CAR INSURANCE DATA

Final Report

Group number : 5
Batch : PGPDSE-FT online Feb22

Team members:

Mitra Karthikeyan
Mohammad Umar Sharieff.M
Manibalan .N
Lalith .S

Table of contents:

Acknowledgement

Firstly, I would like to express my sincere thanks to my mentor Mr Jatinder Bedi for helping and guiding me throughout the entire duration of this project without which it would have been impossible to deliver proper and accurate results within the stipulated time period. Moreover, I would like to express my special thanks to Great Learning who gave me this golden opportunity to learn so many interesting concepts and apply them on such an important topic as a capstone project. This journey has made me more inquisitive about the never-ending possibilities with Machine learning and Data science in the near future and has helped me to successfully identify the path which I would like to continue as my profession in this field itself. Last but not the least, I would like to extend my warm wishes and thanks to all my fellow teammates for their commitments, encouragements, initiative and continued support during the course of this Project.

Industrial Review

Machine learning is popular because there is an abundance of data to learn from today and luckily computation is abundant and cheap today.

Machine learning is the study of making decisions under uncertainty given a training dataset, how should I act when I see something new – and trust me this technology is the new black. But it is truly intimidating to explore these domains, especially if you are a newbie. The reason being, there is no fool proof roadmap to master AI or be a skilled data scientist, the skills and tools needed are dynamic.

The mentioned methods in this research can be utilized in other supply chain cases to forecast The use of machine learning will provide flexibility to the company's decision makers which would result in a better and smooth supply chain process. To deal with diverse characteristics of data, this article aims at using ranged methods for specifying different levels of predicting features. This range is tuneable and will give flexibility to the decision-making authority. Since it is the decision-making problem Decision tree based approach can be used in this. This would make the model interpretable without having expert knowledge.

Tree based machine learning algorithm is chosen which includes Random Forest and Gradient Boosting. With the use of ranged methods approach for imbalanced dataset, the performance of machine learning has improved by 20%. The data contains monthly, quarterly, half yearly sales and forecast sales information, inventory level and flag-based information. The use of ensemble techniques provided much better results when precision and recall were taken as the performance metrics.

The solution then moves on to the EDA which includes making observations based Univariate and Bivariate Analysis of features. As a part of Data processing missing values are fixed using

Imputation technique like Simple Imputer and Miss Forest Imputation. Data being highly imbalanced, oversampling techniques like Random Oversample and SMOTE have been used.

Literature review

Prediction using machine learning algorithms is not well adapted in many parts of the business decision processes due to the lack of clarity and flexibility. The erroneous data as inputs in the prediction process may produce inaccurate predictions. We aim to use machine learning models in the area of the business decision process by predicting products' backorder while providing flexibility to the decision authority, better clarity of the process, and maintaining higher accuracy. A ranged method is used for specifying different levels of predicting features to cope with the diverse characteristics of real-time data which may happen by machine or human errors.

The range is tuneable that gives flexibility to the decision managers. The tree-based machine learning is chosen for better explain ability of the model. The backorders of products are predicted in this study using Distributed Random Forest (DRF) and Gradient Boosting Machine (GBM). We have observed that the performances of the machine learning models have been improved by 20% using this ranged approach when the dataset is highly biased with random error. We have utilized a five-level metric to indicate the inventory level, sales level, forecasted sales level, and a four-level metric for the lead time. A decision tree from one of the constructed models is analysed to understand the effects of the ranged approach. As a part of this analysis, we list major probable backorder scenarios to facilitate business decisions. The mentioned methods in this research can be utilized in other supply chain cases to forecast backorders.

Dataset:

https://www.kaggle.com/datasets/sagnik1511/car-insurance-data

In [3]:
```
data.head()
```
Out[3]:

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE	VEHICLE_OWNERSHIP	VEHICLE_YEAR	MARRIED	CHILDR
0	569520	65+	female	majority	0-9y	high school	upper class	0.629027	owns vehicle	after 2015	unmarried	has ch
1	750365	16-25	male	majority	0-9y	none	poverty	0.357757	no vehicle	before 2015	unmarried	no ch
2	199901	16-25	female	majority	0-9y	high school	working class	0.493146	owns vehicle	before 2015	unmarried	no ch
3	478866	16-25	male	majority	0-9y	university	working class	0.206013	owns vehicle	before 2015	unmarried	has ch
4	731664	26-39	male	majority	10-19y	none	working class	0.388366	owns vehicle	before 2015	unmarried	no ch

In [6]:
```
data.shape
```
Out[6]: (10000, 19)

In [4]: data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 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   RACE                 10000 non-null  object 
 4   DRIVING_EXPERIENCE   10000 non-null  object 
 5   EDUCATION            10000 non-null  object 
 6   INCOME               10000 non-null  object 
 7   CREDIT_SCORE         9018 non-null   float64
 8   VEHICLE_OWNERSHIP    10000 non-null  object 
 9   VEHICLE_YEAR         10000 non-null  object 
 10  MARRIED              10000 non-null  object 
 11  CHILDREN             10000 non-null  object 
 12  POSTAL_CODE          10000 non-null  int64  
 13  ANNUAL_MILEAGE       9043 non-null   float64
 14  VEHICLE_TYPE         10000 non-null  object 
 15  SPEEDING_VIOLATIONS  10000 non-null  int64  
 16  DUIS                 10000 non-null  int64  
 17  PAST_ACCIDENTS       10000 non-null  int64  
 18  OUTCOME              10000 non-null  object 
dtypes: float64(2), int64(5), object(12)
memory usage: 1.4+ MB
```

OverView of the dataset:

- It contains 10000 records and 19 fields.
- That contains 11 numerical columns and 8 categorical features.

Usage of the .describe() function.

In [5]: data.describe()

Out[5]:

	ID	CREDIT_SCORE	POSTAL_CODE	ANNUAL_MILEAGE	SPEEDING_VIOLATIONS	DUIS	PAST_ACCIDENTS
count	10000.000000	9018.000000	10000.000000	9043.000000	10000.000000	10000.000000	10000.000000
mean	500521.906800	0.515813	19864.548400	11697.003207	1.482900	0.239200	1.056300
std	290030.768758	0.137688	18915.613855	2818.434528	2.241966	0.554990	1.652454
min	101.000000	0.053358	10238.000000	2000.000000	0.000000	0.000000	0.000000
25%	249638.500000	0.417191	10238.000000	10000.000000	0.000000	0.000000	0.000000
50%	501777.000000	0.525033	10238.000000	12000.000000	0.000000	0.000000	0.000000
75%	753974.500000	0.618312	32765.000000	14000.000000	2.000000	0.000000	2.000000
max	999976.000000	0.960819	92101.000000	22000.000000	22.000000	6.000000	15.000000

- Describe function returns the statistical summary of the dataframe or series. This includes count, mean, median (or 50th percentile) standard variation, min-max, and percentile values of columns. It helps to grasp a quick statistical overview of the dataset provided.

Views:

- The data is not negative or inconsistent, thus saves time in data cleaning.
- Almost all the features has it original data filled, and few with null values.
- There are a few features that are not required for our model we need to get rid of that.
- And workaround for feature DUIS, which have more null values.

Removing Insignificant variables:

- The "ID" feature has the unique feature for every record, therefore that won't help us for the classification model we're working on. Instead, we may temporarily remove the column from our dataset using the drop() function by specifying the column.

Finding the missing/null values:

- By adding the total number of null values to the length and multiplying that result by 100, we can calculate the proportion of features that have missing data. This enables us to calculate the percentage of values that are missing.

