.

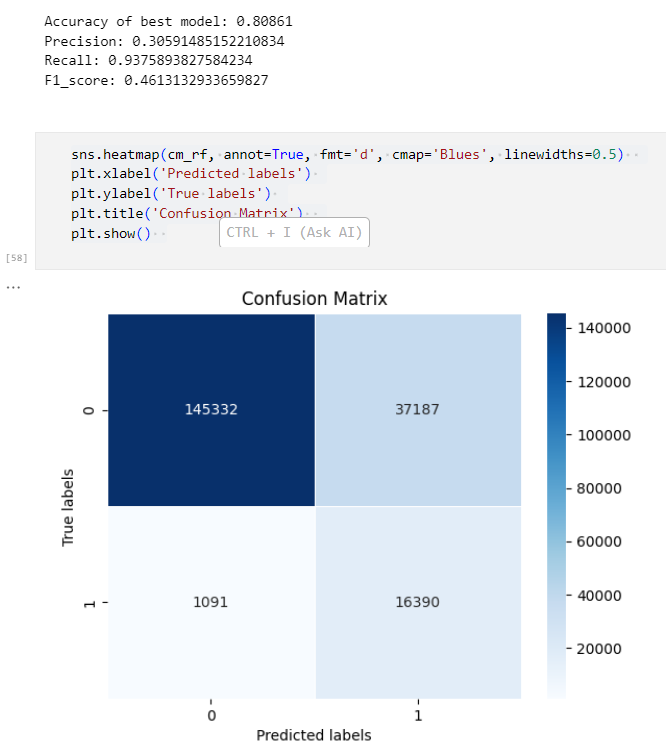
### Training and Evaluation of Best Hypertuned model



The snapshot shown above displays how the best hypertuned model Random Forest model was trained on the training dataset, the hypertuned model will learn the best weights and patterns from the training dataset. Furthermore, the snapshot shown above displays how predictions are generated from the best hypertuned random forest model after passing the testing dataset to the model. These predictions will be helpful in evaluating the performance of the best tuned model by using evaluation metrics such as accuracy\_score, precision, recall etc.



The snapshot shown above displays the evaluation of the best Random Forest model using various evaluation metrics accuracy, precision etc. After evaluation of the model, the performance of the model is shown below.

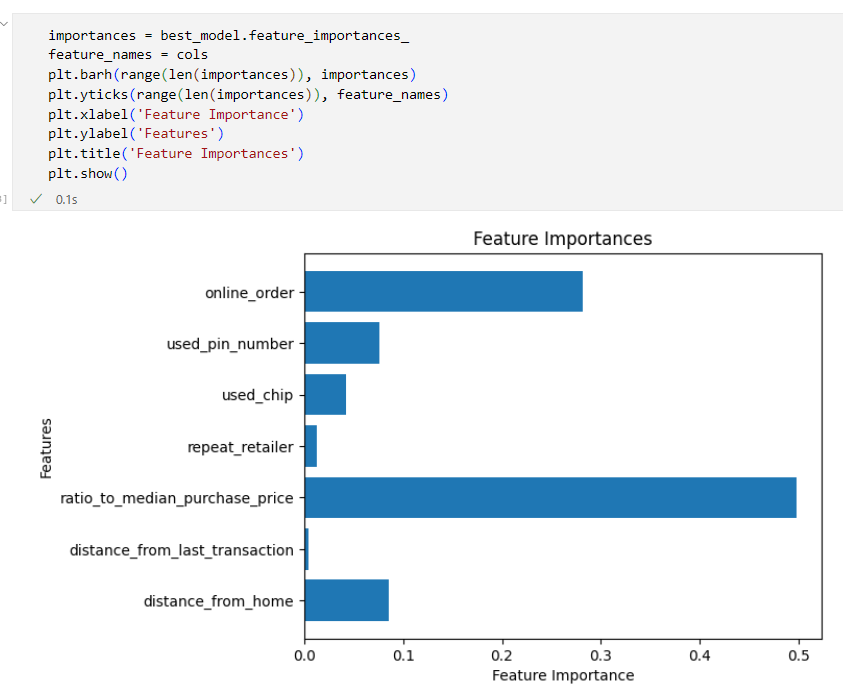


The above shown screenshot displays the results of the best random forest model. After analyzing and visualizing the results from the metrics it can be seen that the recall, precision and f1-score of the model has been increased by 1.5% in precision, 23% in recall and 5.2% in f1-score, highlighting how important it was to performing the hyperparameter tuning of the Random forest model.

## Model Explainability

After evaluating the machine learning models it becomes important to explain the behavior of complex machine learning models to identify the reasons behind complex predictions and decisions made by the machine learning models. These advanced models generate optimized results for a variety of fields, and are named as Black box models due to the complex nature of their internal architecture. To identify the root cause behind the prediction of the Machine learning models, explainability methods came into existence such as feature\_importances, Shap, lime etc. For the current study, feature\_importances attribute and SHAP was used for explaining the reasons behind classification of the transactions as fraudulent or normal.

