Data Science Project – Predictive Modeling

PUNIT KUMAR

Problem Statement

- Data provided is from an insurance company about profiles of individuals, who it had approached earlier for selling caravan insurance policies.

- Expectation is to build a model for predicting whether any specified profile will be likely to buy a caravan insurance policy.

Solution Methods and Steps

1. Data Loading and Cleaning
2. Exploratory Data Analysis (EDA)
3. Multicollinearity Check and Feature Elimination
4. One-Hot Encoding for Categorical Variables
5. Over-sampling using SMOTE
6. Normalization
7. Recursive Feature Elimination
8. Feature Selection using P-Value
9. Logistic Regression Model Fitting
10. Conclusion

Step 1: Data Loading and Cleaning

- Data was **loaded** from the Training Excel file into Jupyter Notebook
- Required Machine Learning **python libraries** were imported and loaded
- Unwanted columns with **garbage data values** were **dropped** from the dataset
- **Null Values** checked in the dataset; and there were none
- Retained **5** Categorical Class columns, and **dropped** their corresponding descriptive columns

Importing Libraries

In [43]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from fast_ml.model_development import train_valid_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
import statsmodels.api as sm
from sklearn.feature_selection import RFE
from imblearn.over_sampling import SMOTE
%matplotlib inline
```

Loading Data

In [2]:
```
df1 = pd.read_excel("Training Set _5822 records 86 attributes including Class.xlsx")
```

In [7]:
```
df1[df1.isnull().any(axis=1)]
```

Step 2: Exploratory Data Analysis (EDA)

Loaded Clean Dataset (Rows: 5,822 Columns: 86)