```
In [7]: data.drop('ID',axis=1,inplace=True)

In [8]: data.head()
```

Out[8]:

	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE	VEHICLE_OWNERSHIP	VEHICLE_YEAR	MARRIED	CHILDREN	POS
0	65+	female	majority	0-9y	high school	upper class	0.629027	owns vehicle	after 2015	unmarried	has child	
1	16-25	male	majority	0-9y	none	poverty	0.357757	no vehicle	before 2015	unmarried	no child	
2	16-25	female	majority	0-9y	high school	working class	0.493146	owns vehicle	before 2015	unmarried	no child	
3	16-25	male	majority	0-9y	university	working class	0.206013	owns vehicle	before 2015	unmarried	has child	
4	26-39	male	majority	10-19y	none	working class	0.388366	owns vehicle	before 2015	unmarried	no child	

In [13]:
```
missing_value_percentage = data. isnull(). sum() * 100 / len(data)
missing_value_percentage
```

Out[13]:
```
AGE                    0.000000
GENDER                 0.000000
RACE                   0.000000
DRIVING_EXPERIENCE     0.000000
EDUCATION              0.000000
INCOME                 0.000000
CREDIT_SCORE           9.820000
VEHICLE_OWNERSHIP      0.000000
VEHICLE_YEAR           0.000000
MARRIED                0.000000
CHILDREN               0.000000
POSTAL_CODE            0.000000
ANNUAL_MILEAGE         9.570000
VEHICLE_TYPE           0.000000
SPEEDING_VIOLATIONS    0.000000
DUIS                   0.000000
PAST_ACCIDENTS         0.000000
OUTCOME                0.000000
dtype: float64
```

- The function .isnull() helps us to find the null values present in the feature. And using the sum() function on it will help us to find the total missing values from the dataset.

In [14]:
```
data.isnull().sum()
```

Out[14]:
```
AGE                      0
GENDER                   0
RACE                     0
DRIVING_EXPERIENCE       0
EDUCATION                0
INCOME                   0
CREDIT_SCORE           982
VEHICLE_OWNERSHIP        0
VEHICLE_YEAR             0
MARRIED                  0
CHILDREN                 0
POSTAL_CODE              0
ANNUAL_MILEAGE         957
VEHICLE_TYPE             0
SPEEDING_VIOLATIONS      0
DUIS                     0
PAST_ACCIDENTS           0
OUTCOME                  0
dtype: int64
```

Treating the missing values:

Either the missing values can be dropped or it can be replaced with the mean or median values. (Droping is not suggested because the data might loss it's key feature elements from the record)

CREDIT_SCORE feature

In [15]:
```
data['CREDIT_SCORE'].mean()
```
Out[15]:
```
0.5158128096021279
```

In [16]:
```
data['CREDIT_SCORE'].median()
```
Out[16]:
```
0.5250327585000001
```

In [17]:
```
data['CREDIT_SCORE']=data['CREDIT_SCORE'].fillna(data['CREDIT_SCORE'].mean())
```

ANNUAL_MILEAGE feature

In [18]:
```
data['ANNUAL_MILEAGE'].mean()
```
Out[18]:
```
11697.003206900365
```

In [19]:
```
data['ANNUAL_MILEAGE'].median()
```
Out[19]:
```
12000.0
```

In [20]:
```
data['ANNUAL_MILEAGE']=data['ANNUAL_MILEAGE'].fillna(data['ANNUAL_MILEAGE'].mean())
```

In [21]:
```
data.isna().sum()
```
Out[21]:
```
AGE                    0
GENDER                 0
RACE                   0
DRIVING_EXPERIENCE     0
EDUCATION              0
INCOME                 0
CREDIT_SCORE           0
VEHICLE_OWNERSHIP      0
VEHICLE_YEAR           0
MARRIED                0
CHILDREN               0
POSTAL_CODE            0
ANNUAL_MILEAGE         0
VEHICLE_TYPE           0
SPEEDING_VIOLATIONS    0
DUIS                   0
PAST_ACCIDENTS         0
OUTCOME                0
dtype: int64
```

1. **Credit score feature.**
   The mean and mean values are almost similar we shall taken either of these are which one is lower to apply on to the null values.

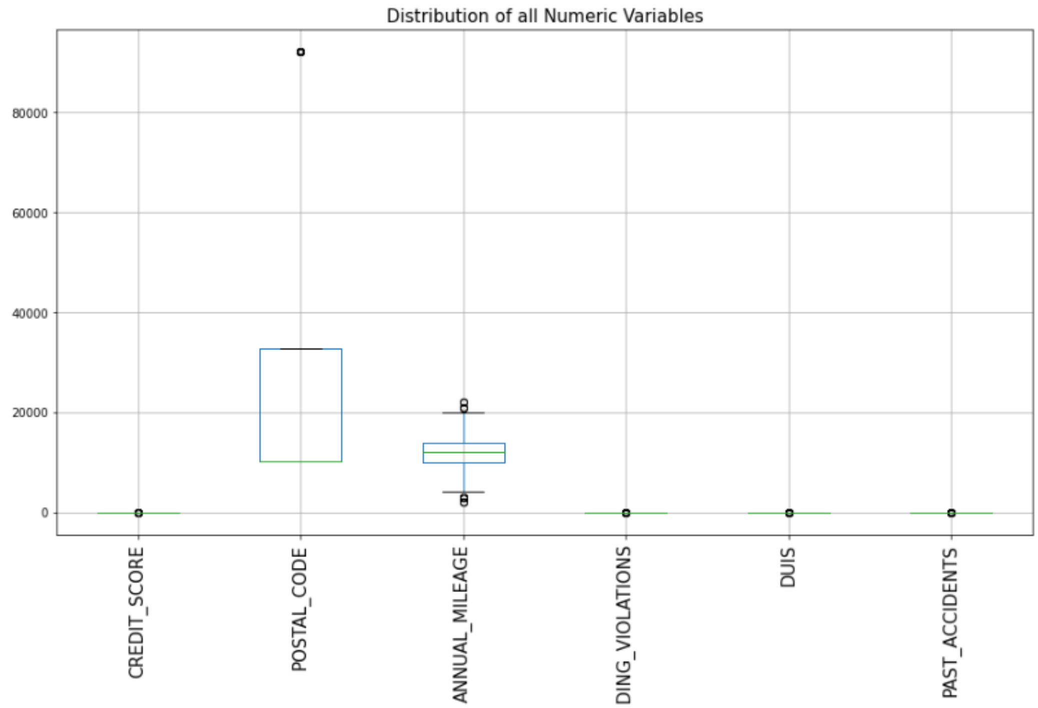
2. **Annual mileage.**
   Same here we are replacing the null values with the median.

Exploratory Data analysis (EDA) and Feature engineering.

Outliers detection:

1. **Distribution of all Numeric Variables.**

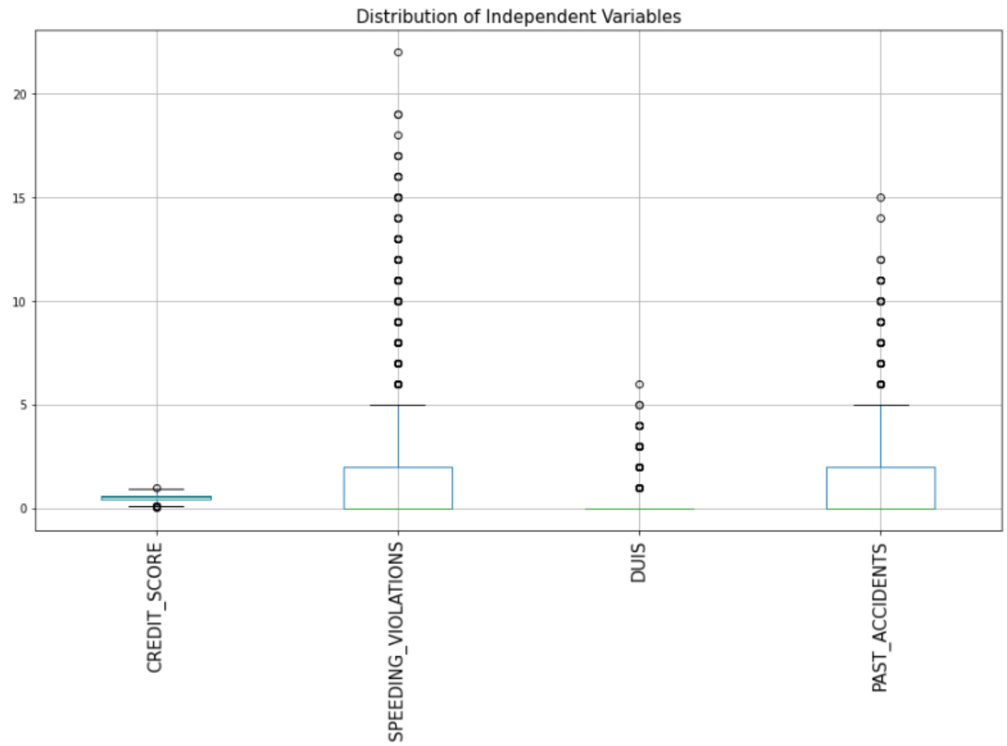
```
In [9]: plt.figure(figsize = (15,8))

