Predictive Analytics

ALY 6020, CRN 80405

Professor Vladimir Shapiro

Predictive Analytics - Final Exam

Submitted By - Richa Umesh Rambhia

Module 6 Final Exam - Analyzing Car Insurance Data

Table of Contents

Theory Questions

Please provide only textual answers and 2-3 sentences around the rationale of why that is (no coding is required)

Q1. Give a definition of impurity. Please follow that up with a business example of low and high impurity.

Q2. Compare and discuss similarities or differences between AIC and R2?

Q3. Name the three activation function techniques mentioned in class for Neural Networks. Provide an example of a business problem leveraging at least one such technique.

Q4. Why is it important to find balance in the learning rate of a Gradient Boosted Model? Is there a learning rate for Random Forest?

Q5. Provide two examples of Machine Learning models that are suitable to support decision-making by executive management. Name two optimization algorithms used in Machine Learning.

Answering Question 1 to Question 5:

Q1. Give a definition of impurity. Please follow that up with a business example of low and high impurity.

A1. Impurity is defined as a measure of the disorder or an uncertainty within a set of data points or values or with respect to a specific node in the decision tree algorithm. It is most **commonly used in the decision tree models** for either classification or regression based problems in order to **determine the optimal splitting criteria for the model.**

Impurity helps in **evaluating the quality of the partitions in the model** and help to decide which attribute or variable to split on, where the goal is to minimize the impurity and achieve the greatest possible separation of classes or reduction of uncertainty.

The two commonly used impurity measures in the decision tree algorithm are **Gini Impurity and Entropy**, where Gini Impurity calculates the probability of misclassifying a randomly chosen element from the dataset, and on the other hand, Entropy measures the average amount of information or uncertainty within a given subset of data.

1. **Low Impurity:** Considering the example of **spam or no spam emails dataset.** The decision tree model is trained on the dataset which uses the Gini impurity measure to determine the evaluation of the element. The node of the algorithm after training might have a very low impurity value which means that the majority of emails in the node are either all spam or not spam. Thus, the low impurity measure indicates that the node is highly pure and effective in separating the two classes.

2. **High Impurity:** On the other hand, considering the same example, if a node in the decision tree model has a high impurity measure or value, it means that the sample data points within that node have a mix of different classes and there is no proper separation between the two classes. This indicates that there is a high level of uncertainty in the decision-making process and the

Q2. Compare and discuss similarities or differences between AIC and R-squared?

A2. AIC score and R-squared are both statistical measures which are used in the regression model analysis that help provide information about the model performance.

AIC Score: AIC Score is used for model selection and comparison in the context of regression analysis and other statistical models where it provides a method to balance the goodness of fit of a model.

R-squared: R-squared on the other hand which is also known as coefficient of determination is a measure that quantifies the proportion of a variance in the dependent variable which is explained by the independent variable in the regression analysis model and is used to represent the goodness of fit of a model.

- **Similarities between AIC and R-squared**

  Both AIC and R-squared are used to **evaluate the goodness of fit** of machine learning or regression models.
- **Differences between AIC and R-squared**

  AIC and R-squared measures have different purposes that provide information about evaluation of regression models.

  AIC is used for **model selection which considers both the goodness of fit of model and the complexity of the model**, whereas R-squared helps in understanding the measure of how well the regression model has performed and how well the model is able to explain the dependent variable's variance, i.e., **it provides insight on how well the model fits the data.**

  AIC score represents a **relative measure allowing the comparison of models** based on their model fit and complexity. On the other hand, R-squared represents the **percentage of the variance of the dependent variable** which is explained by the independent variable.

  A **higher R-squared value** indicates a better fit of model along with a higher level or percentage of variance. On the contrary, **lower AIC scores** indicate better performing models.

  Lastly, AIC score **does not have a specific range of values or scale** of score, whereas R-squared **ranges from 0 to 1** where 0 indicates a poor fit model and that the model is not able to explain the variance and 1 indicates a perfect fit model where the model can explain all its variance.

Q3. Name the three activation function techniques mentioned in class for Neural Networks. Provide an example of a business problem leveraging at least one such technique.

A3. The three activation function techniques mentioned in class for Neural Networks are:

- **Sigmoid or Logistic Activation Function:** The sigmoid function curve looks like a S-shape and is especially used for models where the probability as an ouput needs to be predicted or for binary classification, and since the probability of anything exists between 0 to 1, Sigmoid activation function can be used in such scenarios.
- **Tanh or hyperbolic tangent Activation Function:** The tanh function is similar to sigmoid function which has a S-shape curve too but maps the input between -1 and 1. This function is used for classification between the two classes.
- **ReLU (Rectified Linear Unit) Activation Function:** ReLU is the most common activation function that is used as it is used in all the deep learning and convoluation neural network models. The function outputs the input value if it is positive and zero otherwise.
- **Example of a business problem leveraging Tanh Activation function:** Considering an example where the business wants to predict the **returns of the stock market** for the next trading day which is based on different factors and parameters. A neural network model is built with a tanh activation function in the output layer which predicts the continuous values of the data points. Here, the tanh activation function provides outputs in a way that are centered around zero which helps in analyzing the positive and negative returns and also capture the non linear relationship between the input factor and the stock market performance.

Q4. Why is it important to find balance in the learning rate of a Gradient Boosted Model? Is there a learning rate for Random Forest?

A4. The **learning rate** determines the contribution of each tree in the ensemble model and helps control the step size at each iteration of the boosting process. It is important to find the right balance in the learning rate of a Gradient Boosted Model in order to achieve the optimal performance.

The reasons why it is important to find the right balance in the learning rate of a Gradient Boosted Model are as follows.

1. A higher learning rate will allow the model to learn faster and overfit the training data which will fit the noise or outliers in the model. On the other hand, a lower learning rate will prevent the overfitting of the model but may result in underfitting of the model if there are limitations on the number of iterations. Hence, balancing the learning rate helps in generalizing well to the unseen data and avoiding the problem of overfitting or underfitting of the data.
2. With respect to the noise, a lower learning rate will make the model less sensitive to noise and errors in data points and reduce the impact of outliers, whereas a higher learning rate will make it more sensitive to noise, which will again lead to underfitting or overfitting of data, and hence it is important to balance the learning rate.
3. Lastly, a higher learning rate can lead shorter training times which means that the model quickly learns the patterns in the dataset, whereas a lower learning rate would require more iterations in order to reach the convergence. Thus, in order to avoid the model to overshoot the optimal solution, it is important to find the right balance in the learning rate.

With respect to the **learning rate for Random Forest model**, the concept of **learning rate is not applicable** as it is applied in Gradient Boosted algorithms. Random Forest models are ensemble models which combine multiple decision trees and there is **no iterative optimization process as it is in gradient boosting**. Thus, Random Forest models does not have a learning rate parameter.

Q5. Provide two examples of Machine Learning models that are suitable to support decision-making by executive management. Name two optimization algorithms used in Machine Learning.

A5. The two examples of Machine Learning models that are suitable to support decision-making by executive management are as follows.

1. **Linear Regression model**

   Linear Regression is a supervised machine learning model which is used for prediction of continuous values based on the input features. It is a simple and interpretable model which helps in analyzing the relationship between the independent and the dependent variables. Linear regression model are suitable for exeutive decision-making by executive management in a way to understand the linear relationships and to estimate the numerical values of a dataset. For example, considering the sales forecasting, linear regression can help in predicting the future sales which are based on historical data and the management can use these predictions to make informed decisions. The linear regression model thus is easy to interpret and helps organizations in their decision-making process.
2. **SVM model**

   Support Vector Machine (SVM) model is another supervised machine learning model that can be used for both classification and regression based problems. SVM models can be helpful for executive decision-making in situations where precise forecast or distinct class borders are required. For instance, at a financial institution, SVMs may help in classifying the credit risk and figuring out the probability of the loan defaults, enabling the management to decide on loan approvals and handle the risk with knowledge.

The two optimization algorithms used in Machine Learning are as follows.

1. **Gradient Descent**

   Gradient Descent is an optimization algorithm which is used in machine learning in order to find the optimal parameters of a model which is done by iteratively minimizing a loss function. The algorithm is widely used in various models which include linear regression, logistic regression, neural networks, and support vector machines.
2. **Stochastic Gradient Descent**

   Stochastic Gradient Descent is another optimization algorithm which is a variant of the Gradient Descent algorithm. This algorithm is particularly useful when working with large datasets as it updates the parameters of the model which uses a single randomly selected data point at each iteration rather than using the entire dataset as a whole in Gradient Descent.

Implementation of Machine Learning models

Introduction

Logistic Regression Model

Logistic Regression is a *statistical type of machine learning model* that helps in the **classification and predictive analytics** in order to estimate the probability and likelihood of an event occuring. [1] The difference between a Linear Regression model and a Logistic Regression model is that Linear Regression algorithms are used to identify relationships between a continuous dependent variable and one or more independent variable, whereas Logistic Regression models are used to **predict a category based on categorical type** variable versus the continuous data points.

The different types of Logistic Regression algorithms which are defined based on the categorical data points are as follows.

1. **Binary Logistic Regression**
2. **Multinomial Logistic Regression**
3. **Ordinal Logistic Regression**

Nearest Neighbor Model

K-nearest algorithm or KNN is one of the simplest classification and regression model in machine learning which is categorized into supervised learning and is implemented by calculating the distance between the data points in order to predict the category for the new data point. This algorithm uses the feature similarity concept in order to predict the values of the data points based on how closely the data point matches the points present in the training set. [2]

KNN algorithm calculates the distance between the two points and there are four methods of calculating the promixity of the data points, which are as follows.

1. Euclidean distance
2. Manhattan distance
3. Minkowski distance
4. Hamming distance

The most common method that is used for calculating the distance between the instances is the **Euclidean method**.

After the distance or the K-nearest neighbors are identified, the algorithm will now start predicting the class of the test data points based on the training dataset and the majority class of its K neighbors, where K will be the number of neighbors which is any integer value. [2]

The algorithm thus does not require much training but it can be expensive and memory-intensive when implemented for large datasets. Apart from that, the *K value* and the *distance metric* are important parameters that help to decide the **efficiency of the KNN algorithm**.

SVM Model

A Support Vector Machine (SVM) model is a powerful supervised machine learning algorithm which is used for **classification and regression** based problems. The main idea of the SVM algorithm is to **find the best hyperplane** which *separates the two classes into a high-dimensional feature space*. The algorithm thus tries to find a hyperplane that maximizes the margin between the classes. [3] Hyperplanes are decision boundaries that help classify the data points, and the data points fall on either side of the hyperplane that can be attributed to different classes.

Decision Tree Model

Decision Tree algorithm is a powerful tool for **classification and regression** based problems, which is a **flowchart-like tree structure** where the internal node denotes a test on the attribute, each branch represents the outcome of the test, and each leaf node (terminal node) represents a class label. Decision Tree algorithms are a tree-like model that helps in decision making that is followed by a sequence of if-else questions about the various input data until a prediction is made. [4]

Neural Network Model

Neural Network model is a type of machine learning algorithm which is designed in a way to **mimic the behavior of the human brain** and is composed of layers of the **interconnected nodes or neurons** in order to process and transmit the information. These algorithms are widely used for a variety of machine learning tasks in different applications. [5]

Problem Statement

You have a dataset about whether or not someone got in an accident that year. This insurance company needs to understand which features of a person or vehicle make them more likely to get in an accident.

Clean missing values first, but no exploratory info is needed. You need to predict the Outcome column.

Car Insurance Data Analysis

Car Insurance Data Analysis is based on the **insurance dataset** which has over 17 different features and parameters in order to understand and analyze the data. This will help in recommending the company what parameters are to be considered when someone applies for a car insurance.

The insurance dataset has about **10000 rows of data points and 17 field values** where the goal is to build different classification models based on the target variable to **classify the target variable**. The various factors that affect the response parameter can be further analyzed to understand the parameters of the dataset that are affecting and contributing to the target variable.

Thus, the goal here is to provide specific recommendations to the insurance company of what parameters they need to look out for when someone applies for car insurance with them and how they can reduce that risk when someone uses their insurance.

Analysis

Task 1:

Clean missing values first, but no exploratory info is needed. You need to predict the Outcome column.

Task 2:

Build a logistic regression model and discuss the significant variables. Provide a table of all significant variables and their coefficients (a snippet of the data is not acceptable and if there are no variables at .05 or under, feel free to expand to .1). From your initial thoughts, which variable sticks out to you as intriguing that it is significant and why. How could this information be useful to the insurer?

Task 3:

Run a few non-ensemble models (only ones used in class) using (1000 iterations). Address the accuracy of each model and why you choose that model. Which model is the most accurate?

Task 4:

Build a confusion matrix for each model; discuss which part of the confusion matrix a company would want to reduce and which model does the best at doing so.

Task 5:

In two paragraphs minimum, discuss the features that were important and significant from the models. Use that to provide a specific recommendation to the insurance company of what they need to look out for when someone applies for car insurance with them and how they can reduce that risk when someone uses their insurance.

Installing required packages

In [1]:
```
!pip install pandas_profiling
!pip install featurewiz
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Collecting pandas_profiling
  Downloading pandas_profiling-3.6.6-py2.py3-none-any.whl (324 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 324.4/324.4 kB 6.2 MB/s eta 0:00:00
Collecting ydata-profiling (from pandas_profiling)
  Downloading ydata_profiling-4.1.2-py2.py3-none-any.whl (345 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 345.9/345.9 kB 14.6 MB/s eta 0:00:00
Collecting scipy<1.10,>=1.4.1 (from ydata-profiling->pandas_profiling)
  Downloading scipy-1.9.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (33.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 33.7/33.7 MB 15.5 MB/s eta 0:00:00
Requirement already satisfied: pandas!=1.4.0,<1.6,>1.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (1.5.3)
Collecting matplotlib<3.7,>=3.2 (from ydata-profiling->pandas_profiling)
  Downloading matplotlib-3.6.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.8/11.8 MB 54.4 MB/s eta 0:00:00
Requirement already satisfied: pydantic<1.11,>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (1.10.7)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ydata-
```

In [ ]:
```
!pip install --upgrade scikit-learn
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.9.3)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.1.0)
```

Importing libraries

In [1]:
```
import pandas as pd
import numpy as np
import pandas_profiling
import ydata_profiling
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import KNNImputer
from featurewiz import featurewiz
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import Lasso, LogisticRegression
from sklearn.utils.class_weight import compute_class_weight
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc, precision_score, recall_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import tree
import tensorflow as tf
from tensorflow import keras
```

Loading the dataset

In [2]:
```
insurance_data = pd.read_csv("car_insurance.csv")
insurance_data
```

Out[2]:

	ID	AGE	GENDER	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE	VEHICLE_OWNERSHIP	MARRIED	CHILD
0	569520	65+	female	0-9y	high school	upper class	0.629027	1	0	
1	750365	16-25	male	0-9y	none	poverty	0.357757	0	0	
2	199901	16-25	female	0-9y	high school	working class	0.493146	1	0	
3	478866	16-25	male	0-9y	university	working class	0.206013	1	0	
4	731664	26-39	male	10-19y	none	working class	0.388366	1	0	
...	...	...	...	...	...	...	...	...	...	
9995	323164	26-39	female	10-19y	university	upper class	0.582787	1	0	
9996	910346	26-39	female	10-19y	none	middle class	0.522231	1	0	
9997	468409	26-39	male	0-9y	high school	middle class	0.470940	1	0	
9998	903459	26-39	female	10-19y	high school	poverty	0.364185	0	0	
9999	442696	26-39	female	0-9y	none	working class	0.435225	1	1	

10000 rows × 17 columns

Table 1. Car Insurance Data Analysis

Step 1: Exploratory Data Analysis

EDA is performed on the data in order to analyze various parameters and features of the dataset and to understand the *structure* of the dataset such that various *trends and patterns* between the variables is known. Exploratory Data Analysis helps in understanding the *relationship between the various independent and dependent variables* of the dataset that would further be useful in building the model such as description analysis and statistical analysis.

Descriptive Analysis

In [ ]:
```
# displaying number of rows and columns
print("Total number of Rows and Columns:", insurance_data.shape)

print("\n----------------------------------------------------------")

# displaying field values/column names
print("\nColumn Names:\n")
insurance_data.columns
```

```
Total number of Rows and Columns: (10000, 17)

----------------------------------------------------------

Column Names:
```

Out[55]:
```
Index(['ID', 'AGE', 'GENDER', 'DRIVING_EXPERIENCE', 'EDUCATION', 'INCOME',
       'CREDIT_SCORE', 'VEHICLE_OWNERSHIP', 'MARRIED', 'CHILDREN',
       'POSTAL_CODE', 'ANNUAL_MILEAGE', 'VEHICLE_TYPE', 'SPEEDING_VIOLATIONS',
       'DUIS', 'PAST_ACCIDENTS', 'OUTCOME'],
      dtype='object')
```

In [ ]:
```
# displaying data types
print("Data types:\n")
insurance_data.dtypes
```

```
Data types:
```

Out[56]:
```
ID                      int64
AGE                    object
GENDER                 object
DRIVING_EXPERIENCE     object
EDUCATION              object
INCOME                 object
CREDIT_SCORE          float64
VEHICLE_OWNERSHIP       int64
MARRIED                 int64
CHILDREN                int64
POSTAL_CODE             int64
ANNUAL_MILEAGE        float64
VEHICLE_TYPE           object
SPEEDING_VIOLATIONS     int64
DUIS                    int64
PAST_ACCIDENTS          int64
OUTCOME                 int64
dtype: object
```

Table 2. Data types for Insurance Data

From the *descriptive analysis*, it is observed that there are total **10000 rows of data** and **17 field values** and the data type for each of the field value is displayed in order to understand what data type values are present in the dataset.

Here, there are different types of data points that are present in the dataset which are **numerical data type** having either *'int' or 'float'* values, along with **object data type** which further needs to be corrected to *string or category* type of data.

Statistical Analysis

In [ ]:
```
# dataset info
print("Dataset Info:\n")
insurance_data.info()
```

```
Dataset Info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   10000 non-null  int64
 1   AGE                  10000 non-null  object
 2   GENDER               10000 non-null  object
 3   DRIVING_EXPERIENCE   10000 non-null  object
 4   EDUCATION            10000 non-null  object
 5   INCOME               10000 non-null  object
 6   CREDIT_SCORE         9018 non-null   float64
 7   VEHICLE_OWNERSHIP    10000 non-null  int64
 8   MARRIED              10000 non-null  int64
 9   CHILDREN             10000 non-null  int64
 10  POSTAL_CODE          10000 non-null  int64
 11  ANNUAL_MILEAGE       9043 non-null   float64
 12  VEHICLE_TYPE         10000 non-null  object
 13  SPEEDING_VIOLATIONS  10000 non-null  int64
 14  DUIS                 10000 non-null  int64
 15  PAST_ACCIDENTS       10000 non-null  int64
 16  OUTCOME              10000 non-null  int64
dtypes: float64(2), int64(9), object(6)
memory usage: 1.3+ MB
```

Table 3. Information about the dataset

In [ ]:
```
# describing the dataset
print("Describing the dataset:\n")
round(insurance_data.describe(),1)
```

```
Describing the dataset:
```

Out[58]:

	ID	CREDIT_SCORE	VEHICLE_OWNERSHIP	MARRIED	CHILDREN	POSTAL_CODE	ANNUAL_MILEAGE	SPEEDING_VIOLAT
count	10000.0	9018.0	10000.0	10000.0	10000.0	10000.0	9043.0	10
mean	500521.9	0.5	0.7	0.5	0.7	19864.5	11697.0	
std	290030.8	0.1	0.5	0.5	0.5	18915.6	2818.4	
min	101.0	0.1	0.0	0.0	0.0	10238.0	2000.0	
25%	249638.5	0.4	0.0	0.0	0.0	10238.0	10000.0	
50%	501777.0	0.5	1.0	0.0	1.0	10238.0	12000.0	
75%	753974.5	0.6	1.0	1.0	1.0	32765.0	14000.0	
max	999976.0	1.0	1.0	1.0	1.0	92101.0	22000.0	

Table 4. Dataset Description

Statistical Analysis helps in understanding about each of the numerical field type based on the **total count values, minimum value, maximum value, standard deviation**, etc. giving an overall analysis of the field data points about the various rows present in the dataset.

