Data Mining Project

Sharmila Sengupta Chowdhry

Table of Contents

Introduction

Analysis of the problem

Problem: A tour insurance firm is facing higher claim frequency.

Objective: We have been provided data from over a few years from the tour insurance company, to make a model to predict the likely claim status of any new customer buying insurance.

We will be using supervised learning techniques – Decision Trees, Random Forests and Artificial Neural Networks for making this prediction.

Analysis of the data

It is quite clear that the data is imbalanced (30% of the data is Claim positive, prior to duplicate removal). This is quite expected in real life situations, especially in cases of fraud prevention, cancer detection, etc. where detecting, analysing or predicting the minority class is the objective. In this case, because the model will learn more from the majority class (i.e. no claim or 0 or Claim negative, in this case), we cannot use Accuracy as a means to compare model performance, since the model will only reflect the accuracy of the majority class.

For an imbalanced dataset, other performance metrics should be used, such as the Precision, Recall, AUC score, and F1 score. Since most machine learning techniques are designed to work well with a balanced dataset, data scientists create balanced data using the following methods:

- Undersampling - Creating a new dataset out of the original dataset using undersampling of the majority class, making the two datasets equal in size.

- Oversampling - Resampling of the minority class points to equal the total number of majority points. Repetition of the minority class points is one such type of oversampling technique.

- Creating synthetic data – With the help of SMOTE (Synthetic Minority Oversampling technique) we can create synthetic data points for the minority class. It creates new instances between the points of the minority class.

- Penalize the overclassified - Penalized classification imposes an additional cost on the model for making classification mistakes on the minority class during training. These penalties can bias the model to pay more attention to the minority class.

Our recommendation is to create synthetic data through SMOTE and run the same models again. This falls outside the scope of our learning and this assignment, so we will leave it as a recommendation.

We will focus on analysing certain parameters during model evaluation and comparison, as mentioned in the next point, *Selection of model performance metrics.*

Resampling via **cross-validation** is a used to evaluate machine learning models on a limited data sample. The data is divided in a number of 'folds' or buckets and a part of that is used for training and the remaining is used for testing. This is repeated with different combinations of train and test folds. The scores are then averaged out to arrive at a final score.

Selection of model performance metrics

Accuracy indicates how many predictions have been identified by the model as correctly true and correctly false in all the predictions made.
Accuracy = TP+TN / (TP+FP+TN+FN). In imbalanced datasets, accuracy is not considered fool proof, as it doesn't give the right picture.

Specificity or True Negative Rate (TNR) indicates the proportion of actual negatives that are correctly identified by the model.
Specificity = TN/(TN+FP)

Since we are not very interested in the negative Claim status, we can overlook Specificity as a metric too.

Rather than the above two metrics, the two metrics below will throw more light on what we are expecting from our model:

Precision or Positive Prediction Rate (PPR) is the ability of the model to determine positives correctly. Precision is the accuracy of positive predictions. Precision is low if the model classifies too many negatives as positives.
Precision = TP/(TP+FP)

Sensitivity /Recall or True Positive Rate (TPR) is the fraction of positives that were correctly identified.
Recall = TP / (TP+FN)

Below is our explanation for considering the above two metrics for comparing model performance:

We can aim to maximize precision or recall depending on the task, *so we eventually need to choose between avoiding false positives or false negatives.*

Therefore, we cannot try to maximize both precision and recall because there is a trade-off between them. Increasing precision decreases recall and vice versa.

For an email spam detection model, we try to maximize precision because we want to be correct when an email is detected as spam. Its more disadvantageous to label a normal email as spam (i.e. false positive), than it is to have a spam be treated as a useful email (i.e. false negative). That's because the *cost of false positive is more than the benefit of avoiding a false negative.*

On the other hand, for a tumor detection task, we need to maximize recall because we want to detect positive classes as much as possible. Its more beneficial to label a healthy subject as one with a tumor (i.e. false positive), than it is to have an unhealthy subject as one who is healthy (i.e. false negative). The assumption being to err on the side of caution, that if someone is wrongly classified a having a tumor, somewhere in subsequent prognosis, they will eventually find out and be relieved. The cost of that (on a holistic level) is lesser than the cost of a tumor being left undetected and the medical fallout on a human life. Therefore, here *the cost of identifying a false negative outweighs the benefit or value from avoiding a false positive.*

In our case, is it more important to identify *false positive* or *false negative?*

False negative is a case where a customer actually shows the right patterns for asking for a claim, but the model has not predicted it as so.

False Positive is a case where a customer actually shows the right patterns for *not* asking for a claim, but the model has predicted it as so.

In order of importance, detecting a False Negative is most important, so we will consider Recall of class 1 to be most important. We will follow this by the F1 Score, since that is a harmonic mean of Precision and Recall. Therefore, our top 3 choice of metrics are:

1. Test AUC
2. Test Recall Score of Class 1
3. Test F1 Score of Class 1
4. Test Precision Score of Class 1

Recall is preferred higher than precision because False Negative is more disadvantageous than False Positive because we are trying to avoid claim pay outs.

Q1. Exploratory Descriptive Analytics

Data Ingestion: Read the dataset. Do the descriptive statistics and do null value condition check, write an inference on it.

We import all the necessary libraries. Then we read the dataset. Run basic exploratory data analysis.

```
[4] data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Age            3000 non-null   int64
 1   Agency_Code    3000 non-null   object
 2   Type           3000 non-null   object
 3   Claimed        3000 non-null   object
 4   Commision      3000 non-null   float64
 5   Channel        3000 non-null   object
 6   Duration       3000 non-null   int64
 7   Sales          3000 non-null   float64
 8   Product Name   3000 non-null   object
 9   Destination    3000 non-null   object
dtypes: float64(2), int64(2), object(6)
memory usage: 234.5+ KB
```

There are 3000 datapoints and no null values. Of the 10 features, 6 are object types, 2 are floats and 2 are integers.

```
data.describe()
```

	Age	Commision	Duration	Sales
count	3000.000000	3000.000000	3000.000000	3000.000000
mean	38.091000	14.529203	70.001333	60.249913
std	10.463518	25.481455	134.053313	70.733954
min	8.000000	0.000000	-1.000000	0.000000
25%	32.000000	0.000000	11.000000	20.000000
50%	36.000000	4.630000	26.500000	33.000000
75%	42.000000	17.235000	63.000000	69.000000
max	84.000000	210.210000	4580.000000	539.000000

Fig. Description of Numerical data

Since there is quite a lot of difference between mean and 50% of the case of Duration, Sales and Commission, we can conclude that the data is left skewed. Age seems to be closer to a normal distribution.

```
[ ] data.describe(include='object')
```

	Agency_Code	Type	Claimed	Channel	Product Name	Destination
count	3000	3000	3000	3000	3000	3000
unique	4	2	2	2	5	3
top	EPX	Travel Agency	No	Online	Customised Plan	ASIA
freq	1365	1837	2076	2954	1136	2465

Fig. Description of categorical data

The categorical features have 2, 3, 4 and 5 class divisions. Target variable 'Claimed' is divided into 2 classes, with a majority class being *No*. Our class of interest being *Yes*, is around 30% of the dataset.

Duplicates

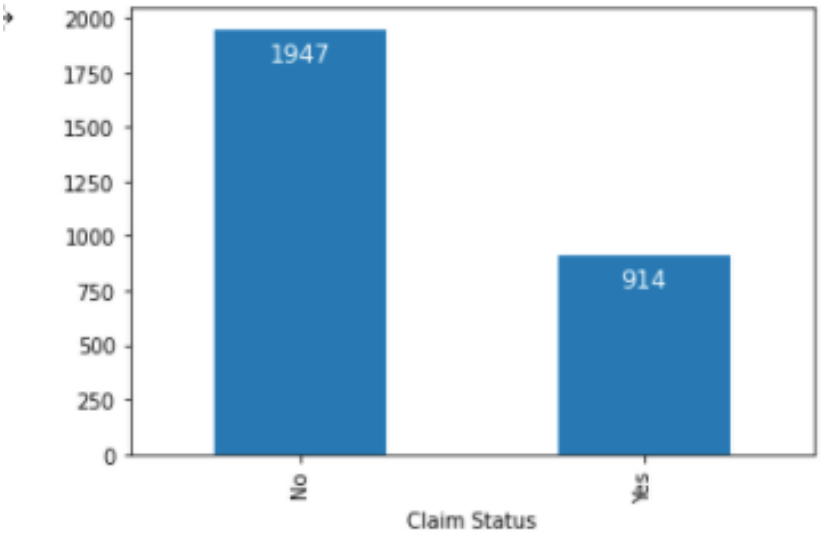


Fig. Distribution of Target Variable 'Claim'

We found 139 duplicates, which we dropped. We now have 2861 records, of which 1947 are No and only 914 are Yes.

Distributions & Classification

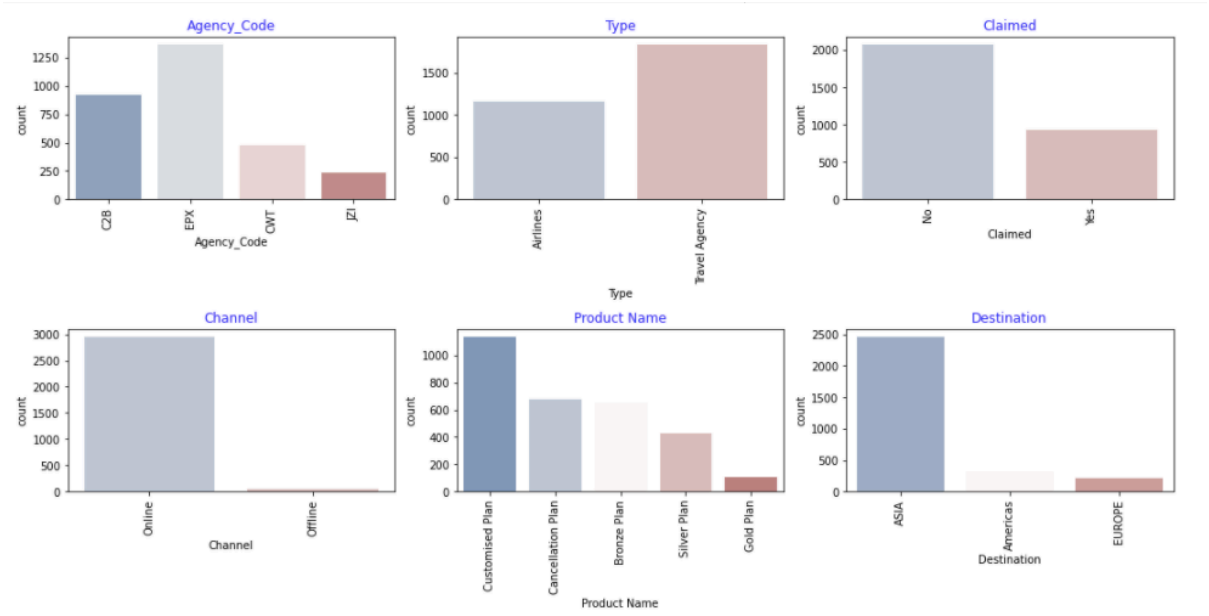


Fig. Countplot of Categorical features

While the data for the target variable is imbalanced, its not as imbalanced as it could get (e.g. very rare diseases, analysing rare astronomical phenomena, etc.). As we have seen before, roughly 1/3 of the data is Claim positive. But whether this is enough to train our models, we will see during model assessment stage.
We will not be dropping any variables at this point, since we don't have any Serial No. or other features that can be redundant to the machine learning models.

Agent_Code has been kept intact so that we can see whether that is important to the pattern of claim positive cases.