        data.boxplot()

        plt.title('Distribution of all Numeric Variables', fontsize = 15)
        # xticks() returns the x-axis ticks
        plt.xticks(rotation = 'vertical', fontsize = 15)
        plt.show()
```



2. **Distribution of Independent Variables.**

With Independent variables we can manipulate, control, or vary in an experimental model.

In [11]:
```python
plt.figure(figsize = (15,8))
data.loc[:,['CREDIT_SCORE','SPEEDING_VIOLATIONS', 'DUIS','PAST_ACCIDENTS']].boxplot()


plt.title('Distribution of Independent Variables', fontsize = 15)


plt.xticks(rotation = 'vertical', fontsize = 15)


plt.show()
```

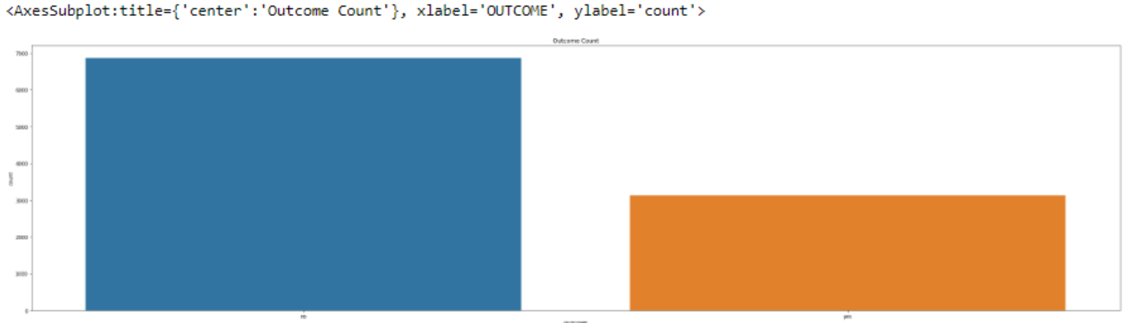


Univariate analysis:

Analyzing data with only one variable “OUTCOME”

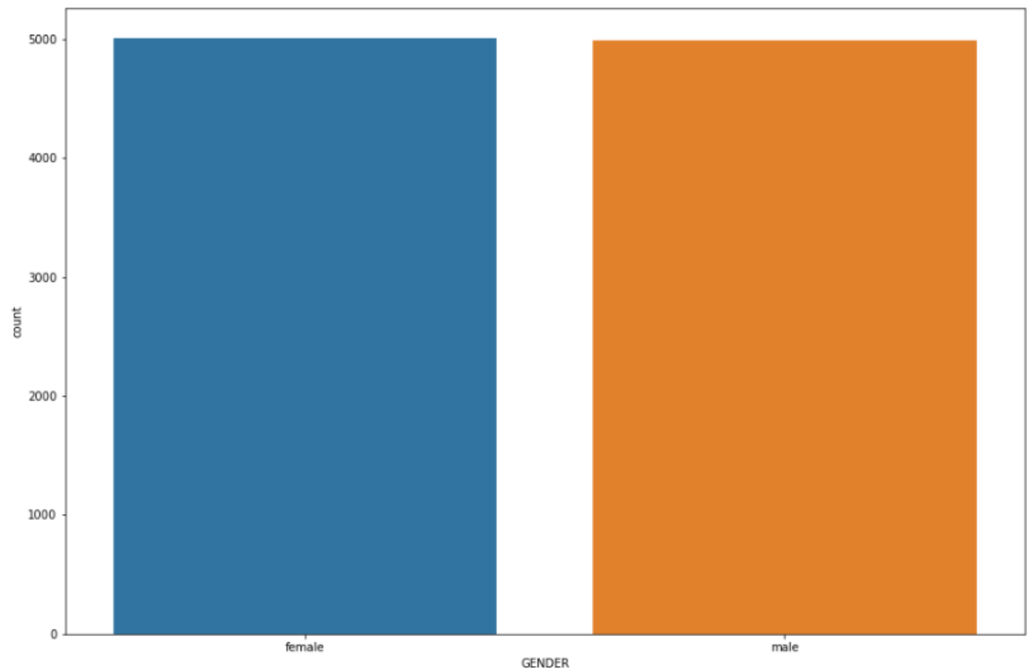
In [22]:
```python
plt.figure(figsize=(40,10))
plt.title('Outcome Count')
sns.countplot(data=data, x ='OUTCOME')
```

Out[22]: <AxesSubplot:title={'center':'Outcome Count'}, xlabel='OUTCOME', ylabel='count'>



Gender analysis:

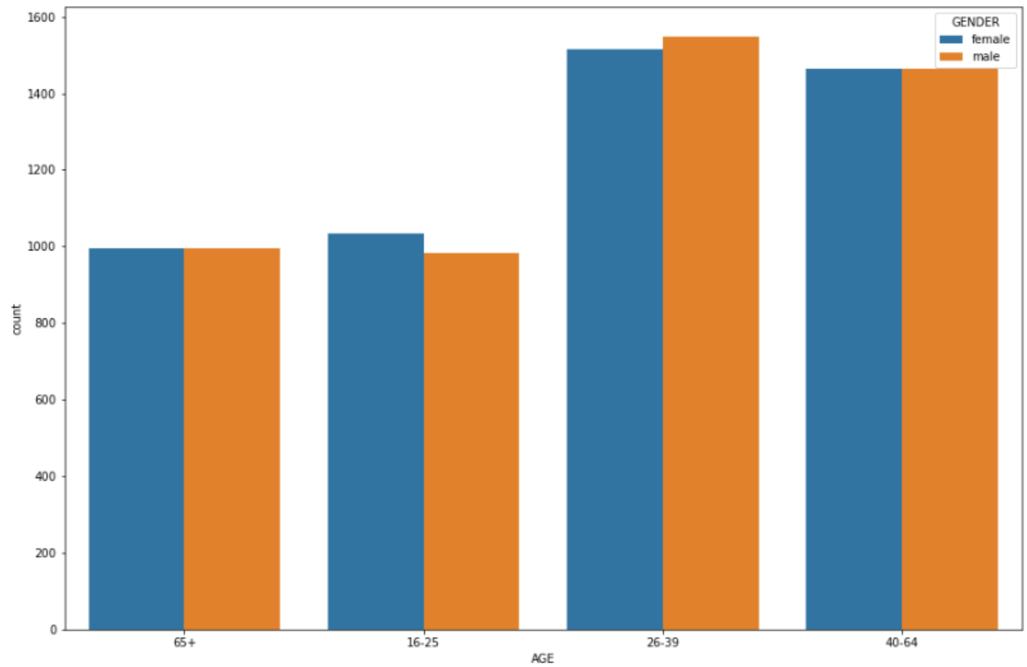
```
In [23]: sns.countplot(x = 'GENDER', data = data)
Out[23]: <AxesSubplot:xlabel='GENDER', ylabel='count'>
```



Inference: Both Male and Female are almost equally distributed.

Gender based on age:

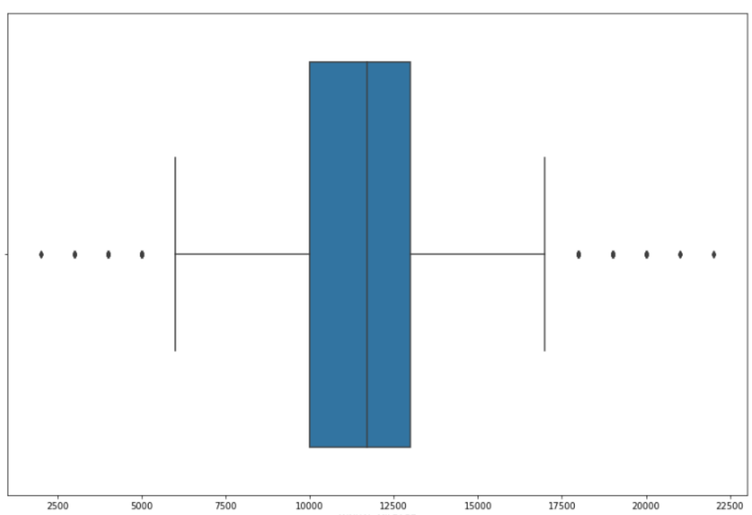
```
In [24]: sns.countplot(x= 'AGE', data= data, hue= 'GENDER')
Out[24]: <AxesSubplot:xlabel='AGE', ylabel='count'>
```



Inference: Most of the people from the data lies between 26-39 and 40-64.

Annual Mileage of the vehicle.

