CS 452 Data Science with Python Assignment 2 Report

Car Rollover Prediction

Author	Uygar KAYA
Instructor	Assistant Prof. Reyhan Aydoğan
Submission Date	19.12.2021

1 Introduction

In this assignment, we implement a Support Vector Machine, Naïve-Bayes, Logistic Regression, & Random Forest models are trained to learn the Car Rollover Prediction.

In this assignment, we access the FARS dataset with the help of the API provided to us on the notebook. We get different results for each different machine learning model by putting the final data set we obtained after making some instructions into 4 different machine learning models.

In this assignment, we use the 7 instructions which are.

1. Data Loading
2. Exploratory Data Analysis
3. Data Cleaning and Feature Engineering
4. Data Splitting and Transformation
5. Feature Selection
6. Modelling and Performance Evaluation
7. Interpretation

1.1 Support Vector Machine

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression, and outliers' detection.

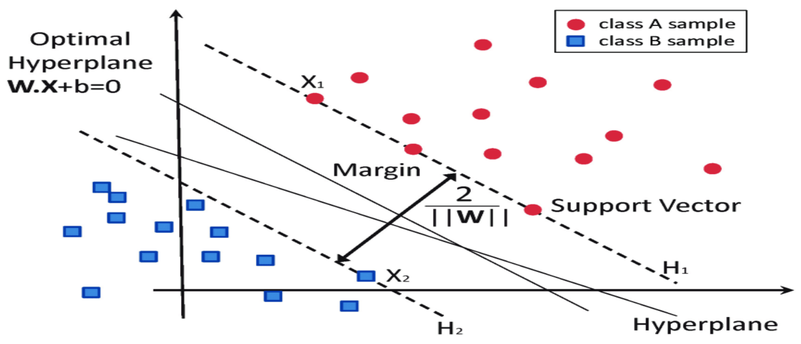


Figure 1 – Example of Support Vector Machines [1]

1.2 Naïve-Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.

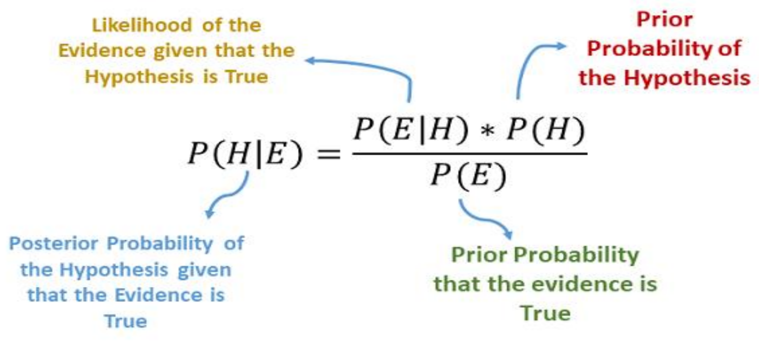

Likelihood of the Evidence given that the Hypothesis is True
Prior Probability of the Hypothesis
$P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$
Posterior Probability of the Hypothesis given that the Evidence is True
Prior Probability that the evidence is True

Figure 2 – Formula of Naïve-Bayes [2]

1.3 Logistic Regression

Logistic Regression is often used for predictive analytics and modeling and extends to applications in machine learning.

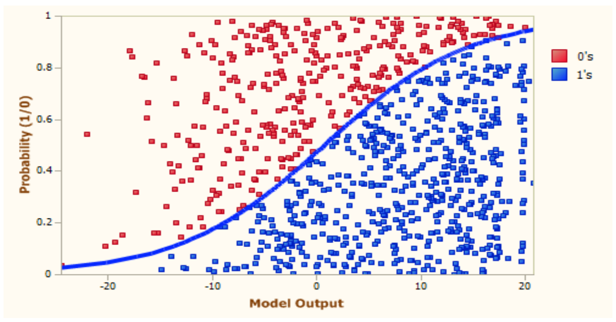


Figure 3 – Example of Logistic Regression [3]

1.4 Random Forest

Random Forest is a meta estimator that fits several decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

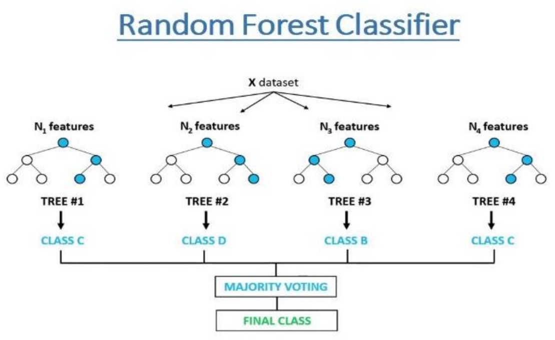


Figure 4 – Example of Random Forest [4]

2 Methodology

In this assignment, we apply a Support Vector Machine, Naïve-Bayes, Logistic Regression, and Random Forest models methodology using Python Programming Language.

During this assignment, using only 5 different Python programming language libraries these are.

1. NumPy
2. Pandas
3. Scikit-Learn
4. Matplotlib
5. Seaborn
6. Requests
7. Io
8. Random were used.

First of all, after pulling two types of datasets named person and vindecode in the FARS database for the state years from 2014 to 2019 with the help of the API, two different DataFrames were created and these two different DataFrames were printed for examination. In addition, after data is cleared Categorical & Numerical features were determined in these raw data, and pre-processing was performed for these determined data. After the data was pre-processed, Dropping & Imputation (Handling with Missing Values) was performed for null values and reducing cardinality of categorical features has been done. Also, Categorical data were converted into numerical data by One-Hot Encoding. Finally, machine learning models was applied after the Data Splitting into two different variables as Test Data and Train Data. In order to find the best model, we choose the model that performs the best out of 2 different models with at most 30 features and the performance of this model is visualized & presented in this report.

3 Implementation Details

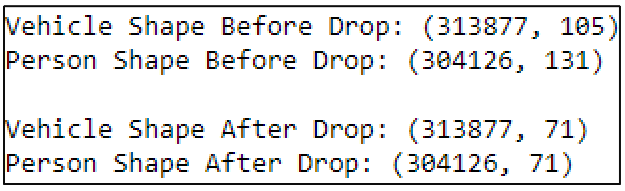
3.1 Data Loading & Cleaning

3.1.1 Get Vehicle & Person Dataset

First, two types of datasets named person and vindecode in the FARS database for the state years from 2014 to 2019 were fetch with the help of API and turned into two different DataFrames. These DataFrames are named PersonDF and VehicleDF and PersonDF raw DataFrame has 498002 rows & 131 columns and VehicleDF DataFrame has 313877 rows & 105 columns.