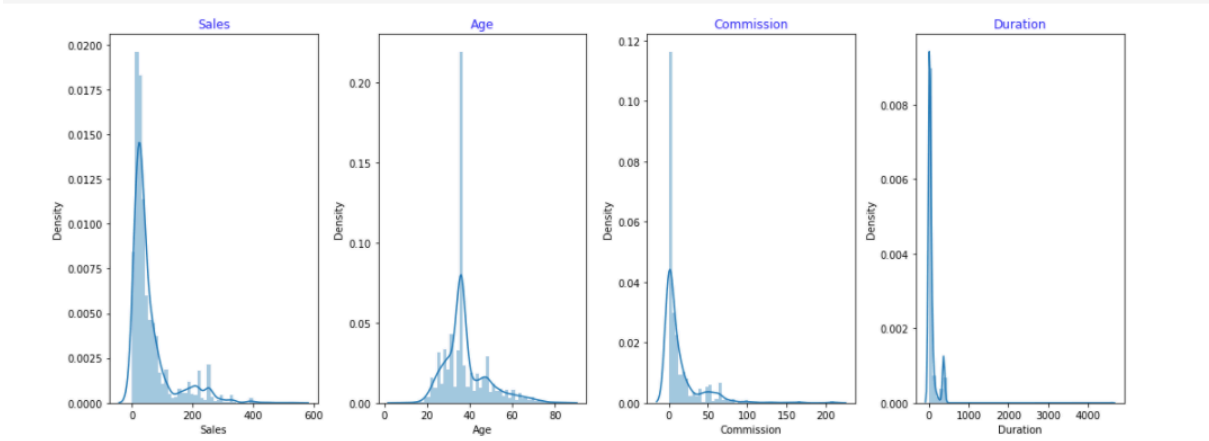


Fig. Distribution of numeric features

In the numerical features, Sales, Commission are right skewed and Age is closer to a normal distribution. Duration has two outliers which we will deal with.

Treating Extreme Values

Although Machine Learning algorithms are robust to outliers, we wanted to purge the data of the two outliers in Duration, -1 and 4580, both which seem improbable. To impute these two values, we separated the 2 classes of with the same categorical data, calculated the mean duration and then replaced the above Durations.

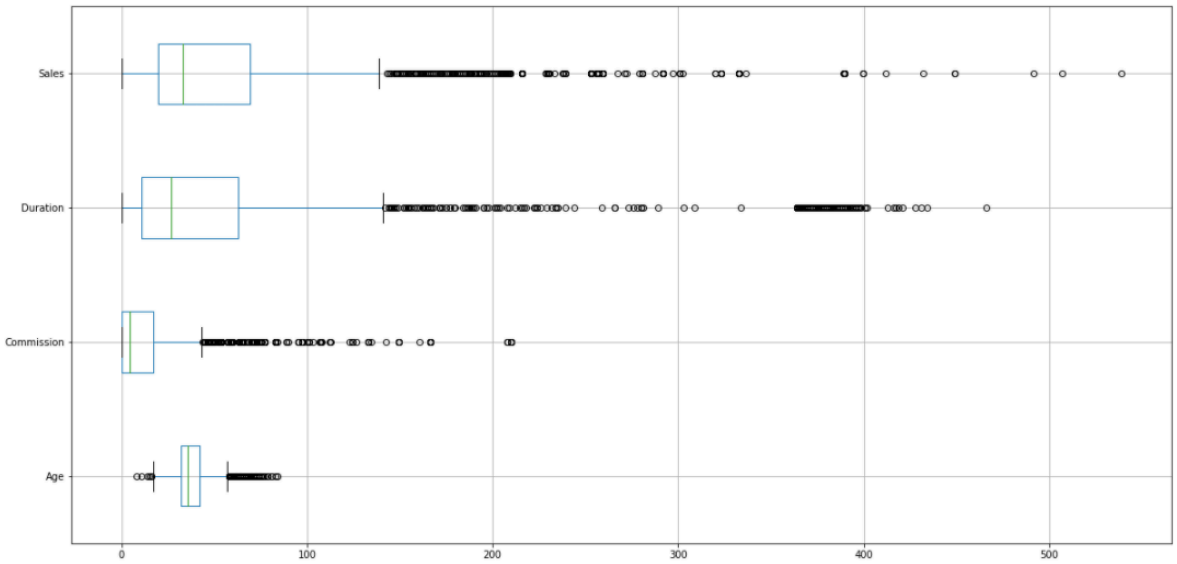


Fig. Boxplot after data cleaning

Even after imputing these two values (which we did to clean the data), there a significant outliers in the data. We will not supress or clean them, since ML uses binning and therefore not very sensitive to outliers.

Correlations

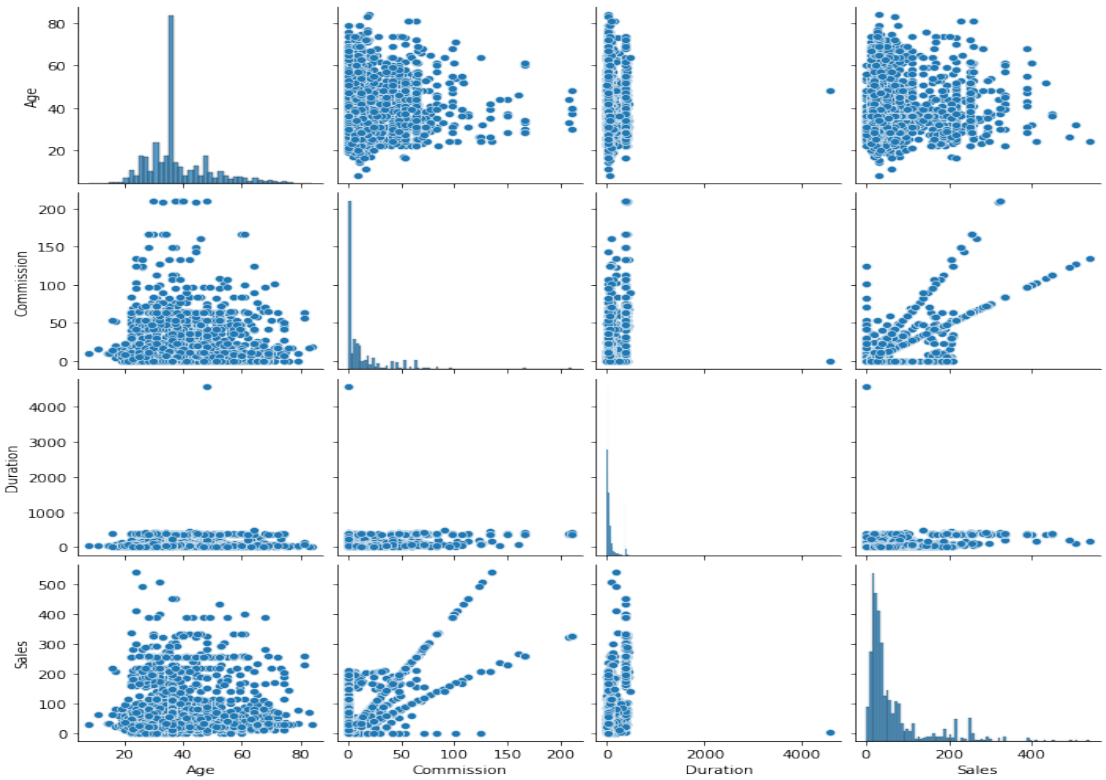


Fig. Pairplot analysing correlation between numerical variables

No correlations exist, except for Sales and Commission, which is quite expected.

Encoding For Machine Learning algorithms, data needs to be put into numerical form to use the data to make predictions. We would be encoding, Agency_Code, Type, Channel, Product_Name, Destination, the object datatypes. No harm in converting our Target variable to a code too, although that's not required.

Q2. Data Split

Split the data into test(30% of the data) and train(70% of the data), build classification model CART, Random Forest, Artificial Neural Network.

Train-Test Split

The dataset is divided by the predictors or the independent variables and the target variable or the dependent variable.
Both the dependent variable and the independent variables are divided into train and test set. The train set is used to train the model. The model is then tested on the test set to check the model performance on unseen data.

```
[203] # Separating the Independent and target variables
      df_predictors = data.drop('Claimed', axis = 1)
      df_target = data['Claimed']
```

```
# Spliting the data in train and test
X_train, X_test, y_train, y_test = train_test_split(df_predictors, df_target, test_size = 0.3, random_state = 123)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

(2002, 9)
(2002,)
(859, 9)
(859,)
```

In the next section, we build different Machine Learning models using CART, Random Forest and Neural Networks techniques.

Q3. Performance Metrics

Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score for each model.

In the last section, we built models with various Machine Learning techniques.

For each technique and model, we use a variety of combinations of hyper parameters, measure their performance, then select the right combination which maximizes results. We try to select the right combination within a technique, then select the best performing model between different techniques to arrive at the most preferred model and technique.

In this section, we evaluate model performances.

Decision Tree (CART)

Although, we know theoretically that we get better results after plugging in the hyperparameters that is found out by the GridSearch, we still ran the vanilla Decision Tree algorithm to confirm this point in this specific case.

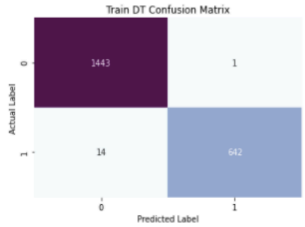
▼ Decision Tree - Train

```
[116] # Performance Scores - Train
      CART_train_AS = accuracy_score(y_train, dt_ytrain_pred, normalize = True)*100
      CART_train_PS = precision_score(y_train, dt_ytrain_pred)*100
      CART_train_F1 = f1_score(y_train, dt_ytrain_pred)*100
      CART_train_RS = recall_score(y_train, dt_ytrain_pred)*100

      print('The Train CART Accuracy Score is %.2f'%CART_train_AS,'%')
      print('The Train CART Precision Score is %.2f'%CART_train_PS,'%')
      print('The Train CART F1 Score is %.2f'%CART_train_F1,'%')
      print('The Train CART Recall Score is %.2f'%CART_train_RS,'%')
```

```
The Train CART Accuracy Score is 99.29 %
The Train CART Precision Score is 99.84 %
The Train CART F1 Score is 98.85 %
The Train CART Recall Score is 97.87 %
```

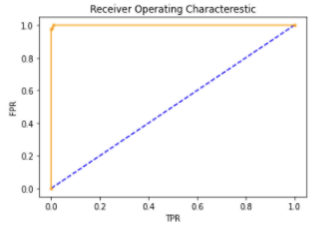
```
[117] # Confusion Report and Classification Matrix
      print(confusion_matrix(y_train, dt_ytrain_pred))
      print(classification_report(y_train, dt_ytrain_pred))
      print('\n')
      print("_____________________________________________________________")
      print('\n')
      sns.heatmap(confusion_matrix(y_train, dt_ytrain_pred),annot=True, fmt='d', cbar=False,cmap='BuPu')
      plt.xlabel('Predicted Label')
      plt.ylabel('Actual Label')
      plt.title('Train DT Confusion Matrix')
      plt.show()
```

```
[[1443    1]
 [  14  642]]
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1444
           1       1.00      0.98      0.99       656

    accuracy                           0.99      2100
   macro avg       0.99      0.99      0.99      2100
weighted avg       0.99      0.99      0.99      2100
```




```
[118] #AUC - ROC for model Performance (Train)
      #AUC
      dt_train_auc = roc_auc_score(y_train, dt_ytrain_pred_prob[:,1])
      print("Train DT AUC %.2f" % dt_train_auc)
      #ROC
      fpr_dt_tr, tpr_dt_tr, th_dt = roc_curve(y_train, dt_ytrain_pred_prob[:,1], pos_label= 1)
      plt.plot ([0,1],[0,1], linestyle = '--', color = 'blue', label = 'Unskilled')
      plt.plot(fpr_dt_tr, tpr_dt_tr, marker = '.', color = 'orange', label = 'Skill')
      plt.ylabel('FPR')
      plt.xlabel('TPR')
      plt.title('Receiver Operating Characterestic')
      plt.show()
```

```
Train DT AUC 1.00
```



The train numbers look very encouraging, at near perfect performance in all parameters and a perfect AUC Score of 1.