```
In [25]: sns.boxplot(data['ANNUAL_MILEAGE'])
Out[25]: <AxesSubplot:xlabel='ANNUAL_MILEAGE'>
```



Inference: The median lies between 100000 to 150000 miles.

People engagement for applying loan.

```
In [27]: dont_claim_loan = len(data[data.OUTCOME == 1])
         claim_loan = len(data[data.OUTCOME == 0])
         print("Percentage of people who apply for a loan: {:.2f}%".format((dont_claim_loan / (len(data.OUTCOME))*100)))
         print("Percentage of people who did not apply for a loan: {:.2f}%".format((claim_loan / (len(data.OUTCOME))*100)))

         Percentage of people who apply for a loan: 0.00%
         Percentage of people who did not apply for a loan: 0.00%
```

Percentage of people who apply for a loan: 0.00%
Percentage of people who did not apply for a loan: 0.00%

Find for any correlation.

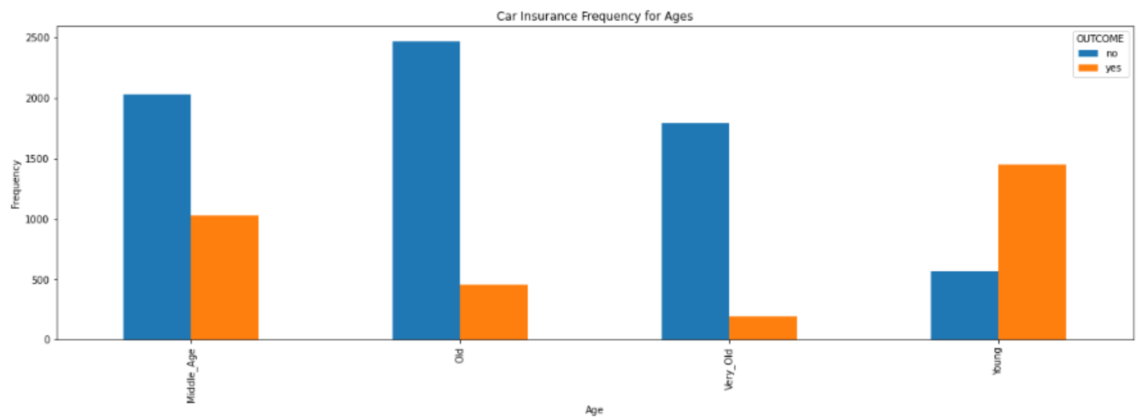
Annotating only with the top 5 variables the selected features.

```
In [28]: corr_matrix = data.corr()
         fig, ax = plt.subplots(figsize=(22, 10))
         ax = sns.heatmap(corr_matrix,annot=True,linewidths=0.5,fmt=".2f",cmap="YlGn");
         bottom, top = ax.get_ylim()
         ax.set_ylim(bottom + 0.5, top - 0.5)

Out[28]: (6.5, -0.5)
```

Car Insurance Frequency for Ages segregated by groups.