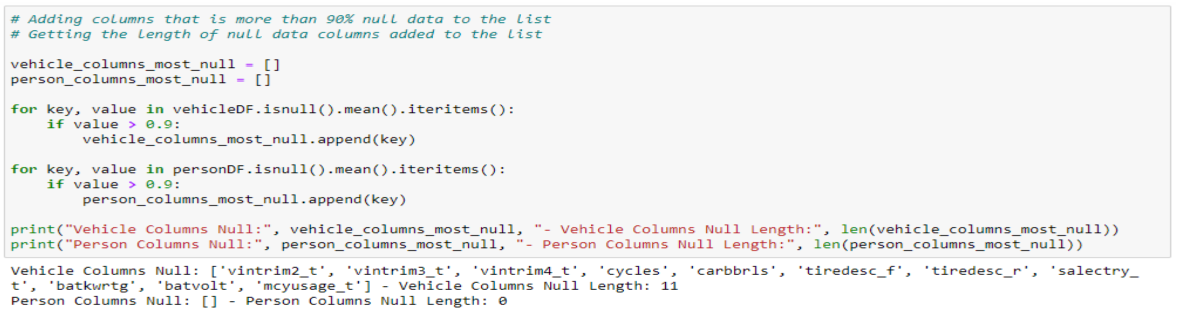
3.1.2 Determine Useless Columns & Drop from DataFrames

After the data is taken, during the cleaning phase of the raw data, firstly, the non-driver people were dropped in the PersonDF DataFrame according to the condition specified in the piazza. In the FARS dataset we have two columns for some features, which are encoded & decoded values. According to the notebook hint 1, we should drop the encoded columns. For this, we should detect the encoded & decoded values and then, we drop the encoded values since we cannot make sense of the encoded values, we cannot reduce of cardinality in other parts of the notebook. Therefore, the Encoded values do not provide us with any information, and when we do One-Hot Encoding and Imputation in our DataFrame later, it would create extra unnecessary columns and rows; hence, we drop these encoded values.



```
Vehicle Shape Before Drop: (313877, 105)
Person Shape Before Drop: (304126, 131)

Vehicle Shape After Drop: (313877, 71)
Person Shape After Drop: (304126, 71)
```

Secondly, according to the notebook hint 2, we should drop the mostly null values & single value. First, two different empty lists are created for both Vehicle and Person DataFrames and we determine the columns and we determine the columns that are more than 90% null in both different DataFrames and append them to the list we created.

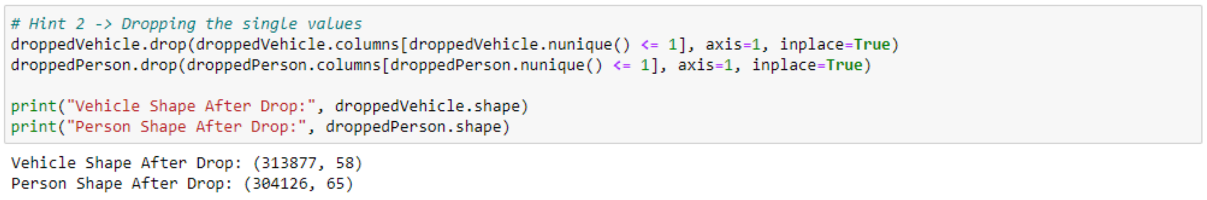


```
# Adding columns that is more than 90% null data to the list
# Getting the length of null data columns added to the list

vehicle_columns_most_null = []
person_columns_most_null = []

for key, value in vehicleDF.isnull().mean().iteritems():
    if value > 0.9:
        vehicle_columns_most_null.append(key)

for key, value in personDF.isnull().mean().iteritems():
    if value > 0.9:
        person_columns_most_null.append(key)

print("Vehicle Columns Null:", vehicle_columns_most_null, "- Vehicle Columns Null Length:", len(vehicle_columns_most_null))
print("Person Columns Null:", person_columns_most_null, "- Person Columns Null Length:", len(person_columns_most_null))
```

```
Vehicle Columns Null: ['vintrim2_t', 'vintrim3_t', 'vintrim4_t', 'cycles', 'carbbrls', 'tiredesc_f', 'tiredesc_r', 'salectry_
t', 'batkwrtg', 'batvolt', 'mcyusage_t'] - Vehicle Columns Null Length: 11
Person Columns Null: [] - Person Columns Null Length: 0
```

Figure 5 – Example of Null Columns List

After dropping Null Columns, we get 313877 rows & 60 columns in the VehicleDF and 304126 rows & 71 columns in the PersonDF. Also, we drop the single values like below,

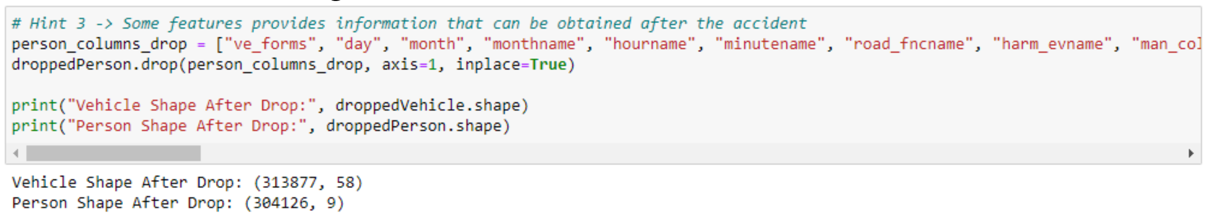


```
# Hint 2 -> Dropping the single values
droppedVehicle.drop(droppedVehicle.columns[droppedVehicle.nunique() <= 1], axis=1, inplace=True)
droppedPerson.drop(droppedPerson.columns[droppedPerson.nunique() <= 1], axis=1, inplace=True)

print("Vehicle Shape After Drop:", droppedVehicle.shape)
print("Person Shape After Drop:", droppedPerson.shape)
```

```
Vehicle Shape After Drop: (313877, 58)
Person Shape After Drop: (304126, 65)
```

Figure 6 – Example of Single Columns List

According to the notebook, some features provide information that can be obtained after the accident. We should detect these columns and dropping them because we cannot use the values after the accident, so we are removing these data from our DataFrames.



```
# Hint 3 -> Some features provides information that can be obtained after the accident
person_columns_drop = ["ve_forms", "day", "month", "monthname", "hourname", "minutename", "road_fncname", "harm_evname", "man_col
droppedPerson.drop(person_columns_drop, axis=1, inplace=True)

print("Vehicle Shape After Drop:", droppedVehicle.shape)
print("Person Shape After Drop:", droppedPerson.shape)
```

```
Vehicle Shape After Drop: (313877, 58)
Person Shape After Drop: (304126, 9)
```

Figure 7 – Example of After Accidents Columns List

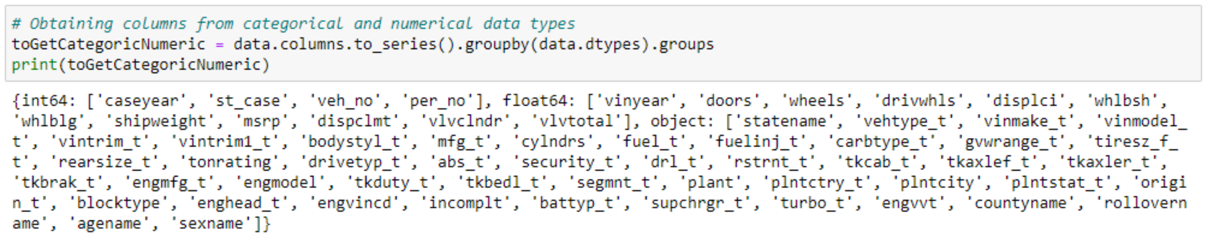
3.1.3 Merging Vehicle & Person DataFrames

In this part, first of all, I return an “accidients” DataFrame by inner joining two different DataFrames according to "caseyear", "st_case", "veh_no", "statename" key columns. Then, by converting this DataFrame to data.csv file, there are approximately 304126 rows & 63 columns of data at the end.