But we know that if a model is returning 100% on all performance metrics on trained data, it indicates that the model has learnt perfectly from the trained data that it has seen. In most cases, it doesn't perform as well on test or validation data, that it has not seen. This mismatch of model performance between train and test data is called over fitting, which Decision Trees in particular, are very prone to.

AUC stands for Area Under the Curve (the ROC Curve). The ROC or Receiver's Characteristic Curve that plots the True Positive Rate on the y-axis and False Positive Rate (1-True Negative Rate) on the x-axis, at various threshold levels of classification.

An excellent model will have an AUC Score of near 1. A poor model has an AUC Score of 0, meaning it is predicting 0 as 1 and 1 as 0.

An AUC Score of 0.5 means the model is not effective, it is no different than random chance. For an AUC Score of .8, there is an 80% chance that the model will predict correctly.

The Decision Tree model performance in Colab Notebook is as follows:

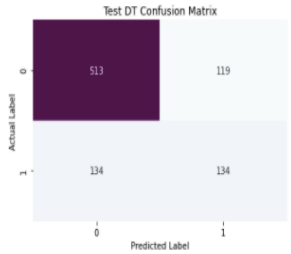
Decision Tree - Test

```
[119] # Performance Scores - Test
      CART_test_AS = accuracy_score(y_test, dt_ytest_pred, normalize = True)*100
      CART_test_PS = precision_score(y_test, dt_ytest_pred)*100
      CART_test_F1 = f1_score(y_test, dt_ytest_pred)*100
      CART_test_RS = recall_score(y_test, dt_ytest_pred)*100

      print('The Test CART Accuracy Score is %.2f'%CART_test_AS,'%')
      print('The Test CART Precision Score is %.2f'%CART_test_PS,'%')
      print('The Test CART F1 Score is %.2f'%CART_test_F1,'%')
      print('The Test CART Recall Score is %.2f'%CART_test_RS,'%')
```

```
The Test CART Accuracy Score is 71.89 %
The Test CART Precision Score is 52.96 %
The Test CART F1 Score is 51.44 %
The Test CART Recall Score is 50.00 %
```

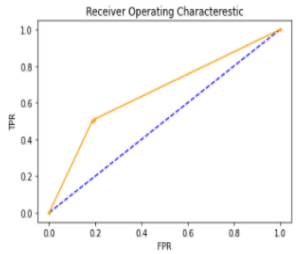
```
# Confusion Report and Classification Matrix (Test)
print(confusion_matrix(y_test, dt_ytest_pred))
print(classification_report(y_test, dt_ytest_pred))
print('\n')
print("_____")
print('\n')
sns.heatmap(confusion_matrix(y_test, dt_ytest_pred),annot=True, fmt='d', cbar=False,cmap='BuPu')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title('Test DT Confusion Matrix')
plt.show()
```

```
[[513 119]
 [134 134]]
              precision    recall  f1-score   support

           0       0.79      0.81      0.80       632
           1       0.53      0.50      0.51       268

    accuracy                           0.72       900
   macro avg       0.66      0.66      0.66       900
weighted avg       0.71      0.72      0.72       900
```




```
#AUC - ROC for model Performance (Test)
dt_test_auc = roc_auc_score(y_test, dt_ytest_pred_prob[:,1])
print("Test DT AUC %.2f" % dt_test_auc)
#ROC
fpr_dt_t, tpr_dt_t, th_dt_t = roc_curve(y_test, dt_ytest_pred_prob[:,1], pos_label= 1)
plt.plot ([0,1],[0,1], linestyle = '--', color = 'blue', label = 'Unskilled')
plt.plot(fpr_dt_t, tpr_dt_t, marker = '.', color = 'orange', label = 'Skill')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.title('Receiver Operating Characterestic')
plt.show()
```

```
Test DT AUC 0.66
```



As expected, model is performing quite unsatisfactorily on test data. We will prune the tree with hyperparameters selected by the GridSearch algorithm.

Once we check the performance of the model on test data, it confirms our suspicion. An AUC of 0.65, and at 0.50 and 0.51 for recall and f1 score for class 1, the model leaves much to be desired. This condition is called overfitting. Overfitting means the model has learnt the train data so well, that it cannot perform well in test data (or any new data). However, the idea of training a model is to predict on new data. So, we will prune the tree with hyperparameter tuning, so that we can leave out some of the noise from the training data that's making the model ineffective.

Decision Tree GridSearch

GridSearch gives us the opportunity to find the right set of hyperparameters with which we can plug in to form the best performing Decision Tree model. We gave a wide range of options for the grid to present us the best hyperparameters.

While selecting hyperparameters given by GridSearchCV, we made sure that the GridSearchCV algorithm had enough options on both sides (lower and upper), so the selection is correct.

For Minimum sample leaf, we intentionally chosen to be no less than 30, since we wanted minimum critical mass on the terminal nodes for correct training.

Below are the 4 different combinations of hyperparameters we chose.

a. param_grid ={'criterion':['gini', 'entropy'],
'max_depth':range(2, 9),
'min_samples_split':[120, 150, 155, 160],
'min_samples_leaf': [30, 35, 40, 50]}

best_grid : {'criterion': 'gini','max_depth': 2,
'min_samples_leaf': 30,'min_samples_split': 120}

b. param_grid ={'criterion':['gini', 'entropy'],
'max_depth':range(4, 9),
'min_samples_split':[90, 100, 120],
'min_samples_leaf': [30, 35, 40, 50]}

best_grid: {'criterion': 'entropy','max_depth': 5,
'min_samples_leaf': 30, 'min_samples_split': 90}

c. param_grid ={'criterion':['gini', 'entropy'],
'max_depth':range(4, 9),
'min_samples_split':[70, 80, 90, 95],
'min_samples_leaf': [30, 35, 40, 50]}

best_grid:{'criterion': 'entropy',
'max_depth': 4,'min_samples_leaf': 30, 'min_samples_split': 70}

d. param_grid ={'criterion':['gini', 'entropy'],
'max_depth':range(4, 6),
'min_samples_split':[60, 70, 80, 90, 95],
'min_samples_leaf': [30, 35]}

best_grid: {'criterion': 'entropy','max_depth': 4,
'min_samples_leaf': 30,'min_samples_split': 60}

When we chose a range of 2-9 for max depth, the algorithm kept picking 2 as max depth. We understand that it should be more than that, of course, so we chose a higher range, first choosing 4 as the minimum which yielded.

DTGrid	a.	b.	c.	d.
Test AUC	0.79	0.79	0.79	0.79
Test Recall	0.60	0.63	0.63	0.63
Test Precision	0.65	0.66	0.66	0.66
Test F1 Score	0.62	0.64	0.64	0.64

Since there is no difference in performance of Decision Tree GridSearch models b., c., d. any of them could be deployed.

The Decision Tree GridSearch performance Jupyter Notebook for model b. is as follows:

▾ Decision Tree GridSearch - Train

```
[227] # Performance Scores
      DTGS_train_AS = accuracy_score(y_train, dtgs_ytrain_pred, normalize = True)*100
      DTGS_train_PS = precision_score(y_train, dtgs_ytrain_pred)*100
      DTGS_train_F1 = f1_score(y_train, dtgs_ytrain_pred)*100
      DTGS_train_RS = recall_score(y_train, dtgs_ytrain_pred)*100

      print('The Train DTGS Accuracy Score is %.2f'%DTGS_train_AS,'%')
      print('The Train DTGS Precision Score is %.2f'%DTGS_train_PS,'%')
      print('The Train DTGS F1 Score is %.2f'%DTGS_train_F1,'%')
      print('The Train DTGS Recall Score is %.2f'%DTGS_train_RS,'%')
```

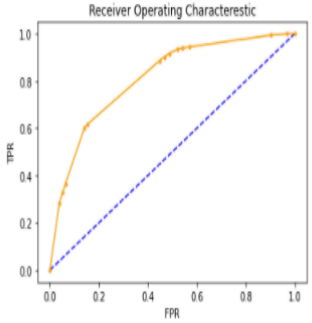
```
The Train DTGS Accuracy Score is 77.57 %
The Train DTGS Precision Score is 67.07 %
The Train DTGS F1 Score is 63.29 %
The Train DTGS Recall Score is 59.91 %
```

```
# Confusion Report and Classification Matrix
print(confusion_matrix(y_train, dtgs_ytrain_pred))
print(classification_report(y_train, dtgs_ytrain_pred))
print('\n')
print("_____")
print('\n')
sns.heatmap(confusion_matrix(y_train, dtgs_ytrain_pred),annot=True, fmt='d', cbar=False,cmap='BuPu')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title('Train DTGS Confusion Matrix')
plt.show()
```

```
[[1166  190]
 [ 259  387]]
              precision    recall  f1-score   support

           0       0.82      0.86      0.84      1356
           1       0.67      0.60      0.63       646

    accuracy                           0.78      2002
   macro avg       0.74      0.73      0.74      2002
weighted avg       0.77      0.78      0.77      2002
```

```
#AUC - ROC for model performance (Train)

#AUC

dtgs_train_auc = roc_auc_score(y_train, dtgs_ytrain_pred_prob[:,1])
print("Train DTGS AUC %.2f" % dtgs_train_auc)

#ROC
fpr_dtgs_tr, tpr_dtgs_tr, th_dtgs_tr = roc_curve(y_train, dtgs_ytrain_pred_prob[:,1], pos_label= 1)
plt.plot ([0,1],[0,1], linestyle = '--', color = 'blue', label = 'Unskilled')
plt.plot(fpr_dtgs_tr, tpr_dtgs_tr, marker = '.', color = 'orange', label = 'Skill')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.title('Receiver Operating Characterestic')
plt.show()
```

```
Train DTGS AUC 0.81
```



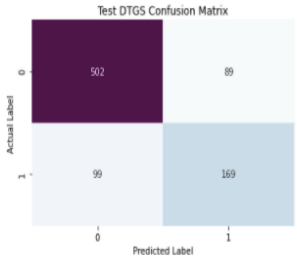
▼ Decision Tree GridSearch - Test

```
[229] # Performance Scores
      DTGS_test_AS = accuracy_score(y_test, dtgs_ytest_pred, normalize = True)*100
      DTGS_test_PS = precision_score(y_test, dtgs_ytest_pred)*100
      DTGS_test_F1 = f1_score(y_test, dtgs_ytest_pred)*100
      DTGS_test_RS = recall_score(y_test, dtgs_ytest_pred)*100

      print('Test DTGS Accuracy Score is %.2f'%DTGS_test_AS,'%')
      print('Test DTGS Precision Score is %.2f'%DTGS_test_PS,'%')
      print('Test DTGS F1 Score is %.2f'%DTGS_test_F1,'%')
      print('Test DTGS Recall Score is %.2f'%DTGS_test_RS,'%')
```

```
Test DTGS Accuracy Score is 78.11 %
Test DTGS Precision Score is 65.50 %
Test DTGS F1 Score is 64.26 %
Test DTGS Recall Score is 63.06 %
```

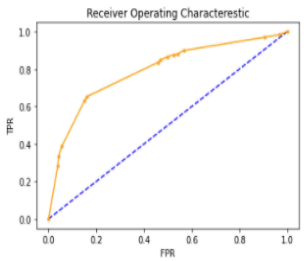
```
[230] # Confusion Report and Classification Matrix to check model performance
      print(confusion_matrix(y_test, dtgs_ytest_pred))
      print(classification_report(y_test, dtgs_ytest_pred))

      print('\n')
      print("_____________________________________________________________")
      print('\n')
      sns.heatmap(confusion_matrix(y_test, dtgs_ytest_pred),annot=True, fmt='d', cbar=False,cmap='BuPu')
      plt.xlabel('Predicted Label')
      plt.ylabel('Actual Label')
      plt.title('Test DTGS Confusion Matrix')
      plt.show()
```

```
[[502  89]
 [ 99 169]]
              precision    recall  f1-score   support

           0       0.84      0.85      0.84       591
           1       0.66      0.63      0.64       268

    accuracy                           0.78       859
   macro avg       0.75      0.74      0.74       859
weighted avg       0.78      0.78      0.78       859
```




```
#AUC - ROC for model performance (Test)

#AUC
dtgs_test_auc  = roc_auc_score(y_test, dtgs_ytest_pred_prob[:,1])
print("Test DTGS AUC %.2f" % dtgs_test_auc)

#ROC
fpr_dtgs_t, tpr_dtgs_t, th_dtgs_t = roc_curve(y_test, dtgs_ytest_pred_prob[:,1], pos_label= 1)
plt.plot ([0,1],[0,1], linestyle = '--', color = 'blue', label = 'Unskilled')
plt.plot(fpr_dtgs_t, tpr_dtgs_t, marker = '.', color = 'orange', label = 'Skill')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.title('Receiver Operating Characterestic')
plt.show()
```

```
Test DTGS AUC 0.79
```