```
In [29]: data["AGE"].replace({"16-25": "Young", "26-39": "Middle_Age","40-64":"Old","65+":"Very_Old"}, inplace=True)
```

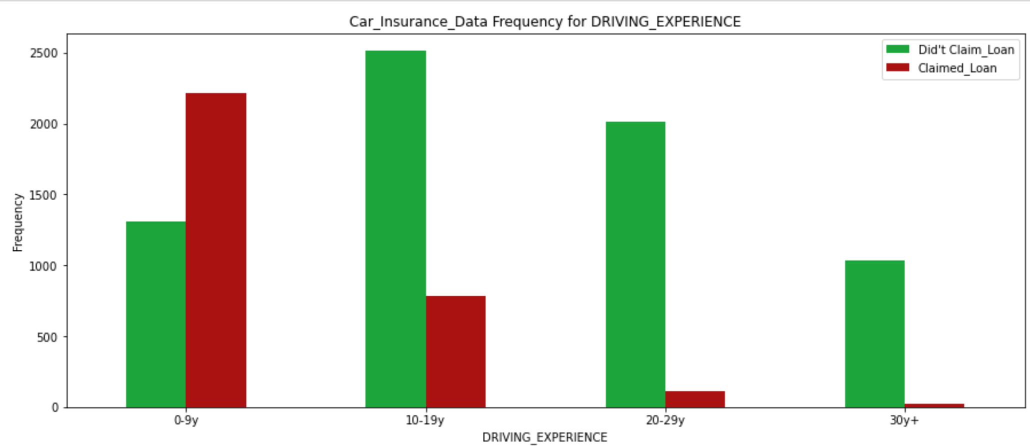
```
In [30]: pd.crosstab(data.AGE,data.OUTCOME).plot(kind="bar",figsize=(20,6))
         plt.title('Car Insurance Frequency for Ages')
         plt.xlabel('Age')
         plt.ylabel('Frequency')
         plt.show()
```



Car Insurance Data Frequency for DRIVING EXPERIENCE.

Analysis performed for people who claimed and did not claim the loan.

```
In [31]: pd.crosstab(data.DRIVING_EXPERIENCE,data.OUTCOME).plot(kind="bar",figsize=(15,6),color=['#1CA53B','#AA1111', '#FFA500' ])
         plt.title('Car_Insurance_Data Frequency for DRIVING_EXPERIENCE')
         plt.xlabel('DRIVING_EXPERIENCE')
         plt.xticks(rotation=0)
         plt.legend(["Did't Claim_Loan", "Claimed_Loan"])
         plt.ylabel('Frequency')
         plt.show()
```



Inference: People from the age of 0-9 and 10-19 have claimed the maximum insurance.

Feature engineering.

Adjusting the features will help the model to perform better when put across different algorithms. Segregation the numarical and categoerical variables fo the feature selection.

```
In [33]: #all feature without target var
         data_feature = data.drop('OUTCOME', axis = 1)

In [34]: #cat
         data_cat=data_feature.select_dtypes(include=np.object)

In [35]: data_cat.columns

Out[35]: Index(['AGE', 'GENDER', 'RACE', 'DRIVING_EXPERIENCE', 'EDUCATION', 'INCOME',
                'VEHICLE_OWNERSHIP', 'VEHICLE_YEAR', 'MARRIED', 'CHILDREN',
                'VEHICLE_TYPE'],
               dtype='object')

In [36]: #num
         data_numarical=data_feature.select_dtypes(include=np.number)

In [37]: data_numarical.columns

Out[37]: Index(['CREDIT_SCORE', 'POSTAL_CODE', 'ANNUAL_MILEAGE', 'SPEEDING_VIOLATIONS',
                'DUIS', 'PAST_ACCIDENTS'],
               dtype='object')
```

Verifying for the class imbalance of the data.

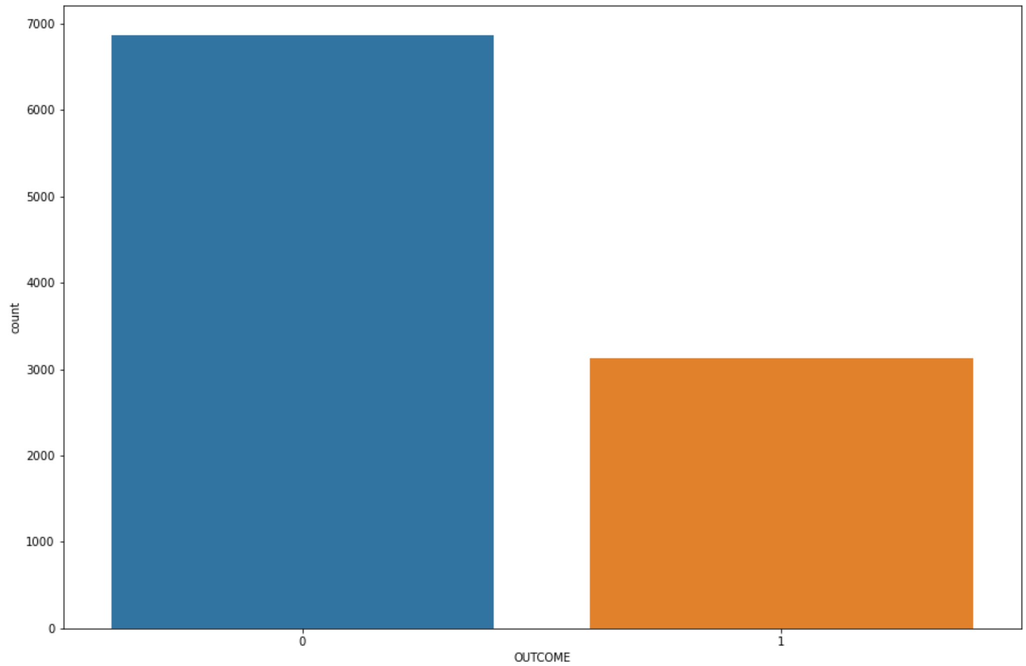
Observing the counts and info of target feature.

```
In [38]: data['OUTCOME'].value_counts()

Out[38]: no     6867
         yes    3133
         Name: OUTCOME, dtype: int64

In [39]: data['OUTCOME']=data['OUTCOME'].map({'no':0,'yes':1})

In [40]: #target var
         data_target=data['OUTCOME']
         data_target.info()

         <class 'pandas.core.series.Series'>
         RangeIndex: 10000 entries, 0 to 9999
         Series name: OUTCOME
         Non-Null Count  Dtype
         --------------  -----
         10000 non-null  int64
         dtypes: int64(1)
         memory usage: 78.2 KB
```

Count plot of Target feature.

```
In [71]: sns.countplot(data['OUTCOME'])
```

```
Out[71]: <AxesSubplot:xlabel='OUTCOME', ylabel='count'>
```



Encoding for categorical variables.

Performed to transform data so that it can be properly (and safely) consumed by a different type of algorithms.

```
In [41]: data_dummy_var= pd.get_dummies(data=data_cat,drop_first = True)
         data_dummy_var.head()
```

Out[41]:

	AGE_Old	AGE_Very_Old	AGE_Young	GENDER_male	RACE_minority	DRIVING_EXPERIENCE_10-19y	DRIVING_EXPERIENCE_20-29y	DRIVING_EXPERIENCE_30y+	E
0	0	1	0	0	0	0	0	0	
1	0	0	1	1	0	0	0	0	
2	0	0	1	0	0	0	0	0	
3	0	0	1	1	0	0	0	0	
4	0	0	0	1	0	1	0	0	

Scaling for the numerical variable.

Performed to prevent the model to work properly if the features have different orders of magnitude.