3.2 Exploratory Data Analysis (EDA)

In this part, explore Categorical & Numerical features, detecting missing values, dropping duplicate and empty rows/columns, plotting the histograms, scatter & bar plots, printing the value counts of categorical features and printing the statistical tables

In the final “data” DataFrame, there are 63 different features in total, some of them are of Int & Float data types, while others are of Object data type. Also, our target name is rollovername.

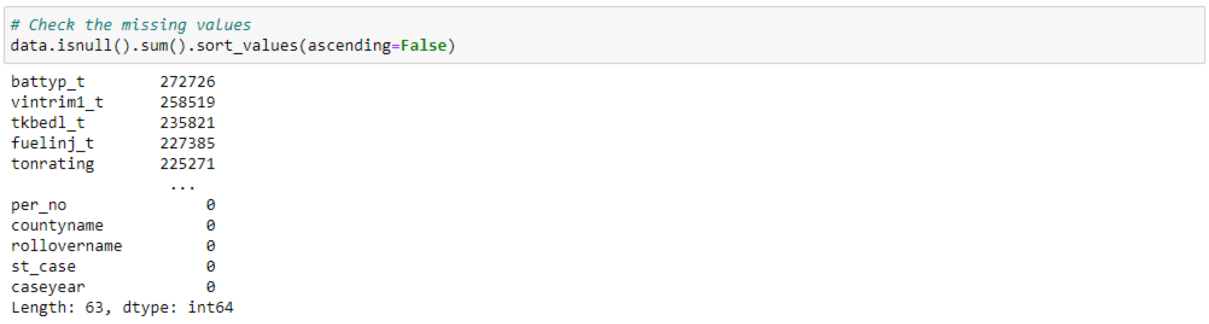


```
# Obtaining columns from categorical and numerical data types
toGetCategoricNumeric = data.columns.to_series().groupby(data.dtypes).groups
print(toGetCategoricNumeric)
```

```
{int64: ['caseyear', 'st_case', 'veh_no', 'per_no'], float64: ['vinyear', 'doors', 'wheels', 'drivwhls', 'displci', 'whlbsh',
'whlblg', 'shipweight', 'msrp', 'dispclmt', 'vlvclndr', 'vlvtotal'], object: ['statename', 'vehtype_t', 'vinmake_t', 'vinmodel_
t', 'vintrim_t', 'vintrim1_t', 'bodystyl_t', 'mfg_t', 'cylndrs', 'fuel_t', 'fuelinj_t', 'carbtype_t', 'gvwrange_t', 'tiresz_f_
t', 'rearsize_t', 'tonrating', 'drivetyp_t', 'abs_t', 'security_t', 'drl_t', 'rstrnt_t', 'tkcab_t', 'tkaxlef_t', 'tkaxler_t',
'tkbrak_t', 'engmfg_t', 'engmodel', 'tkduty_t', 'tkbedl_t', 'segmnt_t', 'plant', 'plntctry_t', 'plntcity', 'plntstat_t', 'origi
n_t', 'blocktype', 'enghead_t', 'engvincd', 'incomplt', 'battyp_t', 'supchrgr_t', 'turbo_t', 'engvvt', 'countyname', 'rollovern
ame', 'agename', 'sexname']}
```

Figure 8 – To Get the Categorical & Numerical Dictionary

There are number of missing rows in the some features.



```
# Check the missing values
data.isnull().sum().sort_values(ascending=False)
```

```
battyp_t        272726
vintrim1_t      258519
tkbedl_t        235821
fuelinj_t       227385
tonrating       225271
                 ...
per_no               0
countyname           0
rollovername         0
st_case              0
caseyear             0
Length: 63, dtype: int64
```

Figure 9 – Checking the Missing Values

The most NaN values are available in the battyp_t, vintrim1_t, tkbedl_t, fuelinj_t, tonrating and more columns. In the preprocessing part, the relationship of these NaN values with selectKbest values is checked and the machine learning model is applied without including the unrelated data in the train set. The remaining NaN values were processed with the Handling with Missing Values (Dropping & Imputation) method and used in the machine learning models by generating data for the missing numerical values. Also, we are dropping the duplicates values if exist. At the end of the drop_duplicates() method, we detect the there is no duplicates values in the “data” DataFrame. At the end of the process, we have 304126 rows & 63 columns.

Moreover, I checked the emptiest Rows & Columns data in order to apply this process, we calculate the mean of the null values of each column, we remove those columns when they are more than 80%. At the end of the process, I get the only two columns have the most null values which are, “vintrim1_t”, “battyp_t”. After dropping these columns, we get the shape 304126 rows & 61 columns. After this process, I rearrange the categorical & numerical lists. Also, I suppressed the histogram for the numeric values.

```
# Drawing a histogram for each numeric feature
data[numeric].hist(figsize=(15,20))

# Clear the text "residue"
plt.show()
```