Random Forest

Ensemble learning methods such as Random Forests help to overcome vulnerability Decision Trees to overfit the data by employing different algorithms and combining their output.

a. n=300
   msl=50
   mss=100
   md=6
   mf=9
   criterion= 'gini'

b. n=300
   msl=30
   mss=90
   md=6
   mf=9
   criterion= 'entropy'

c. n=500
   msl=30
   mss=100
   md=5
   mf=8
   criterion= 'entropy'

d. n=600
   msl=30
   mss=100
   md=4
   mf=7
   criterion= 'gini'

RF	a.	b.	c.	d.
Test AUC	0.79	0.80	0.80	0.80
Test Recall	0.55	0.60	0.61	0.60
Test Precision	0.66	0.67	0.66	0.66
Test F1 Score	0.60	0.63	0.63	0.63

Random Forest GridSearch

We again tried a variety of combinations of hyperparameters with the Random Forest GridSearch.

```
param_grid = {'n_estimators': [300, 500],
              'max_depth': [5,7],
              'min_samples_split' : [90, 150],
              'min_samples_leaf':[30, 50],
              'max_features': [8, 10],
              'criterion': ['gini', 'criterion']}
```

{'criterion': 'gini','max_depth': 7, 'max_features': 8, 'min_samples_leaf': 50, 'min_samples_split': 150,'n_estimators':300}

b. param_grid = {'n_estimators': [200, 150, 300], 'max_depth': [6,7,8],'min_samples_split' : [90, 150, 160],'min_samples_leaf':[30, 40, 50, 60],
'max_features': [6,7, 8],
'criterion': ['gini', 'criterion']}

{'criterion': 'gini','max_depth': 8, 'max_features': 6, 'min_samples_leaf': 60, 'min_samples_split': 90, 'n_estimators': 200}

The time taken is 2011.9405431747437

c. param_grid = {'n_estimators': [200, 150, 300], 'max_depth': [6,7,8],'min_samples_split' : [90, 150, 160],'min_samples_leaf':[30, 40, 50, 60],
'max_features': [6,7, 8],
'criterion': ['gini', 'criterion']}

param = {'criterion': 'gini','max_depth': 7, 'max_features': 6,'min_samples_leaf': 40, 'min_samples_split': 150, 'n_estimators': 200}

The time taken is 2006.6199448108673

```
{'criterion': 'gini',
 'max_depth': 7,
 'max_features': 6,
 'min_samples_leaf': 40,
 'min_samples_split': 150,
 'n_estimators': 200}
```

The time taken is 2006.6199448108673

d. param_grid = {'n_estimators': [100, 150, 200, 250],
'max_depth': [6,7,8],
'min_samples_split' : [90, 120, 150, 155],

```
        'min_samples_leaf':[30, 40, 50],
        'max_features': [4,5, 6,7],
                'criterion': ['gini', 'criterion']}

{'criterion': 'gini',
 'max_depth': 7,
 'max_features': 4,
 'min_samples_leaf': 40,
 'min_samples_split': 120,
 'n_estimators': 200}
```

Needless to say, this was much more computationally intensive than the Decision Tree technique. However, Test Recall, Test F1 Score and Test Precision, all were lower than Decision Tree GridSearch.

RF Grid	a.	b	c.	d.
Test AUC	0.80	0.80	0.80	0.80
Test Recall	0.56	0.54	0.53	0.54
Test Precision	0.66	0.67	0.65	0.65
Test F1 Score	0.61	0.60	0.59	0.59

The Random Forest Performance of Model a. in Jupyter Notebook is as follows:

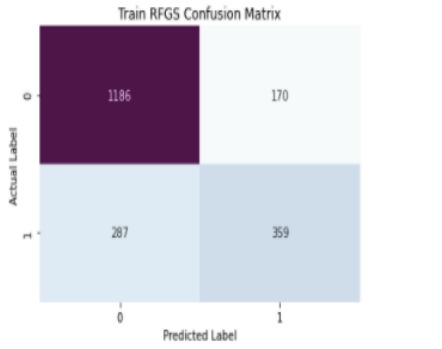
Random Forest - Grid Search - Train

[185]
```
# Performance Scores - Train
RFGS_train_AS = accuracy_score(y_train, rfgs_ypred_train, normalize = True)*100
RFGS_train_PS = precision_score(y_train, rfgs_ypred_train)*100
RFGS_train_F1 = f1_score(y_train, rfgs_ypred_train)*100
RFGS_train_RS = recall_score(y_train, rfgs_ypred_train)*100

print('Train RFGS Accuracy Score is %.2f'%RFGS_train_AS,'%')
print('Train RFGS Precision Score is %.2f'%RFGS_train_PS,'%')
print('Train RFGS F1 Score is %.2f'%RFGS_train_F1,'%')
print('Train RFGS Recall Score is %.2f'%RFGS_train_RS,'%')
```

```
Train RFGS Accuracy Score is 77.17 %
Train RFGS Precision Score is 67.86 %
Train RFGS F1 Score is 61.11 %
Train RFGS Recall Score is 55.57 %
```

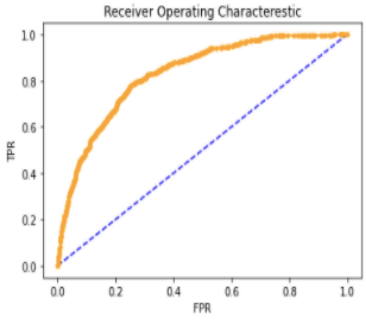
[186]
```
# Confusion Report and Classification Matrix to check model performance
print(confusion_matrix(y_train, rfgs_ypred_train))
print(classification_report(y_train, rfgs_ypred_train))
print("      ")
print("_____________________________________________________")

print("           ")
sns.heatmap(confusion_matrix(y_train, rfgs_ypred_train),annot=True, fmt='d', cbar=False,cmap='BuPu')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title('Train RFGS Confusion Matrix')
plt.show()
```

```
[[1186  170]
 [ 287  359]]
              precision    recall  f1-score   support

           0       0.81      0.87      0.84      1356
           1       0.68      0.56      0.61       646

    accuracy                           0.77      2002
   macro avg       0.74      0.72      0.72      2002
weighted avg       0.76      0.77      0.77      2002
```



[187]
```
#Random forest AUC (Train)
rfgs_train_auc = roc_auc_score(y_train, rfgs_ypred_train_prob[:,1])
print("Train RFGS AUC %.2f" % rfgs_train_auc)

#ROC
fpr_rfgs_tr, tpr_rfgs_tr, th_rfgs_tr = roc_curve(y_train, rfgs_ypred_train_prob[:,1], pos_label= 1)
plt.plot ([0,1],[0,1], linestyle = '--', color = 'blue', label = 'Unskilled')
plt.plot(fpr_rfgs_tr, tpr_rfgs_tr, marker = '.', color = 'orange', label = 'Skill')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.title('Receiver Operating Characterestic')
plt.show()
```

```
Train RFGS AUC 0.83
```



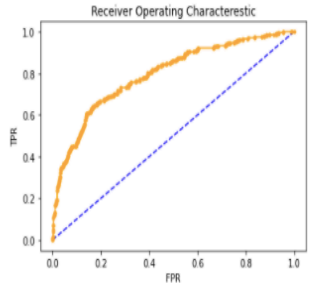
▾ Random Forest - GridSearch - Test

```
# Performance Scores - Train
RFGS_test_AS = accuracy_score(y_test, rfgs_ypred_test, normalize = True)*100
RFGS_test_PS = precision_score(y_test, rfgs_ypred_test)*100
RFGS_test_F1 = f1_score(y_test, rfgs_ypred_test)*100
RFGS_test_RS = recall_score(y_test, rfgs_ypred_test)*100

print('Test RFGS Accuracy Score is %.2f'%RFGS_test_AS,'%')
print('Test RFGS Precision Score is %.2f'%RFGS_test_PS,'%')
print('Test RFGS F1 Score is %.2f'%RFGS_test_F1,'%')
print('Test RFGS Recall Score is %.2f'%RFGS_test_RS,'%')
```

```
Test RFGS Accuracy Score is 77.18 %
Test RFGS Precision Score is 65.65 %
Test RFGS F1 Score is 60.64 %
Test RFGS Recall Score is 56.34 %
```

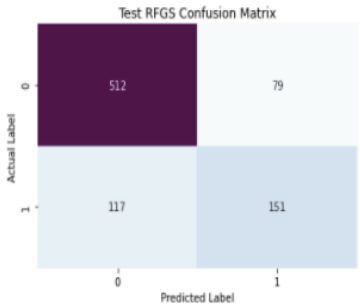
```
#Random Forest Grid AUC Test
rfgs_test_auc = roc_auc_score(y_test, rfgs_ypred_test_prob[:,1])
print("Test RF AUC %.2f" % rfgs_test_auc)

#RF ROC Test

fpr_rfgs_t, tpr_rfgs_t, th_rfgs_t = roc_curve(y_test, rfgs_ypred_test_prob[:,1], pos_label= 1)
plt.plot ([0,1],[0,1], linestyle = '--', color = 'blue', label = 'Unskilled')
plt.plot(fpr_rfgs_t, tpr_rfgs_t, marker = '.', color = 'orange', label = 'Skill')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.title('Receiver Operating Characterestic')
plt.show()
```

```
Test RF AUC 0.80
```




```
# Confusion Report and Classification Matrix to check model performance
print(classification_report(y_test, rfgs_ypred_test))
print("      ")
print("_____________________________________________________")

print("          ")
sns.heatmap(confusion_matrix(y_test, rfgs_ypred_test),annot=True, fmt='d', cbar=False,cmap='BuPu')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title('Test RFGS Confusion Matrix')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84       591
           1       0.66      0.56      0.61       268

    accuracy                           0.77       859
   macro avg       0.74      0.71      0.72       859
weighted avg       0.76      0.77      0.77       859

          
_____________________________________________________
          
```



Neural Network

With a network of neurons emulating the human brain, usually neural networks have an inbuilt mechanism to reduce the error or loss function, called backpropagation. However, a lot depends on the right combination of number of neurons, layers, iterations or epochs, type of activation function and gradient descent.

a. MLP_model = MLPClassifier(hidden_layer_sizes= (20, 30, 50),

max_iter = 51,

activation = 'tanh',

solver = 'adam',

```
        learning_rate = 'adaptive',
        tol = 0.0001,
        verbose = True)
```

With Scaling

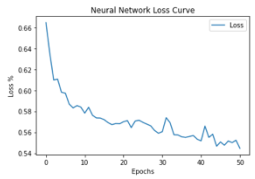
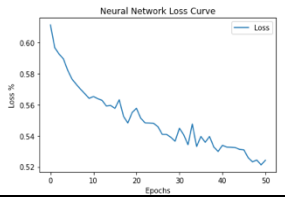
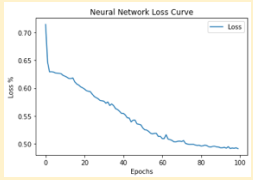
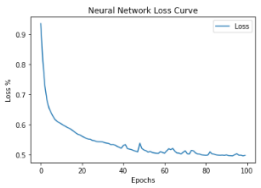
b.
```
MLP_model = MLPClassifier(hidden_layer_sizes= (20, 30, 50),
        max_iter = 51,
                                activation = 'tanh',
        random_state= 567,
                                solver = 'adam',
        learning_rate = 'adaptive',
        tol = 0.0001,
        verbose = True)
```

Without Scaling

c.
```
MLP_model = MLPClassifier(hidden_layer_sizes= (20,30, 50),
        max_iter = 100,
        activation = 'logistic',
        solver = 'adam',
        random_state= 567,
        learning_rate = 'adaptive',
        tol = 0.0001,
        verbose = True)
```

d.
```
MLP_model = MLPClassifier(hidden_layer_sizes= (40, 60, 100),
        max_iter = 200,
        activation = 'tanh',
        solver = 'adam',
                                random_state= 567,
                                learning_rate = 'adaptive',
        tol = 0.0001,
        verbose = True)
```

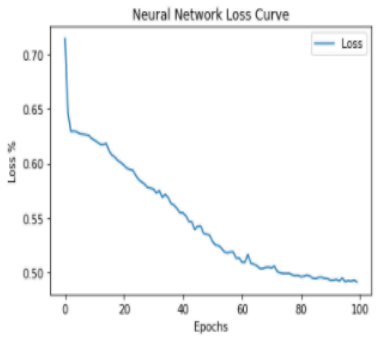
NN	a.	b	c.	d.