For example, as observed in the dataset, we see that there are multiple field values having the *minimum, maximum values* along with the *total count of values* which is **10000** and *standard deviation* of the column values. It can be observed that the maximum value of *Annual_Mileage* is **22000.0** and the maximum value of *Speeding_Violations* is **22.0**.

Thus, similarly, other parameters of the dataset can be analyzed based on their statistical values.

Data Profiling

In [4]:
```
insurance_data_report = insurance_data.profile_report(title='Car Insurance Data Analysis Report', explorative
insurance_data_report
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Out[4]:

In [5]:
```
# Saving the profile report
insurance_data_report.to_file(output_file="Car Insurance Data Analysis Report.html")
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

The data profiling report generated for the dataset helps in understanding various parameters such as the data type of the field values, the missing and duplicate values present in the dataset, the correlation between each of the field value, and the analysis of each of the field value on a individual basis based on correlation plot, histogram, and interaction graphs.

From the profiling report, it is observed that there are **6 numerical variable type, and 11 categorical data type** field values present in the dataset of which the numerical data type have **integer and float values**, along with string objects. Apart from this, there are **missing values** present in the dataset, and the missing values visualization or plot also helps in understanding the same, for which the percent of missing cells is **1.1% i.e., 1939 missing cells** are present in the dataset. For each field value a separate visualization is also displayed in order to specifially analyze a particular field value, and there are **no duplicate values present in the dataset.**

The data profiling report also helps in analyzing the correlation between the variables, the class imbalance for each field value, and the number of missing values or zeros present in the dataset in each column. This will help in understanding the variables in much more detail in order to train the model.

Further cleaning of the data is implemented in the below steps.

Step 2: Data Cleaning

Task 1: Clean the dataset and discuss how you cleaned each variable with missing values and why you chose that method.

1. Checking for null values in each column of the dataset, i.e., missing values
2. a. Dropping data values or fields having maximum null values

   b. Dropping irrelevant columns
3. Checking for incorrect data types in field values and correcting the data type of the column
4. Checking for outliers in the dataset

   a. *Boxplot*

   b. *Distribution Plot*

1. Checking for null values in each column of the dataset, i.e., missing values

In [ ]:
```
for x in range(17):
    print("%-45s %10d" % (insurance_data.columns.values[x], insurance_data.iloc[:,x].isna().sum()))
```

```
ID                                            0
AGE                                           0
GENDER                                        0
DRIVING_EXPERIENCE                            0
EDUCATION                                     0
INCOME                                        0
CREDIT_SCORE                                982
VEHICLE_OWNERSHIP                             0
MARRIED                                       0
CHILDREN                                      0
POSTAL_CODE                                   0
ANNUAL_MILEAGE                              957
VEHICLE_TYPE                                  0
SPEEDING_VIOLATIONS                           0
DUIS                                          0
PAST_ACCIDENTS                                0
OUTCOME                                       0
```

Table 5. Missing or null values

As observed from the table above, there are missing values present in the **Credit_Score** and **Annual_Mileage** columns that need to be addressed by either imputation methods or dropping the rows of data if there are less than 20% of missing values. Hence, the data is analyzed in order to understand which imputation method needs to be implemented.

2.a. Dropping data values or fields having maximum null values

In [ ]:
```
# displaying unique data

print("Displaying the unique data present in columns\n")
insurance_data.nunique()
```

```
Displaying the unique data present in columns
```

Out[15]:
```
ID                    10000
AGE                       4
GENDER                    2
DRIVING_EXPERIENCE        4
EDUCATION                 3
INCOME                    4
CREDIT_SCORE           9018
VEHICLE_OWNERSHIP         2
MARRIED                   2
CHILDREN                  2
POSTAL_CODE               4
ANNUAL_MILEAGE           21
VEHICLE_TYPE              2
SPEEDING_VIOLATIONS      21
DUIS                      7
PAST_ACCIDENTS           15
OUTCOME                   2
dtype: int64
```

Table 6. Unique Data Count

In [ ]:
```
# checking for data distribution of field values 'Credit_Score' as it has missing values


# distribution plot for Credit_Score column

plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(insurance_data['CREDIT_SCORE'])
plt.title("Distribution Plot for Credit Score")
plt.show()
```

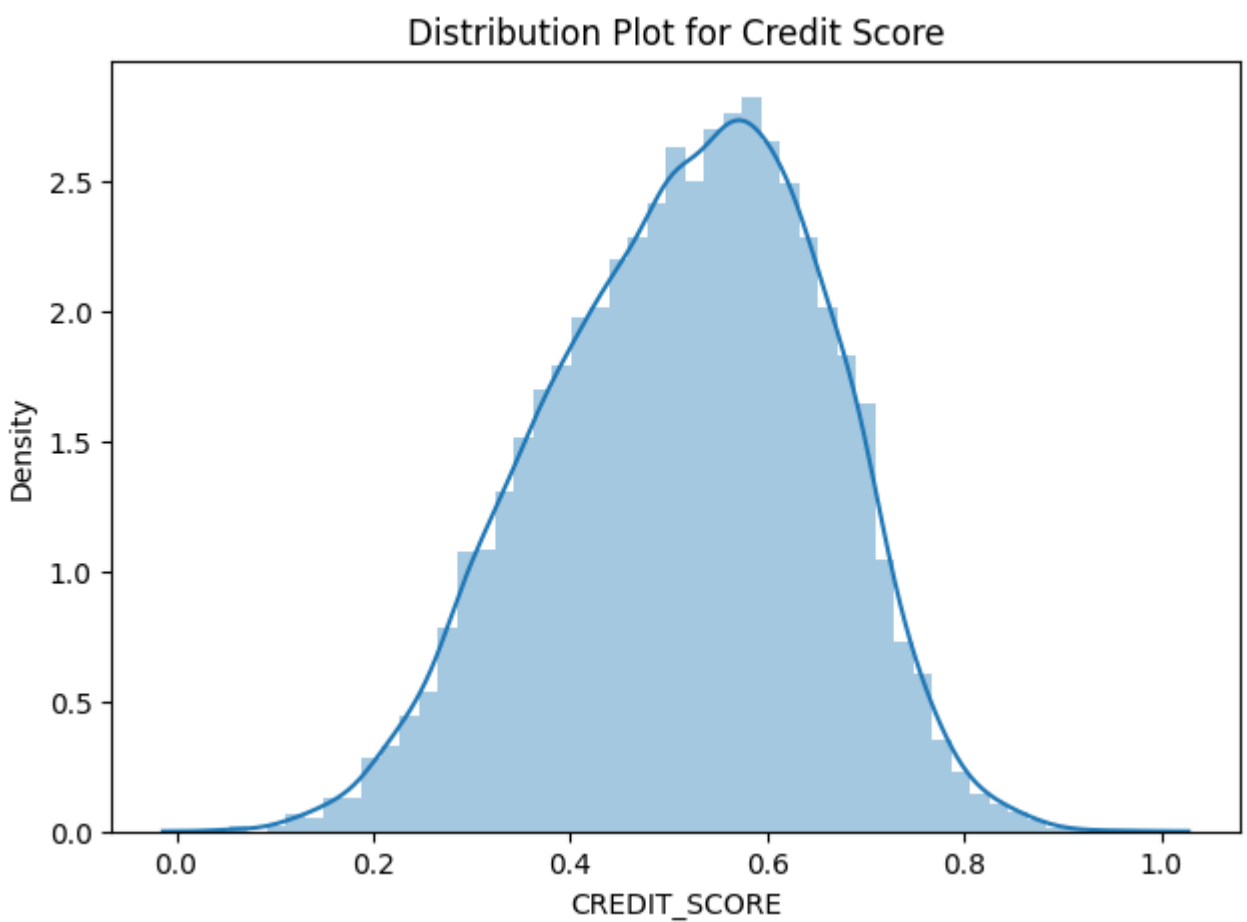

Figure 1. Distribution Plot for Credit Score variable

The **distribution plot for the Credit Score variable** above shows that the data points is **normally distributed** and hence if applying a mean-median imputation, the mean method needs to be applied to replace the missing values present in the dataset.

In [ ]:
```
# checking for data distribution of field values 'Annual_Mileage' as it has missing values


# distribution plot for Annual_Mileage column

plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(insurance_data['ANNUAL_MILEAGE'])
plt.title("Distribution Plot for Annual Mileage")
plt.show()
```

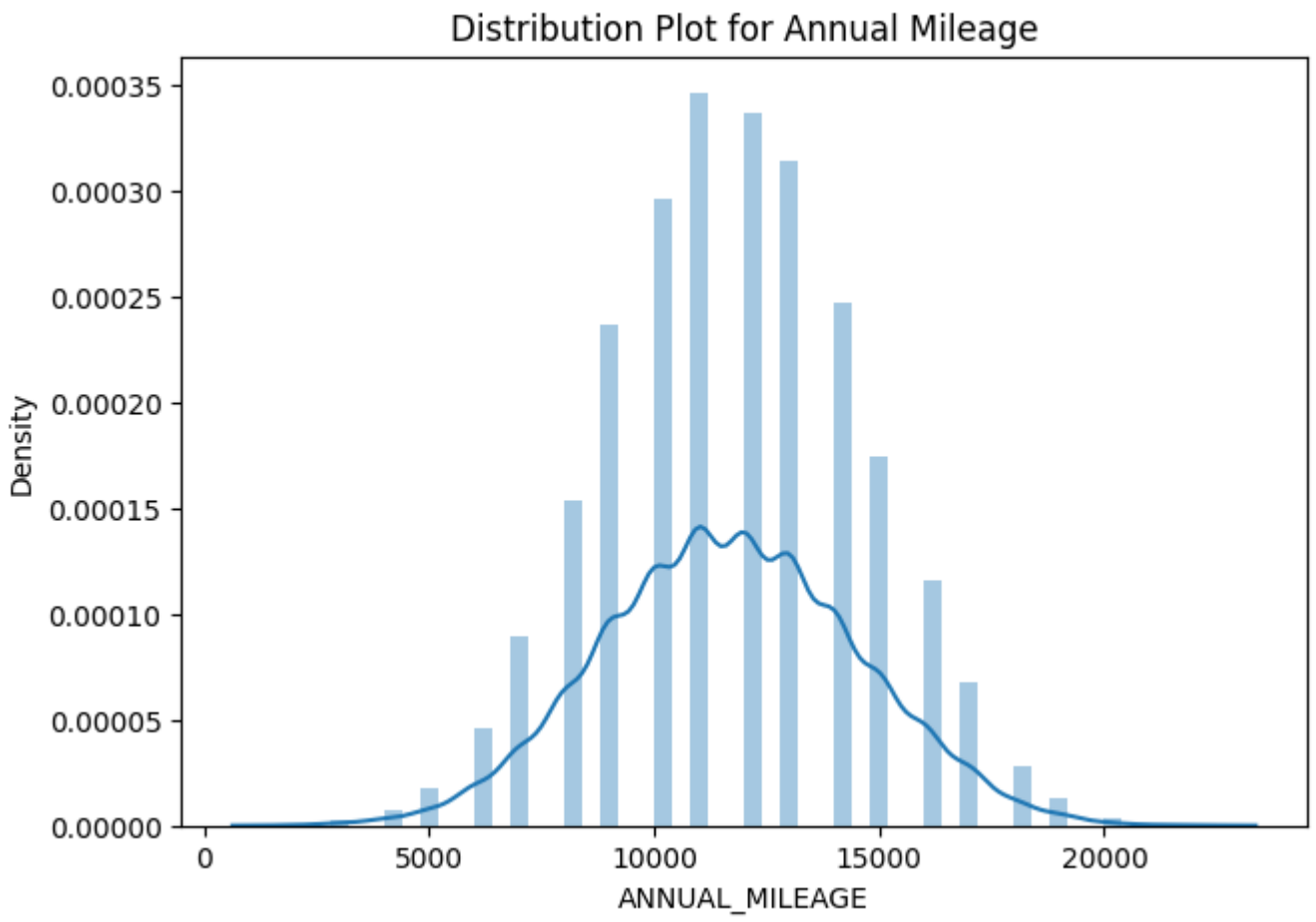

Figure 2. Distribution Plot for Annual Mileage variable

Dropping the columns having maximum null values

In [3]:
```
# dropping columns having maximum null values

insurance_data = insurance_data.drop(['CREDIT_SCORE', 'ANNUAL_MILEAGE'], axis=1)
print("Dropped columns having maximum null values.")
```

```
Dropped columns having maximum null values.
```

Dropping **'Credit Score'** and **'Annual Mileage'** field value from the dataset as it has maximum null values present in the dataset of the total data points which can be considered irrelevant for the training of the model and for which imputation methods will not be effective to implement.

In [7]:
```
# checking for any missing values after data cleaning & imputations
insurance_data.shape
```

Out[7]:
```
(10000, 15)
```

In [8]:
```
for x in range(15):
    print("%-45s %10d" % (insurance_data.columns.values[x], insurance_data.iloc[:,x].isna().sum()))
```

```
ID                                                0
AGE                                               0
GENDER                                            0
DRIVING_EXPERIENCE                                0
EDUCATION                                         0
INCOME                                            0
VEHICLE_OWNERSHIP                                 0
MARRIED                                           0
CHILDREN                                          0
POSTAL_CODE                                       0
VEHICLE_TYPE                                      0
SPEEDING_VIOLATIONS                               0
DUIS                                              0
PAST_ACCIDENTS                                    0
OUTCOME                                           0
```

Table 7. Missing Values Count

2. b. Dropping irrelevant columns

In [4]:
```
insurance_data = insurance_data.drop(['ID'], axis=1)
print("Column Dropped.")
```

```
Column Dropped.
```

In [10]:
```
insurance_data.shape
```

Out[10]:
```
(10000, 14)
```

3. Checking for incorrect data types in field values and correcting the data type of the column

In [5]:
```
# correcting the data types for the variables of the dataset which are of object type to string/category

insurance_data['AGE'] = insurance_data['AGE'].astype('category')
insurance_data['GENDER'] = insurance_data['GENDER'].astype('category')
insurance_data['DRIVING_EXPERIENCE'] = insurance_data['DRIVING_EXPERIENCE'].astype('category')
insurance_data['EDUCATION'] = insurance_data['EDUCATION'].astype('category')
insurance_data['INCOME'] = insurance_data['INCOME'].astype('category')
insurance_data['VEHICLE_TYPE'] = insurance_data['VEHICLE_TYPE'].astype('category')

print("Data Type conversion successful.")
```

```
Data Type conversion successful.
```

In [ ]:
```
# checking for the correct data type of the variable

insurance_data.dtypes
```

Out[9]:
```
AGE                   category
GENDER                category
DRIVING_EXPERIENCE    category
EDUCATION             category
INCOME                category
VEHICLE_OWNERSHIP        int64
MARRIED                  int64
CHILDREN                 int64
POSTAL_CODE              int64
VEHICLE_TYPE          category
SPEEDING_VIOLATIONS      int64
DUIS                     int64
PAST_ACCIDENTS           int64
OUTCOME                  int64
dtype: object
```

Table 8. Data type of Insurance Data

In [ ]:
```
# checking for data values after data type conversion

insurance_data
```

Out[66]:

	AGE	GENDER	DRIVING_EXPERIENCE	EDUCATION	INCOME	VEHICLE_OWNERSHIP	MARRIED	CHILDREN	POSTAL_CODE	VE
0	65+	female	0-9y	high school	upper class	1	0	1	10238	
1	16-25	male	0-9y	none	poverty	0	0	0	10238	
2	16-25	female	0-9y	high school	working class	1	0	0	10238	
3	16-25	male	0-9y	university	working class	1	0	1	32765	
4	26-39	male	10-19y	none	working class	1	0	0	32765	
...	...	...	...	...	...	...	...	...	...	
9995	26-39	female	10-19y	university	upper class	1	0	0	10238	
9996	26-39	female	10-19y	none	middle class	1	0	1	32765	
9997	26-39	male	0-9y	high school	middle class	1	0	1	10238	
9998	26-39	female	10-19y	high school	poverty	0	0	1	10238	
9999	26-39	female	0-9y	none	working class	1	1	1	10238	

10000 rows × 14 columns

Table 9. Insurance Data after data type conversion

4. Checking for outliers in the dataset

a. Boxplot

The below code creates **boxplots** for the various field values of the marketing dataset in order to check for outliers present in the dataset. Here, the boxplots are implemented for the variables **VEHICLE_OWNERSHIP**, **MARRIED**, **CHILDREN**, and **SPEEDING_VIOLATIONS** as shown in the figures below.

There are outliers present in the variables as observed in the boxplot below, which will not be removed as each of the data point is important for analysis and model building.