- Categorical Data (Rows: 5,822 Columns: 5)
- Continuous Data (Rows: 5,822 Columns: 80)
- Output Variable (Rows: 5,822 Columns: 1)

Unique Counts of values in **Categorical Variables**

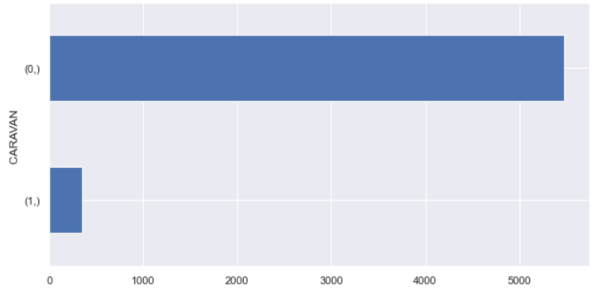
```
Out[10]: MOSTYPE  Description     40
         MGEMLEEF  Description     6
         MOSHOOFD Description     10
         MGODRK Description       10
         PWAPART Description       4
         dtype: int64
```

Statistical information of top 5 **Continuous variables**

	count	mean	std	min	25%	50%	75%	max
MAANTHUI	5822.0	1.110615	0.405842	1.0	1.0	1.0	1.0	10.0
MGEMOMV	5822.0	2.678805	0.789835	1.0	2.0	3.0	3.0	5.0
MGODPR	5822.0	4.626932	1.715843	0.0	4.0	5.0	6.0	9.0
MGODOV	5822.0	1.069907	1.017503	0.0	0.0	1.0	2.0	5.0
MGODGE	5822.0	3.258502	1.597647	0.0	2.0	3.0	4.0	9.0

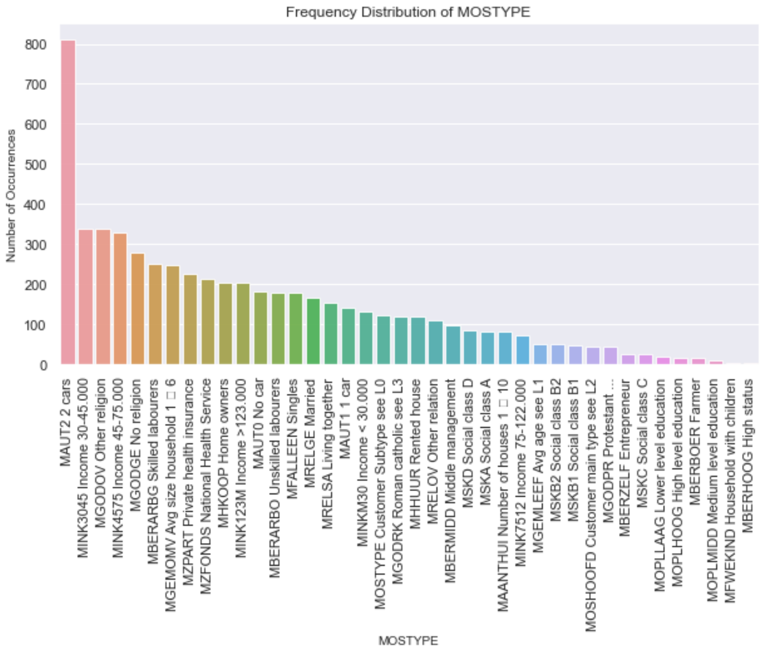
...

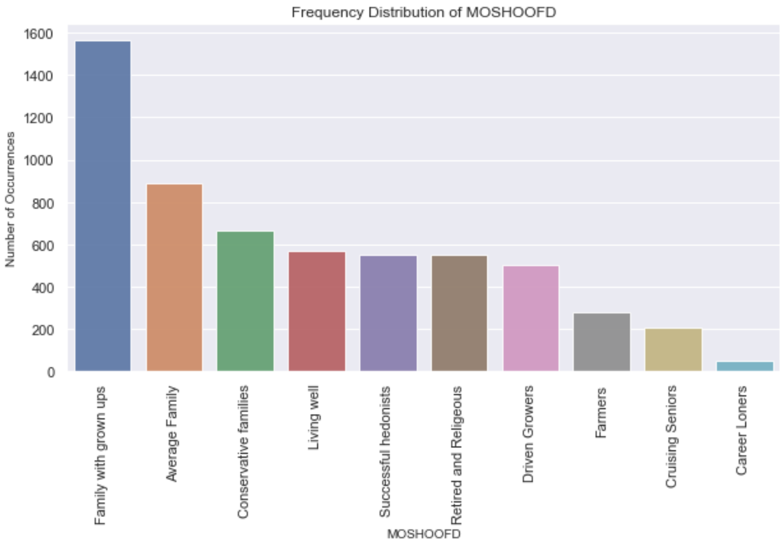
Frequency distribution of **Output variable**



Step 2: Exploratory Data Analysis (EDA)

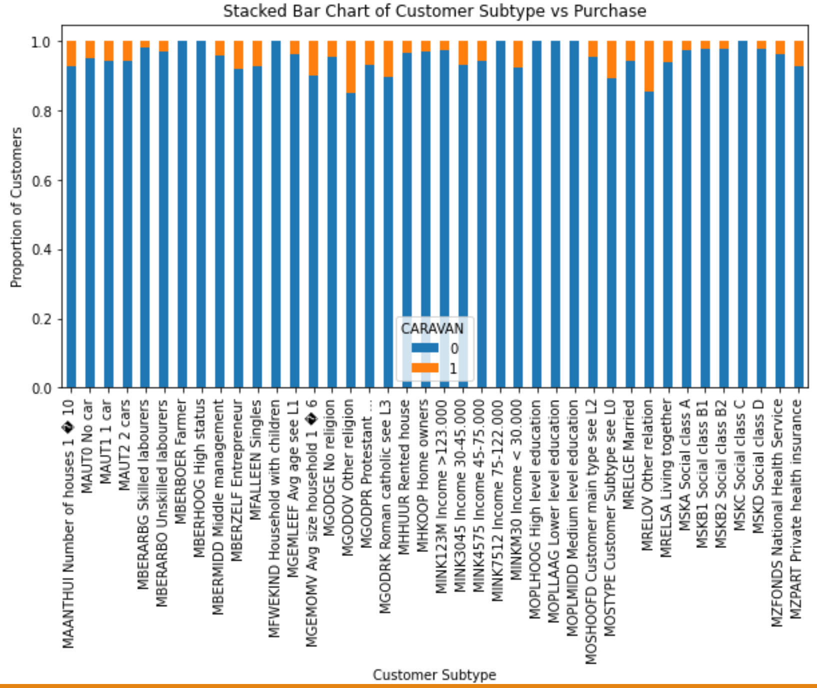
Frequency Distribution of top **2** categorical variables based on their unique value count

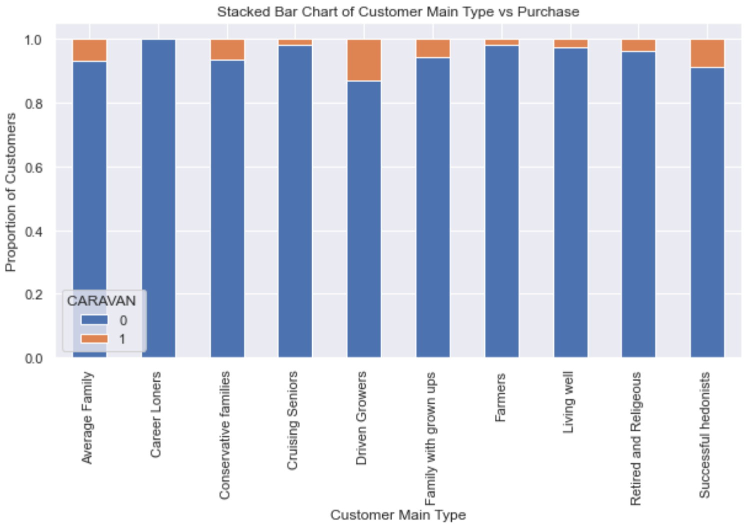




Step 2: Exploratory Data Analysis (EDA)

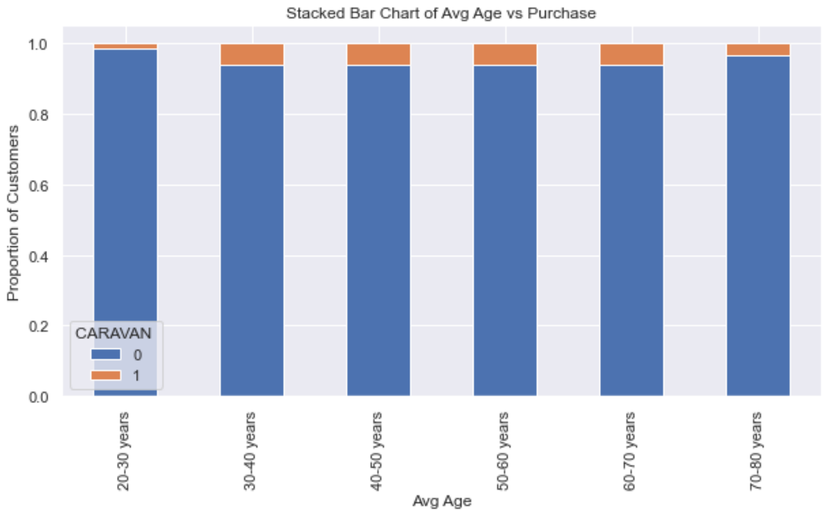
Both Customer Type & Sub Type look **strong predictor** for the output variable by looking at the changing proportions amongst their values

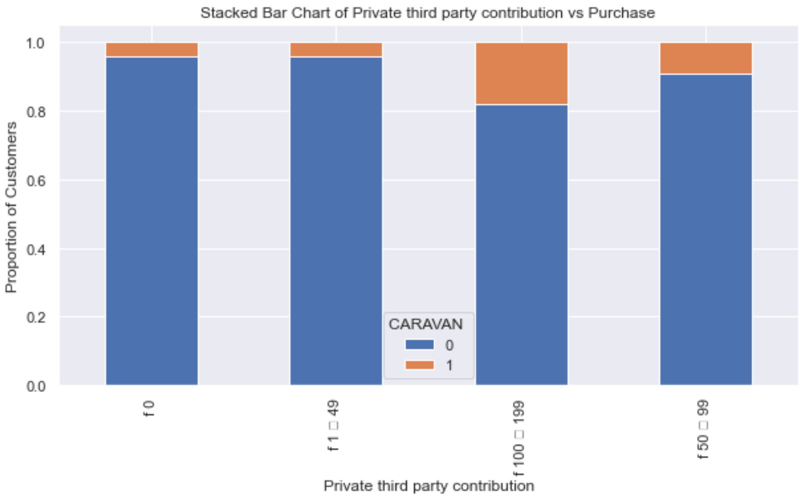




Step 2: Exploratory Data Analysis (EDA)

Private 3rd Party contribution seems **better predictor** as compared to Average Age





Step 3: Multicollinearity Check and Feature Elimination

- **Multicollinearity** happens when one predictor variable in a multiple regression model can be linearly predicted from the others with a high degree of accuracy.