Loss function	0.54 	0.52 	0.49 	0.49 
Test AUC	0.70	0.77	0.80	0.78
Test Recall	0.31	0.52	0.58	0.45
Test Precision	0.66	0.59	0.67	0.69
Test F1 Score	0.42	0.55	0.62	0.55

The Model c. is performing better than the others.

The Neural Network Performance of Model c. in Jupyter Notebook is as follows:

```
# Plot the loss curve to see how the model is performing
loss_values = MLP_model.loss_curve_
plt.plot(loss_values, label = 'Loss')
plt.xlabel("Epochs")
plt.ylabel('Loss %')
plt.title("Neural Network Loss Curve")
plt.legend()
plt.show()
```




```
[233] # Make train and test class predictions based on the model
      nn_ytrain_pred = MLP_model.predict(X_train)
      nn_ytest_pred = MLP_model.predict(X_test)

      # Make train and test probability predictions based on the model
      nn_ytrain_pred_prob = MLP_model.predict_proba(X_train)
      nn_ytest_pred_prob = MLP_model.predict_proba(X_test)
```

Neural Network - Train

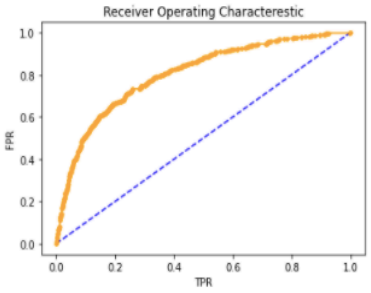
```
[234] # Performance Scores - Train
      NN_train_AS = accuracy_score(y_train, nn_ytrain_pred, normalize = True)*100
      NN_train_PS = precision_score(y_train, nn_ytrain_pred)*100
      NN_train_F1 = f1_score(y_train, nn_ytrain_pred)*100
      NN_train_RS = recall_score(y_train, nn_ytrain_pred)*100

      print('The Train NN Accuracy Score is %.2f'%NN_train_AS,'%')
      print('The Train NN Precision Score is %.2f'%NN_train_PS,'%')
      print('The Train NN F1 Score is %.2f'%NN_train_F1,'%')
      print('The Train NN Recall Score is %.2f'%NN_train_RS,'%')
```

```
The Train NN Accuracy Score is 77.27 %
The Train NN Precision Score is 67.27 %
The Train NN F1 Score is 62.05 %
The Train NN Recall Score is 57.59 %
```

```
# Confusion Report and Classification Matrix to check model performance
print(classification_report(y_train, nn_ytrain_pred))

print("      ")
print("_____________________________________________________")

print("           ")
sns.heatmap(confusion_matrix(y_train, nn_ytrain_pred),annot=True, fmt='d', cbar=False,cmap='BuPu')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title('Train NN Confusion Matrix')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84      1356
           1       0.67      0.58      0.62       646

    accuracy                           0.77      2002
   macro avg       0.74      0.72      0.73      2002
weighted avg       0.77      0.77      0.77      2002
```

```
Train NN AUC 0.80
```



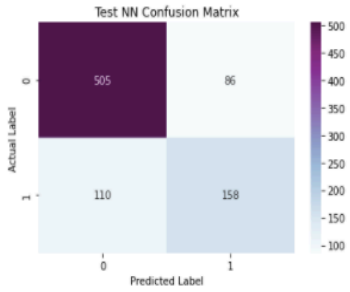
Neural Network - Test

```
# Performance Scores - Test
NN_test_AS = accuracy_score(y_test, nn_ytest_pred, normalize = True)*100
NN_test_PS = precision_score(y_test, nn_ytest_pred)*100
NN_test_F1 = f1_score(y_test, nn_ytest_pred)*100
NN_test_RS = recall_score(y_test, nn_ytest_pred)*100

print('The Test NN Accuracy Score is %.2f'%NN_test_AS,'%')
print('The Test NN Precision Score is %.2f'%NN_test_PS,'%')
print('The Test NN F1 Score is %.2f'%NN_test_F1,'%')
print('The Test NN Recall Score is %.2f'%NN_test_RS,'%')
```

```
The Test NN Accuracy Score is 77.18 %
The Test NN Precision Score is 64.75 %
The Test NN F1 Score is 61.72 %
The Test NN Recall Score is 58.96 %
```

```
[238] # Confusion Report and Classification Matrix to check model performance
      print(classification_report(y_test, nn_ytest_pred))
      print("      ")
      print("_____________________________________________________")
      print("           ")
      sns.heatmap(confusion_matrix(y_test, nn_ytest_pred), annot = True, fmt ='d', cmap = 'BuPu' )
      #sns.heatmap(confusion_matrix(y_train, nn_y_train_pred),annot=True, fmt='d', cbar=False,cmap='BuPu')
      plt.xlabel('Predicted Label')
      plt.ylabel('Actual Label')
      plt.title('Test NN Confusion Matrix')
      plt.show()
```

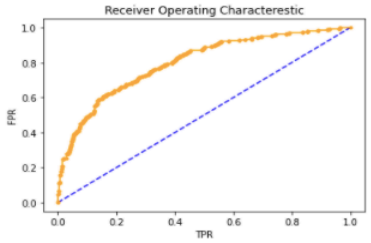
```
              precision    recall  f1-score   support

           0       0.82      0.85      0.84       591
           1       0.65      0.59      0.62       268

    accuracy                           0.77       859
   macro avg       0.73      0.72      0.73       859
weighted avg       0.77      0.77      0.77       859
```




```
[239] #AUC - NN TEST
      nn_test_auc = roc_auc_score(y_test, nn_ytest_pred_prob[:,1])
      print("Test NN AUC %.2f" % nn_test_auc)

      #ROC - NN TEST
      fpr_nn_t, tpr_nn_t, th_nn_t = roc_curve(y_test, nn_ytest_pred_prob[:,1], pos_label= 1)
      plt.plot ([0,1],[0,1], linestyle = '--', color = 'blue', label = 'Unskilled')
      plt.plot(fpr_nn_t, tpr_nn_t, marker = '.', color = 'orange', label = 'Skill')
      plt.ylabel('FPR')
      plt.xlabel('TPR')
      plt.title('Receiver Operating Characterestic')
      plt.show()
```

```
#AUC - NN TEST
nn_test_auc = roc_auc_score(y_test, nn_ytest_pred_prob[:,1])
print("Test NN AUC %.2f" % nn_test_auc)

#ROC - NN TEST
fpr_nn_t, tpr_nn_t, th_nn_t = roc_curve(y_test, nn_ytest_pred_prob[:,1], pos_label= 1)
plt.plot ([0,1],[0,1], linestyle = '--', color = 'blue', label = 'Unskilled')
plt.plot(fpr_nn_t, tpr_nn_t, marker = '.', color = 'orange', label = 'Skill')
plt.ylabel('FPR')
plt.xlabel('TPR')
plt.title('Receiver Operating Characterestic')
plt.show()
```

```
Test NN AUC 0.80
```



Neural Network GridSearch

With GridSearch we hope to find the right hyperparameters, again trying a combination of inputs.

```
parameters = {'solver': ['ibfgs','adam', 'sdg'],
                                'max_iter': [700, 800],
                                'hidden_layer_sizes': np.arange(10,15),
                                 'activation': ['relu', 'logistic', 'tanh', 'identity'],
                                  'tol': [0.001, 0.0001],
                                  'learning_rate': ['constant', 'adaptive']}
```

```
{'activation': 'logistic',
                        'hidden_layer_sizes': 11,
```

'learning_rate': 'constant',
'max_iter': 800,
'solver': 'adam',
'tol': 0.0001}

The Time taken is 718.4615943431854

b. parameters = {'solver': ['ibfgs','adam', 'sdg'],
'max_iter': [800, 900, 1000],
'hidden_layer_sizes': np.arange(9,13),
'activation': ['relu', 'logistic', 'tanh', 'identity'],
'tol': [0.001, 0.0001],
'learning_rate': ['constant', 'adaptive']}

{'activation': 'logistic',
'hidden_layer_sizes': 12,
'learning_rate': 'constant',
'max_iter': 800,
'solver': 'ibfgs',
'tol': 0.001}

(random_state = 123)

The Time taken is 1818.1898846626282

c. parameters = {'solver': ['ibfgs','adam', 'sdg'],
'max_iter': [950, 1000, 1050],
'hidden_layer_sizes': np.arange(5,11),
'activation': ['relu', 'logistic', 'tanh', 'identity'],
'tol': [0.001, 0.0001],
'learning_rate': ['constant', 'adaptive']}

{'activation': 'logistic',
'hidden_layer_sizes': 12,
'learning_rate': 'constant',
'max_iter': 1000,
'solver': 'lbfgs',
'tol': 0.001}

The Time taken is 1756.2832567691803

d. parameters = {'solver': ['ibfgs','adam', 'sdg'],
'max_iter': [800, 950],
'hidden_layer_sizes': np.arange(9,13),
'activation': ['relu', 'logistic', 'tanh', 'identity'],
'tol': [0.001, 0.0001],
'learning_rate': ['constant', 'adaptive']}

{'activation': 'logistic',
'hidden_layer_sizes': 8,
'learning_rate': 'constant',
'max_iter': 950,
'solver': 'ibfgs',
'tol': 0.001}

The Time taken is 845.8169329166412

NN Grid	a.	b.	c.	d.
Test AUC	0.82	0.81	0.83	0.78
8Test Recall	0.54	0.60	0.62	0.60
Test Precision	0.70	0.68	0.69	0.67
Test F1 Score	0.61	0.64	0.62	0.63

The Model c. is performing better than the others. Of the Neural Network GridSearch selected Models, c. is performing better in Jupyter Notebook.