```
In [43]: X_scaler = StandardScaler()

         num_scaled = X_scaler.fit_transform(data_numarical)

         data_numarical_scaled_var = pd.DataFrame(num_scaled, columns = data_numarical.columns)

In [44]: data_numarical_scaled_var.shape

Out[44]: (10000, 6)
```

Concatenate scaled numerical and dummy encoded categorical variables.

Done in order to perform the train test split on the whole of the dataset.

```
In [45]: X = pd.concat([data_numarical_scaled_var, data_dummy_var],axis=1)

         X.head()

Out[45]:
```

	CREDIT_SCORE	POSTAL_CODE	ANNUAL_MILEAGE	SPEEDING_VIOLATIONS	DUIS	PAST_ACCIDENTS	AGE_Old	AGE_Very_Old	AGE_Young	GENDER
0	0.865914	-0.508946	0.113057	-0.661462	-0.431020	-0.639263	0	1	0	
1	-1.208879	-0.508946	1.605576	-0.661462	-0.431020	-0.639263	0	0	1	
2	-0.173367	-0.508946	-0.260073	-0.661462	-0.431020	-0.639263	0	0	1	
3	-2.369485	0.682034	-0.260073	-0.661462	-0.431020	-0.639263	0	0	1	
4	-0.974770	0.682034	0.113057	0.230657	-0.431020	-0.034072	0	0	0	

```
In [46]: X.shape

Out[46]: (10000, 24)
```

Performing Train Test Split.

Done in order to estimate the performance of machine learning algorithms that are applicable for prediction-based Algorithms/Applications.

```
In [47]: X = sm.add_constant(X)

In [48]: X.shape

Out[48]: (10000, 25)

In [49]: X_train, X_test, Y_train, Y_test = train_test_split(X, data_target, random_state = 10, test_size = 0.2)

In [50]: print(X_train.shape)
         #X_test
         #Y_train
         #Y_test

         (8000, 25)
```

Here the split of 80-20 was performed in order to train the data on the large scale.

Handling the class imbalance for train data.

With the help of SMOTEENN from imblearn we have smoothen the trained data.

```
In [72]: from imblearn.combine import SMOTEENN

In [73]: smt=SMOTEENN()

In [74]: X_train,Y_train=smt.fit_resample(X_train,Y_train)

In [75]: Y_train.value_counts()

Out[75]: 1    4203
         0    3486
         Name: OUTCOME, dtype: int64
```

Model Building:

Base model - Logistic regression.

```
In [76]: logreg = sm.Logit(Y_train, X_train)
         result = logreg.fit()
         result.summary()

         Optimization terminated successfully.
                  Current function value: 0.148187
                  Iterations 10
```

Out[76]:

Logit Regression Results

Dep. Variable:	OUTCOME	No. Observations:	7689
Model:	Logit	Df Residuals:	7664
Method:	MLE	Df Model:	24
Date:	Tue, 08 Nov 2022	Pseudo R-squ.:	0.7849
Time:	16:19:54	Log-Likelihood:	-1139.4
converged:	True	LL-Null:	-5296.1
Covariance Type:	nonrobust	LLR p-value:	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	2.4519	0.257	9.558	0.000	1.949	2.955
CREDIT_SCORE	-0.2536	0.084	-3.029	0.002	-0.418	-0.089
POSTAL_CODE	1.0270	0.070	14.746	0.000	0.891	1.164
ANNUAL_MILEAGE	0.2298	0.074	3.102	0.002	0.085	0.375
SPEEDING_VIOLATIONS	0.5626	0.093	6.044	0.000	0.380	0.745
DUIS	-0.1873	0.075	-2.508	0.012	-0.334	-0.041
PAST_ACCIDENTS	-0.2959	0.099	-2.983	0.003	-0.490	-0.101
AGE_Old	-1.0510	0.175	-5.999	0.000	-1.394	-0.708
AGE_Very_Old	-0.4099	0.215	-1.910	0.056	-0.831	0.011
AGE_Young	0.0678	0.234	0.290	0.771	-0.390	0.526
GENDER_male	1.4997	0.125	12.025	0.000	1.255	1.744
RACE_minority	-0.8974	0.220	-4.088	0.000	-1.328	-0.467
DRIVING_EXPERIENCE_10-19y	-3.9922	0.192	-20.787	0.000	-4.369	-3.616
DRIVING_EXPERIENCE_20-29y	-7.2414	0.335	-21.618	0.000	-7.898	-6.585
DRIVING_EXPERIENCE_30y+	-10.4986	0.807	-13.010	0.000	-12.080	-8.917
EDUCATION_none	0.3108	0.170	1.832	0.067	-0.022	0.643
EDUCATION_university	-0.3962	0.130	-3.046	0.002	-0.651	-0.141
INCOME_poverty	-0.3745	0.238	-1.575	0.115	-0.840	0.091
INCOME_upper class	-0.2320	0.163	-1.422	0.155	-0.552	0.088
INCOME_working class	-0.3137	0.175	-1.797	0.072	-0.656	0.028
VEHICLE_OWNERSHIP_owns vehicle	-2.9382	0.142	-20.690	0.000	-3.217	-2.660
VEHICLE_YEAR_before 2015	3.0485	0.151	20.208	0.000	2.753	3.344
MARRIED_unmarried	0.6518	0.129	5.070	0.000	0.400	0.904
CHILDREN_no child	0.2856	0.135	2.114	0.035	0.021	0.550
VEHICLE_TYPE_sports car	-0.3921	0.308	-1.275	0.202	-0.995	0.211

Finding out the Significant features that has the p value of < 0.05 from the dataframe in order to perform the predictions.

In [77]:
```
significant_feat = pd.DataFrame()
significant_feat['Feature'] = X.columns
significant_feat['P_Value'] = result.pvalues.values
significant_feat[significant_feat['P_Value'] < 0.05]
```

Out[77]:

	Feature	P_Value
0	const	0.000000
1	CREDIT_SCORE	0.002456
2	POSTAL_CODE	0.000000
3	ANNUAL_MILEAGE	0.001924
4	SPEEDING_VIOLATIONS	0.000000
5	DUIS	0.012147
6	PAST_ACCIDENTS	0.002859
7	AGE_Old	0.000000
10	GENDER_male	0.000000
11	RACE_minority	0.000043
12	DRIVING_EXPERIENCE_10-19y	0.000000
13	DRIVING_EXPERIENCE_20-29y	0.000000
14	DRIVING_EXPERIENCE_30y+	0.000000
16	EDUCATION_university	0.002322
20	VEHICLE_OWNERSHIP_owns vehicle	0.000000
21	VEHICLE_YEAR_before 2015	0.000000
22	MARRIED_unmarried	0.000000
23	CHILDREN_no child	0.034533

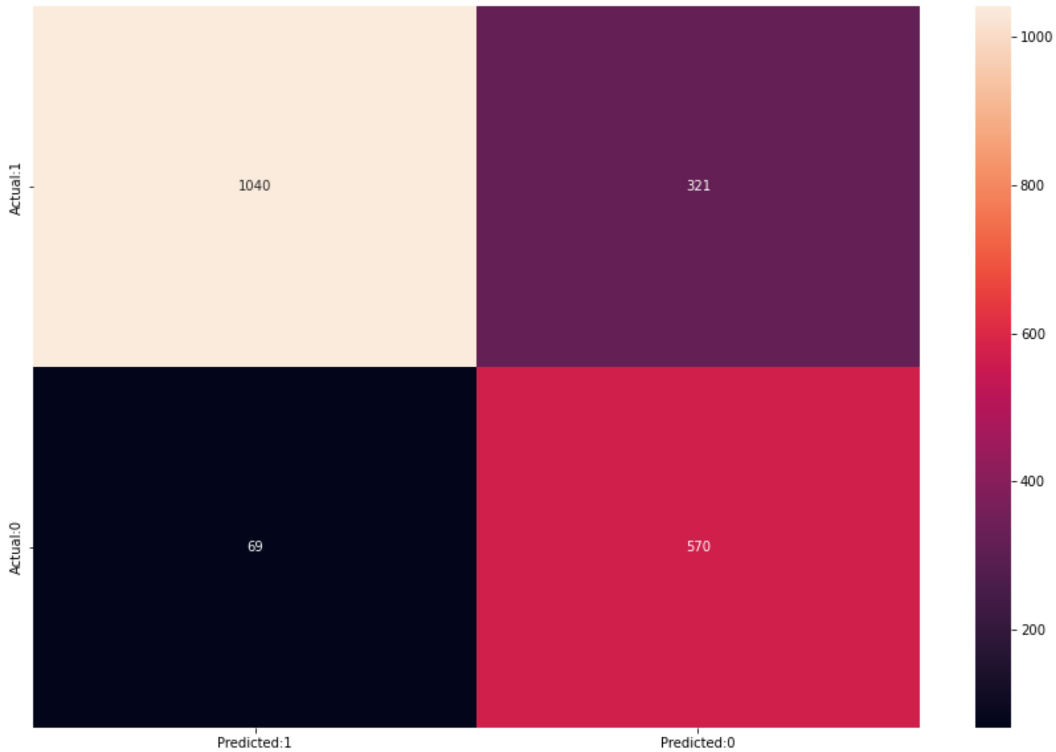
Predictions on the test set.

In [78]:
```
y_pred_prob = result.predict(X_test)