In [ ]:
```
# creating boxplot for 'VEHICLE_OWNERSHIP' and 'MARRIED' variable

fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].boxplot(insurance_data['VEHICLE_OWNERSHIP'])
axs[1].boxplot(insurance_data['MARRIED'])
axs[0].set_title('Boxplot for column, VEHICLE_OWNERSHIP')
axs[1].set_title('Boxplot for column, MARRIED')
axs[0].set_ylabel('Data Values')
axs[1].set_ylabel('Data Values')

plt.show()
```

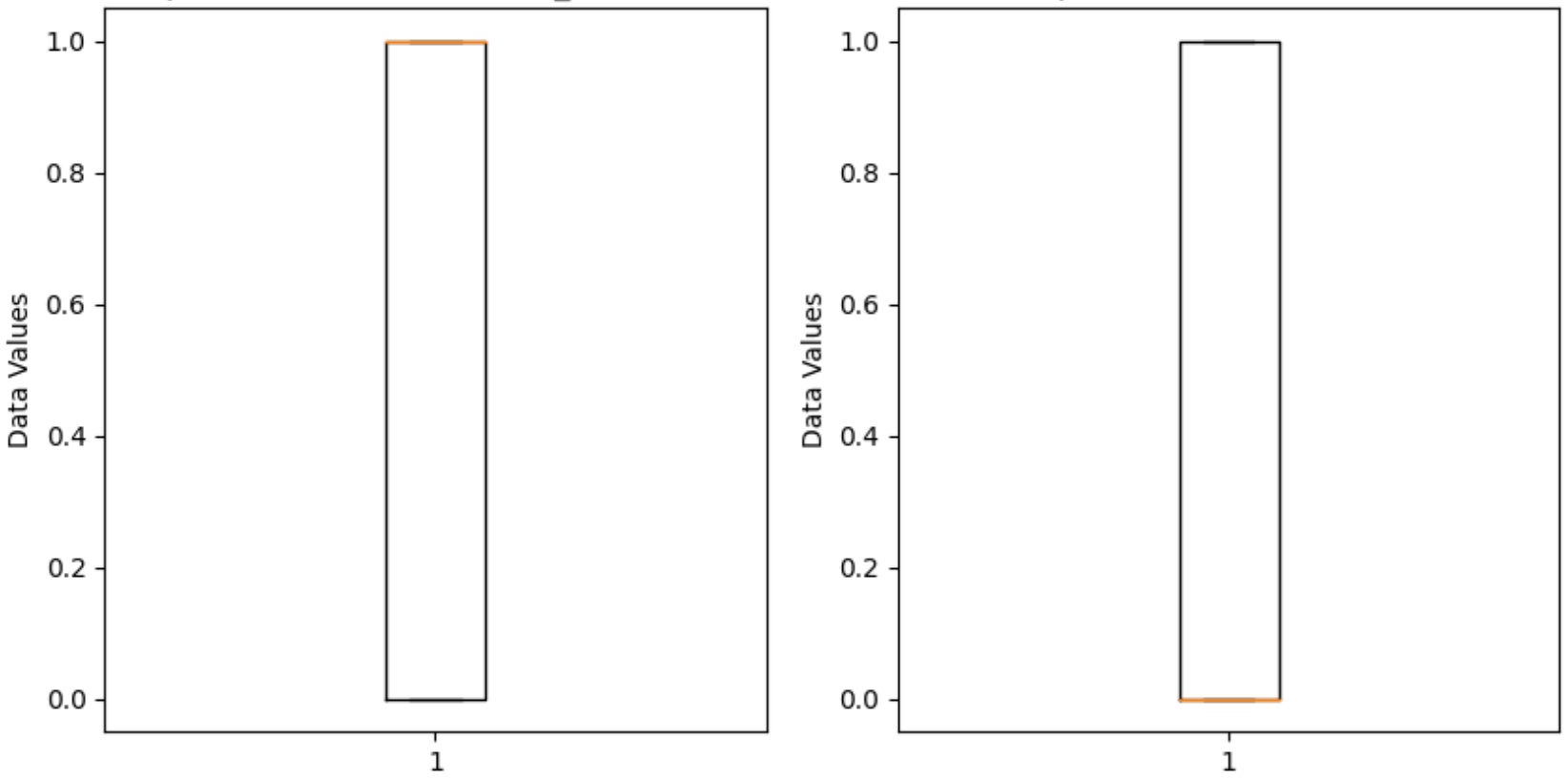

Figure 3. Boxplot for Vehicle Ownership & Married column

In [ ]:
```
# creating boxplot for 'CHILDREN' and 'SPEEDING_VIOLATIONS' variable

fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].boxplot(insurance_data['CHILDREN'])
axs[1].boxplot(insurance_data['SPEEDING_VIOLATIONS'])
axs[0].set_title('Boxplot for column, CHILDREN')
axs[1].set_title('Boxplot for column, SPEEDING_VIOLATIONS')
axs[0].set_ylabel('Data Values')
axs[1].set_ylabel('Data Values')

plt.show()
```

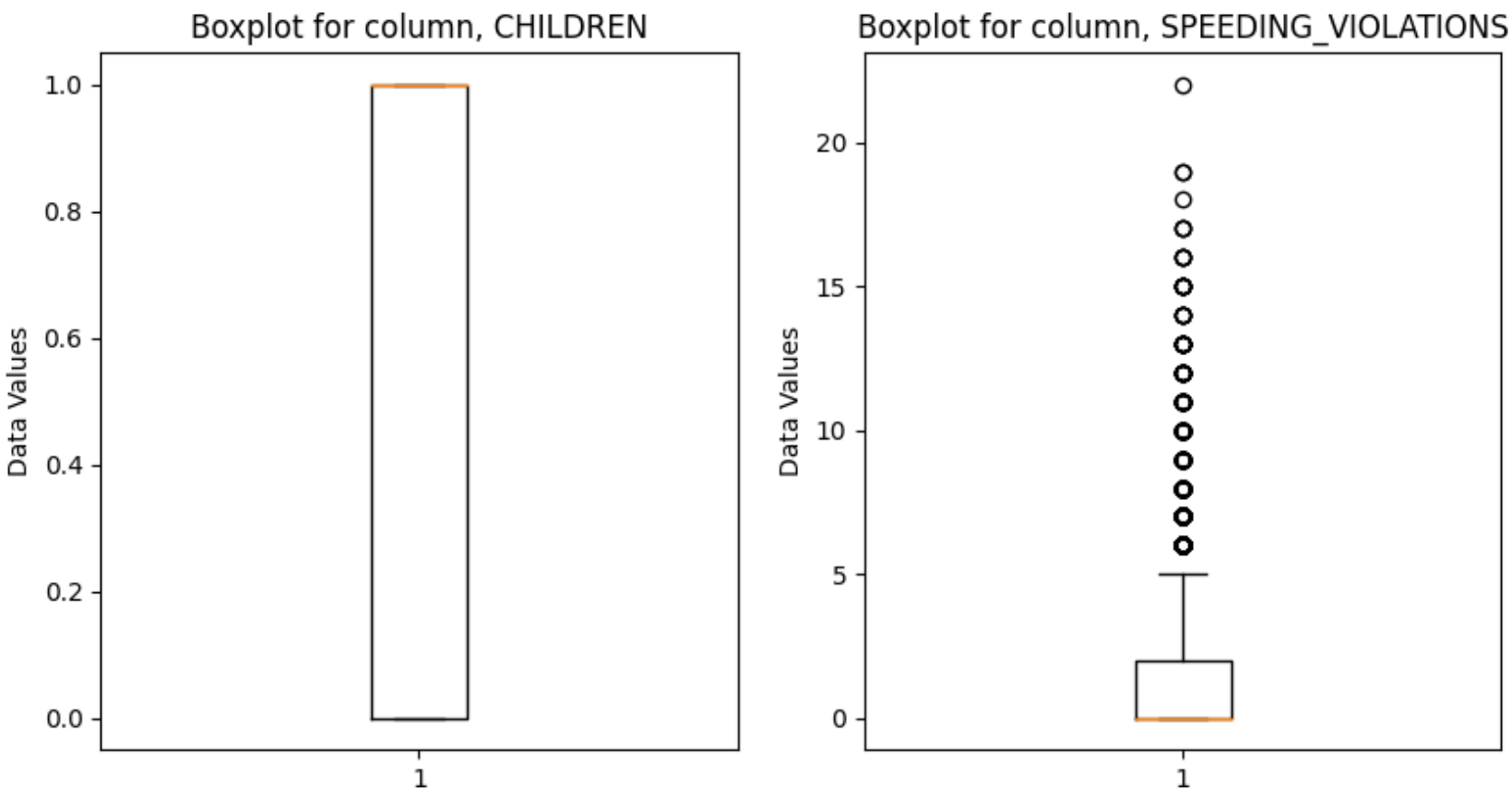

Figure 4. Boxplot for Children & Speeding Violations column

b. Distribution Plot

The distribution plot for the various parameters of the dataset values gives an overview of the outliers that are present and the distribution of the data points across present in the dataset.

The plot below for *Childer* shows that the data is **normally distributed** across the data points, meaning that the data points are evenly distributed around the mean value and the plot for the *Married* variable also indicates that the data is **normally distributed**.

In [ ]:
```
# distribution plot for Children & Married

plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(insurance_data['CHILDREN'])
plt.subplot(1,2,2)
sns.distplot(insurance_data['MARRIED'])
plt.show()
```

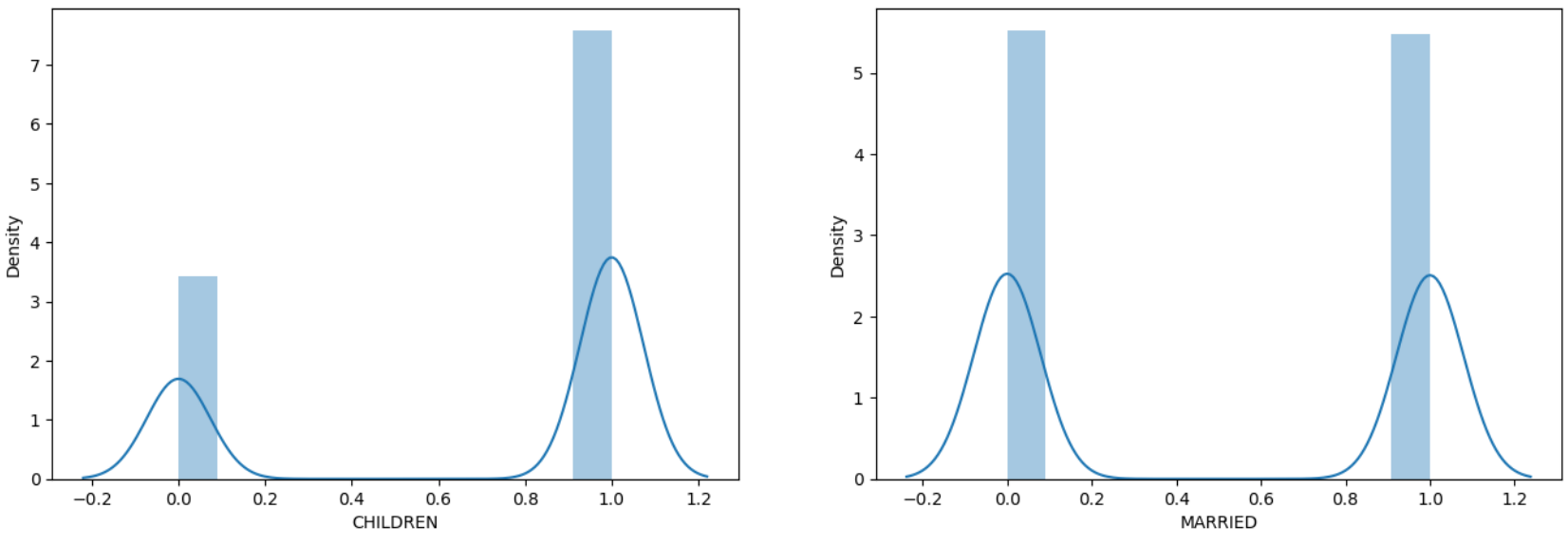

Figure 5. Distribution Plot for Children & Married

In [ ]:
```
# distribution plot for Speeding Violations & Past Accidents

plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(insurance_data['SPEEDING_VIOLATIONS'])
plt.subplot(1,2,2)
sns.distplot(insurance_data['PAST_ACCIDENTS'])
plt.show()
```

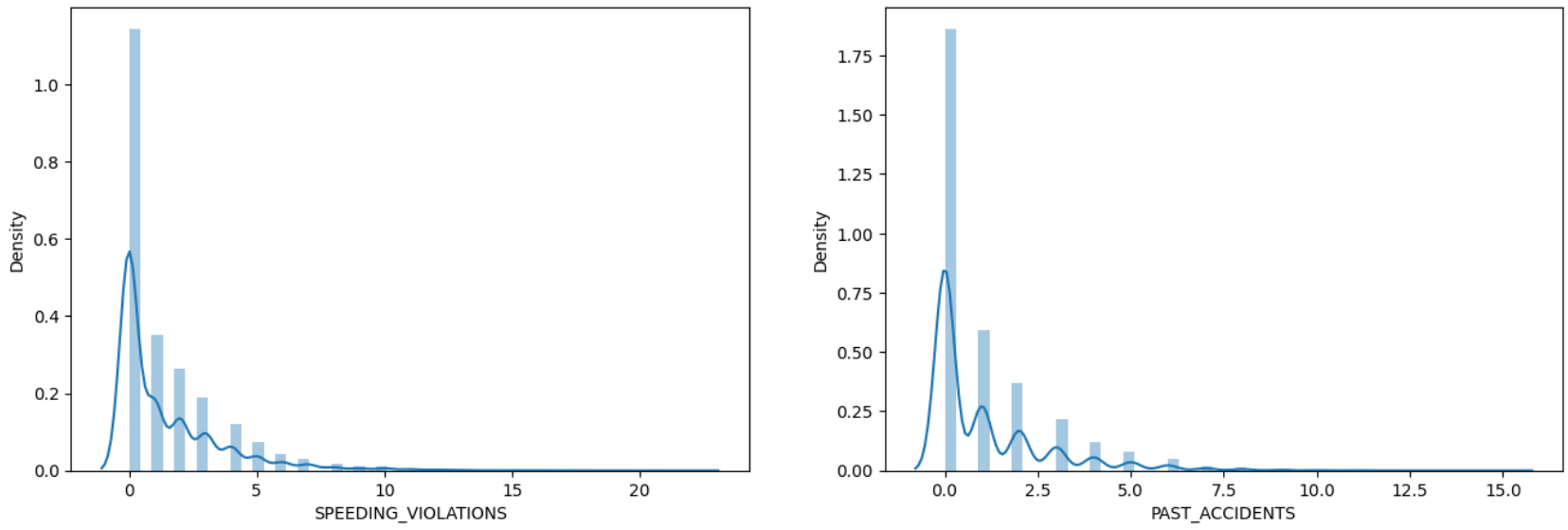

Figure 6. Distribution Plot for Speeding Violations & Past Accidents

Here, the data points for *Speeding Violations* and *Past Accidents* are right skewed and the data is concentrated towards a range of values and not normally distributed across the mean values.

Step 3: Data Visualization

In [ ]:
```
# 1. Outcome Count Analysis

plt.figure(figsize=(6,5))
sns.countplot(x='OUTCOME', data=insurance_data, palette="pastel")
plt.title('\nOutcome Count Analysis')
plt.show()
```

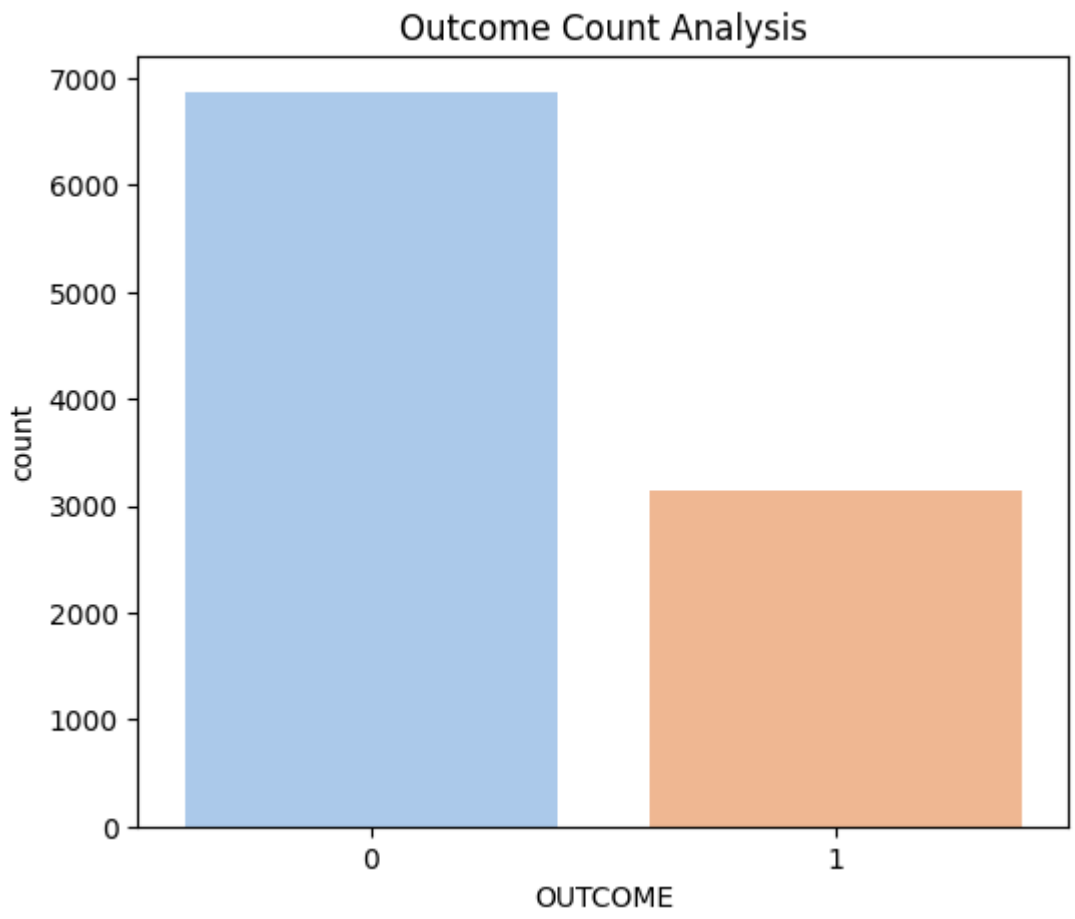

Figure 7. Outcome Count Analysis

From the above graph of *Outcome Count Analysis*, it is observed that the data is **biased towards one class** of no outcome, and hence there is a possibility that the model will be biased to this class which will classify majority of the data points into no outcome category.

In [ ]:
```
# 2. Count Analysis of Number of Children and Married

fig, axs = plt.subplots(1, 2, figsize=(12, 5))
sns.countplot(x='CHILDREN', data=insurance_data, palette="pastel", ax=axs[0])
axs[0].set_title('\nAnalysis of Number of Children')
sns.countplot(x='MARRIED', data=insurance_data, palette="pastel", ax=axs[1])
axs[1].set_title('\nAnalysis of Number of Married')
plt.show()
```

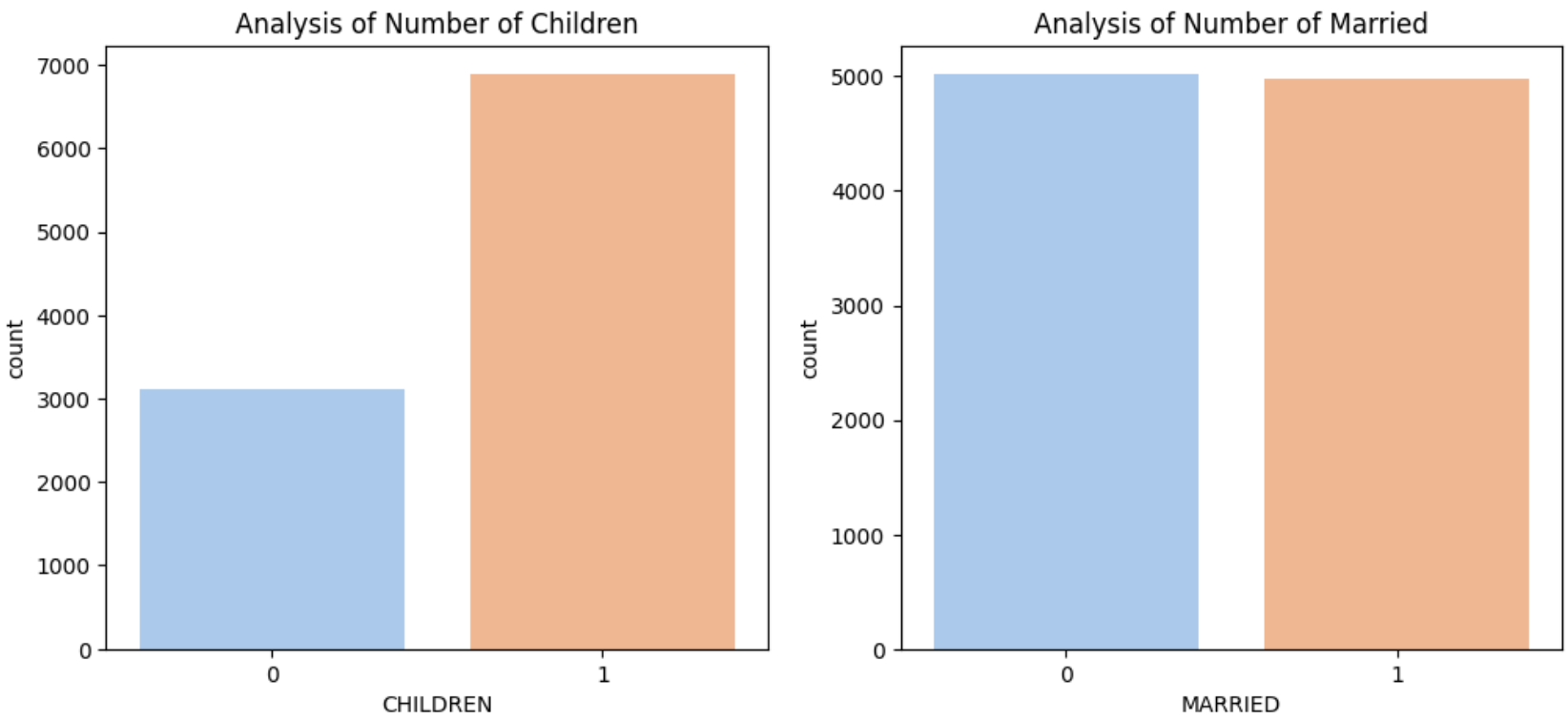

Figure 8. Analysis of Number of Children & Number of Married

The above visualization plots for the **Children graph** and **Married graph** helps in analyzing the count of each variable and as observed the count of children value being 1 is high as compared to value being 0. On the other hand, the count analysis of Married show that the class is not biased and has equal number of married and not married people.