▾ Neural Network - GridSearch

```
[52] # Define the hyperparameters
     parameters = {'solver': ['lbfgs','adam', 'sdg'],
                   'max_iter': [950, 1000, 1050],
                   'hidden_layer_sizes': np.arange(9,13),
                   'activation': ['relu', 'logistic', 'tanh', 'identity'],
                   'tol': [0.001, 0.0001],
                   'learning_rate': ['constant', 'adaptive']}
```

```
[53] # Train the model, then fit it with training data
     import time
     start_time = time.time()
     NN_grid_model = GridSearchCV(MLPClassifier(random_state = 123), parameters, n_jobs=-1, cv=10)
     NN_grid_model.fit(X_train, y_train)
     print('The Time taken is', (time.time()-start_time))
```

```
     The Time taken is 1756.2832567691803
```

```
[54] # Measure Model score on train data
     print(NN_grid_model.score(X_train,y_train))
```

```
     0.7938095238095239
```

```
[64] # Best parameters in GS
     NN_grid_model.best_params_
```

```
     {'activation': 'logistic',
      'hidden_layer_sizes': 12,
      'learning_rate': 'constant',
      'max_iter': 1000,
      'solver': 'lbfgs',
      'tol': 0.001}
```

```
[65] # Best parameters in NN Grid Search
     NN_grid_model.best_estimator_
```

```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=12, learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=1000,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=123, shuffle=True, solver='lbfgs',
              tol=0.001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

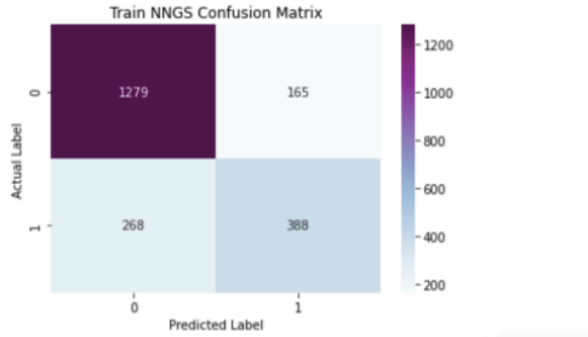
```
[66] # Make train and test class predictions based on the model
     nngs_ytrain_pred = NN_grid_model.best_estimator_.predict(X_train)
     nngs_ytest_pred= NN_grid_model.best_estimator_.predict(X_test)

     # Make train and test probability predictions based on the model
     nngs_ytrain_pred_prob = NN_grid_model.best_estimator_.predict_proba(X_train)
     nngs_ytest_pred_prob = NN_grid_model.best_estimator_.predict_proba(X_test)
```

```
[67] # Performance Scores - Train
     NNGS_train_AS = accuracy_score(y_train, nngs_ytrain_pred, normalize = True)*100
     NNGS_train_PS = precision_score(y_train, nngs_ytrain_pred,)*100
     NNGS_train_F1 = f1_score(y_train, nngs_ytrain_pred,)*100
     NNGS_train_RS = recall_score(y_train, nngs_ytrain_pred,)*100

     print('Train NNGS Accuracy Score is %.2f'%NNGS_train_AS,'%')
     print('Train NNGS Precision Score is %.2f'%NNGS_train_PS,'%')
     print('Train NNGS F1 Score is %.2f'%NNGS_train_F1,'%')
     print('Train NNGS Recall Score is %.2f'%NNGS_train_RS,'%')
```

```
Train NNGS Accuracy Score is 79.38 %
Train NNGS Precision Score is 70.16 %
Train NNGS F1 Score is 64.19 %
Train NNGS Recall Score is 59.15 %
```

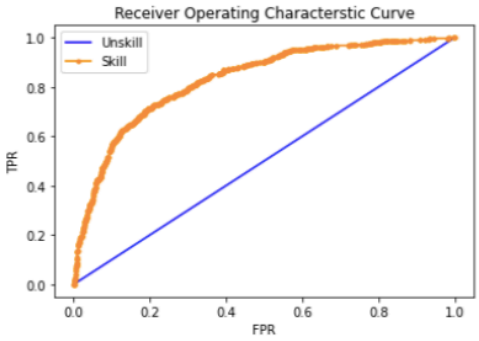
```
#Confusion Report and Classification Matrix to check model performance
print(classification_report(y_train, nngs_ytrain_pred))
print("      ")
print("_____________________________________________________________")
print("           ")
sns.heatmap(confusion_matrix(y_train, nngs_ytrain_pred), annot = True, fmt ='d', cmap = 'BuPu' )
#sns.heatmap(confusion_matrix(y_train, nn_y_train_pred),annot=True, fmt='d', cbar=False,cmap='BuPu')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title('Train NNGS Confusion Matrix')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1444
           1       0.70      0.59      0.64       656

    accuracy                           0.79      2100
   macro avg       0.76      0.74      0.75      2100
weighted avg       0.79      0.79      0.79      2100
```



```
[69]
# ROC and AUC - NN GRID Train
nngs_train_auc = roc_auc_score(y_train, nngs_ytrain_pred_prob[:,1])
print('Train NNGS AUC Score: %.2f' % nngs_train_auc)

fpr_nngs_tr, tpr_nngs_tr, th_nngs_tr = roc_curve(y_train, nngs_ytrain_pred_prob[:,1], pos_label = 1)
plt.plot([0,1], [0,1], linestyle = '-', label = 'Unskill', color = 'blue')
plt.plot(fpr_nngs_tr, tpr_nngs_tr, marker = '.', color = 'darkorange', label = "Skill")
plt.xlabel ("FPR")
plt.ylabel ("TPR")
plt.title ("Receiver Operating Characterstic Curve")
plt.legend()
plt.show()
```

```
Train NNGS AUC Score: 0.83
```

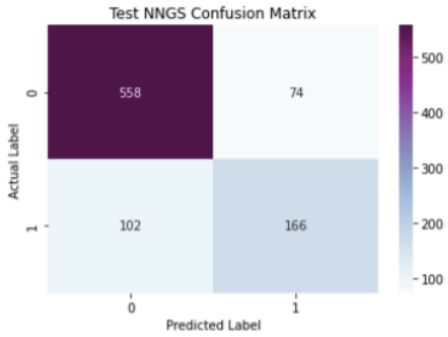



```
[70] # Performance Scores - Test
NNGS_test_AS = accuracy_score(y_test, nngs_ytest_pred, normalize = True)*100
NNGS_test_PS = precision_score(y_test, nngs_ytest_pred)*100
NNGS_test_F1 = f1_score(y_test, nngs_ytest_pred)*100
NNGS_test_RS = recall_score(y_test, nngs_ytest_pred)*100

print('Test NNGS Accuracy Score is %.2f'%NNGS_test_AS,'%')
print('Test NNGS Precision Score is %.2f'%NNGS_test_PS,'%')
print('Test NNGS F1 Score is %.2f'%NNGS_test_F1,'%')
print('Test NNGS Recall Score is %.2f'%NNGS_test_RS,'%')
```

```
Test NNGS Accuracy Score is 80.44 %
Test NNGS Precision Score is 69.17 %
Test NNGS F1 Score is 65.35 %
Test NNGS Recall Score is 61.94 %
```

```
#Confusion Report and Classification Matrix to check model performance
print(classification_report(y_test, nngs_ytest_pred))
print("      ")
print("_____________________________________________________________")
print("           ")
sns.heatmap(confusion_matrix(y_test, nngs_ytest_pred), annot = True, fmt ='d', cmap = 'BuPu' )
#sns.heatmap(confusion_matrix(y_train, nn_y_train_pred),annot=True, fmt='d', cbar=False,cmap='BuPu')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title('Test NNGS Confusion Matrix')
plt.show()
```

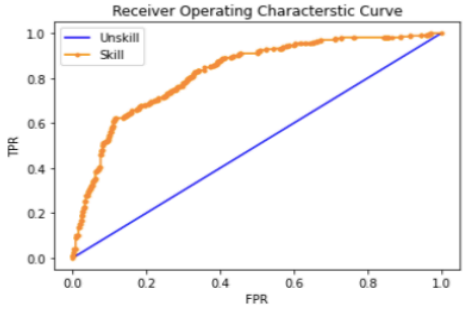
```
              precision    recall  f1-score   support

           0       0.85      0.88      0.86       632
           1       0.69      0.62      0.65       268

    accuracy                           0.80       900
   macro avg       0.77      0.75      0.76       900
weighted avg       0.80      0.80      0.80       900
```




```
# AUC NN GRID TEST
nngs_test_auc = roc_auc_score(y_test, nngs_ytest_pred_prob[:,1])
print('Test NNGS AUC Score: %.2f' % nngs_test_auc)


# ROC NN GRID TEST
fpr_nngs_t, tpr_nngs_t, th_nngs_t = roc_curve(y_test, nngs_ytest_pred_prob[:,1], pos_label = 1)
plt.plot([0,1], [0,1], linestyle = '-', label = 'Unskill', color = 'blue')
plt.plot(fpr_nngs_t, tpr_nngs_t, marker = '.', color = 'darkorange', label = "Skill")
plt.xlabel ("FPR")
plt.ylabel ("TPR")
plt.title ("Receiver Operating Characterstic Curve")
plt.legend()
plt.show()
```

```
[72] # AUC NN GRID TEST
     nngs_test_auc = roc_auc_score(y_test, nngs_ytest_pred_prob[:,1])
     print('Test NNGS AUC Score: %.2f' % nngs_test_auc)


     # ROC NN GRID TEST
     fpr_nngs_t, tpr_nngs_t, th_nngs_t = roc_curve(y_test, nngs_ytest_pred_prob[:,1], pos_label = 1)
     plt.plot([0,1], [0,1], linestyle = '-', label = 'Unskill', color = 'blue')
     plt.plot(fpr_nngs_t, tpr_nngs_t, marker = '.', color = 'darkorange', label = "Skill")
     plt.xlabel ("FPR")
     plt.ylabel ("TPR")
     plt.title ("Receiver Operating Characterstic Curve")
     plt.legend()
     plt.show()
```

```
Test NNGS AUC Score: 0.83
```



Q4. Final Model

Compare all the model and write an inference which model is best/optimized.

In this section, we share all the different models that we have built, namely-

- Decision Tree / CART
- Decision Tree GridSearch
- Random Forest GridSearch

- Artificial Neural Network
- ANN GridSearch

Decision Tree / CART Model

Decision Tree is an algorithm that can be used for both classification or regression problems. It does this by creating homogeneous nodes or leaves, so we can predict the outcome of a set of conditions.

Decision Tree (CART) Model

```
#Declare the CART model
dtcl = DecisionTreeClassifier(criterion = 'gini', random_state = 12)
#Fit the data to the model
modell = dtcl.fit(X_train, y_train)
```

```
#Feature importances according the the CART model
modell.feature_importances_
# See Feature Importance visually
modell_a = pd.DataFrame(modell.feature_importances_, columns = ["Importance"], index = df_predictors.columns).sort_values("Importance", ascending = False)
plt.figure(figsize=(10,7))
sns.barplot(x = modell_a.Importance, y = modell_a.index, palette='vlag')
plt.ylabel('Feature Name')
plt.xlabel('Feature Importance')
plt.title('Feature Importance Plot')
plt.show()
```

With the feature_importances_ attribute helps identify the features that are important to the target variable in terms of identifying the patterns that are leading to cases being claims positive.

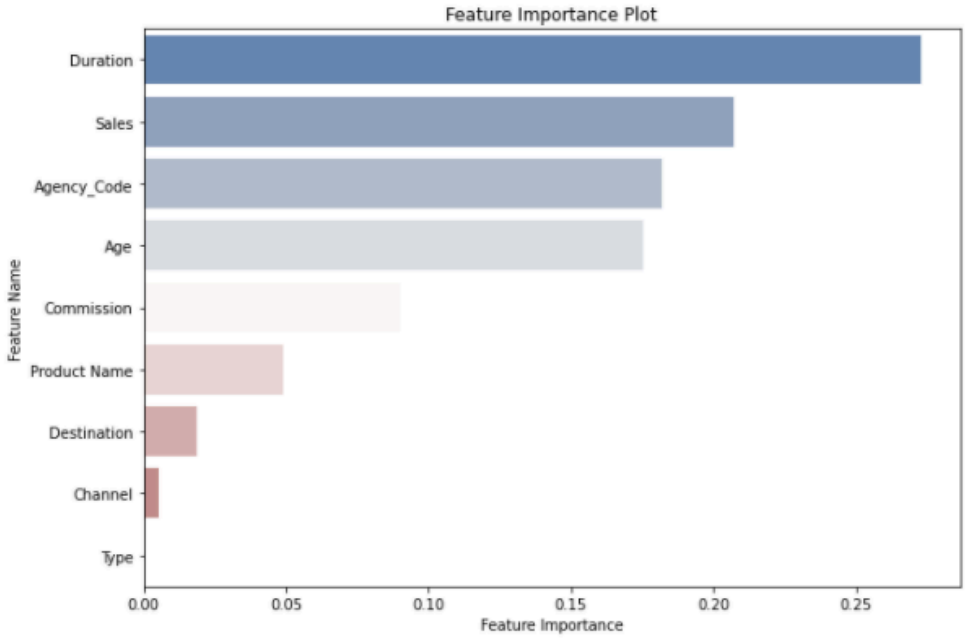
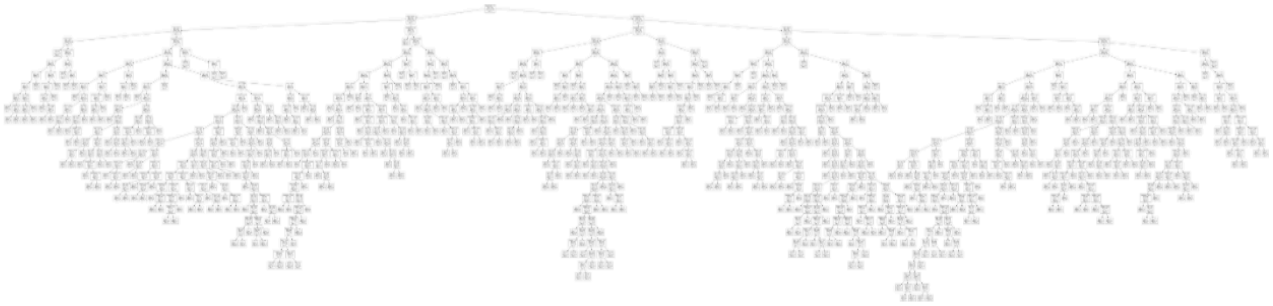


Fig. CART Feature importance

When we construct the Tree, it grows to its fullest possible extent (as expected). Growing a Decision Tree to its fullest leads to ineffective results, as the model learns so perfectly from the training data, that it doesn't perform well in the unseen datasets. The model learning from the 'noise' of the data causes the model to underperform in unseen data.