y_pred_prob.head()
```

Out[78]:
```
937     0.999281
9355    0.000046
2293    0.000094
192     0.347800
8675    0.993100
dtype: float64
```

Confusion Matrix on Logistic regression.

In [80]:
```
cm = confusion_matrix(Y_test, y_pred)

conf_mat=pd.DataFrame(data=cm,columns=['Predicted:1','Predicted:0'],index=['Actual:1','Actual:0'])
sns.heatmap(conf_mat,annot=True,fmt='d')

plt.show()
```



Compute various performance matrix on the predectied model.

Precision

In [82]:
```
precision = TP / (TP+FP)

precision
```

Out[82]: 0.6397306397306397

Recall

In [83]:
```
recall = TP / (TP+FN)
recall
```

Out[83]: 0.892018779342723

Specificity

In [84]:
```
specificity = TN / (TN+FP)
specificity
```

Out[84]: 0.7641440117560617

f1_score

In [85]:
```
f1_score = 2*((precision*recall)/(precision+recall))
f1_score
```

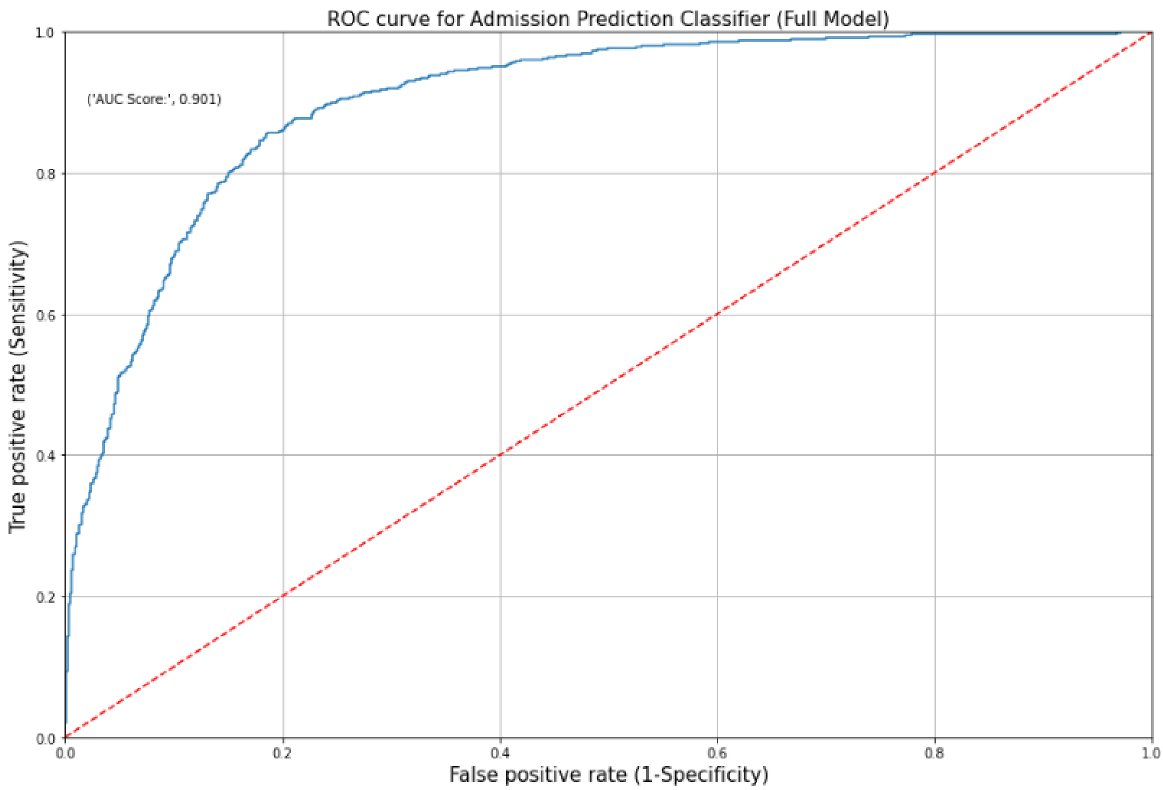
Out[85]: 0.7450980392156863

Accuracy:

In [86]:
```
accuracy = (TN+TP) / (TN+FP+FN+TP)

accuracy
```

Out[86]: 0.805

ROC curve for Admission Prediction Classifier (Full Model).



Train and test accuracy.

Precicting for the train set and calculating the accuracy

In [96]:
```
y_pred_lr_trn=clf_lr.predict(X_train)
print(f'Train Accuracy = {accuracy_score(Y_train,y_pred_lr_trn)}')
```

```
Train Accuracy = 0.9462869033684485
```

Precicting for the test set and calculating the accuracy

In [97]:
```
y_pred_lr_tes=clf_lr.predict(X_test)
print(f'Test Accuracy = {accuracy_score(Y_test,y_pred_lr_tes)}')
```

```
Test Accuracy = 0.802
```

Decision tree classifier.

The main advantage of the decision tree classifier is its ability to using different feature subsets and decision rules at different stages of classification.