In [ ]:
```
# 3. Count Analysis of Number of Past_Accidents and Speeding_Violations

fig, axs = plt.subplots(1, 2, figsize=(12, 5))
sns.countplot(x='PAST_ACCIDENTS', data=insurance_data, palette="pastel", ax=axs[0])
axs[0].set_title('\nAnalysis of Number of Past Accidents')
sns.countplot(x='SPEEDING_VIOLATIONS', data=insurance_data, palette="pastel", ax=axs[1])
axs[1].set_title('\nAnalysis of Number of Speeding Violations')
plt.show()
```

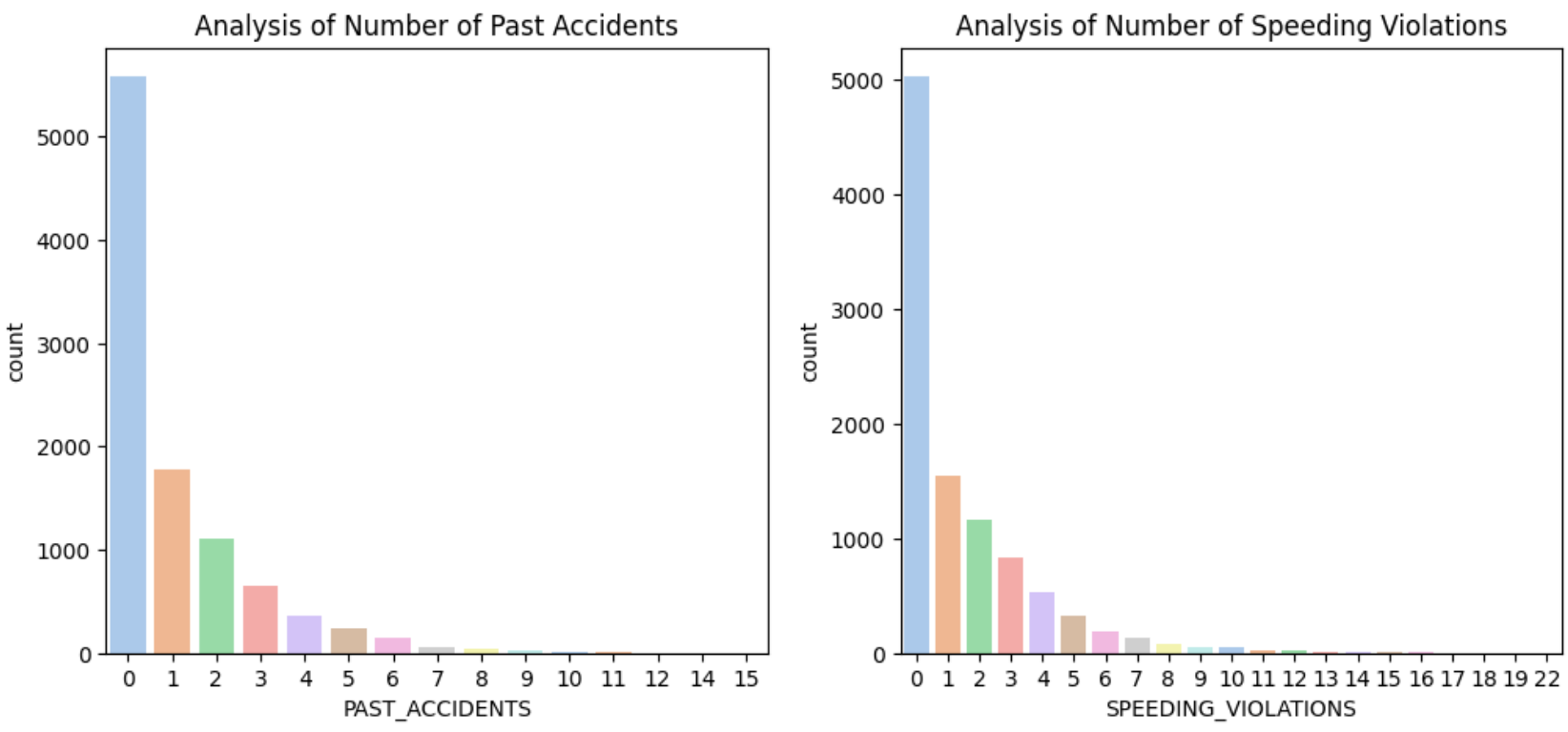

Figure 9. Analysis of Number of Past Accidents and Speeding Violations

The above visualization plots for the **Past Accidents graph** and **Speeding Violations graph** helps in analyzing the count of past accidents and speeding violations where the various count values of the percentage are displayed that can show which percent value has a higher count. This helps in understanding the count of number of past accidents and speeding violations of a person.

Step 4: Pre-Modeling Steps

1. Label Encoding
2. Correlation Plot
3. Defining the features for model training
4. Spliting the dataset into train & test set
5. Class Bias
6. Standardization

1. Label Encoding

Since some of the selected features for prediction are categorical data and most of the ML models require the data to be numerical or binary, it is important that these features are converted into **binary** or **numerical** type data in order for the model to be able to classify the classes.

Label Encoding is a method which helps to **convert the categorical variables** into **numerical values**, thus helping to transform the data point into a format where the algorithm is able to process the data for classification. *LabelEncoder()* function is used to encode the categorical type data to numerical type, where new columns of data are created for the categorical field value in the dataset which will be used in the training of the model.

In [6]:
```
labelencoder = LabelEncoder()

insurance_data['AGE_LABEL'] = labelencoder.fit_transform(insurance_data["AGE"])
insurance_data['GENDER_LABEL'] = labelencoder.fit_transform(insurance_data["GENDER"])
insurance_data['DRIVING_EXPERIENCE_LABEL'] = labelencoder.fit_transform(insurance_data["DRIVING_EXPERIENCE"])
insurance_data['EDUCATION_LABEL'] = labelencoder.fit_transform(insurance_data["EDUCATION"])
insurance_data['INCOME_LABEL'] = labelencoder.fit_transform(insurance_data["INCOME"])
insurance_data['VEHICLE_TYPE_LABEL'] = labelencoder.fit_transform(insurance_data["VEHICLE_TYPE"])
print("Label Encoding successful.")
```

```
Label Encoding successful.
```

In [ ]:
```
# checking for data values and field names after Label Encoding

insurance_data
```

Out[68]:

	AGE	GENDER	DRIVING_EXPERIENCE	EDUCATION	INCOME	VEHICLE_OWNERSHIP	MARRIED	CHILDREN	POSTAL_CODE	VE
0	65+	female	0-9y	high school	upper class	1	0	1	10238	
1	16-25	male	0-9y	none	poverty	0	0	0	10238	
2	16-25	female	0-9y	high school	working class	1	0	0	10238	
3	16-25	male	0-9y	university	working class	1	0	1	32765	
4	26-39	male	10-19y	none	working class	1	0	0	32765	
...	...	...	...	...	...	...	...	...	...	
9995	26-39	female	10-19y	university	upper class	1	0	0	10238	
9996	26-39	female	10-19y	none	middle class	1	0	1	32765	
9997	26-39	male	0-9y	high school	middle class	1	0	1	10238	
9998	26-39	female	10-19y	high school	poverty	0	0	1	10238	
9999	26-39	female	0-9y	none	working class	1	1	1	10238	

10000 rows × 20 columns

Table 10. Insurance Data after Label Encoding

In [ ]:
```
insurance_data.columns
```

Out[69]:
```
Index(['AGE', 'GENDER', 'DRIVING_EXPERIENCE', 'EDUCATION', 'INCOME',
       'VEHICLE_OWNERSHIP', 'MARRIED', 'CHILDREN', 'POSTAL_CODE',
       'VEHICLE_TYPE', 'SPEEDING_VIOLATIONS', 'DUIS', 'PAST_ACCIDENTS',
       'OUTCOME', 'AGE_LABEL', 'GENDER_LABEL', 'DRIVING_EXPERIENCE_LABEL',
       'EDUCATION_LABEL', 'INCOME_LABEL', 'VEHICLE_TYPE_LABEL'],
      dtype='object')
```

2. Corelation Plot

A **correlation plot** or matrix is a *visual representation of the variables* present in the dataset which helps in understanding the *relationship* between the different variables and how highly the variables are corelated to each other.

The values of the correlation plot range from **-1 to 1**, where -1 indicates a **negative correlation** between the variables, 0 indicates **no correlation**, and 1 indicates a **positive correlation**.

The variables that have positive correlation are said to be highly correlated to each and hence either of the two variables must be removed for the model building as it may lead to **multicollinearity** where the efficiency of the model may reduce.

In [ ]:
```
# plotting correlation matrix

plt.figure(figsize = (10,7))
ax = plt.subplot()
sns.heatmap(insurance_data.corr(),annot=True, fmt='.1f', ax=ax, cmap="Blues")
ax.set_title('Correlation Plot');
```

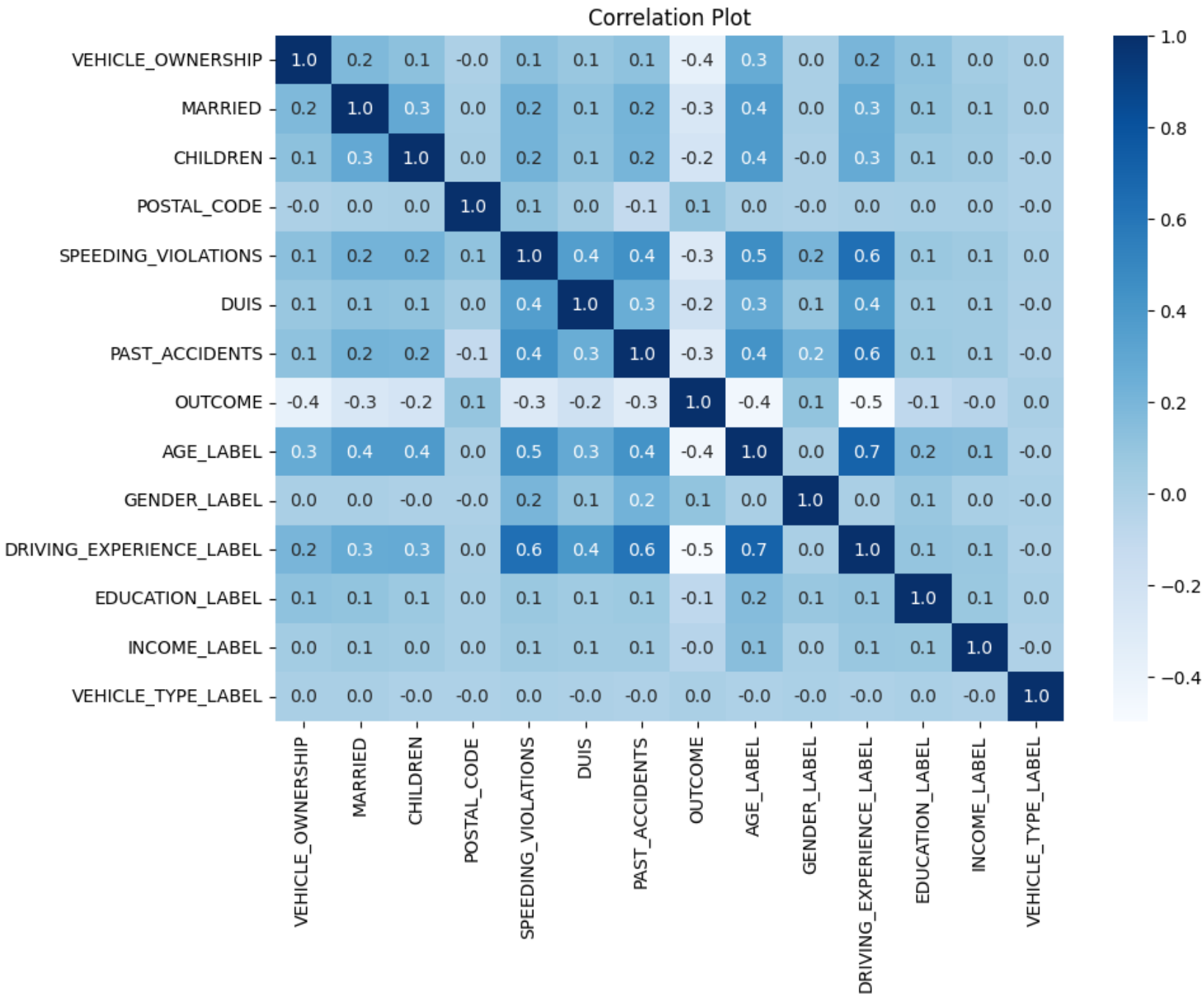

Figure 10. Correlation Matrix

Looking for strong corelations between independent & dependent variables

As observed in the correlation matrix above, we see that there are many variables or features that are *highly correlated* to each other and hence we need to analyze the features that are strongly correlated such that these features are excluded from the training of the model in order to avoid **multicollinearity** and *improve the efficiency of the model*. The following features are strongly correlated with the other features in the dataset and can be excluded from model building to avoid multicollinearity.

Correlation among the variables:

1. Variable '**Married**' is correlated with '**Age**' with a correlation value of 0.4
2. Variable '**Children**" is correlated with '**Age**' with a correlation value of 0.4
3. '**Speeding Violations**' is highly correlated with '**Age**' and '**Driving Experience**'
4. '**Age**' is highly correlated with **Driving Experience** with correlation value of 0.7

Checking VIF score for Multicollinearity

VIF score i.e., **Variance Inflation Factor** is a *measure of multicollinearity* between the independent variables in the regression analysis. This calculates the variance of the variables which helps in understanding the coefficient value and how much the variable is inflated due to collinearity in the model. The VIF score from **range 0 to 5** can be accepted to be considered for the training of the model, while values above 5 are considered to have high multicollinearity which would affect the accuracy and performance of the model, hence should be excluded.

In [ ]:
```
# checking VIF score for field values

numerical_insurance_data = insurance_data.select_dtypes(include=[np.number])
vif_score1 = pd.DataFrame()
vif_score1["Feature"] = numerical_insurance_data.columns
vif_score1["VIF Score"] = [variance_inflation_factor(numerical_insurance_data.values, i) for i in range(numer
print(vif_score1)
```

```
                    Feature  VIF Score
0          VEHICLE_OWNERSHIP   3.248795
1                    MARRIED   2.431381
2                   CHILDREN   3.579133
3                POSTAL_CODE   2.134626
4         SPEEDING_VIOLATIONS   2.657020
5                       DUIS   1.448913
6             PAST_ACCIDENTS   2.487127
7                    OUTCOME   1.699445
8                  AGE_LABEL   7.429417
9               GENDER_LABEL   2.261966
10  DRIVING_EXPERIENCE_LABEL   7.404375
11           EDUCATION_LABEL   2.215011
12              INCOME_LABEL   3.076806
13        VEHICLE_TYPE_LABEL   1.046299
```

Table 11. VIF Score Table

From the above VIF score table it is observed that the VIF score **is in the range from 0 to 5**, except for the *Age & Driving Experience variable*, with a VIF score of 7.4, which will be excluded for model training, and thus there is **no issue of multicollinearity** and that the remaining variables are not correlated to each other. The features of the dataset thus can be used for training the model except for Age and Driving Experience variable avoiding the issue of multicollinearity where the performance of the model can be improved.

Since, **Age and Driving_Experience** have a high VIF score and are also highly correlated with other independent variables of the dataset, the two features are excluded from the training of the model in order to avoid multicollinearity.

a. Understanding the top features selected by Correlation Matrix

In [ ]:
```
corr_result = insurance_data.corr()
correlation_response = corr_result['OUTCOME'].sort_values(ascending=False)
topfeatures = correlation_response[1:6]
print("The top features selected by correlation matrix are:\n")
print(topfeatures)
```

```
The top features selected by correlation matrix are:

GENDER_LABEL          0.107208
POSTAL_CODE           0.095889
VEHICLE_TYPE_LABEL    0.005620
INCOME_LABEL         -0.047560
EDUCATION_LABEL      -0.092643
Name: OUTCOME, dtype: float64
```

b. Lasso Regression to select the most important features for model training

In [ ]:
```
# creating a new dataframe excluding categorical variable

new_insurancedata_lasso = pd.DataFrame()
new_insurancedata_lasso = insurance_data.drop(columns=['AGE', 'GENDER','DRIVING_EXPERIENCE','EDUCATION', 'INC
print("Dataframe created.")
```

```
Dataframe created.
```

In [ ]:
```
A = new_insurancedata_lasso.drop(['OUTCOME'], axis=1)
B = new_insurancedata_lasso['OUTCOME']
lasso_result = Lasso(alpha=0.1)
lasso_result.fit(A, B)
coef = pd.Series(lasso_result.coef_, index=A.columns)
features_lasso = coef.abs().sort_values(ascending=False).head(5).index
print("The top features selected by Lasso regression:\n")
print(features_lasso)
```

```
The top features selected by Lasso regression:

Index(['DRIVING_EXPERIENCE_LABEL', 'AGE_LABEL', 'SPEEDING_VIOLATIONS',
       'POSTAL_CODE', 'VEHICLE_OWNERSHIP'],
      dtype='object')
```

c. Features selected for model building

The features that are selected for the model building based on the Correlation Plot values, Lasso Regression, and VIF Score are as follows:

Correlation Plot Values	Lasso Regression	Features with VIF Scores in range 0 to 5
Gender	Driving_Experience	Vehicle_Ownership
Postal_Code	Age	Married
Vehicle_Type	Speeding_Violations	Children
Income	Postal_Code	Postal_Code
Education	Vehicle_Ownership	Speeding_Violations
		Duis
		Past_Accidents
		Gender
		Education
		Income
		Vehicle_Type

Table 12. Feature Selection & Extraction Table

3. Defining the features for model training (Dimensionality Reduction)

The model is trained & built on the features that are selected from the analysis of the Feature selection and extraction, Correlation matrix, Lasso Regression, and VIF score for multicollinearity.

In [7]:
```
X = insurance_data.drop(columns=['AGE', 'GENDER','DRIVING_EXPERIENCE','EDUCATION', 'INCOME', 'VEHICLE_TYPE',
y = insurance_data['OUTCOME']
```

4. Splitting the dataset into train & test set

The dataset is split into training and testing data with a random split of **80%** train set and **20%** for test data.