```
[114] #Construct the tree
      from sklearn.tree import export_graphviz
      from IPython.display import Image
      import pydotplus

      tree_viz = export_graphviz(model1, feature_names = df_predictors.columns, class_names = ['0','1'])
      tree_plot = pydotplus.graph_from_dot_data(tree_viz)

      #Display the Decision Tree
      Image(tree_plot.create_png())
```



```
# Make class predictions
dt_ytrain_pred = model1.predict(X_train)
dt_ytest_pred = model1.predict(X_test)

# Make probability predictions
dt_ytrain_pred_prob = model1.predict_proba(X_train)
dt_ytest_pred_prob = model1.predict_proba(X_test)
```

Decision Tree GridSearch

To machine learning model perform properly on test or unseen data, the model can be pruned, meaning, some aspects of the data to be muted by externally controlling some parameters called the hyperparameters. Therefore, the hyperparameters, which are defined by the data scientist before the model is trained, are particularly important in a data science project.

▾ Decision Tree (CART) - GridSearch

```
[215] dtcl1 = DecisionTreeClassifier()

      #Lets set the grid of hyperparameters
      param_grid ={'criterion':['gini', 'entropy'],
                   'max_depth':range(4, 6),
                   'min_samples_split':[60, 70, 80, 90, 95],
                   'min_samples_leaf': [30, 35]}
      dtgrid = GridSearchCV(estimator = dtcl1, param_grid = param_grid, cv = 10, verbose = 1)
      dtgrid.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 40 candidates, totalling 400 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 400 out of 400 | elapsed:    2.2s finished
GridSearchCV(cv=10, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(4, 6), 'min_samples_leaf': [30, 35],
                         'min_samples_split': [60, 70, 80, 90, 95]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=1)
```

```
[216] ##Identify the best hyperparameters assessed by GridSearch
      dtgrid.best_params_
```

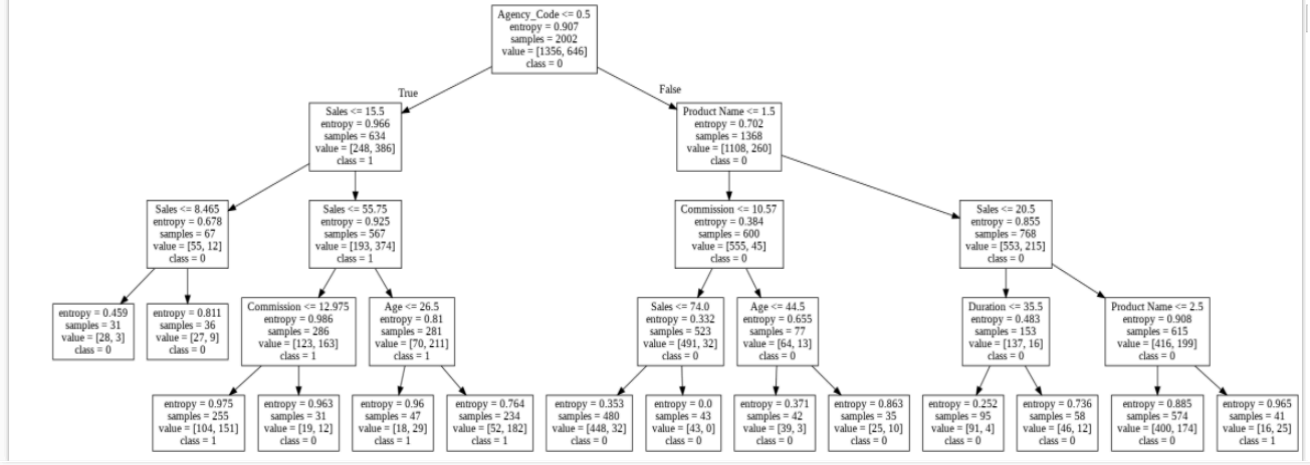
```
{'criterion': 'entropy',
 'max_depth': 4,
 'min_samples_leaf': 30,
 'min_samples_split': 60}
```

The Decision Tree GridSearch has calculated the best hyperparameters to be {'criterion': 'entropy',
'max_depth': 4,
'min_samples_leaf': 30,
'min_samples_split': 60}

In the GridSearch, max_depth, an important hyperparameter which decides the level to which the tree will grow, has been defined between 4-10. Similarly, we have chosen min_sample_leaf to be no less than 30, to ensure we have critical mass for our terminal nodes and accuracy is better.

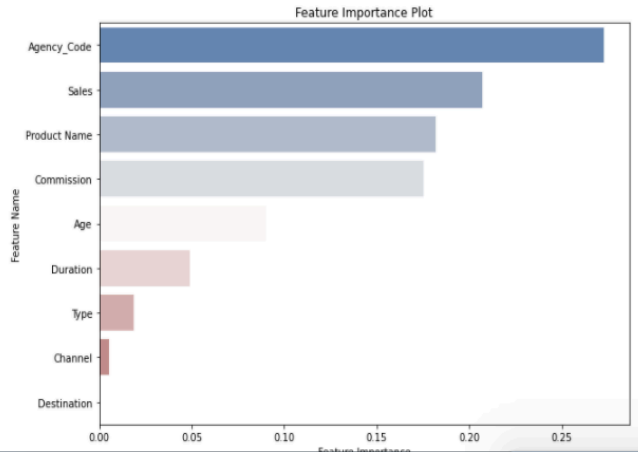
```
[217] # Build the model using the best parameters and fit the data
      dtgrid1 = DecisionTreeClassifier(criterion = dtgrid.best_params_.get('criterion'), random_state = 12,
                                       max_depth = dtgrid.best_params_.get('max_depth'), min_samples_leaf = dtgrid.best_params_.get('min_samples_leaf'),
                                       min_samples_split = dtgrid.best_params_.get('min_samples_split'))
      dtgrid2 = dtgrid1.fit(X_train, y_train)
```

```
#Construct the tree

tree_viz_gs = export_graphviz(dtgrid2, feature_names = df_predictors.columns, class_names = ['0','1'])

tree_plot = pydotplus.graph_from_dot_data(tree_viz_gs)

#Display the Decision Tree
Image(tree_plot.create_png())
```




```
[212] # Make class predictions based on the model
      dtgs_ytrain_pred = dtgrid2.predict(X_train)
      dtgs_ytest_pred = dtgrid2.predict(X_test)

      # Make probability predictions
      dtgs_ytrain_pred_prob = dtgrid2.predict_proba(X_train)
      dtgs_ytest_pred_prob = dtgrid2.predict_proba(X_test)
```

```
[213] #Feature importances according the CART model
      dtgrid2.feature_importances_
      # See Feature Importance visially
      dtgrid2_a = pd.DataFrame(dtgrid2.feature_importances_, columns = ["Importance"], index = df_predictors.columns).sort_values("Importance", ascending = False)
      plt.figure(figsize=(10,7))
      sns.barplot(x = model1_a.Importance, y = dtgrid2_a.index, palette='vlag')
      plt.ylabel('Feature Name')
      plt.xlabel('Feature Importance')
      plt.title('Feature Importance Plot')
      plt.show()
```



Notice that the features considered important are quite different in DT GridSearch than in DT. If we want we can use only the top 5, which would account for 90% relative importance. We do not do that here, since the number of datapoints are very less, and added to that we have an imbalanced dataset.

Random Forest

Random Forest is an ensemble supervised learning that uses multiple decision trees for making decisions (hence the name). The advantage of a Random Forest is because it grows many Decision Tress, it doesn't rely on the result of one, but hundreds of tress, therefore maximizing the chances of arriving at the right model.

However, in contrast to Decision Trees, which are quite intuitive and easy to interpret and explain, Random Forests are considered 'black box' algorithms, meaning that we cannot explain the working of the algorithm intuitively.

Although, we will evaluate model performance in the next section. A preliminary OOB test suggests that the Random Forest seems to be performing at around 76%. We would like to better the performance by tuning the hyperparameters with GridSearch.

▾ Random Forest - GridSearch

```
[ ] # Define parameters for the RF Grid Search
    param_grid = {'n_estimators': [100, 150, 200, 250],
                  'max_depth': [6,7,8],
                  'min_samples_split' : [90, 120, 150, 155],
                  'min_samples_leaf':[30, 40, 50],
                  'max_features': [4,5, 6,7],
                  'criterion': ['gini', 'criterion']}
```

```
[ ] # Call the model, define the hyperparamters
    rf_model1 = RandomForestClassifier()
    rfgrid = GridSearchCV(estimator = rf_model1, param_grid =param_grid, cv = 10)
```

```
[ ] # Fit the model with train data, calculate the model making time
    start_time = time.time()
    rfgrid.fit(X_train, y_train)
    print('The time taken is', (time.time() - start_time))
```

```
    The time taken is 2695.7062180042267
```

```
[ ] # Finding the best parameters
    rfgrid.best_params_
```

```
    {'criterion': 'gini',
     'max_depth': 7,
     'max_features': 6,
     'min_samples_leaf': 50,
     'min_samples_split': 90,
     'n_estimators': 100}
```

```
[ ] # Finding the best model
    rfbestgrid = rfgrid.best_estimator_
```

```
[ ] # Make class predictions with the best model
    rfgs_ypred_train = rfbestgrid.predict(X_train)
    rfgs_ypred_test = rfbestgrid.predict(X_test)

    # Make probability predictions with the best model
    rfgs_ypred_train_prob = rfbestgrid.predict_proba(X_train)
    rfgs_ypred_test_prob = rfbestgrid.predict_proba(X_test)
```