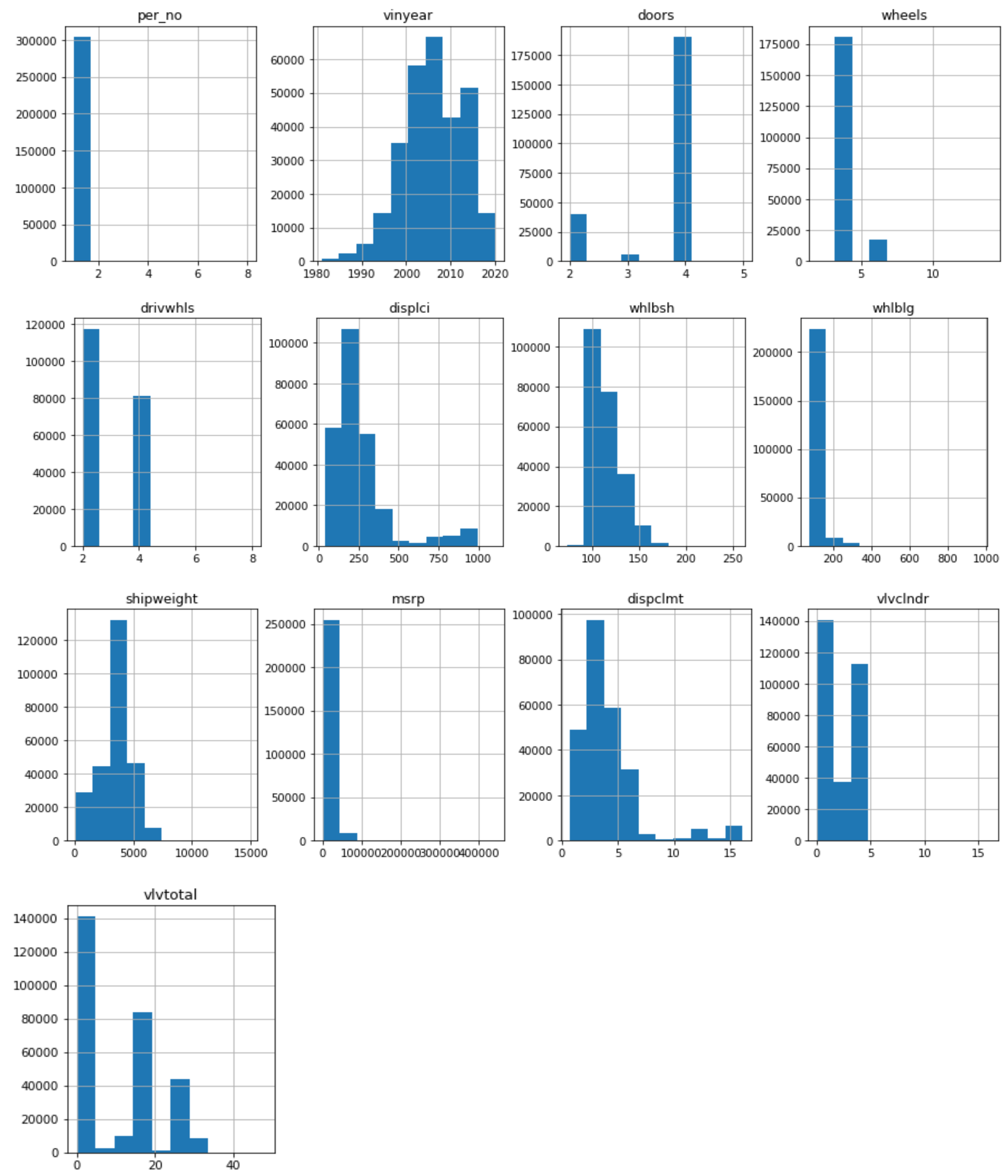


Figure 10 – Histogram of the Numerical Values

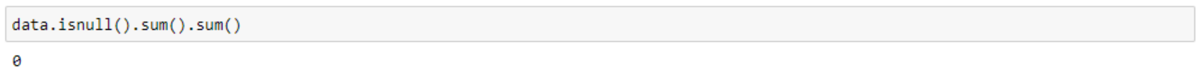
Then, the value_counts() of the Categorical data were suppressed and the barplot graphs of these data were drawn. Numeric features are described and their values such as count, mean, std are examined. At the end of the 3.**2 Exploratory Data Analysis (EDA)** part Target Variable barplot was examined. (Please go to part **Part 2 - Exploratory Data Analysis (EDA)** of the notebook to see the visualizations of Categorical & Numerical values).

3.3 Data Cleaning and Feature Engineering

3.3.1 Handling with Missing Values: Dropping and/or Imputation

In this section, we give new values to some NaN numeric values by applying the imputation methodology and maybe we can drop some columns or rows.

First of all, I drop some unnecessary columns because the columns here have too many rows, so when we do reduce of cardinality, unnecessary values would be difficult for us, so I dropped the columns that are not related to target_name and will not affect the accuracy of the model. After dropping the "data" DataFrame, we rearrange the Categorical & Numeric list's and at the end of the process, we get the 304126 rows and 53 columns. After dropping some columns, I create the mean_value_list & mode_value_list. In the mean_value_list, we fill the numeric NaN values with the column's mean() values. The main reason for me to do mean() is because it represents the numeric data more accurately than mean(), mode() & median(), I filled these NaN values by taking mean. In the categorical data, I apply the most used value. After applying the this part, we fill the all NaN values with their mean & mode and when we look at the isnull() method for categorical & numerical data, we get the 0 null value



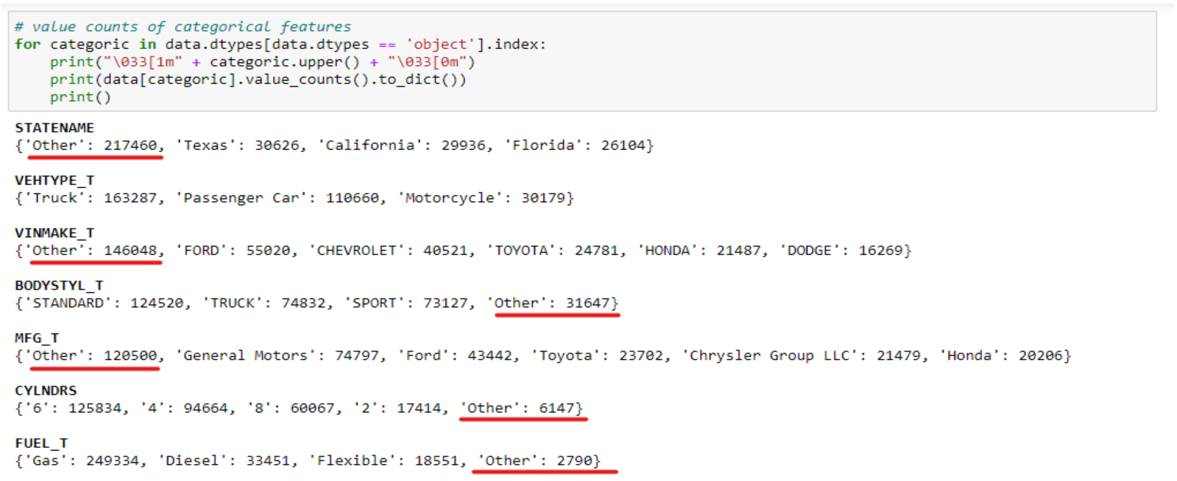
```
data.isnull().sum().sum()
```

```
0
```

Figure 10 – Showing the Sum of the Null data

3.3.2 Reducing Cardinality of Categorical Features

Some categorical features have a high cardinality, resulting in many dummy variables. As a result, some operations on those features are unavoidably required. In this section, we create a single column by combining the equal feature values meaning of the columns, and by one-hot encoding this column to the machine learning model, we allow the accuracy of the model to increase. In this part, I just apply the these methodology in the categorical data which are, vehtype_t, carbtype_t, abs_t, drl_t, origin_t, turbo_t, supchrgr_t, rollovername, blocktype, tkbrak_t, fuel_t, drivetyp_t, bodystyl_t, cylndrs, segmnt_t, tiresz_f_t, rearsize_t, agename. In these categorical columns, before applying the main operation, I manually change the values of only some columns by doing the pre-porcessing operation. After these pre-porcesing part, I apply the main methodology which is by looking at the values of all categorical column data and making the values less than 5% of these values "Other", I reduce the number of the values of these columns and prevent excessive number of columns when I do one-hot encoding. I then analyze the new values by suppressing the value_counts() of this newly edited categorical data. (Please go to part **Part 3 - Data Cleaning and Feature Engineering** of the notebook to see the rearranging values)



```
# value counts of categorical features
for categoric in data.dtypes[data.dtypes == 'object'].index:
    print("\033[1m" + categoric.upper() + "\033[0m")
    print(data[categoric].value_counts().to_dict())
    print()
```

```
STATENAME
{'Other': 217460, 'Texas': 30626, 'California': 29936, 'Florida': 26104}

VEHTYPE_T
{'Truck': 163287, 'Passenger Car': 110660, 'Motorcycle': 30179}

VINMAKE_T
{'Other': 146048, 'FORD': 55020, 'CHEVROLET': 40521, 'TOYOTA': 24781, 'HONDA': 21487, 'DODGE': 16269}

BODYSTYL_T
{'STANDARD': 124520, 'TRUCK': 74832, 'SPORT': 73127, 'Other': 31647}

MFG_T
{'Other': 120500, 'General Motors': 74797, 'Ford': 43442, 'Toyota': 23702, 'Chrysler Group LLC': 21479, 'Honda': 20206}

CYLNDRS
{'6': 125834, '4': 94664, '8': 60067, '2': 17414, 'Other': 6147}

FUEL_T
{'Gas': 249334, 'Diesel': 33451, 'Flexible': 18551, 'Other': 2790}
```