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 42)
```

5. Handling imbalance data (Class Bias)

Class Bias in a dataset occurs when **distribution of data points in a dataset is uneven**, with one or more classes being overrepresented or underrepresented, which means that either of the category has majority of the data points and thus while training of the model, the prediction will be biased to that category. Hence, it is important to **handle class imbalance** in the dataset which can be performed using various methods, and one such method that is implemented below is the **class weights**.

In [9]:
```
class_weights = compute_class_weight(
                                        class_weight = "balanced",
                                        classes = np.unique(y_train),
                                        y = y_train
                                    )
class_weights = dict(zip(np.unique(y_train), class_weights))
print(class_weights)
```

```
{0: 0.7272727272727273, 1: 1.6}
```

6. Standardization

Standardization is performed on the split dataset in order to make the features selected comparable to a standardized scale.

In [10]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print("Standardization successful")
```

```
Standardization successful
```

Label Count in train & test dataset

In [17]:
```
print('Labels count in y:', np.bincount(y))
print('Labels count in y_train:', np.bincount(y_train))
print('Labels count in y_test:', np.bincount(y_test))
```

```
Labels count in y: [6867 3133]
Labels count in y_train: [5500 2500]
Labels count in y_test: [1367  633]
```

Step 5: Model Building - Logistic Regression Model

Task 2: Build a logistic regression model and discuss the significant variables. Provide a table of all significant variables and their coefficients (a snippet of the data is not acceptable and if there are no variables at .05 or under, feel free to expand to .1). From your initial thoughts, which variable sticks out to you as intriguing that it is significant and why. How could this information be useful to the insurer?

Building a Logistic Regression model to predict the Outcome variable.

Fitting the Logistic Regression model

The LR model is fit with a balanced class weight method in order to avoid class imbalance or bias data present in the training dataset.

In [11]:
```
logisticreg_model = LogisticRegression(solver='newton-cg', class_weight='balanced')
logisticreg_model.fit(X_train_scaled, y_train)
```

Out[11]:
```
LogisticRegression(class_weight='balanced', solver='newton-cg')
```

Displaying the coefficients & intercepts after fitting the model

As observed from the below code, the coefficient values of the variables are displayed which are either positive or negative, which indicates that the variables with **positive** value have a **positive relationship with the target variable** whereas values having a **negative sign** indicate that there is a **negative relationship between the independent variable and the target variable.**

In [ ]:
```
coefficient_values = pd.DataFrame({'Feature': X.columns, 'Coefficient': logisticreg_model.coef_[0]})
print('Coefficients:')
print(coefficient_values)
```

```
Coefficients:
                Feature  Coefficient
0     VEHICLE_OWNERSHIP    -0.790778
1               MARRIED    -0.319276
2              CHILDREN    -0.191781
3           POSTAL_CODE     0.308743
4   SPEEDING_VIOLATIONS    -0.654364
5                  DUIS    -0.174767
6        PAST_ACCIDENTS    -1.006465
7          GENDER_LABEL     0.590882
8       EDUCATION_LABEL    -0.091516
9          INCOME_LABEL    -0.025174
10   VEHICLE_TYPE_LABEL     0.009815
```

Table 13. Coefficients Table

Summary Report of the Logistic Regression model

Summary report of the Logistic Regression model provides an overview of the model build and how accurately the model fits the data for each independent variable to predict or classify the target variable or dependent variable. The report is used to evaluate the overall fit of the model, identify which independent variables are most important in predicting the dependent variable, and analyze the statistical significance of each coefficients.

- From the summary report for the Logistic Regression model below, it is observed that the p-value for two features of the dataset is **greater that the significance value of 0.05**, hence the variable or feature is considered **statistically insignificant**. The variables that are statistically insignificant are **Education** and **Vehicle Type**. Thus, this indicates that the variable is not contributing in the classification of the response variable.
- The remaining features have **p-value less than the significance value of 0.05**, hence are consider as **statistically significant variables**, indicating that these features are contributing in the prediction of the target variable.
- Of the statistically significant variables, features having highest coefficient values are **Gender, Income, and Vehicle Type**, indicating that they have the **highest positive influence on the target variable**, whereas **Vehicle Ownership and Past Accidents** have the **highest negative impact on the target variable**, which means that if there are Past Accidents that have occured, changes of the outcome variable to be positive are less.
- **Hence, the variables that are significant which have an impact on the business are Gender, Income, and Vehicle Type, and the variables that have the highest negative impact are Vehicle Ownership and Past Accidents.**

In [ ]:
```
model_summary = sm.Logit(endog=y, exog=X)
summary_result = model_summary.fit()
print(summary_result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.437113
         Iterations 7
                          Logit Regression Results
==============================================================================
Dep. Variable:                OUTCOME   No. Observations:                10000
Model:                          Logit   Df Residuals:                     9989
Method:                           MLE   Df Model:                           10
Date:                Wed, 17 May 2023   Pseudo R-squ.:                 0.2969
Time:                        22:34:47   Log-Likelihood:                -4371.1
converged:                       True   LL-Null:                       -6217.2
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                         coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
VEHICLE_OWNERSHIP      -1.4851      0.053    -27.771      0.000      -1.590      -1.380
MARRIED                -0.5629      0.056     -9.974      0.000      -0.674      -0.452
CHILDREN               -0.1946      0.054     -3.618      0.000      -0.300      -0.089
POSTAL_CODE          1.928e-05   1.28e-06     15.060      0.000    1.68e-05    2.18e-05
SPEEDING_VIOLATIONS    -0.3190      0.022    -14.516      0.000      -0.362      -0.276
DUIS                   -0.3862      0.072     -5.386      0.000      -0.527      -0.246
PAST_ACCIDENTS         -0.5768      0.032    -17.859      0.000      -0.640      -0.513
GENDER_LABEL            1.3238      0.054     24.602      0.000       1.218       1.429
EDUCATION_LABEL        -0.0115      0.030     -0.386      0.699      -0.070       0.047
INCOME_LABEL            0.1150      0.022      5.132      0.000       0.071       0.159
VEHICLE_TYPE_LABEL      0.1992      0.121      1.642      0.101      -0.039       0.437
==============================================================================
```

Variable Significance

Table of all significant variables and their coefficients.

In [ ]:
```
coefficients = summary_result.params[0:]
p_values = summary_result.pvalues

results = pd.DataFrame({'Coefficient': coefficients, 'p-value': p_values})
results['p-value'] = results['p-value'].apply(lambda x: f'{x:.4f}')
results['p-value'] = pd.to_numeric(results['p-value'])
significant_vars = results[results['p-value'] < 0.05]
print(significant_vars)
```

```
                    Coefficient  p-value
VEHICLE_OWNERSHIP     -1.485120   0.0000
MARRIED               -0.562949   0.0000
CHILDREN              -0.194605   0.0003
POSTAL_CODE            0.000019   0.0000
SPEEDING_VIOLATIONS   -0.319029   0.0000
DUIS                  -0.386187   0.0000
PAST_ACCIDENTS        -0.576754   0.0000
GENDER_LABEL           1.323815   0.0000
INCOME_LABEL           0.115034   0.0000
```

Table 14. Variable Significance Table

Ranking the top three variables by the highest coefficient (by absolute value).

In [ ]:
```
coef_sort = abs(summary_result.params).sort_values(ascending=False).head(3)
table2 = pd.DataFrame({'Coefficient (abs)': coef_sort})
table2 = table2.loc[coef_sort.index]
print(table2)
```

```
                  Coefficient (abs)
VEHICLE_OWNERSHIP          1.485120
GENDER_LABEL               1.323815
PAST_ACCIDENTS             0.576754
```

Discussing the significant variables

Task 2. Question. From your initial thoughts, which variable sticks out to you as intriguing that it is significant and why. How could this information be useful to the insurer?

Answer. Based on the p-values and coefficients values, the variables that are statistically significant are as shown in Table 14, which have p-value less than 0.05, hence indicate that the variables have a positive impact in the prediction of the target variable.

The variables that sticks out as intriguing and which are statistically significant are **Gender, Income, and Vehicle Type as they have the highest positive coefficient value, whereas Vehicle Ownership and Past Accidents have the highest negative coefficient value.**

This information will be useful to the insurer in a way that they can analyze this data in order to minimize the risk of insurance and also can use this features to understand the data when a person is applying for a insurance. For example, it is observed that if the Income is high, there is likelihood that the insurance outcome is positive. Also, depending upon a certain vehicle type, the outcome variable changes. Hence, these parameters need to be considered by the company in order to check when someone applies for a insurance.

Model Testing

In [20]:
```
y_pred1 = logisticreg_model.predict(X_test_scaled)
```

Evaluating the performance of the model

1. Accuracy of the model on training and testing dataset
2. Confusion Matrix
3. Classification Report
4. AUC-ROC curve

In [21]:
```
# Accuracy of the model on training and testing set

print('Accuracy of Logistic Regressor model on training set: {:.3f}'.format(logisticreg_model.score(X_train_s
print('Accuracy of Logistic Regressor model on test set:     {:.3f}'.format(logisticreg_model.score(X_test_sc

model_result1 = logisticreg_model.score(X_test_scaled, y_test)
model_result1 = round(model_result1,4)
print("Overall Accuracy of the model is ", model_result1)
```

```
Accuracy of Logistic Regressor model on training set: 0.769
Accuracy of Logistic Regressor model on test set:     0.761
Overall Accuracy of the model is  0.761
```

In [22]:
```
# Confusion Matrix

confusionmatrix_LR = confusion_matrix(y_test, y_pred1)

fig = sns.heatmap(confusionmatrix_LR, annot=True,  annot_kws={"size": 15}, cmap = 'Blues', fmt='g')
fig.xaxis.set_ticklabels(['No Outcome','Outcome'])
fig.yaxis.set_ticklabels(['No Outcome','Outcome'])
fig.set_xlabel('Predicted Values')
fig.set_ylabel('Actual Values ')
fig.set_title('Confusion Matrix for the Logistic Regression Model')
sns.set(font_scale=1.0)
```

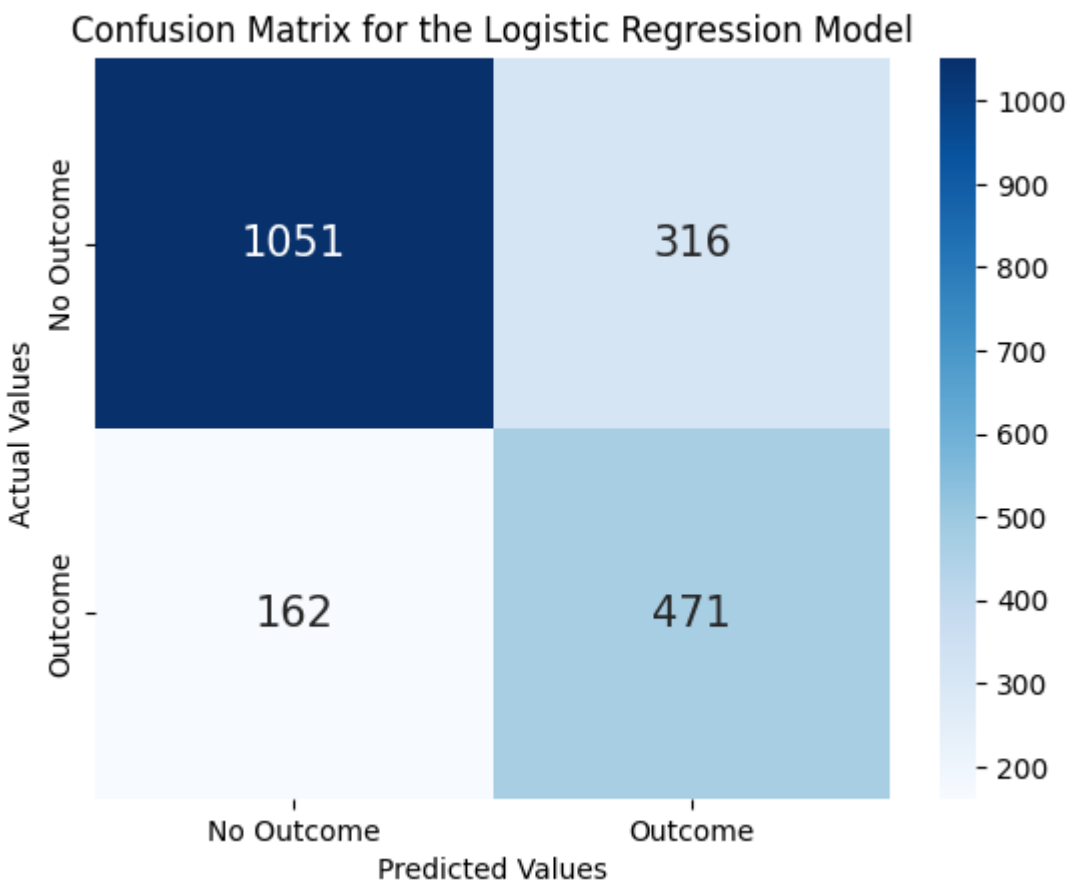

Figure 11. Confusion Matrix for Logistic Regression Model

In [23]:
```
# Classification Report

print("\n Classification report %s:\n%s\n" % (logisticreg_model, metrics.classification_report(y_test, y_pred
```

```
 Classification report LogisticRegression(class_weight='balanced', solver='newton-cg'):
              precision    recall  f1-score   support

           0       0.87      0.77      0.81      1367
           1       0.60      0.74      0.66       633

    accuracy                           0.76      2000
   macro avg       0.73      0.76      0.74      2000
weighted avg       0.78      0.76      0.77      2000
```

In [24]:
```
# AUC-ROC Curve

ypred_prob = logisticreg_model.predict_proba(X_test_scaled)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, ypred_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8,5))
plt.title('ROC Curve')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.xlabel('False Positive Rate')
plt.show()
```

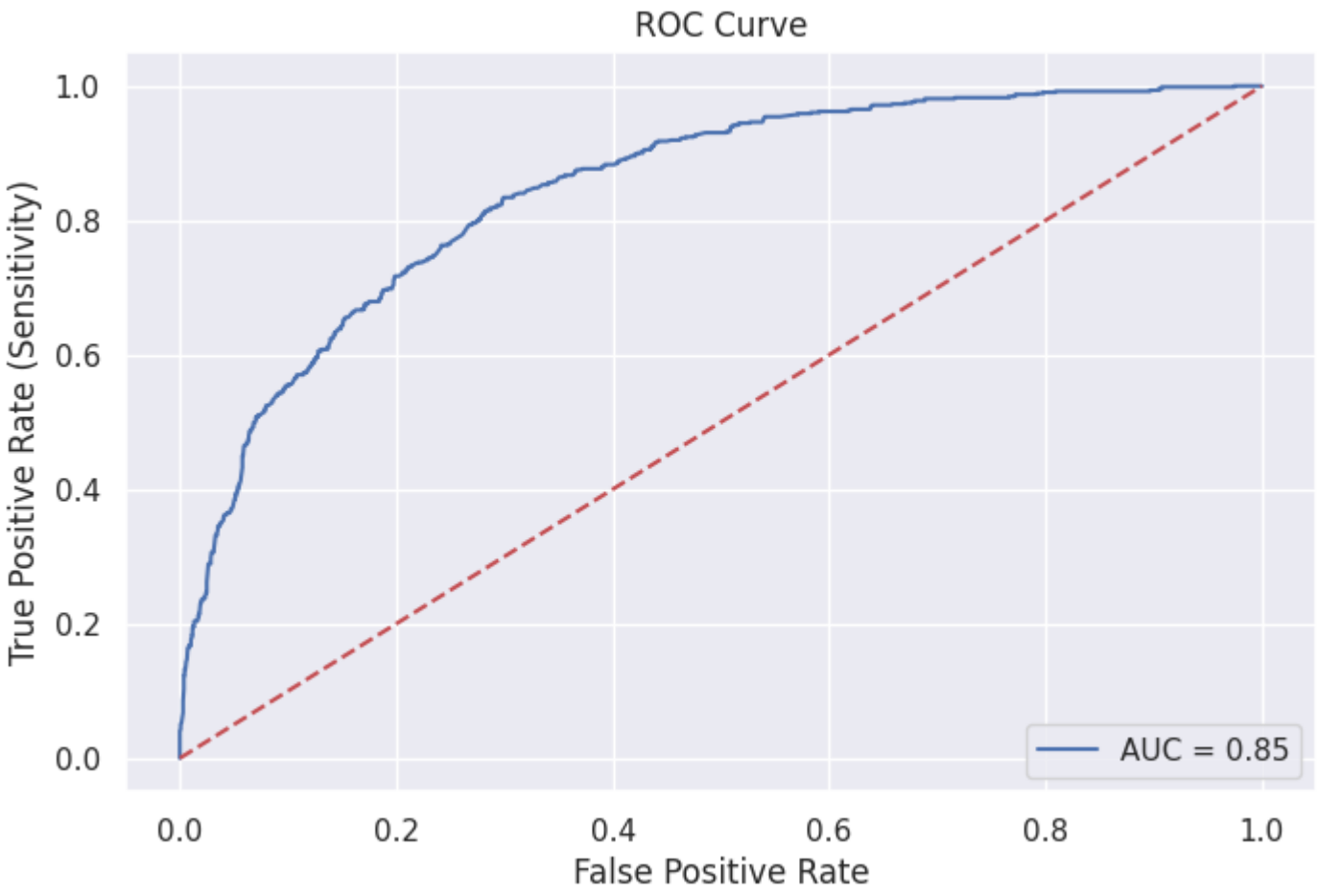

Figure 12. ROC Curve

Accuracy metric:

From the above evaluation metrics it is observed that the Logistic Regression model performed well in classifying the outcome variable of the insurance data with an accuracy of **76.9% for training data and 76.1% for testing data,** which indicates that the model has good accuracy in classification. Also, the accuracy difference between the training and testing data is almost the same which indicates that the model is neither overfitted nor underfitted and it is able to make predictions on the training data as well as accurately predict on the new or unseen data.

Confusion matrix:

The confusion matrix of the Logistic Regression model indicates that the **no outcome category is correctly classified 1051 times** whereas the **outcome class is correctly classified 471 times**, which is a good percent of values where the data is been correctly classified. However, the **no outcome category is wrongly classified times as outcome class 316 times** and **outcome class is classified as no outcome category 162 times**. Hence, the false positive and false negative values need to addressed to avoid inefficiency of the category classification.

Classification Report:

From the classification report we understand the model performance in terms of training and testing data based on the precision and recall values. The precision score for **no outcome and outcome class is 87% and 60% respectively**, and as it can be observed the precision score is good for both the categories. Similarly, the recall score for the **no outcome class is 60%** and for **outcome class is 74%** which indicates that the model is performing better to classify the classes.

ROC Curve:

The AUC score for the Logistic Regression model built to classify the house price range is **85%** which is a good AUC score indicating that the model performs well in classification of the outcome variable and is a good fit model.

Step 5: Model Building - Non-ensemble models

Task 3: Run a few non-ensemble models (only ones used in class) using (1000 iterations). Address the accuracy of each model and why you choose that model. Which model is the most accurate?

Building Non-ensemble models to predict the outcome variable.

1. KNN model
2. SVM model
3. Decision Tree model
4. Neural Networks

1. KNN Model

Fitting the KNN model

The KNN model is fit with value of n-nearest neighbors 3 and balanced class weight method in order to avoid class imbalance or bias data present in the training dataset.