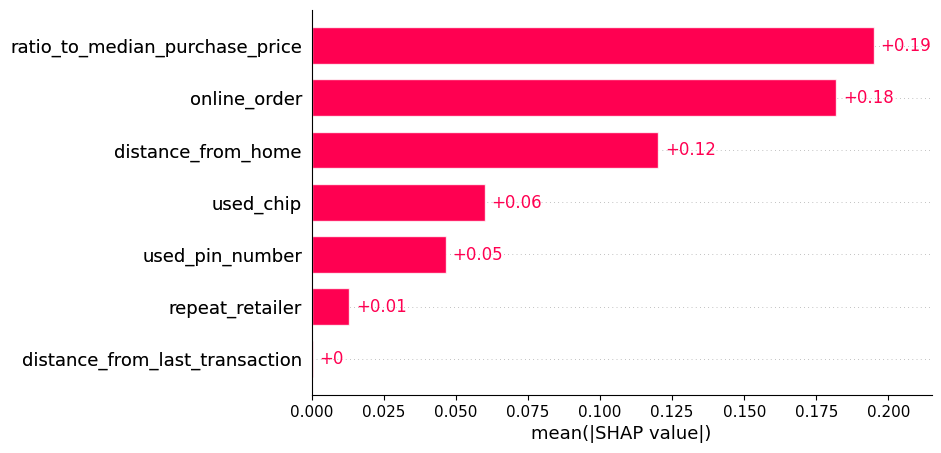
### Model Explainability using feature importances\_ attribute of the ensemble models



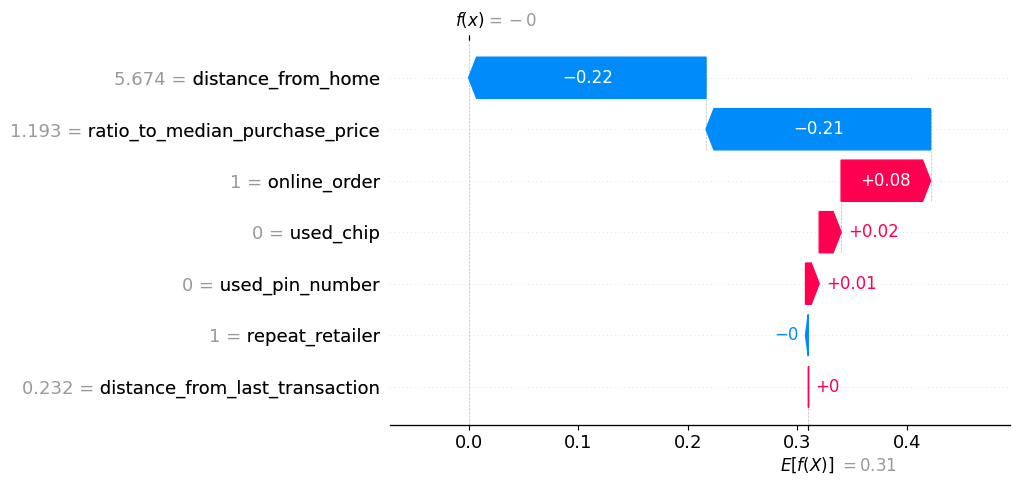
The snapshot shown above displays the importance of each feature of the dataset, according to the best Random Forest model. In python the “feature\_importances\_” attribute is used to identify the importance of features in determining the output. After analyzing the above results it can be concluded that rato\_to\_median\_purchase\_price is the most important feature with feature importance of 0.49861632 and distance\_from\_last\_transaction is the least important feature in determining the output with a feature importance of 0.004. The overall findings are displaying that columns like rato\_to\_median\_purchase\_price, online order are showing highest contribution in determining the output whereas distance\_from\_last\_transaction and repeat\_retailer are displaying lowest contribution in determining the prediction as normal or fraudulent.

### Model Explainability using SHAP

Shapley Additive Explanations (SHAP) is model explainability method which helps in the global interpretation of the complex machine learning models (Black box). This method helps in identifying the reason behind why certain predictions were classified as normal and why they were classified fraudulent. The SHAP algorithm uses the concept of game theory to identify the contribution of each player (feature) in determining the output (target column).



The snapshot shown above displays the contribution/importance of each variable in the dataset in classifying the transaction as normal or fraudulent. After analyzing the results from the above it can be concluded that columns like rato\_to\_median\_purchase\_price, online order play a major role in classifying prediction as normal or fraudulent with an importance of +0.19 and +0.18 respectively. While distance\_from\_last\_transaction is showing almost nothing contribution in classifying the prediction with feature importance of 0.



The snapshot shown above displays the contribution of various features on the ist prediction of the Random Forest model. Each bar in the plot represents the average impact of a feature on the Random Forests prediction on the first instance. The red color in the plot means the column shows positive correlation with the prediction whereas blue color means the feature showed the negative correlation with the prediction. For example, the feature "distance\_from\_home" has a negative impact on the ist prediction, while the feature "online\_order" has a positive impact on the ist prediction from the model. The longer the bar, the stronger the impact. The baseline prediction of the Random forest model as per SHAP is 0.31. This means Random Forest predicts a value of 0.31 when all features are having their average values. The SHAP summary plot helps to understand the relationship between various features and the predictions generated by the Random Forest model.