- This can lead to **skewed** or misleading results. Logistic Regression or Linear Regression are not immune to that problem and we should fix it before training the model.
- We checked for multicollinearity using **Correlation matrix** on the **continuous variables.**
- We compared the **correlation** between features and removed one of two features that have a correlation **higher** than **0.9**
- We **eliminated** a total of **14** features through this method and are remained with **66** continuous variables now.

Heat Map of Correlation Matrix of first 10 continuous variables

Step 4: One-Hot Encoding for Categorical Variables

- **Label encoding** is already done for the Categorical variables in the dataset i.e, a number is given to each category of the field.
- However, the numerical values can be misinterpreted by the algorithm. A category tagged as 5 can be given higher weightage than a category tagged as 2.
- To solve this issue there is a popular way to encode the categories via something called **one-hot encoding**.
- The basic strategy is to **convert** each category value into a new column and assign a **1 or 0** (True/False) value to the column. This has the benefit of **not weighting** a value improperly.
- There are many libraries out there that support one-hot encoding but the simplest one is using 'pandas' **.get_dummies()** method.

Create dummy variables

In [109]:
```
for col in categorical_features:
    dummies = pd.get_dummies(df_final[col],prefix=col)
    df_final = pd.concat([df_final,dummies], axis =1)
    df_final.drop(col, axis=1, inplace = True)
```

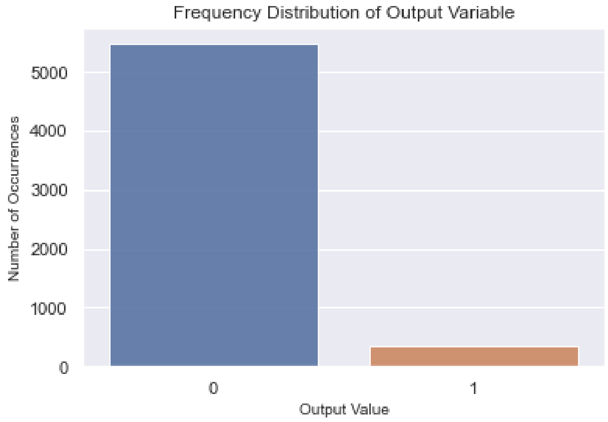
In [150]:
```
df_final.head()
```

Out[150]:

E	MRELSA	MRELOV	MFALLEEN	MFGEKIND	...	MGODRK_4	MGODRK_5	MGODRK_6	MGODRK_7	MGODRK_8	MGODRK_9	PWAPART_0	PWAPART_1
7	0	2	1	2	...	0	0	0	0	0	0	1	0
6	2	2	0	4	...	0	0	0	0	0	0	0	0
3	2	4	4	4	...	0	0	0	0	0	0	0	0
5	2	2	2	3	...	0	0	0	0	0	0	1	0
7	1	2	2	4	...	0	0	0	0	0	0	1	0

Step 5: Over-sampling using SMOTE

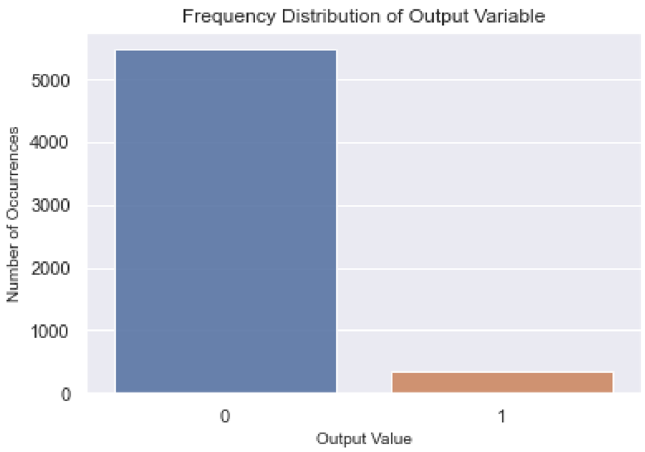
- After looking at the frequency distribution of the Output variable, we can see that our classes are **imbalanced**, and the ratio of no-subscription to subscription instances is **94:6**
- We need to **up-sample** the no-subscription using the **SMOTE** algorithm(**Synthetic Minority Oversampling Technique**)
- SMOTE Works by creating **synthetic** samples from the minor class (no-subscription) instead of creating copies.
- It randomly chooses one of the **k-nearest-neighbors** and using it to create a similar, but randomly tweaked, new observations.



percentage of no subscription is 94.02%
percentage of subscription 5.98%

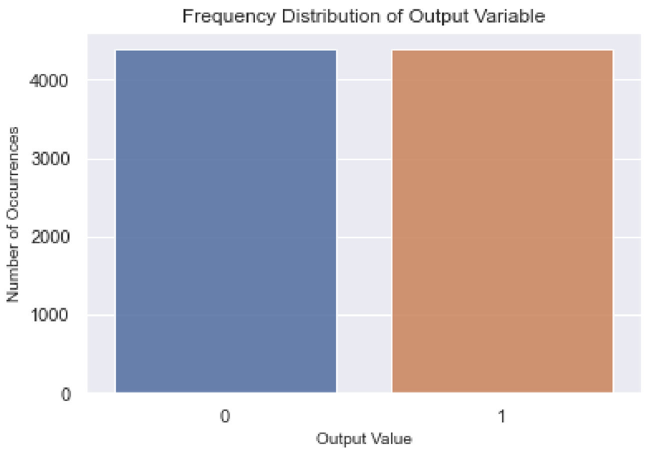
Step 5: Over-sampling using SMOTE

Original Frequency Distribution of Output Variable



percentage of no subscription is 94.02%
percentage of subscription 5.98%

Frequency Distribution of Output Variable POST SMOTE



percentage of no subscription is 50%
percentage of subscription 50%

Now we have a perfect **balanced data**! It's noticeable that data is **over-sampled** only on the training data, because by oversampling only on the training data, none of the information in the test data is being used to create synthetic observations, therefore, no information will bleed from test data into the model training.

Step 6: Normalization