In [12]:
```
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train_scaled, y_train)
```

Out[12]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

Model Testing

In [26]:
```
y_pred2 = knn_model.predict(X_test)
```

Evaluating the performance of the model

1. Accuracy of the model on training and testing dataset
2. Confusion Matrix
3. Classification Report
4. Feature Importance

In [27]:
```
# Accuracy of the model on training and testing set

print('Accuracy of KNN model on training set: {:.3f}'.format(knn_model.score(X_train_scaled, y_train)))
print('Accuracy of KNN model on test set:     {:.3f}'.format(knn_model.score(X_test_scaled, y_test)))


model_result2 = knn_model.score(X_test_scaled, y_test)
model_result2 = round(model_result2,4)
print("Overall Accuracy of the model is ", model_result2)
```

```
Accuracy of KNN model on training set: 0.836
Accuracy of KNN model on test set:     0.772
Overall Accuracy of the model is  0.7725
```

In [28]:
```
# Confusion Matrix

confusionmatrix_KNN = confusion_matrix(y_test, y_pred2)

fig = sns.heatmap(confusionmatrix_KNN, annot=True,  annot_kws={"size": 15}, cmap = 'Blues', fmt='g')
fig.xaxis.set_ticklabels(['No Outcome','Outcome'])
fig.yaxis.set_ticklabels(['No Outcome','Outcome'])
fig.set_xlabel('Predicted Values')
fig.set_ylabel('Actual Values ')
fig.set_title('Confusion Matrix for the KNN Model')
sns.set(font_scale=1.0)
```

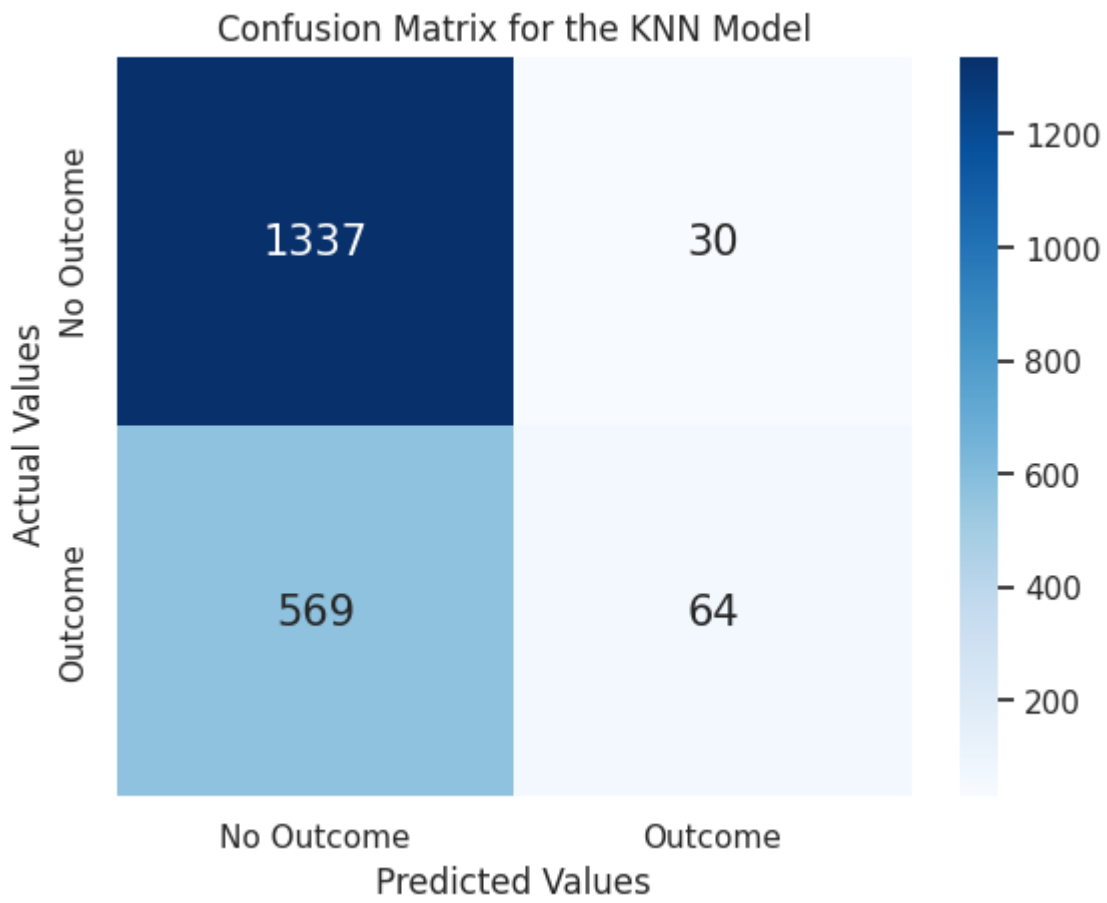

Figure 13. Confusion Matrix for KNN model

In [29]:
```
# Classification Report

print("\n Classification report %s:\n%s\n" % (knn_model, metrics.classification_report(y_test, y_pred2)))
```

```
 Classification report KNeighborsClassifier(n_neighbors=3):
              precision    recall  f1-score   support

           0       0.70      0.98      0.82      1367
           1       0.68      0.10      0.18       633

    accuracy                           0.70      2000
   macro avg       0.69      0.54      0.50      2000
weighted avg       0.69      0.70      0.61      2000
```

Accuracy metric:

From the above evaluation metrics it is observed that the KNN model performed well in classifying the outcome variable of the insurance data with an accuracy of **83.6% for training data and 77.2% for testing data,** which indicates that the model has good accuracy in classification. Also, the accuracy difference between the training and testing data slightly differs which indicates that the model is overfitted and it is not able to make predictions on the testing data or the new unseen data.

Confusion matrix:

The confusion matrix of the KNN model indicates that the **no outcome category is correctly classified 1337 times** whereas the **outcome class is correctly classified 64 times**, which is a good percent of values where the data is been correctly classified. However, the **no outcome category is wrongly classified times as outcome class 30 times** and **outcome class is classified as no outcome category 569 times**. Hence, the false positive and false negative values need to addressed to avoid inefficiency of the category classification.

Classification Report:

From the classification report we understand the model performance in terms of training and testing data based on the precision and recall values. The precision score for **no outcome and outcome class is 70% and 68% respectively**, and as it can be observed the precision score is good for both the categories. Similarly, the recall score for the **no outcome class is 98%** and for **outcome class is 10%** which indicates that the model is performing better to classify the classes, but has a low recall score for the outcome category class.

KNN model in comparison with Logistic Regression model

- As compared to the goodness of fit metrics and evaluation metrics for the Logistic Regression, it is observed that **KNN model performed better in classifying the outcome variable** as compared to the Logistic Regression model.
- KNN model **performed well** as compared to the Logistic Regression model.
- This is because the **accuracy of the training and testing data slightly increased with respect to the KNN model** as compared to that of the Logistic Regression model.
- Apart from that, when analyzed the **confusion matrix we see that high number of classes are correctly classified** using the KNN model as compared to the LR model, which is important because if the less number of false positive and false negative values are there, it will be less of a job to manually address that and hence would be effective in analyzing the data.
- Also, the **precision and recall score is relatively high for the KNN model** in comparison with the Logistic Regression model, indicating that the prediction of classes is better performed in KNN model.

2. SVM Model

Fitting the SVM model

The SVM model is fit with a linear kernel and balanced class weight method in order to avoid class imbalance or bias data present in the training dataset.

In [13]:
```
svm_model = SVC(kernel='linear', C=1, random_state=42, class_weight='balanced')
svm_model.fit(X_train_scaled, y_train)
```

Out[13]:
```
SVC(C=1, class_weight='balanced', kernel='linear', random_state=42)
```

Model Testing

In [31]:
```
y_pred3 = svm_model.predict(X_test_scaled)
```

Evaluating the performance of the model

1. Accuracy of the model on training and testing dataset
2. Confusion Matrix
3. Classification Report
4. Feature Importance

In [32]:
```
# Accuracy of the model on training and testing set

print('Accuracy of SVM model on training set: {:.3f}'.format(svm_model.score(X_train_scaled, y_train)))
print('Accuracy of SVM model on test set:     {:.3f}'.format(svm_model.score(X_test_scaled, y_test)))

model_result3 = svm_model.score(X_test_scaled, y_test)
model_result3 = round(model_result3,4)
print("Overall Accuracy of the model is ", model_result3)
```

```
Accuracy of SVM model on training set: 0.771
Accuracy of SVM model on test set:     0.761
Overall Accuracy of the model is  0.7615
```

In [33]:
```
# Confusion Matrix

confusionmatrix_SVM = confusion_matrix(y_test, y_pred3)

fig = sns.heatmap(confusionmatrix_SVM, annot=True,  annot_kws={"size": 15}, cmap = 'Blues', fmt='g')
fig.xaxis.set_ticklabels(['No Outcome','Outcome'])
fig.yaxis.set_ticklabels(['No Outcome','Outcome'])
fig.set_xlabel('Predicted Values')
fig.set_ylabel('Actual Values ')
fig.set_title('Confusion Matrix for the SVM Model')
sns.set(font_scale=1.0)
```

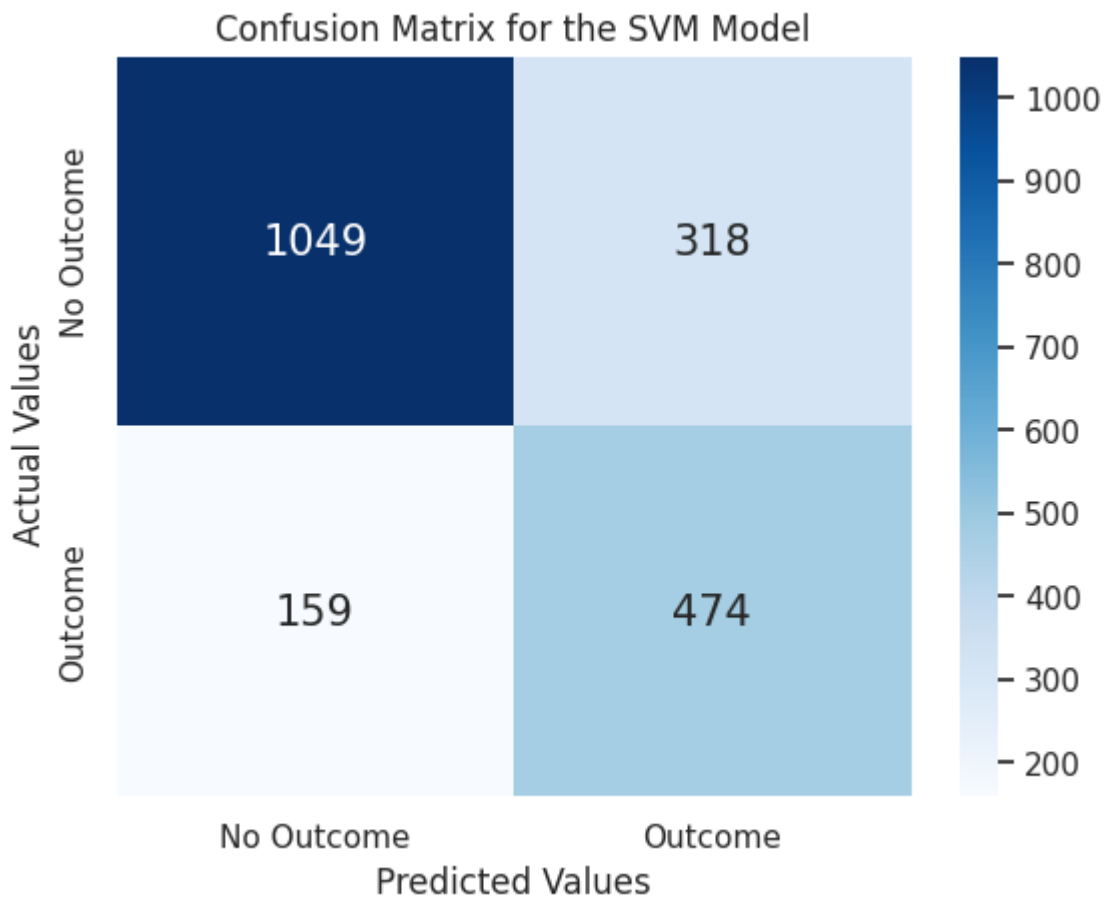

Figure 14. Confusion Matrix for SVM Model

In [34]:
```
# Classification Report

print("\n Classification report %s:\n%s\n" % (svm_model, metrics.classification_report(y_test, y_pred3)))
```

```
 Classification report SVC(C=1, class_weight='balanced', kernel='linear', random_state=42):
              precision    recall  f1-score   support

           0       0.87      0.77      0.81      1367
           1       0.60      0.75      0.67       633

    accuracy                           0.76      2000
   macro avg       0.73      0.76      0.74      2000
weighted avg       0.78      0.76      0.77      2000
```

Feature Importance Visualization for SVM Model

In [35]:
```
feature_importances = pd.Series(abs(svm_model.coef_[0]), index=X.columns)
feature_importances.nlargest(10).plot(kind='barh', color="darkblue", title="Feature Importance for SVM model")
```

Out[35]: <Axes: title={'center': 'Feature Importance for SVM model'}>

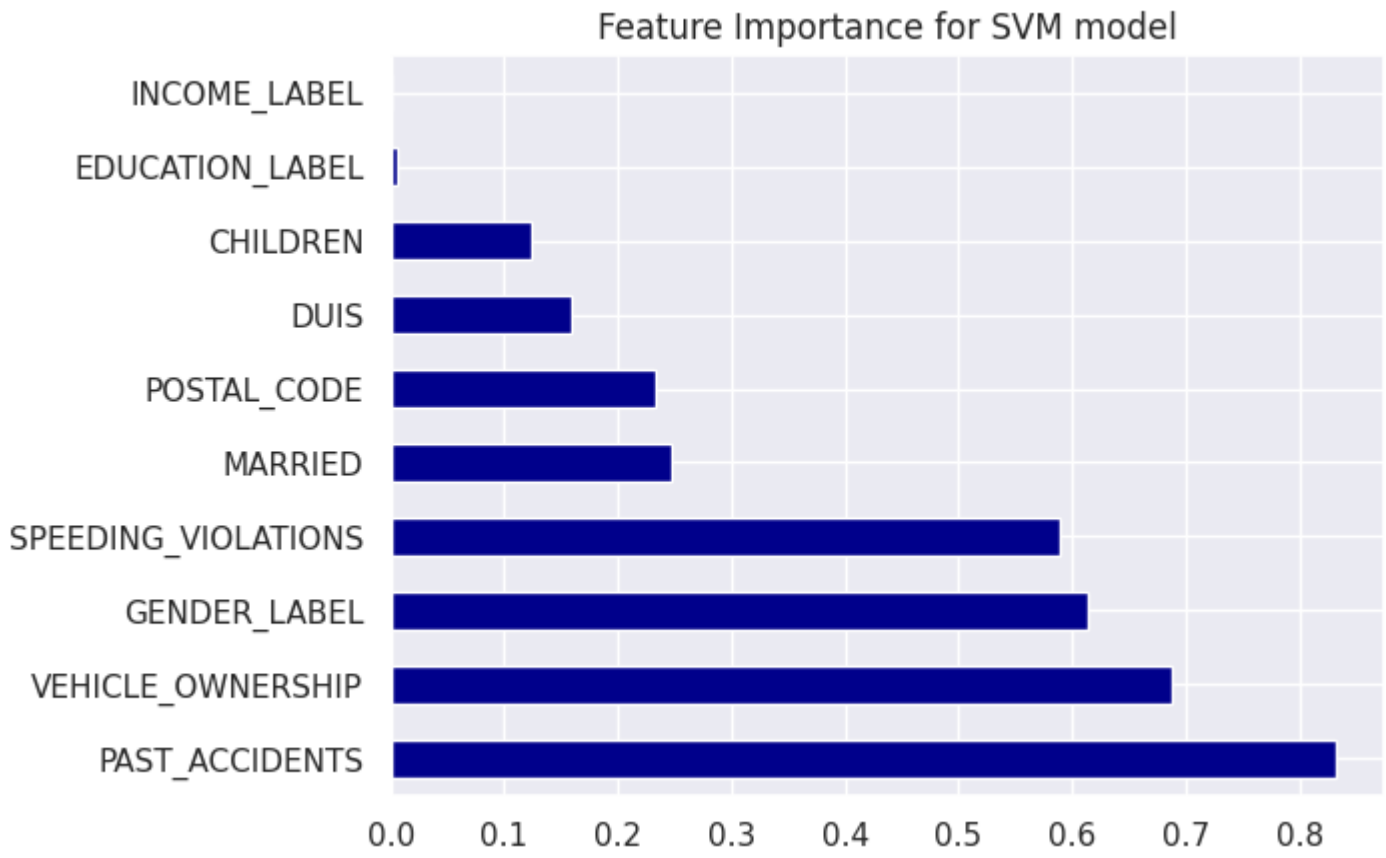

Figure 15. Feature Importance for SVM Model

Accuracy metric:

From the above evaluation metrics it is observed that the SVM model performed well in classifying the outcome variable of the insurance data with an accuracy of **77.1% for training data and 76.1% for testing data,** which indicates that the model has good accuracy in classification. Also, the accuracy difference between the training and testing data is the same which indicates that the model is neither overfitted nor underfitted and it is able to make predictions on the training data as well as accurately predict on the new or unseen data.

Confusion matrix:

The confusion matrix of the SVM model indicates that the **no outcome category is correctly classified 1049 times** whereas the **outcome class is correctly classified 474 times**, which is a good percent of values where the data is been correctly classified. However, the **no outcome category is wrongly classified times as outcome class 318 times** and **outcome class is classified as no outcome category 159 times**. Hence, the false positive and false negative values need to addressed to avoid inefficiency of the category classification.

Classification Report:

From the classification report we understand the model performance in terms of training and testing data based on the precision and recall values. The precision score for **no outcome and outcome class is 87% and 60% respectively**, and as it can be observed the precision score is good for both the categories. Similarly, the recall score for the **no outcome class is 77%** and for **outcome class is 75%** which indicates that the model is performing better to classify the classes.

Feature Importance Score:

The feature importance graph shows that variables **Past_Accidents, Vehicle_Ownership, and Gender** have the highest feature importance with **score of 0.8, 0.7, and 0.6 respectively** indicating that the model classification for insurance data is based on these features and that they are highly contributing in the prediction of the target variable. Thus, **it is recommended that the company should focus on these parameters and features while analyzing the insurance data.**

SVM model comparison with Logistic Regression & KNN model

- As compared to the goodness of fit metrics and evaluation metrics for the Logistic Regression and KNN model, it is observed that **KNN model performed better in classifying the outcome variable** as compared to the Logistic Regression and SVM model.
- KNN model **performed well** as compared to the Logistic Regression and SVM model.
- This is because the **accuracy of the training and testing data slightly increased with respect to the KNN model** as compared to that of the Logistic Regression and SVM model.
- Apart from that, when analyzed the **confusion matrix we see that high number of classes are correctly classified** using the KNN model as compared to the LR and SVM model, which is important because if the less number of false positive and false negative values are there, it will be less of a job to manually address that and hence would be effective in analyzing the data.
- Also, the **precision and recall score is relatively high for the KNN model** in comparison with the Logistic Regression and SVM model, indicating that the prediction of classes is better performed in KNN model.
- Although, SVM model performed better after KNN model as compared to the LR model.

3. Decision Tree Model

Fitting the Decision Tree model

The Decision Tree model is fit with a max depth of 4 and balanced class weight method in order to avoid class imbalance or bias data present in the training dataset.