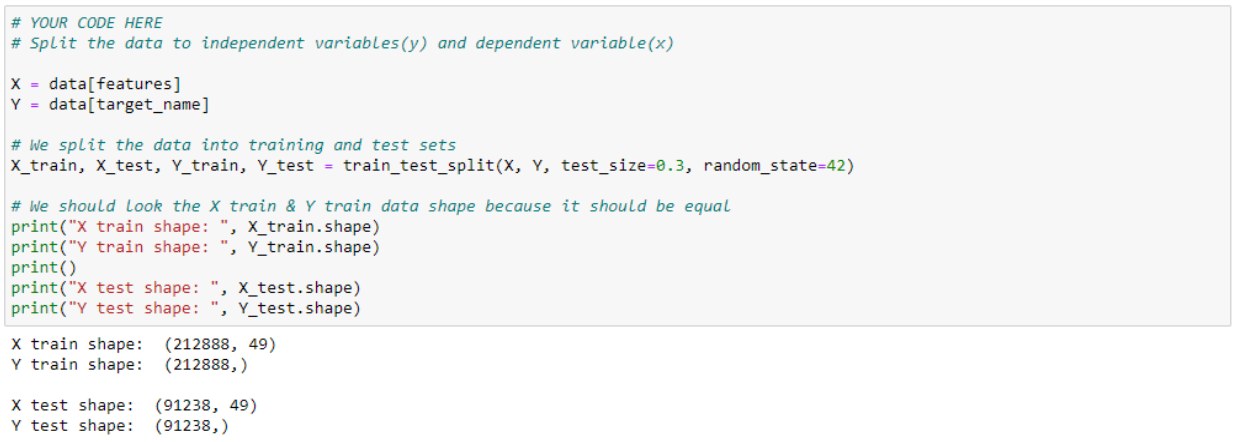
Figure 11 – Showing the Counts of Categorial Data

Before the **Data Splitting and Transformation** part and the After the **Data Cleaning and Feature Engineering** part, I applied the replace the target name to 0 & 1, which is 'No Rollover': 0 & 'Rollover': 1

3.4 Data Splitting and Transformation

In this section, I split the "data" DataFrame data into training & test datasets before the standard scaler. After this process, I scale the numerical features and after these I encode the categorical features based on the train dataset.

In order to split the data in the train & test set, we specify the X and Y. In the X, we set the data[features] which is features = Numerical + Categorical lists. After that, we should describe the X_train, X_test, Y_train & Y_test to split the data train & test set and then we print the shape of these veriables. Then we look at the shapes of the training sets we have created, because the shapes of the train sets, we have separated must be the same.



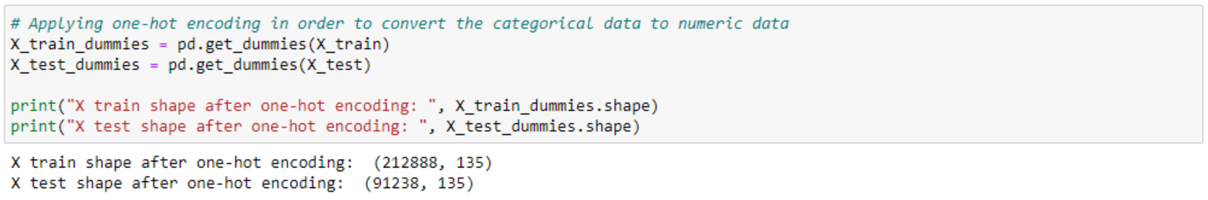
```
# YOUR CODE HERE
# Split the data to independent variables(y) and dependent variable(x)

X = data[features]
Y = data[target_name]

# We split the data into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=42)

# We should look the X train & Y train data shape because it should be equal
print("X train shape: ", X_train.shape)
print("Y train shape: ", Y_train.shape)
print()
print("X test shape: ", X_test.shape)
print("Y test shape: ", Y_test.shape)
```

```
X train shape:  (212888, 49)
Y train shape:  (212888,)

X test shape:  (91238, 49)
Y test shape:  (91238,)
```

Figure 12 – Example of Split the Data into Training & Test Dataset

As seen above, a train set of 70% was created for each different DataFrame and the shape of each is the same size.

After this process, we scale these data using the StandartScaler operation to ensure that each dependent variable train sets are in the transforms the data in such a manner that it has mean as 0 and standard deviation as 1, and we complete the Standardization process by using the fit_transform() & transform() method in the StandartScaler for Standardization. I choose the StandartScaler because StandartScaler arranges the data in a standard normal distribution. At the end of this section, we apply the One-Hot Encoding in order to convert the categorical data to numerical data.



```
# Applying one-hot encoding in order to convert the categorical data to numeric data
X_train_dummies = pd.get_dummies(X_train)
X_test_dummies = pd.get_dummies(X_test)

print("X train shape after one-hot encoding: ", X_train_dummies.shape)
print("X test shape after one-hot encoding: ", X_test_dummies.shape)
```

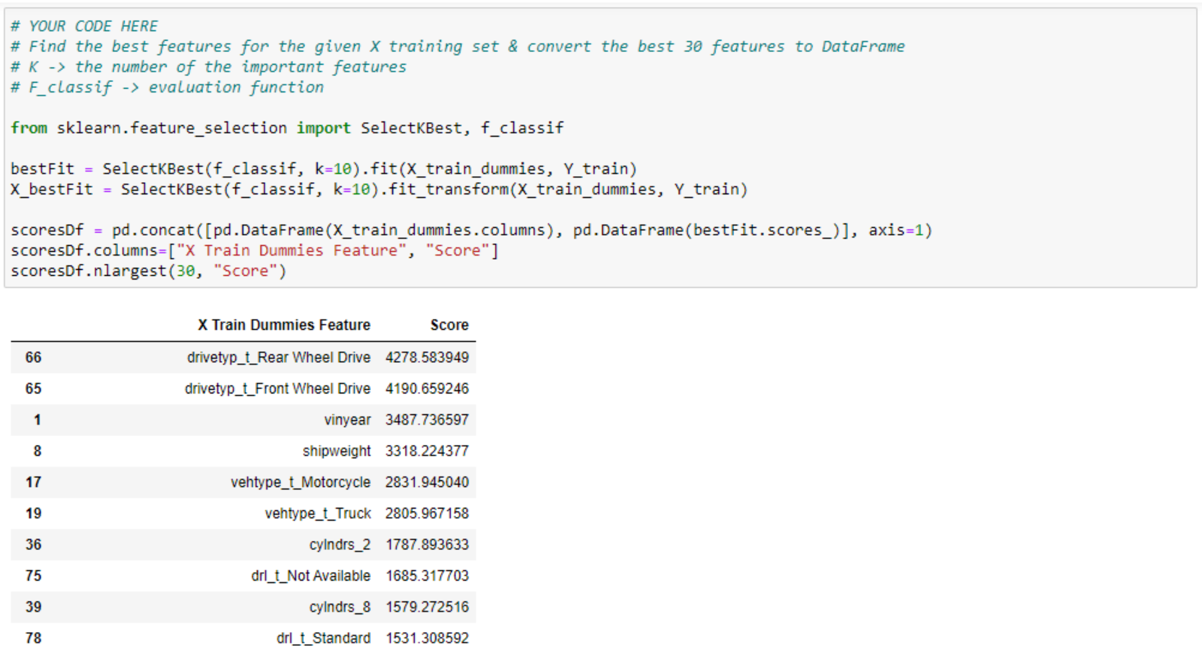
```
X train shape after one-hot encoding:  (212888, 135)
X test shape after one-hot encoding:  (91238, 135)
```

Figure 13 – Example of One-Hot Encoding

3.5 Feature Selection

In this section, I select the best features for our machine learning model to achieve a higher accuracy rate. We can find these features with manually or automatically. In manually, we can use the Thresholding Correlations, Multicollinearity Inspection or more. In automatically, we can use the

SelectKBest, Recursive Feature Elimination or more. I prefer the automatic way in order to find the best features, I use the selectKBest Library because the run time of the SelectKBest is more efficient than the Recursive Feature Elimination and SelectKBest more reliable than the manually selection therefore, I select to use SelectKBest Library for selecting the features according to the k highest score. Also, after selection the best 30 features, I create the best feature DataFrame in order to see more clearly to these features and their scores.