- To give equal importance to all features, we need to scale the continuous features using **scikit-learn's** **MinMaxScaler,** as the feature matrix is a mix of both binary and continuous variables.
- All the variables would be normalized to have values between 0 and 1.

In [44]:
```
mms = MinMaxScaler()
mms.fit(os_data_X)
os_data_X1 = mms.transform(os_data_X)
columns = os_data_X.columns
os_data_X1 = pd.DataFrame(os_data_X1,columns=columns)
os_data_X1.head()
```

Out[44]:

	MAANTHUI	MGEMOMV	MGODPR	MGODOV	MGODGE	MRELGE	MRELSA	MRELOV	MFALLEEN	MFGEKIND	...	MGODRK_4	MGODRK_5	MGODRK_6	MGODRI_
0	0.000000	0.25	0.444444	0.4	0.444444	0.777778	0.142857	0.222222	0.333333	0.444444	...	0.0	0.0	0.0	0.
1	0.000000	0.25	0.555556	0.2	0.222222	0.444444	0.571429	0.222222	0.111111	0.666667	...	0.0	0.0	0.0	0.
2	0.000000	0.50	0.444444	0.0	0.555556	1.000000	0.000000	0.000000	0.000000	0.222222	...	0.0	0.0	0.0	0.
3	0.000000	0.75	0.555556	0.4	0.222222	0.555556	0.000000	0.444444	0.333333	0.000000	...	0.0	0.0	0.0	0.
4	0.111111	0.25	0.333333	0.4	0.444444	0.222222	0.571429	0.333333	0.333333	0.333333	...	0.0	0.0	0.0	0.

Step 7: Recursive Feature Elimination

- Recursive Feature Elimination (**RFE**) is based on the idea to repeatedly construct a model and choose either the best or worst performing feature, setting the feature aside and then repeating the process with the rest of the features.
- This process is applied until all features in the dataset are exhausted. The goal of RFE is to select features by **recursively** considering smaller and smaller sets of features.
- With this method, we got our **top 20** variables.

In [45]:
```
logreg = LogisticRegression()
rfe = RFE(logreg, 20)
rfe = rfe.fit(os_data_X1, os_data_y.values.ravel())