In [14]:
```
decisiontree_model = DecisionTreeClassifier(max_depth=4, random_state=42, class_weight="balanced")
decisiontree_model.fit(X_train_scaled, y_train)
```

Out[14]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=4, random_state=42)
```

Plotting the Decision Tree

In [37]:
```
# plotting the decision tree

plt.figure(figsize=(18,8))
plot_tree(decisiontree_model, filled=True, rounded=True, feature_names=X_train.columns, class_names=["No Outc
plt.show()
```

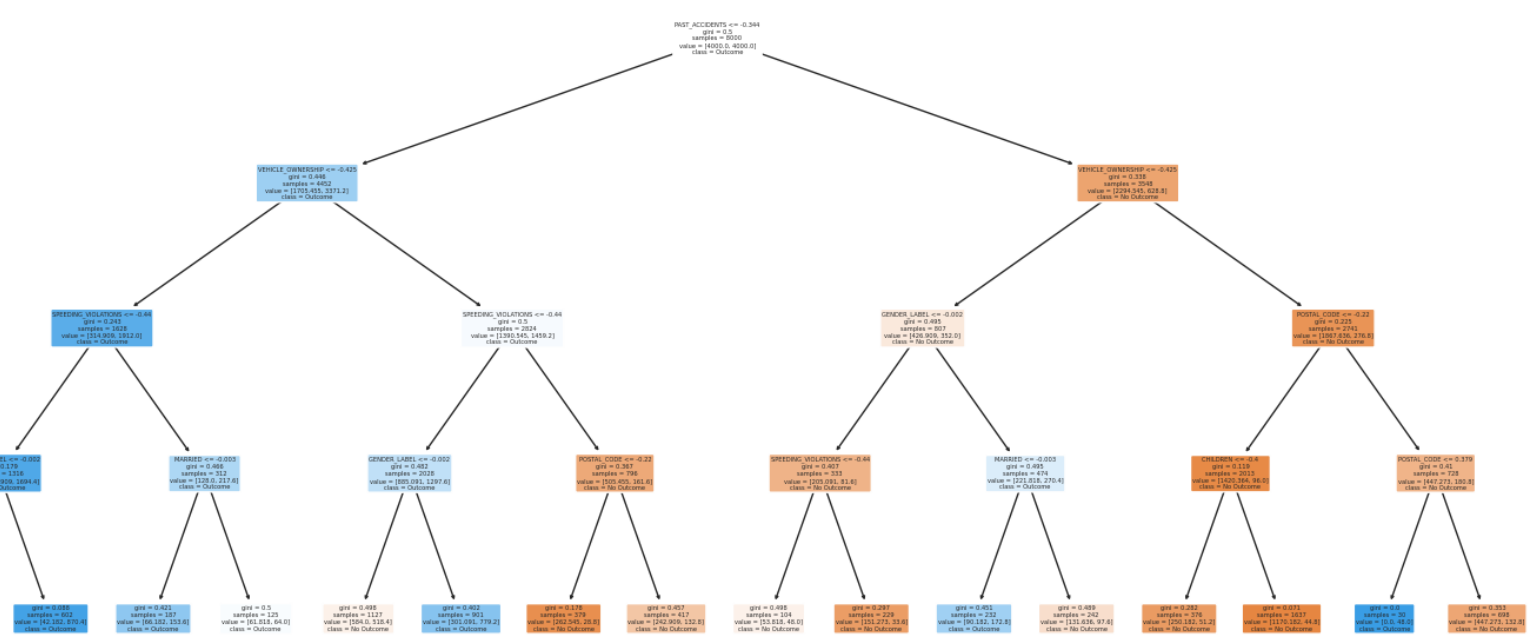

Figure 16. Decision Tree

Model Testing

In [38]:
```
y_pred4 = decisiontree_model.predict(X_test_scaled)
```

Evaluating the performance of the model

1. Accuracy of the model on training and testing dataset
2. Confusion Matrix
3. Classification Report
4. Feature Importance

In [39]:
```
# Accuracy of the model on training and testing set

print('Accuracy of Decision Tree model on training set: {:.3f}'.format(decisiontree_model.score(X_train_scale
print('Accuracy of Decision Tree model on test set:     {:.3f}'.format(decisiontree_model.score(X_test_scaled

model_result4 = decisiontree_model.score(X_test_scaled, y_test)
model_result4 = round(model_result4,4)
print("Overall Accuracy of the model is ", model_result4)
```

```
Accuracy of Decision Tree model on training set: 0.794
Accuracy of Decision Tree model on test set:     0.773
Overall Accuracy of the model is  0.7735
```

In [40]:
```
# Confusion Matrix

confusionmatrix_DT = confusion_matrix(y_test, y_pred4)

fig = sns.heatmap(confusionmatrix_DT, annot=True,  annot_kws={"size": 15}, cmap = 'Blues', fmt='g')
fig.xaxis.set_ticklabels(['No Outcome','Outcome'])
fig.yaxis.set_ticklabels(['No Outcome','Outcome'])
fig.set_xlabel('Predicted Values')
fig.set_ylabel('Actual Values ')
fig.set_title('Confusion Matrix for the Decision Tree Model')
sns.set(font_scale=1.0)
```

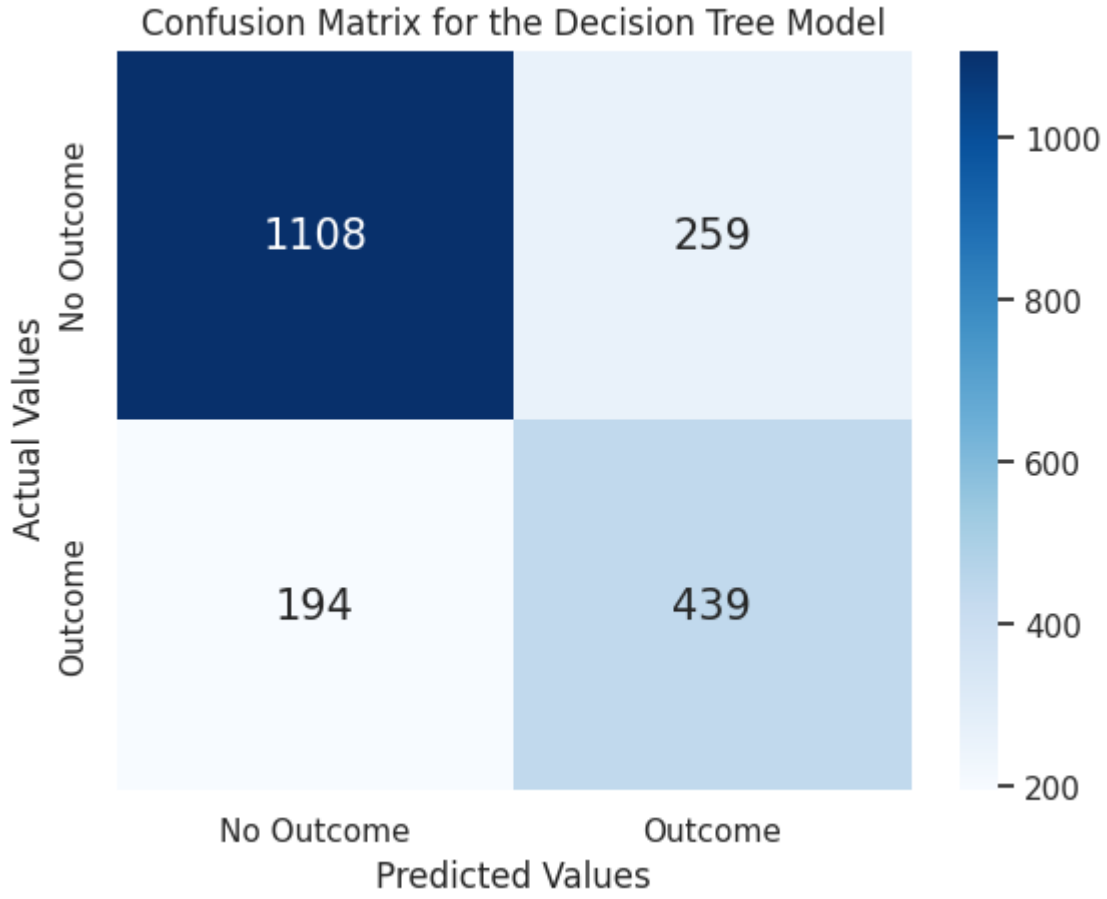

Figure 17. Confusion Matrix for Decision Tree model

In [41]:
```
# Classification Report

print("\n Classification report %s:\n%s\n" % (decisiontree_model, metrics.classification_report(y_test, y_pre
```

```
 Classification report DecisionTreeClassifier(class_weight='balanced', max_depth=4, random_state=42):
              precision    recall  f1-score   support

           0       0.85      0.81      0.83      1367
           1       0.63      0.69      0.66       633

    accuracy                           0.77      2000
   macro avg       0.74      0.75      0.74      2000
weighted avg       0.78      0.77      0.78      2000
```

Feature Importance for Decision Tree model

In [42]:
```
feature_importances = pd.Series(decisiontree_model.feature_importances_, index=X.columns)
feature_importances.nlargest(10).plot(kind='barh', color="darkblue", title = "Feature Importance for Decision
```

Out[42]: <Axes: title={'center': 'Feature Importance for Decision Tree model'}>

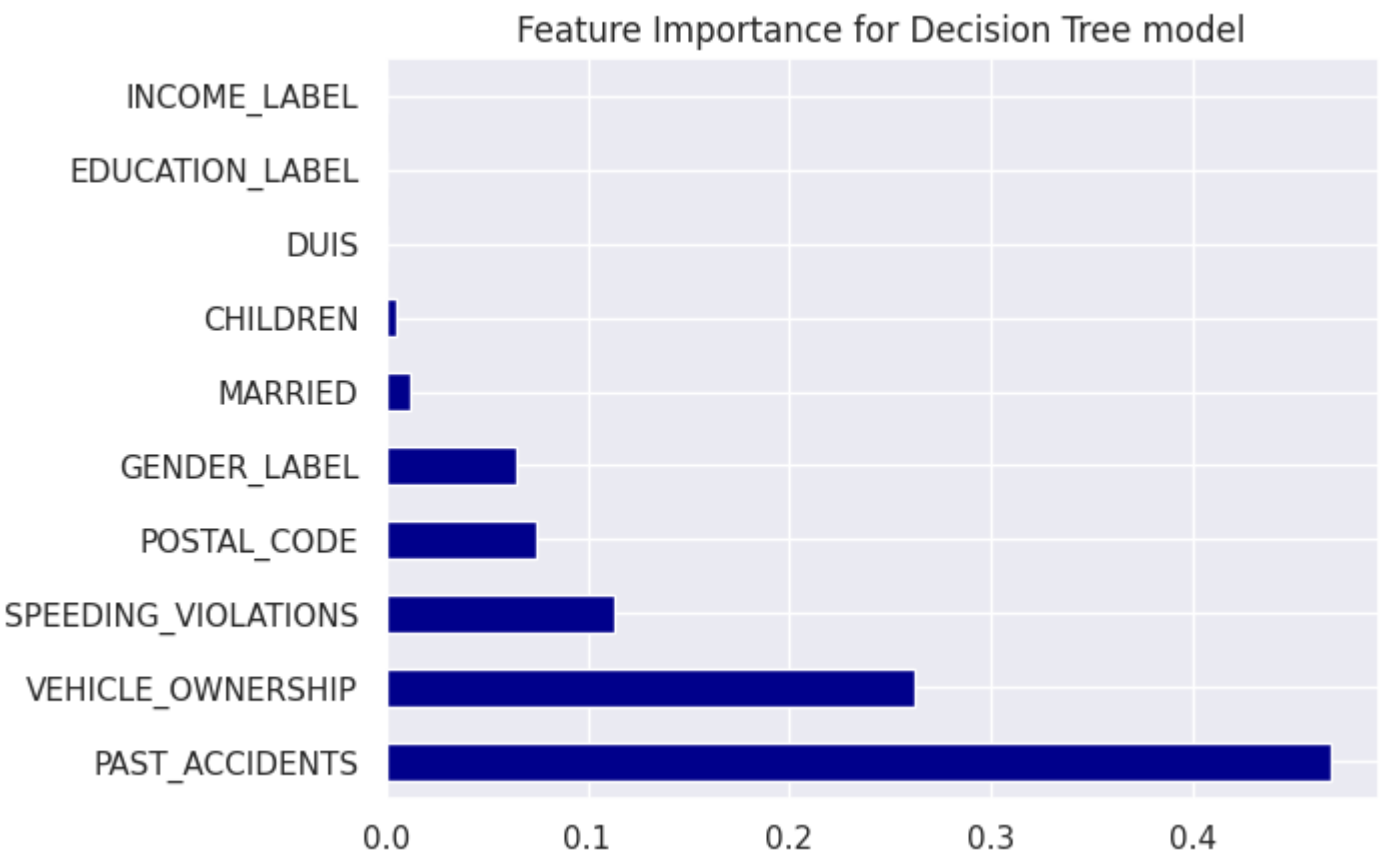

Figure 18. Feature Importance for Decision Tree model

Feature Importance Score

In [43]:
```
# extracting feature importance

feature_importance = pd.DataFrame({'Feature': X_train.columns, 'Importance Score' : np.round(decisiontree_mod
feature_importance.sort_values('Importance Score', ascending=False, inplace = True)
print(feature_importance)
```

	Feature	Importance Score
6	PAST_ACCIDENTS	0.468
0	VEHICLE_OWNERSHIP	0.262
4	SPEEDING_VIOLATIONS	0.113
3	POSTAL_CODE	0.074
7	GENDER_LABEL	0.065
1	MARRIED	0.012
2	CHILDREN	0.005
5	DUIS	0.000
8	EDUCATION_LABEL	0.000
9	INCOME_LABEL	0.000
10	VEHICLE_TYPE_LABEL	0.000

Table 13. Feature Importance for Decision Tree model

Accuracy metric:

From the above evaluation metrics it is observed that the Decision Tree model well in classifying the outcome variable of the insurance data with an accuracy of **79.4% for training data and 77.3% for testing data,** which indicates that the model has good accuracy in classification. Also, the accuracy difference between the training and testing data is the same which indicates that the model is neither overfitted nor underfitted and it is able to make predictions on the training data as well as accurately predict on the new or unseen data.

Confusion matrix:

The confusion matrix of the Decision Tree model indicates that the **no outcome category is correctly classified 1108 times** whereas the **outcome class is correctly classified 439 times**, which is a good percent of values where the data is been correctly classified. However, the **no outcome category is wrongly classified times as outcome class 259 times** and **outcome class is classified as no outcome category 194 times**. Hence, the false positive and false negative values need to addressed to avoid inefficiency of the category classification.

Classification Report:

From the classification report we understand the model performance in terms of training and testing data based on the precision and recall values. The precision score for **no outcome and outcome class is 85% and 63% respectively**, and as it can be observed the precision score is good for both the categories. Similarly, the recall score for the **no outcome class is 81%** and for **outcome class is**

69% which indicates that the model is performing better to classify the classes.

Feature Importance Score:

The feature importance graph shows that variables **Past_Accidents, Vehicle_Ownership, and Speeding_Violations** have the highest feature importance with **score of 0.468, 0.262, and 0.113 respectively** indicating that the model classification for insurance data is based on these features and that they are highly contributing in the prediction of the target variable. Thus, **it is recommended that the company should focus on these parameters and features while analyzing the insurance data.**

Decision Tree model comparison with Logistic Regression, KNN, and SVM model

---

- As compared to the goodness of fit metrics and evaluation metrics for the Logistic Regression, KNN, and SVM model, it is observed that **Decision Tree model performed better in classifying the outcome variable** as compared to the Logistic Regression, KNN, and SVM model.
- Decision Tree model **performed well** as compared to the Logistic Regression, KNN, and SVM model.
- This is because the **accuracy of the training and testing data slightly increased with respect to the Decision Tree model** as compared to that of the Logistic Regression, KNN, and SVM model.
- Apart from that, when analyzed the **confusion matrix we see that high number of classes are correctly classified** using the Decision Tree model as compared to the LR, KNN, and SVM model, which is important because if the less number of false positive and false negative values are there, it will be less of a job to manually address that and hence would be effective in analyzing the data.
- Also, the **precision and recall score is relatively high for the Decision Tree model** in comparison with the Logistic Regression, KNN, and SVM model, indicating that the prediction of classes is better performed in Decision Tree model.

4. Neural Network Model

In [44]:
```
nnmodel = keras.Sequential([
    keras.layers.Dense(64, activation='tanh', input_shape=(X_train_scaled.shape[1],)),
    keras.layers.Dense(32, activation='tanh'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

Fitting the Neural Network model

The Neural Network model is fit with a batch size of 32, validation split of 0.2, and 10 epochs.

In [45]:
```
nnmodel.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [46]:
```
nnmodel.fit(X_train_scaled, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

```
Epoch 1/20
200/200 [==============================] - 2s 5ms/step - loss: 0.4567 - accuracy: 0.7852 - val_loss: 0.4341
- val_accuracy: 0.7987
Epoch 2/20
200/200 [==============================] - 0s 2ms/step - loss: 0.4229 - accuracy: 0.8052 - val_loss: 0.4264
- val_accuracy: 0.8056
Epoch 3/20
200/200 [==============================] - 1s 3ms/step - loss: 0.4164 - accuracy: 0.8102 - val_loss: 0.4225
- val_accuracy: 0.8037
Epoch 4/20
200/200 [==============================] - 0s 2ms/step - loss: 0.4105 - accuracy: 0.8100 - val_loss: 0.4233
- val_accuracy: 0.8062
Epoch 5/20
200/200 [==============================] - 1s 3ms/step - loss: 0.4086 - accuracy: 0.8112 - val_loss: 0.4248
- val_accuracy: 0.8037
Epoch 6/20
200/200 [==============================] - 0s 2ms/step - loss: 0.4064 - accuracy: 0.8147 - val_loss: 0.4190
- val_accuracy: 0.8125
Epoch 7/20
200/200 [==============================] - 1s 3ms/step - loss: 0.4041 - accuracy: 0.8123 - val_loss: 0.4216
- val_accuracy: 0.8100
Epoch 8/20
200/200 [==============================] - 0s 2ms/step - loss: 0.4025 - accuracy: 0.8142 - val_loss: 0.4159
- val_accuracy: 0.8125
Epoch 9/20
200/200 [==============================] - 1s 3ms/step - loss: 0.4004 - accuracy: 0.8153 - val_loss: 0.4178
- val_accuracy: 0.8144
Epoch 10/20
200/200 [==============================] - 0s 2ms/step - loss: 0.4002 - accuracy: 0.8141 - val_loss: 0.4163
- val_accuracy: 0.8188
Epoch 11/20
200/200 [==============================] - 0s 2ms/step - loss: 0.3977 - accuracy: 0.8170 - val_loss: 0.4163
- val_accuracy: 0.8156
Epoch 12/20
200/200 [==============================] - 1s 3ms/step - loss: 0.3959 - accuracy: 0.8216 - val_loss: 0.4239
- val_accuracy: 0.8056
Epoch 13/20
200/200 [==============================] - 1s 3ms/step - loss: 0.3962 - accuracy: 0.8188 - val_loss: 0.4215
- val_accuracy: 0.8138
Epoch 14/20
200/200 [==============================] - 1s 3ms/step - loss: 0.3946 - accuracy: 0.8169 - val_loss: 0.4190
- val_accuracy: 0.8094
Epoch 15/20
200/200 [==============================] - 1s 3ms/step - loss: 0.3933 - accuracy: 0.8147 - val_loss: 0.4176
- val_accuracy: 0.8069
Epoch 16/20
200/200 [==============================] - 0s 2ms/step - loss: 0.3930 - accuracy: 0.8183 - val_loss: 0.4156
- val_accuracy: 0.8106
Epoch 17/20
200/200 [==============================] - 1s 3ms/step - loss: 0.3909 - accuracy: 0.8188 - val_loss: 0.4159
- val_accuracy: 0.8112
Epoch 18/20
200/200 [==============================] - 1s 3ms/step - loss: 0.3885 - accuracy: 0.8195 - val_loss: 0.4190
- val_accuracy: 0.8119
Epoch 19/20
200/200 [==============================] - 1s 3ms/step - loss: 0.3885 - accuracy: 0.8180 - val_loss: 0.4163
- val_accuracy: 0.8100
Epoch 20/20
200/200 [==============================] - 1s 4ms/step - loss: 0.3882 - accuracy: 0.8188 - val_loss: 0.4146
- val_accuracy: 0.8094
```