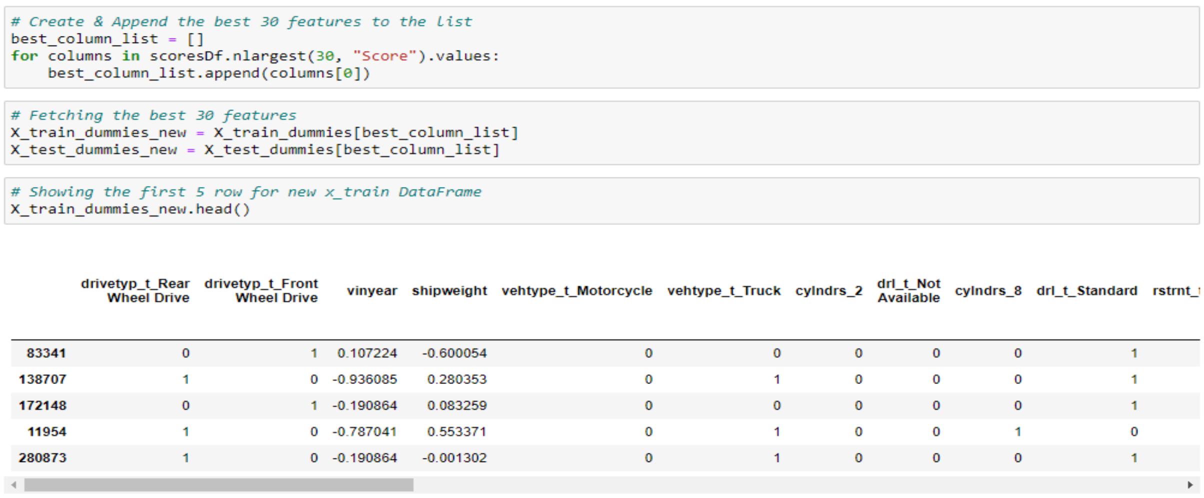
```python
# YOUR CODE HERE
# Find the best features for the given X training set & convert the best 30 features to DataFrame
# K -> the number of the important features
# F_classif -> evaluation function

from sklearn.feature_selection import SelectKBest, f_classif

bestFit = SelectKBest(f_classif, k=10).fit(X_train_dummies, Y_train)
X_bestFit = SelectKBest(f_classif, k=10).fit_transform(X_train_dummies, Y_train)

scoresDf = pd.concat([pd.DataFrame(X_train_dummies.columns), pd.DataFrame(bestFit.scores_)], axis=1)
scoresDf.columns=["X Train Dummies Feature", "Score"]
scoresDf.nlargest(30, "Score")
```

	X Train Dummies Feature	Score
66	drivetyp_t_Rear Wheel Drive	4278.583949
65	drivetyp_t_Front Wheel Drive	4190.659246
1	vinyear	3487.736597
8	shipweight	3318.224377
17	vehtype_t_Motorcycle	2831.945040
19	vehtype_t_Truck	2805.967158
36	cylndrs_2	1787.893633
75	drl_t_Not Available	1685.317703
39	cylndrs_8	1579.272516
78	drl_t_Standard	1531.308592

Figure 14 – Example of the Best Feature

As you can see the most important feature is the **drivetyp_t_Rear Wheel Drive** which score is "4278" and **drivetyp_t_Rear Wheel Drive** "4190" and goes like this. In the SelectKBest Library, I use the "f_classif" which is the evaluation function. F_classif computes the analysis of variance F-value for the provided sample. F_classif works the feature by feature that means f_classif ratio the one feature to another feature and computes the anova f-value.

After getting the best 30 features, I create the best_column_list in order to keep these 30 features in the list and use them to the training of the machine learning models.



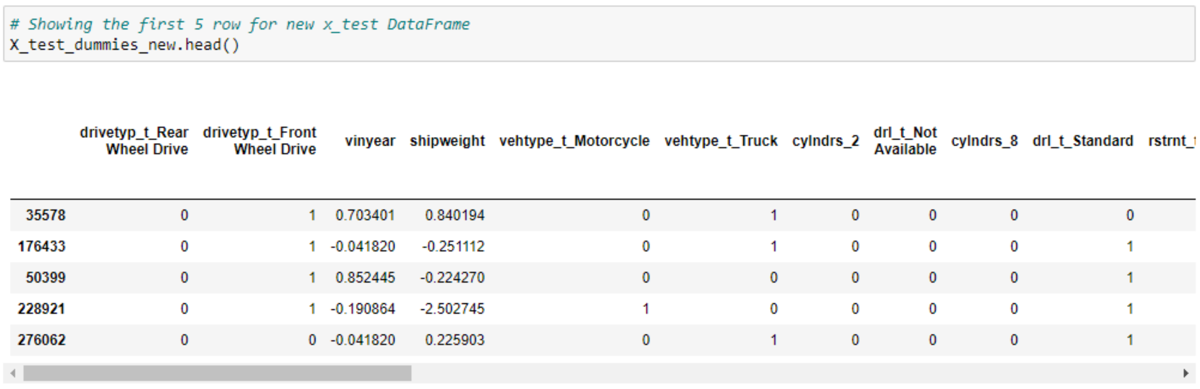
```python
# Create & Append the best 30 features to the List
best_column_list = []
for columns in scoresDf.nlargest(30, "Score").values:
    best_column_list.append(columns[0])
```

```python
# Fetching the best 30 features
X_train_dummies_new = X_train_dummies[best_column_list]
X_test_dummies_new = X_test_dummies[best_column_list]
```

```python
# Showing the first 5 row for new x_train DataFrame
X_train_dummies_new.head()
```

	drivetyp_t_Rear Wheel Drive	drivetyp_t_Front Wheel Drive	vinyear	shipweight	vehtype_t_Motorcycle	vehtype_t_Truck	cylndrs_2	drl_t_Not Available	cylndrs_8	drl_t_Standard	rstrnt_
83341	0	1	0.107224	-0.600054	0	0	0	0	0	1	
138707	1	0	-0.936085	0.280353	0	1	0	0	0	1	
172148	0	1	-0.190864	0.083259	0	0	0	0	0	1	
11954	1	0	-0.787041	0.553371	0	1	0	0	1	0	
280873	1	0	-0.190864	-0.001302	0	1	0	0	0	1	