```
In [106]: decision_tree_classification = DecisionTreeClassifier(criterion = 'entropy', random_state = 10)

          decision_tree = decision_tree_classification.fit(X_train, Y_train)
```

```
In [107]: model = tree.DecisionTreeClassifier()
          model = model.fit(X_train, Y_train)
          predicted_value = model.predict(X_test)
```

```
In [108]: DTC= (cross_val_score(estimator = model, X = X_train, y = Y_train, cv = 10).mean())
```

```
In [109]: print('Score:',DTC)

          Score: 0.9581221892609448
```

Model results after Hyperparameter tuning.

```
In [113]: y_pred_train = tree_cv.predict(X_train)
          print(metrics.classification_report(Y_train, y_pred_train))

                        precision    recall  f1-score   support

                     0       0.92      0.91      0.91      3486
                     1       0.93      0.93      0.93      4203

              accuracy                           0.92      7689
             macro avg       0.92      0.92      0.92      7689
          weighted avg       0.92      0.92      0.92      7689
```

```
In [114]: y_pred_test = tree_cv.predict(X_test)
          print(metrics.classification_report(Y_test, y_pred_test))

                        precision    recall  f1-score   support

                     0       0.93      0.74      0.82      1361
                     1       0.61      0.88      0.72       639

              accuracy                           0.78      2000
             macro avg       0.77      0.81      0.77      2000
          weighted avg       0.83      0.78      0.79      2000
```

Interpretation:

From the above output, we can see that there is slight significant difference between the train and test accuracy; thus, we can conclude that the decision tree is less over-fiited after specifying some of the hyperparameters.

Random Forest for Classification.

Performed to identify large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

```
In [130]: rf_classification = RandomForestClassifier(n_estimators = 10, random_state = 10)

          rf_model = rf_classification.fit(X_train, Y_train)
          RFC= (cross_val_score(estimator = rf_model, X = X_train, y = Y_train, cv = 10).mean())
          print('Score:',RFC)

          Score: 0.9719083563610749
```

Performance measures on the train set.¶ ¶

```
In [131]: train_report = get_train_report(rf_model)

          print(train_report)

                        precision    recall  f1-score   support

                     0       1.00      1.00      1.00      3486
                     1       1.00      1.00      1.00      4203

              accuracy                           1.00      7689
             macro avg       1.00      1.00      1.00      7689
          weighted avg       1.00      1.00      1.00      7689
```

```
In [132]: test_report = get_test_report(rf_model)

          print(test_report)

                        precision    recall  f1-score   support

                     0       0.91      0.80      0.85      1361
                     1       0.66      0.84      0.74       639

              accuracy                           0.81      2000
             macro avg       0.79      0.82      0.79      2000
          weighted avg       0.83      0.81      0.81      2000
```

Interpretation:

From the above output, we can see that there is a difference between the train and test accuracy; thus, we can conclude that the Random Forest for Classification is over-fitted on the train data.

If we tune the hyperparameters in the Random Forest for Classification, it helps to avoid the over-fitting of the Random Forest for Classification.

XGBClassifier.

Performed in order to determine,

- **Numeric features should be scaled**
- **Categorical features should be encoded**

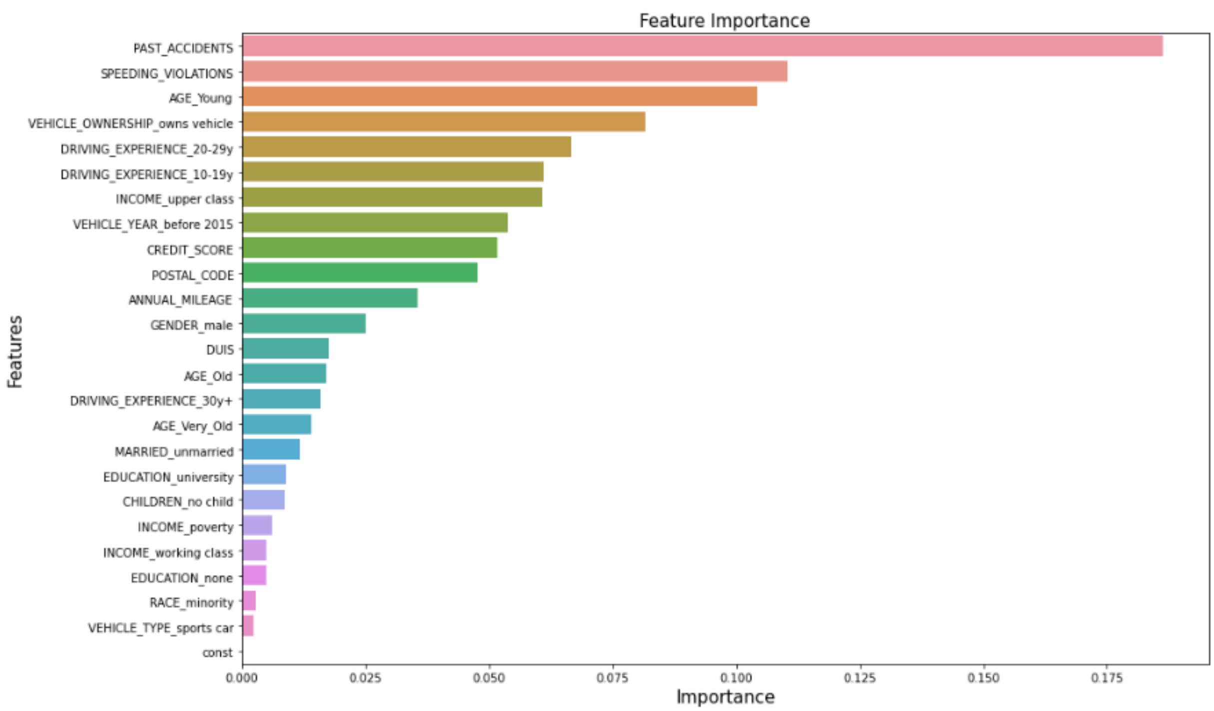
In [138]:
```python
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train, Y_train)
y_prd_xgb = xgb.predict(X_test)
XGB = (cross_val_score(estimator = xgb, X = X_train, y = Y_train, cv = 10).mean())
print(XGB)
```

```
0.976330868010403
```

In [142]:
```python
y_pred_test_XGB = xgb.predict(X_test)
print(metrics.classification_report(Y_test, y_pred_test_XGB))
```

```
              precision    recall  f1-score   support

           0       0.92      0.81      0.86      1361
           1       0.67      0.85      0.75       639

    accuracy                           0.82      2000
   macro avg       0.80      0.83      0.81      2000
weighted avg       0.84      0.82      0.82      2000
```

Feature importance chart on XGBoost.



Accuracy of all the models performed/calculated and their scores.

Out[137]:

	Models	Score
3	XGBClassifier	0.976331
0	Random Forest Classifier	0.971908
1	Decision Tree Classifier	0.958122
2	Logistic Model	0.944984

Interpretation:

So the best performing model is the **XGBClassifier**

Receiver Operating Characteristic XGBOOST.