print(rfe.support_)
print(rfe.ranking_)
```

In [48]:
```
os_data_X1.columns[rfe.support_]
```

Out[48]:
```
Index(['MFALLEEN ', 'MFGEKIND ', 'MFWEKIND ', 'MOPLHOOG ', 'MOPLMIDD ',
       'MOPLLAAG ', 'MHHUUR ', 'MHKOOP ', 'MZFONDS ', 'MZPART ', 'AWAPART ',
       'MOSTYPE _7', 'MOSTYPE _35', 'MOSTYPE _36', 'MOSHOOFD _4',
       'MOSHOOFD _5', 'MOSHOOFD _6', 'MOSHOOFD _7', 'MOSHOOFD _10',
       'PWAPART _0'],
      dtype='object')
```

Step 8: Feature Selection using P-Value

- Ran **Logit** model on the selected features to find **P-value** for all the features.
- Removed the feature **MOSHOOFD_4** as it is having P-value more than **0.05** and re-ran the model to finalize the features.

Initial Model Output

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
MFALLEEN	-6.5129	0.4179	-15.5833	0.0000	-7.3320	-5.6937
MFGEKIND	-6.9969	0.4478	-15.6247	0.0000	-7.8746	-6.1192
MFWEKIND	-7.0797	0.4474	-15.8231	0.0000	-7.9567	-6.2028
MOPLHOOG	-1.9602	0.4204	-4.6623	0.0000	-2.7842	-1.1361
MOPLMIDD	-2.2914	0.4267	-5.3705	0.0000	-3.1276	-1.4551
MOPLLAAG	-3.3617	0.4249	-7.9113	0.0000	-4.1946	-2.5289
MHHUUR	4.5784	0.6488	7.0563	0.0000	3.3067	5.8501
MHKOOP	4.8553	0.6448	7.5296	0.0000	3.5915	6.1192
MZFONDS	8.3341	0.6513	12.7969	0.0000	7.0576	9.6105
MZPART	7.6235	0.6676	11.4191	0.0000	6.3150	8.9320
AWAPART	-4.7953	0.2329	-20.5869	0.0000	-5.2519	-4.3388
MOSTYPE _7	-2.9103	0.6140	-4.7396	0.0000	-4.1138	-1.7068
MOSTYPE _35	-2.4930	0.2911	-8.5647	0.0000	-3.0634	-1.9225
MOSTYPE _36	-2.0602	0.2297	-8.9696	0.0000	-2.5104	-1.6100
MOSHOOFD _4	-21.7595	7042.9983	-0.0031	0.9975	-13825.7824	13782.2634
MOSHOOFD _5	-2.3285	0.1572	-14.8162	0.0000	-2.6365	-2.0205
MOSHOOFD _6	-5.1285	1.0070	-5.0931	0.0000	-7.1021	-3.1549