Figure 15 – Example of the X Train DataFrame with Best Feature



```
# Showing the first 5 row for new x_test DataFrame
X_test_dummies_new.head()
```

	drivetyp_t_Rear Wheel Drive	drivetyp_t_Front Wheel Drive	vinyear	shipweight	vehtype_t_Motorcycle	vehtype_t_Truck	cylndrs_2	drl_t_Not Available	cylndrs_8	drl_t_Standard	rstrnt_
35578	0	1	0.703401	0.840194	0	1	0	0	0	0	
176433	0	1	-0.041820	-0.251112	0	1	0	0	0	1	
50399	0	1	0.852445	-0.224270	0	0	0	0	0	1	
228921	0	1	-0.190864	-2.502745	1	0	0	0	0	1	
276062	0	0	-0.041820	0.225903	0	1	0	0	0	1	

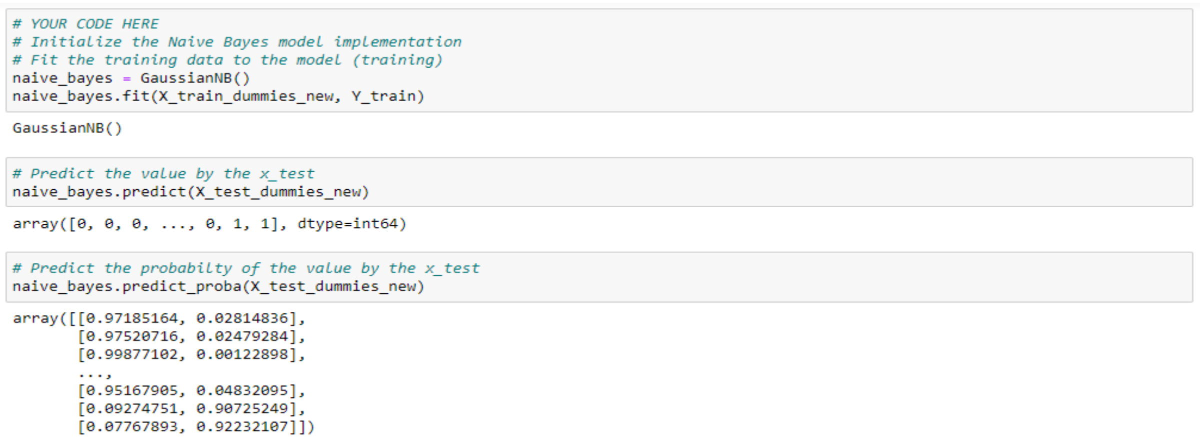
Figure 16 – Example of the X Test DataFrame with Best Feature

3.6 Modelling and Performance Evaluation

In this part, we evaluate the performance of these models by applying 4 different machine learning models which are **Support Vector Machine, Naïve-Bayes, Logistic Regression, and Random Forest**.

3.6.1 Naïve Bayes

First of all, we initialize Naïve Bayes and fit the train data we created, and we get a GaussianNB() model as an output and we predict the test value. Also, we predict the probabilities of this model.



```
# YOUR CODE HERE
# Initialize the Naive Bayes model implementation
# Fit the training data to the model (training)
naive_bayes = GaussianNB()
naive_bayes.fit(X_train_dummies_new, Y_train)
```

```
GaussianNB()
```

```
# Predict the value by the x_test
naive_bayes.predict(X_test_dummies_new)
```

```
array([0, 0, 0, ..., 0, 1, 1], dtype=int64)
```

```
# Predict the probabilty of the value by the x_test
naive_bayes.predict_proba(X_test_dummies_new)
```

```
array([[0.97185164, 0.02814836],
       [0.97520716, 0.02479284],
       [0.99877102, 0.00122898],
       ...,
       [0.95167905, 0.04832095],
       [0.09274751, 0.90725249],
       [0.07767893, 0.92232107]])
```

Figure 17 – Implementation of the GaussianNB

I select the GaussianNB since the GaussianNB implements the Gaussian Naïve Bayes algorithm for classification models. After applying the GaussianNB, I plot the confusion matrix for visualization in order to see the precision, recall and f1 score.

```
# Confusion Matrix Visualization in order to see precission, recall & f1 score
from sklearn.metrics import ConfusionMatrixDisplay, classification_report, confusion_matrix
ConfusionMatrixDisplay(confusion_matrix = confusion_matrix(Y_test, naive_bayes.predict(X_test_dummies_new))).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1641254c910>
```

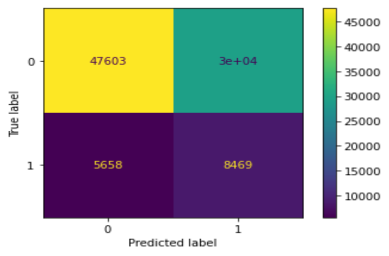


Figure 18 – Visualization of the Confusion Matrix

Furthermore, I print the classification report to see the accuracy & average value to evaluate the performance of the model.

```
# Classification_report in order to see the accurracy & avarage value
print(classification_report(Y_test, naive_bayes.predict(X_test_dummies_new)))
```

```
              precision    recall  f1-score   support

           0       0.89      0.62      0.73     77111
           1       0.22      0.60      0.33     14127

    accuracy                           0.61     91238
   macro avg       0.56      0.61      0.53     91238
weighted avg       0.79      0.61      0.67     91238
```

Figure 19 – Classification Report of GaussianNB

At the end of the Naïve Bayes model, we convert the classification report to dictionary in order to plot the bar chart for the training performance of the model.

3.6.2 Logistic Regression

First of all, we initialize Logistic Regression and fit the train data we created, and we get a LogisticRegression(C=1, class_weight = “balanced”) model as an output and we predict the test value. Also, we predict the probabilities of this model. I select the class_weight = “balanced” in order to get more balanced model. Also, I print the Regression Model Score which is 0.60

```
# Initialize the Logistic Regression model implementation
# Fit the training data to the model (training)
# C is the regularization parameter to avoid the overfitting

logisticRegModel = LogisticRegression(C=1, class_weight='balanced')
logisticRegModel.fit(X_train_dummies_new, Y_train)
```

```
# Predict the value by the x_test
logisticRegModel.predict(X_test_dummies_new)
```

```
# Predict the probabilty of the value by the x_test
logisticRegModelProba = logisticRegModel.predict_proba(X_test_dummies_new)
```

Figure 20 – Implementation of the Logistic Regression

I plot the confusion matrix for visualization in order to see the precision, recall and f1 score. Furthermore, I print the classification report to see the accuracy & average value to evaluate the performance of the model. At the end of the Logistic Regression model, we convert the classification report to dictionary in order to plot the bar chart for the training performance of the model.