Out[46]:
```
<keras.callbacks.History at 0x7fdebbc19390>
```

In [47]:
```
test_loss, test_acc = nnmodel.evaluate(X_test_scaled, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
63/63 [==============================] - 0s 2ms/step - loss: 0.4203 - accuracy: 0.7980
Test loss: 0.42032545804977417
Test accuracy: 0.7979999780654907
```

Model Testing

In [48]:
```
y_pred5 = nnmodel.predict(X_test_scaled)
```

```
63/63 [==============================] - 0s 1ms/step
```

Evaluating the performance of the model

1. Accuracy of the model
2. Confusion Matrix
3. Classification Report
4. Feature Importance

In [49]:
```
# converting predicted values to binary values

y_pred5 = (y_pred5 > 0.5).astype(int)
```

In [50]:
```
# Confusion Matrix

confusionmatrix_NN = confusion_matrix(y_test, y_pred5)
fig = sns.heatmap(confusionmatrix_NN, annot=True,  annot_kws={"size": 15}, cmap = 'Blues', fmt='g')
fig.xaxis.set_ticklabels(['No Outcome','Outcome'])
fig.yaxis.set_ticklabels(['No Outcome','Outcome'])
fig.set_xlabel('Predicted Values')
fig.set_ylabel('Actual Values ')
fig.set_title('Confusion Matrix for the Neural Network Model')
sns.set(font_scale=1.0)
```

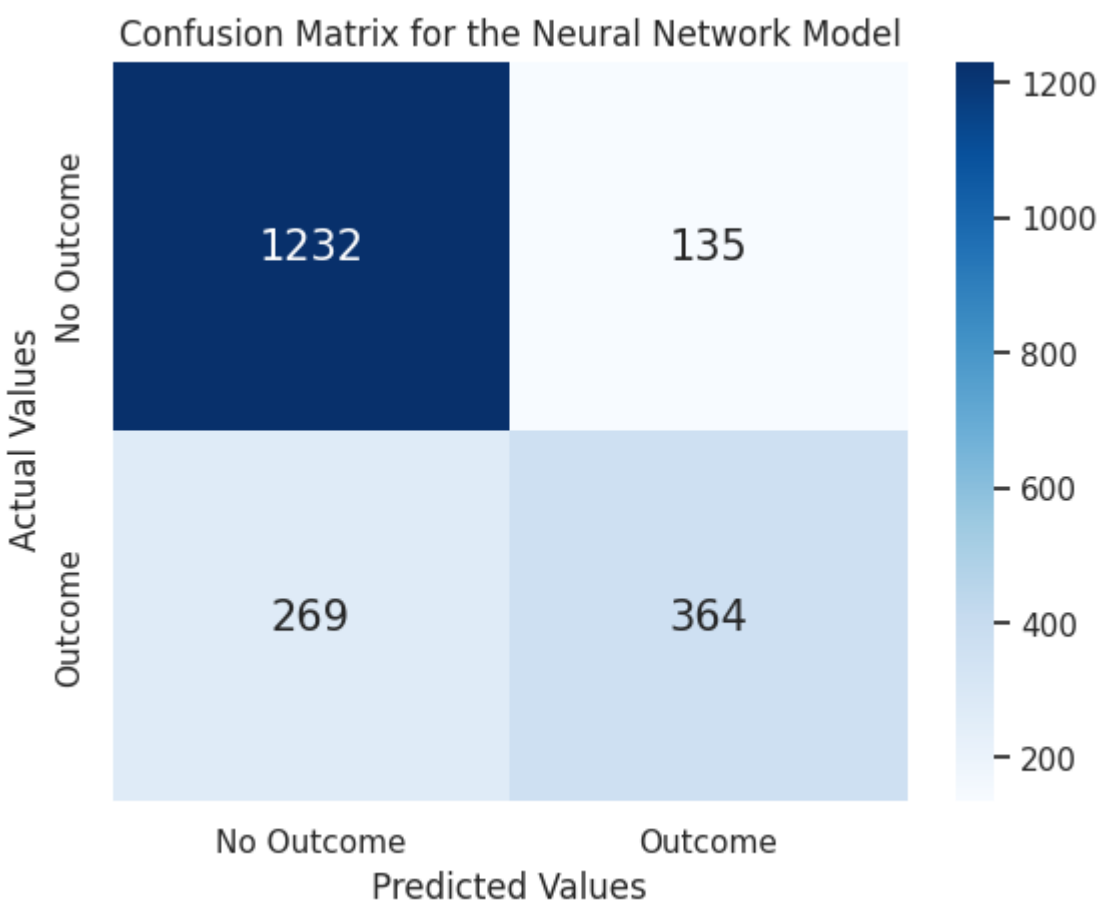

Figure 19. Confusion Matrix for Neural Network model

In [51]:
```
# Classification Report

print("\n Classification report %s:\n%s\n" % (nnmodel, metrics.classification_report(y_test, y_pred5)))
```

```
 Classification report <keras.engine.sequential.Sequential object at 0x7fdebe760e80>:
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      1367
           1       0.73      0.58      0.64       633

    accuracy                           0.80      2000
   macro avg       0.78      0.74      0.75      2000
weighted avg       0.79      0.80      0.79      2000
```

Feature Importance for Neural Network model

In [52]:
```
nnmodel_weights = nnmodel.get_weights()
w = nnmodel_weights[0]
feature_importance_nn = np.mean(np.abs(w), axis=1)
feature_importances_nn = pd.Series(feature_importance_nn, index=X.columns)
feature_importances_nn.nlargest(10).plot(kind='barh', color="darkblue", title = "Feature Importance for Neura
```

Out[52]: <Axes: title={'center': 'Feature Importance for Neural Network model'}>

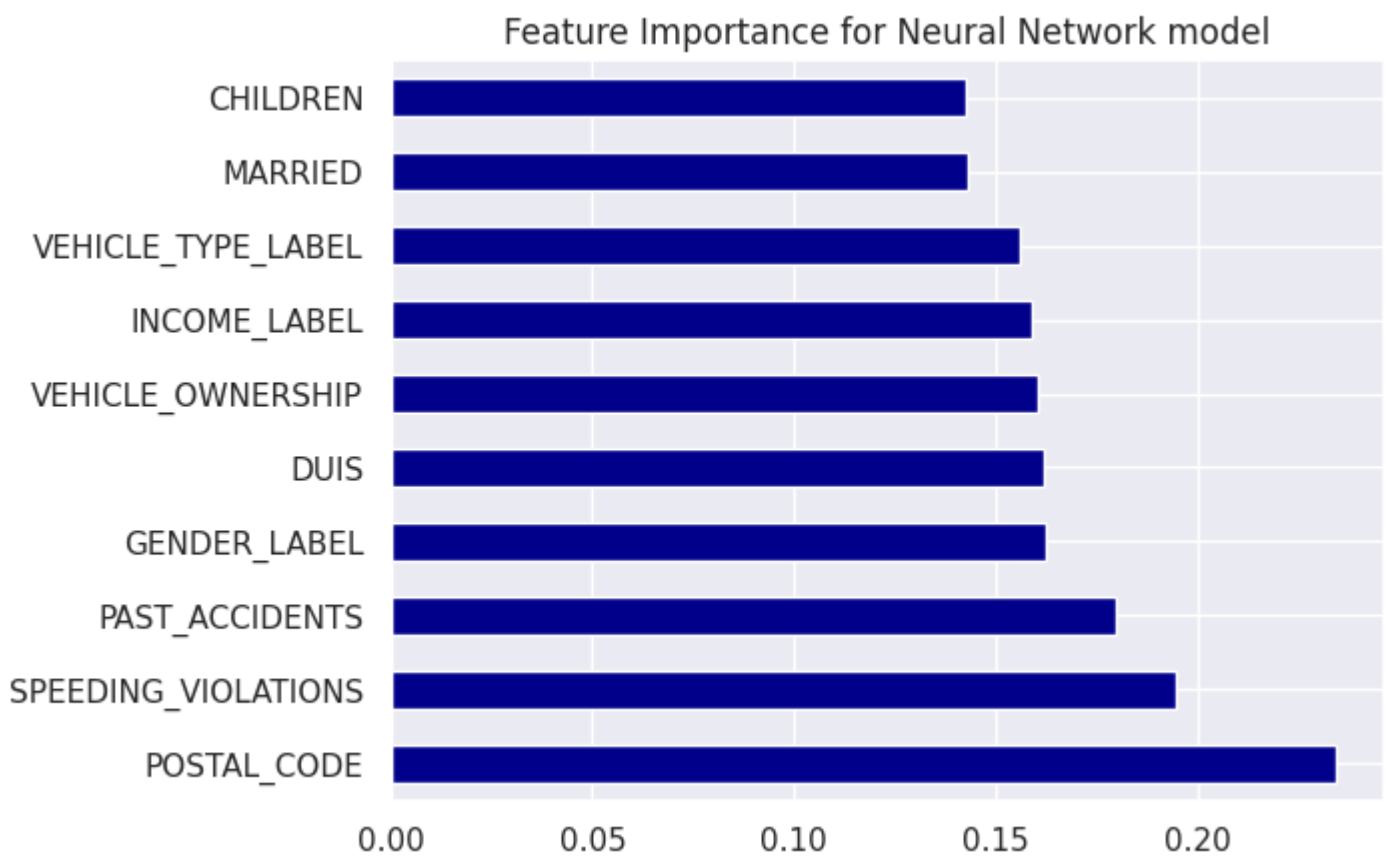

Figure 20. Feature Importance for Neural Network model

Accuracy metric:

From the above evaluation metrics it is observed that the Neural Network model well in classifying the outcome variable of the insurance data with a test accuracy of **79.79%** which indicates that the model has good accuracy in classification.

Confusion matrix:

The confusion matrix of the Neural Network model indicates that the **no outcome category is correctly classified 1232 times** whereas the **outcome class is correctly classified 364 times**, which is a good percent of values where the data is been correctly classified. However, the **no outcome category is wrongly classified times as outcome class 135 times** and **outcome class is classified as no outcome category 269 times**. Hence, the false positive and false negative values need to addressed to avoid inefficiency of the category classification.

Classification Report:

From the classification report we understand the model performance in terms of training and testing data based on the precision and recall values. The precision score for **no outcome and outcome class is 82% and 73% respectively**, and as it can be observed the precision score is good for both the categories. Similarly, the recall score for the **no outcome class is 90%** and for **outcome class is 58%** which indicates that the model is performing better to classify the classes.

Feature Importance Score:

The feature importance graph shows that variables **Postal_Code**, **Speeding_Violations**, and **Past_Accidents** have the highest feature importance indicating that the model classification for insurance data is based on these features and that they are highly contributing in the prediction of the target variable. Thus, **it is recommended that the company should focus on these parameters and features while analyzing the insurance data.**

Neural Network model in comparison with Logistic Regression, KNN, SVM, and Decision Tree model

---

- As compared to the goodness of fit metrics and evaluation metrics for the Logistic Regression, KNN, SVM, and Decision Tree model, it is observed that **Neural Network model performed better in classifying the outcome variable** as compared to the Logistic Regression, KNN, SVM, and Decision Tree model.
- Neural Network model **performed well** as compared to the Logistic Regression, KNN, SVM, and Decision Tree model.
- This is because the **accuracy of the training and testing data slightly increased with respect to the Neural Network model** as compared to that of the Logistic Regression, KNN, SVM, and Decision Tree model.
- Apart from that, when analyzed the **confusion matrix we see that high number of classes are correctly classified** using the Neural Network model as compared to the LR, KNN, SVM, and Decision Tree model, which is important because if the less number of false positive and false negative values are there, it will be less of a job to manually address that and hence would be

effective in analyzing the data.

- Also, the **precision and recall score is relatively high for the Neural Network model** in comparison with the Logistic Regression, KNN, SVM, and Decision Tree model, indicating that the prediction of classes is better performed in Neural Network model.

Task 3. Question. Address the accuracy of each model and why you choose that model. Which model is the most accurate?

Answer.

- The Logistic Regression and non-ensemble classification models are built and implemented in order to classify the target variable of the insurance data which will help the company analyze the features when someone applies for an insurance and also help to minimize the risk of insurance.
- The accuracy of each model is addressed below, which indicates that Neural Network and Decision Tree model are the most accurate models in predicting the outcome variable for the insurance data.
- The significant features for each of the model are listed below which will help the company focus on those parameters when analyzing the data.

Comparing the accuracy of the models built (accuracy, precision, recall)

In [54]:
```
metrics_data = []
metrics_data.append(['Logistic Regression Model', model_result1, '60%', '74%', 'Gender, Income, Vehicle Type'
metrics_data.append(['KNN Model', model_result2, '68%', '10%'])
metrics_data.append(['SVM Model', model_result3, '60%', '75%', 'Past_Accidents, Vehicle_Ownership, Gender'])
metrics_data.append(['Decision Tree Model', model_result4, '63%', '69%', 'Past_Accidents, Vehicle_Ownership,
metrics_data.append(['Neural Network Model', test_acc, '73%', '58%', 'Postal_Code, Speeding_Violations, Past_
metrics_df = pd.DataFrame(metrics_data, columns=['Model', 'Accuracy', 'Precision', 'Recall', 'Significant Var
metrics_df = metrics_df.sort_values("Accuracy", ascending=False)
metrics_df
```

Out[54]:

	Model	Accuracy	Precision	Recall	Significant Variables
4	Neural Network Model	0.7980	73%	58%	Postal_Code, Speeding_Violations, Past_Accidents
3	Decision Tree Model	0.7735	63%	69%	Past_Accidents, Vehicle_Ownership, Speeding_Vi...
1	KNN Model	0.7725	68%	10%	None
2	SVM Model	0.7615	60%	75%	Past_Accidents, Vehicle_Ownership, Gender
0	Logistic Regression Model	0.7610	60%	74%	Gender, Income, Vehicle Type

Table 15. Model Comparison Table

Task 4: Build a confusion matrix for each model; discuss which part of the confusion matrix a company would want to reduce and which model does the best at doing so

In [55]:
```
def print_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    tn, fp, fn, tp = cm.ravel()
    print(f"\nConfusion Matrix for {model_name}:")
    print(cm)

# Print confusion matrices
print_confusion_matrix(y_test, y_pred1, "Logistic Regression")
print_confusion_matrix(y_test, y_pred2, "KNN model")
print_confusion_matrix(y_test, y_pred3, "SVM model")
print_confusion_matrix(y_test, y_pred4, "Decision Tree Classifier")
print_confusion_matrix(y_test, y_pred5, "Neural Network model")
```

```
Confusion Matrix for Logistic Regression:
[[1051  316]
 [ 162  471]]

Confusion Matrix for KNN model:
[[1337   30]
 [ 569   64]]

Confusion Matrix for SVM model:
[[1049  318]
 [ 159  474]]

Confusion Matrix for Decision Tree Classifier:
[[1108  259]
 [ 194  439]]

Confusion Matrix for Neural Network model:
[[1232  135]
 [ 269  364]]
```

- The above code represents the confusion matrix for all the models built for the classification of the outcome variable of the insurance dataset.
- The part of the confusion matrix that a company should focus on and would want to reduce are the false positives and false negative values which are often of high importance.
- This is because these values are classified into wrong classes and it will cause errors while analyzing the data for the decision making process.
- The model which does the best at doing so, i.e, classifying less number of classes as false positives and false negatives is KNN model and Neural Network model as it classifies majority of the classes into their right category.
- Also, Decision Tree model classifies majority of the outcome class into its right category, reducing the number of false positive and false negative values.

Results

Task 5: In two paragraphs minimum, discuss the features that were important and significant from the models. Use that to provide a specific recommendation to the insurance company of what they need to look out for when someone applies for car insurance with them and how they can reduce that risk when someone uses their insurance.

1. Discussing the models to be recommended based on the evaluation metrics

Based on the evaluation metrics as shown in the table above, it is observed that **Neural Network and Decision Tree model has performed the best** as compared to the other models for the classification of the outcome variable. This is because the accuracy and precision - recall score for the Neural Network and Decision Tree model is higher as compared to the other models and **hence the model that would be recommended based on the evaluation metrics is the Neural Network model.**

2. Discussing the key variables they should focus on their business context

The key variables based on the model selected which is the Neural Network model are **'Postal_Code', 'Speeding_Violations', 'Past_Accidents'.** The feature importance score for the Neural Network model for the three variables is highest and hence they are the key variables that the company should focus on that will help them look out for when someone applies for car insurance and also will help them analyze and reduce the risk when someone uses their insurance.

Conclusion

Recommendations

- Based on the analysis and results, we conclude that **Neural Network model** is recommended to be implemented for the classification of the outcome variable The Neural Network model is neither overfitted nor underfitted based on the accuracy values of the train and test dataset, but the performance of the model can be improved such that it can be used in future for further prediction and classification, which can be done by updating new features and data points that will increase the efficiency and performance of the model, implying a *best-fit model for training*.
- The features that should be focused upon are **'Postal_Code', 'Speeding_Violations', 'Past_Accidents'** based on the feature importance score of the Neural Network model. Hence, the company should focus on these features for analyzing the outcome parameter and to understand the risk when someone uses their insurance, as these features influence the classification of the target variable in the dataset.

Future Scope

The Neural Network model selected is able to classify the target variable and also extract the features that contribute to the prediction, but the performance and efficiency of the model can be improved and thus requires reevaluating the performance of the model by adding new features to the model and increasing the training data. Based on the results, it is observed that the dataset has some amount of null values and thus it is important that both the quantity and quality of data is improved to increase the efficiency of the model.

Thus, the model can be improved and updated based on new features being added to the dataset that can help to better analyze the target variable.

References

[1] What is Logistic regression? | IBM. (n.d.). https://www.ibm.com/topics/logistic-regression (https://www.ibm.com/topics/logistic-regression)

[2] KNN Algorithm - Finding Nearest Neighbors. (n.d.). https://www.tutorialspoint.com/machine_learning_with_python/knn_algorithm_finding_nearest_neighbors.htm (https://www.tutorialspoint.com/machine_learning_with_python/knn_algorithm_finding_nearest_neighbors.htm)

[3] Gandhi, R. (2022, November 14). Support Vector Machine — Introduction to Machine Learning Algorithms. Medium. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47)

[4] GeeksforGeeks. (2023). Decision Tree. GeeksforGeeks. https://www.geeksforgeeks.org/decision-tree/ (https://www.geeksforgeeks.org/decision-tree/)

[5] Knocklein, O. (2021, December 10). Classification Using Neural Networks - Towards Data Science. Medium. https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f (https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f)

[6] Singh, K. (2022). How to Improve Class Imbalance using Class Weights in Machine Learning. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/ (https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/)

[7] Brownlee, J. (2020b). Cost-Sensitive SVM for Imbalanced Classification. MachineLearningMastery.com. https://machinelearningmastery.com/cost-sensitive-svm-for-imbalanced-classification/#:~:text=Perhaps%20the%20simplest%20and%20most,level%20weighted%20modification%20was%20proposed (https://machinelearningmastery.com/cost-sensitive-svm-for-imbalanced-classification/#:~:text=Perhaps%20the%20simplest%20and%20most,level%20weighted%20modification%20was%20proposed).