MOSHOOFD _7	-2.3334	0.1893	-12.3269	0.0000	-2.7045	-1.9624
MOSHOOFD _10	-3.4840	0.4252	-8.1944	0.0000	-4.3173	-2.6507
PWAPART _0	-3.2361	0.1179	-27.4397	0.0000	-3.4672	-3.0049

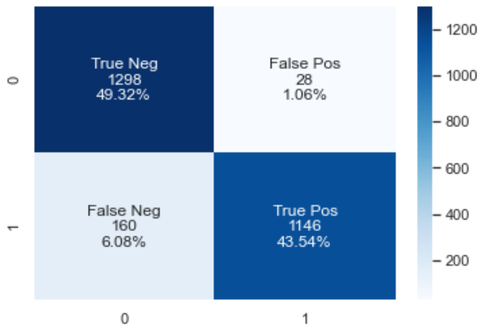
Final Model Output

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
MFALLEEN	-6.5941	0.4163	-15.8402	0.0000	-7.4101	-5.7782
MFGEKIND	-6.7903	0.4445	-15.2758	0.0000	-7.6615	-5.9190
MFWEKIND	-6.8888	0.4445	-15.4986	0.0000	-7.7599	-6.0176
MOPLHOOG	-2.1696	0.4183	-5.1872	0.0000	-2.9894	-1.3498
MOPLMIDD	-2.3306	0.4247	-5.4877	0.0000	-3.1629	-1.4982
MOPLLAAG	-3.3815	0.4231	-7.9924	0.0000	-4.2108	-2.5523
MHHUUR	4.5026	0.6468	6.9616	0.0000	3.2349	5.7702
MHKOOP	4.8143	0.6430	7.4878	0.0000	3.5541	6.0744
MZFONDS	8.2784	0.6494	12.7476	0.0000	7.0056	9.5512
MZPART	7.5932	0.6661	11.3995	0.0000	6.2877	8.8987
AWAPART	-4.8165	0.2320	-20.7599	0.0000	-5.2713	-4.3618
MOSTYPE _7	-2.8978	0.6148	-4.7131	0.0000	-4.1029	-1.6927
MOSTYPE _35	-2.4610	0.2900	-8.4870	0.0000	-3.0293	-1.8927
MOSTYPE _36	-2.0473	0.2303	-8.8878	0.0000	-2.4987	-1.5958
MOSHOOFD _5	-2.2451	0.1561	-14.3851	0.0000	-2.5510	-1.9392
MOSHOOFD _6	-5.0117	1.0065	-4.9793	0.0000	-6.9844	-3.0390
MOSHOOFD _7	-2.3114	0.1892	-12.2197	0.0000	-2.6821	-1.9407
MOSHOOFD _10	-3.5101	0.4254	-8.2519	0.0000	-4.3439	-2.6764
PWAPART _0	-3.2430	0.1175	-27.5972	0.0000	-3.4733	-3.0127

Step 8: Logistic Regression Model Fitting – Validation Set

- Accuracy of **logistic regression** classifier on validation set: **0.93**
- The precision is the ratio **tp / (tp + fp)** where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier to not label a sample as positive if it is negative

Confusion Matrix



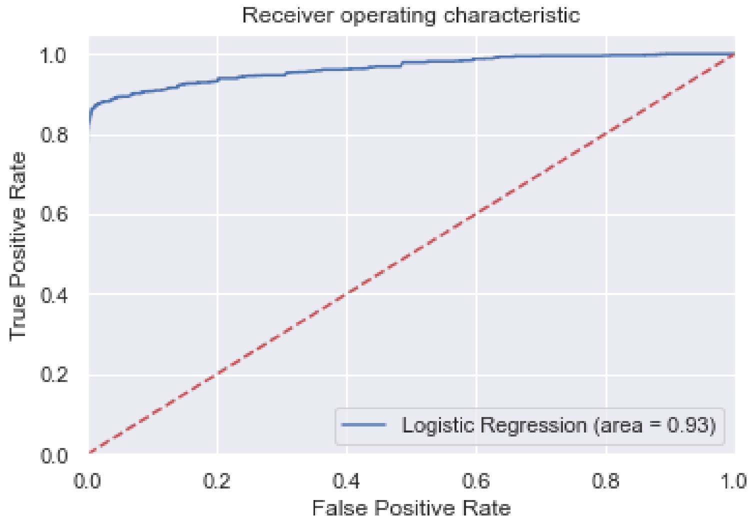