3.6.3 Support Vector Machines

First of all, we initialize Support Vector Machines and fit the train data we created, and we get a SVC(C=0.1, kernel='linear', random_state=0) model as an output and we predict the test value. Also, we predict the probabilities of this model. In the Logistic Regression part I select the class_weight = “balanced” in order to get more balanced model but in this model, I not select because of the run-time.

```
# Confusion Matrix Visualization in order to see precission, recall & f1 score
ConfusionMatrixDisplay(confusion_matrix = confusion_matrix(Y_test, supportVectorMach.predict(X_test_dummies_new))).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x158998ae0a0>
```

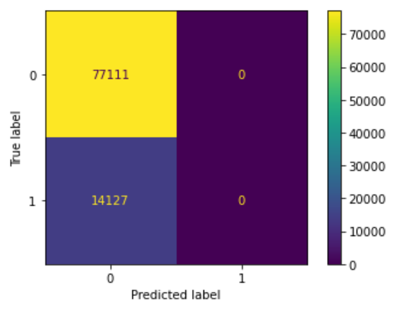
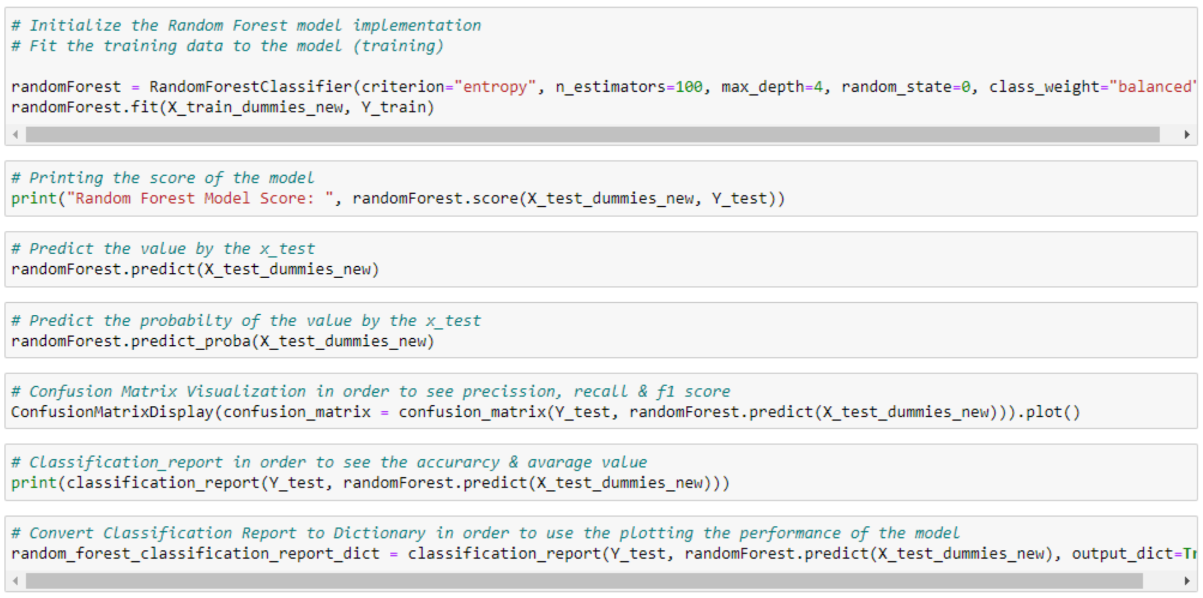


Figure 21 – Implementation of the Support Vector Machines

I plot the confusion matrix for visualization in order to see the precision, recall and f1 score. Furthermore, I print the classification report to see the accuracy & average value to evaluate the performance of the model. At the end of the Support Vector Machine model, we convert the classification report to dictionary in order to plot the bar chart for the training performance of the model.

3.6.4 Random Forest

First of all, we initialize Random Forest and fit the train data we created, and we get a model as an RandomForestClassifier(criterion='entropy', max_depth=4, random_state=0, class_weight='balanced') output and we predict the test value. Also, we predict the probabilities of this model. I select the class_weight = "balanced" in order to get more balanced model.



```python
# Initialize the Random Forest model implementation
# Fit the training data to the model (training)

randomForest = RandomForestClassifier(criterion="entropy", n_estimators=100, max_depth=4, random_state=0, class_weight="balanced"
randomForest.fit(X_train_dummies_new, Y_train)
```

```python
# Printing the score of the model
print("Random Forest Model Score: ", randomForest.score(X_test_dummies_new, Y_test))
```

```python
# Predict the value by the x_test
randomForest.predict(X_test_dummies_new)
```

```python
# Predict the probabilty of the value by the x_test
randomForest.predict_proba(X_test_dummies_new)
```

```python
# Confusion Matrix Visualization in order to see precission, recall & f1 score
ConfusionMatrixDisplay(confusion_matrix = confusion_matrix(Y_test, randomForest.predict(X_test_dummies_new))).plot()
```

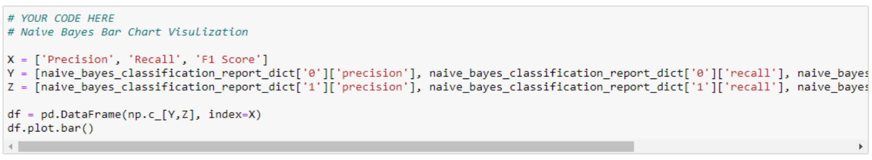
```python
# Classification_report in order to see the accurracy & avarage value
print(classification_report(Y_test, randomForest.predict(X_test_dummies_new)))
```

```python
# Convert Classification Report to Dictionary in order to use the plotting the performance of the model
random_forest_classification_report_dict = classification_report(Y_test, randomForest.predict(X_test_dummies_new), output_dict=Tr
```

Figure 22 – Implementation of the Random Forest

I plot the confusion matrix for visualization in order to see the precision, recall and f1 score. Furthermore, I print the classification report to see the accuracy & average value to evaluate the performance of the model. At the end of the Random Forest model, we convert the classification report to dictionary in order to plot the bar chart for the training performance of the model.

After applying the machine learning models, I plot training performance of the model with a bar chart.



```python
# YOUR CODE HERE
# Naive Bayes Bar Chart Visulization

X = ['Precision', 'Recall', 'F1 Score']
Y = [naive_bayes_classification_report_dict['0']['precision'], naive_bayes_classification_report_dict['0']['recall'], naive_bayes
Z = [naive_bayes_classification_report_dict['1']['precision'], naive_bayes_classification_report_dict['1']['recall'], naive_bayes

df = pd.DataFrame(np.c_[Y,Z], index=X)
df.plot.bar()
```

```
<AxesSubplot:>
```

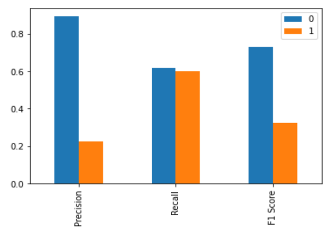


Figure 22 – Example of the Naïve Bayes Performance Evaluation Bar Chart

In the notebook as you can see, the Logistic Regression and the Random Forest predictions are similar, we expect that if the Support Vector Machine has the “class_weight = ‘balanced’” parameter. The whole performance is equal.

4 Appendix

You Can Find the Final “data” Dataset in the Fallowing Google Drive Link.

https://drive.google.com/file/d/1pu7Xa1vkTT3FBZwOqHlrVKEODKKg1qI5/view?usp=sharing

5 Reference

[1] García-Gonzalo, E., Fernández-Muñiz, Z., Nieto, P. J. G., Menéndez, M., Wei, Z.-lei, Lu, Q., Sun, H.-yue, &amp; Zhenlei, W. (2021, November 17). Figure 2. classification of data by support Vector Machine (SVM). ResearchGate. Retrieved December 18, 2021, from https://www.researchgate.net/figure/Classification-of-data-by-support-vector-machine-SVM_fig8_304611323

[2] Gamal, B. (2020, December 17). Naïve Bayes Algorithm. LaptrinhX. https://laptrinhx.com/naive-bayes-algorithm-4229686008/

[3] Joy, A. (2021, December 12). Logistic Regression Using Python - Abin Joy. Medium. https://medium.com/@abinmj656/logistic-regression-using-python-28a269152f53

[4] Nazlı, R. (2021, June 22). R ile Random Forest - Rümeysa Nazlı. Medium. https://medium.com/@rmys.nzl/r-ile-random-forest-3909b0706d86