```
In [56]: print(classification_report(y_test1, y_pred))

              precision    recall  f1-score   support

           0       0.89      0.98      0.93      1326
           1       0.98      0.88      0.92      1306

    accuracy                           0.93      2632
   macro avg       0.93      0.93      0.93      2632
weighted avg       0.93      0.93      0.93      2632
```

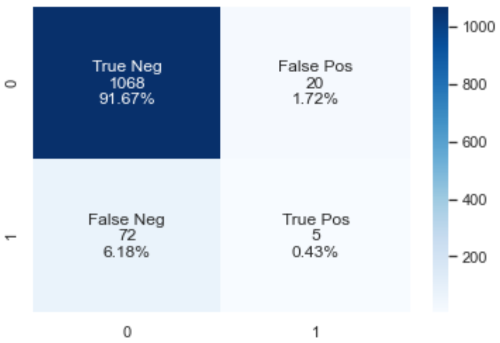
ROC Curve



Step 8: Logistic Regression Model Fitting – Test Set

- **Accuracy** of logistic regression classifier on Test set: **0.92**

Confusion Matrix



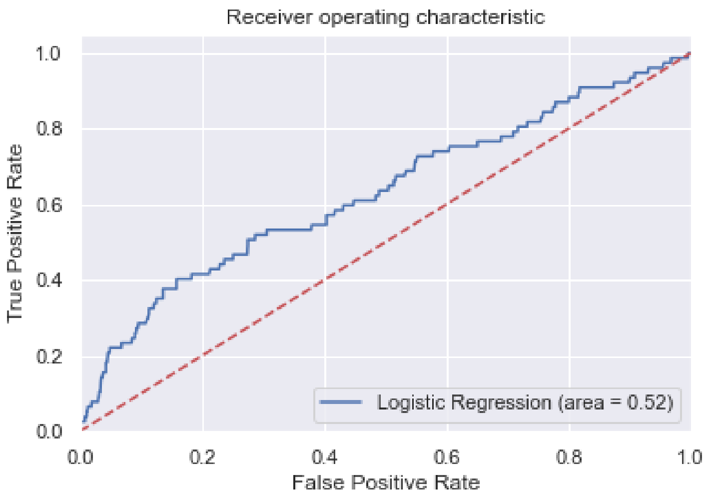

```
In [70]: print(classification_report(y_test2, y_pred2))

              precision    recall  f1-score   support

           0       0.94      0.98      0.96      1088
           1       0.20      0.06      0.10        77

    accuracy                           0.92      1165
   macro avg       0.57      0.52      0.53      1165
weighted avg       0.89      0.92      0.90      1165
```

ROC Curve



Conclusions

1. Data is **positively skewed** for output variable and needed to be treated. But due to very **less data** for policy subscription, model is unable to fit properly for such records.
2. After applying **SMOTE** algorithm, the accuracy of model for policy subscription increased in validation data set.
3. Various Feature selection/elimination methods were used to reduce the number of independent variables.
4. The category variables had too many category types leading to the high dimensionality issue. **Dimensionality reduction** approaches specific to the categorical variables needs to be applied in future work.

Thank You!