```
    0    1494
    1     508
    dtype: int64
```

```
[350] pd.Series(rfgs_ypred_test).value_counts()
```

```
    0    636
    1    223
    dtype: int64
```

It is important to note again that the model is using only around 500 datapoints to learn. We will investigate this in model performance evaluation in the next question, however, this may be insufficient in model building.

Neural Network

Neural networks are algorithms that mimic the operations of a human brain in processing data and producing a model. A neural network is a network of nodes or neurons (like in the brain), where each node is a connected with the other. These nodes are called *perceptrons*. Just like our brain has neurons that help in building and connecting thought, the ANN has perceptrons

that accepts information from the input or a previous hidden layer, processes them and passes it on to another hidden layer or the output layer.
The connection with each node or neuron is calculated by assigning a random weight. To that a *bias* is added, to protect the functioning of the neuron, just in case the weights are selected as zero in the backpropagation.

Much like the neurons in our brain which activate if something is important, *activation functions* are non-linear mathematical equations that are attached to the perceptron. They determine whether a neural network should be activated or not, based on whether each neuron's input is relevant for the model's prediction. Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1.
Like the brain 'learns' from trial and error, after the result is acquired, the algorithm compares it with the actual result and then goes back to update the weights to improve on its performance and minimize the error. This is called *backpropagation* and a since forward and backward cycle is called *epoch*.
Neural Network uses Gradient Descent as an optimization algorithm to find the values of coefficients that minimizes a cost function or error. In other words, this is how the Neural Network 'learns' and tries to better its predictive power.
Unlike Decision Trees, Neural Networks are a black box ML technique, because it provides no insights for interpretation of the results. In case, the business needs explanation or any consultation for course correction, Neural Network do not provide any intuition or understanding
Scaling is necessary before Neural Network model creation, since it is very sensitive to unscaled data, assigning more importance or weights to larger numbers than smaller.

▾ NEURAL NETWORK

```
# Scaling
scaling = StandardScaler()
X_train = scaling.fit_transform(X_train)
X_test = scaling.transform(X_test)

# Separating the Independent and target variables
df_predictors = data.drop('Claimed', axis = 1)
df_target = data['Claimed']


X_train, X_test, y_train, y_test = train_test_split(df_predictors, df_target, test_size = 0.3, random_state = 123)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)


# Define hyperparamters
MLP_model = MLPClassifier(hidden_layer_sizes= (40, 60, 100),
                          max_iter = 200,
                          activation = 'tanh',
                          solver = 'adam',
                          learning_rate = 'adaptive',
                          tol = 0.0001,
                          verbose = True)
```

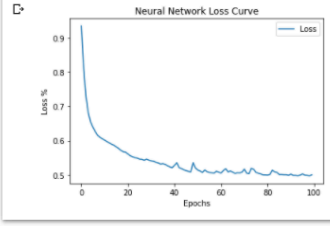
```
(2100, 9)
(2100,)
(900, 9)
(900,)
```

```
[ ] # Fit the Neural Network model with train data
    MLP_model.fit(X_train, y_train)
```

```
Iteration 1, loss = 0.62755595
Iteration 2, loss = 0.56214135
Iteration 3, loss = 0.55793968
Iteration 4, loss = 0.54346305
Iteration 5, loss = 0.54000668
Iteration 6, loss = 0.54576328
Iteration 7, loss = 0.55655130
Iteration 8, loss = 0.56365260
Iteration 9, loss = 0.54269304
Iteration 10, loss = 0.53298223
Iteration 11, loss = 0.52769773
Iteration 12, loss = 0.51908626
Iteration 13, loss = 0.51644325
Iteration 14, loss = 0.51568168
Iteration 15, loss = 0.50387478
Iteration 16, loss = 0.51222218
Iteration 17, loss = 0.50908064
Iteration 18, loss = 0.49960810
Iteration 19, loss = 0.49601207
Iteration 20, loss = 0.49493660
Iteration 21, loss = 0.50328221
Iteration 22, loss = 0.51682882
Iteration 23, loss = 0.52903692
Iteration 24, loss = 0.49400535
Iteration 25, loss = 0.49288987
Iteration 26, loss = 0.48935537
Iteration 27, loss = 0.48944667
Iteration 28, loss = 0.48573169
Iteration 29, loss = 0.52149707
Iteration 30, loss = 0.50242024
Iteration 31, loss = 0.48976065
Iteration 32, loss = 0.48631448
Iteration 33, loss = 0.48905477
Iteration 34, loss = 0.48828601
Iteration 35, loss = 0.49835371
Iteration 36, loss = 0.54556063
Iteration 37, loss = 0.54334613
Iteration 38, loss = 0.49305346
Iteration 39, loss = 0.50115681
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
MLPClassifier(activation='tanh', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(40, 60, 100), learning_rate='adaptive',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=123, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=True,
              warm_start=False)
```

After multiple iterations, we have found that the model is not able to learn very successfully. As it shows above, the inconsistency in the loss function signify that it is struggling to learn appropriately.
We ran the model with multiple combinations of hyperparameters. These are listed in a table in the next section.

```
# Plot the loss curve to see how the model is performing
loss_values = MLP_model.loss_curve_
plt.plot(loss_values, label = 'Loss')
plt.xlabel("Epochs")
plt.ylabel('Loss %')
plt.title("Neural Network Loss Curve")
plt.legend()
plt.show()
```




```
[42] # Make train and test class predictions based on the model
     nn_ytrain_pred = MLP_model.predict(X_train)
     nn_ytest_pred = MLP_model.predict(X_test)

     # Make train and test probability predictions based on the model
     nn_ytrain_pred_prob = MLP_model.predict_proba(X_train)
     nn_ytest_pred_prob = MLP_model.predict_proba(X_test)
```

Next, we run Neural Network with GridSearch, hoping to identify the correct parameters for better model performance.

Neural Network - GridSearch

```
# Define the hyperparameters
parameters = {'solver': ['ibfgs','adam', 'sdg'],
              'max_iter': [950, 1000, 1050],
              'hidden_layer_sizes': np.arange(9,13),
              'activation': ['relu', 'logistic', 'tanh', 'identity'],
              'tol': [0.001, 0.0001],
              'learning_rate': ['constant', 'adaptive']}
```

```
# Train the model, then fit it with training data
import time
start_time = time.time()
NN_grid_model = GridSearchCV(MLPClassifier(), parameters, n_jobs=-1, cv=10)
NN_grid_model.fit(X_train, y_train)
print('The Time taken is', (time.time()-start_time))
```

```
The Time taken is 845.8169329166412
```

```
# Measure Model score on train data
print(NN_grid_model.score(X_train,y_train))
```

```
0.7871428571428571
```

```
# Best parameters in GS
NN_grid_model.best_params_
```

```
{'activation': 'logistic',
 'hidden_layer_sizes': 10,
 'learning_rate': 'adaptive',
 'max_iter': 1000,
 'solver': 'adam',
 'tol': 0.0001}
```

```
# Best parameters in NN Grid Search
NN_grid_model.best_estimator_
```

```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=10, learning_rate='adaptive',
              learning_rate_init=0.001, max_fun=15000, max_iter=1000,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

```
# Make train and test class predictions based on the model
nngs_ytrain_pred = NN_grid_model.best_estimator_.predict(X_train)
nngs_ytest_pred= NN_grid_model.best_estimator_.predict(X_test)

# Make train and test probability predictions based on the model
nngs_ytrain_pred_prob = NN_grid_model.best_estimator_.predict_proba(X_train)
nngs_ytest_pred_prob = NN_grid_model.best_estimator_.predict_proba(X_test)
```

We have built various models in this section, all performance metrics are outlined in the next section.

Final Model Comparison

The performance metric we had selected were:

Test AUC
Test Recall Score of Class 1
Test F1 Score of Class 1
Test Precision Score of Class 1

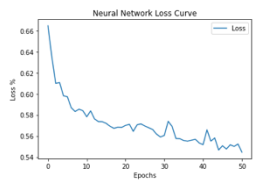
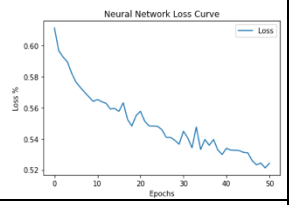
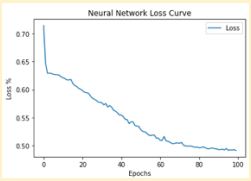
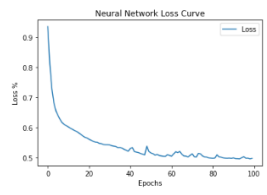
Let us compare how the different models and techniques have performed on these parameters:

DT	a.
Test AUC	0.66
Test Recall	0.50
Test Precision	0.53
Test F1 Score	0.51

DT Grid	a.	b.	c.	d.
Test AUC	0.79	0.79	0.79	0.79
Test Recall	0.60	0.63	0.63	0.63
Test Precision	0.65	0.66	0.66	0.66
Test F1 Score	0.62	0.64	0.64	0.64

RF	a.	b.	c.	d.
Test AUC	0.79	0.80	0.80	0.80
Test Recall	0.55	0.60	0.61	0.60
Test Precision	0.66	0.67	0.66	0.66
Test F1 Score	0.60	0.63	0.63	0.63

RF Grid	a.	b	c.	d.
Test AUC	0.84	0.80	0.80	0.80
Test Recall	0.59	0.54	0.53	0.54
Test Precision	0.67	0.67	0.65	0.65
Test F1 Score	0.63	0.60	0.59	0.59

NN	a.	b	c.	d.
Loss function	0.54 	0.52 	0.46 	0.49 
Test AUC	0.70	0.77	0.80	0.78
Test Recall	0.31	0.52	0.59	0.45
Test Precision	0.66	0.59	0.70	0.69
Test F1 Score	0.42	0.55	0.64	0.55

NN Grid	a.	b.	c.	d.
Test AUC	0.82	0.81	0.83	0.78
8Test Recall	0.54	0.60	0.62	0.60
Test Precision	0.70	0.68	0.69	0.67
Test F1 Score	0.61	0.64	0.62	0.63

Below is the summary performance by the best models of all the techniques:

Best Parameters	DT	DTGrid	RF	RFGrid	NN	NNGrid

Test AUC	0.66	0.79	0.80	0.84	0.80	0.83
Test Recall(Class1)	0.50	0.63	0.61	0.59	0.59	0.62
Test Precision(Class1)	0.53	0.66	0.66	0.67	0.70	0.69
Test F1 Score(Class1)	0.51	0.64	0.63	0.63	0.64	0.62

	DT Train	DT Test	DTGS Train	DTGS Test	RF Train	RF Test	RFGS Train	RFGS Test	NN Train	NN Test	NNGS Train	NNGS Test
Accuracy	99.29	71.89	78.38	71.89	78.76	79.33	78.81	79.22	78.86	80.33	79.38	80.44
AUC	1.00	0.66	0.81	0.83	0.83	0.83	0.83	0.84	0.82	0.82	0.83	0.83
Recall	97.87	50.00	97.87	50.00	57.01	59.70	56.55	58.96	55.64	58.96	59.15	61.94
Precision	99.84	52.96	67.91	52.96	69.52	67.23	69.87	67.23	70.46	70.22	70.16	69.17
F1 Score	98.85	51.44	62.79	51.44	62.65	63.24	62.51	62.82	62.18	64.10	64.19	65.35

According to the above numbers, the best model was found to be model c. of Neural Network GridSearch.

Although Test recall is higher with DT Grid, Test AUC and Test Precision both outperformed with the Neural network model.

We started earlier that Recall is very important to this prediction after AUC Score, so we chose the one with relatively high AUC (highest being RF Grid), because Recall, Precision and F1 Score are well performing in the NNGrid model.

Q5. Inference

Basis on these predictions, what are the business insights and recommendations?

Generally for a data mining project a large dataset is necessary for the model development. Our dataset being fairly small, (at 2861 it is very small, compared to 50,000, 1,00,000 or 1 mn rows).
Perhaps that is the reason, despite making 21 models using 3 Machine Learning Techniques, we haven't yet been able to teach a model beyond 0.62/0.63 rate of success how to not predict a Claims positive case as a Claims negative one.

The recommendation to the business is to
Gather more than 5-10X the data
Use the chosen Neural Network model in the meantime to predict Claims status of each case.
The initial Decision Tree had stated Duration, Sales, Agency Code and Age to be the top 4 in feature importance and Agency Code, Sales, Product Name and Commission to be the top 4 in feature importance according to DT GridSearch algorithm

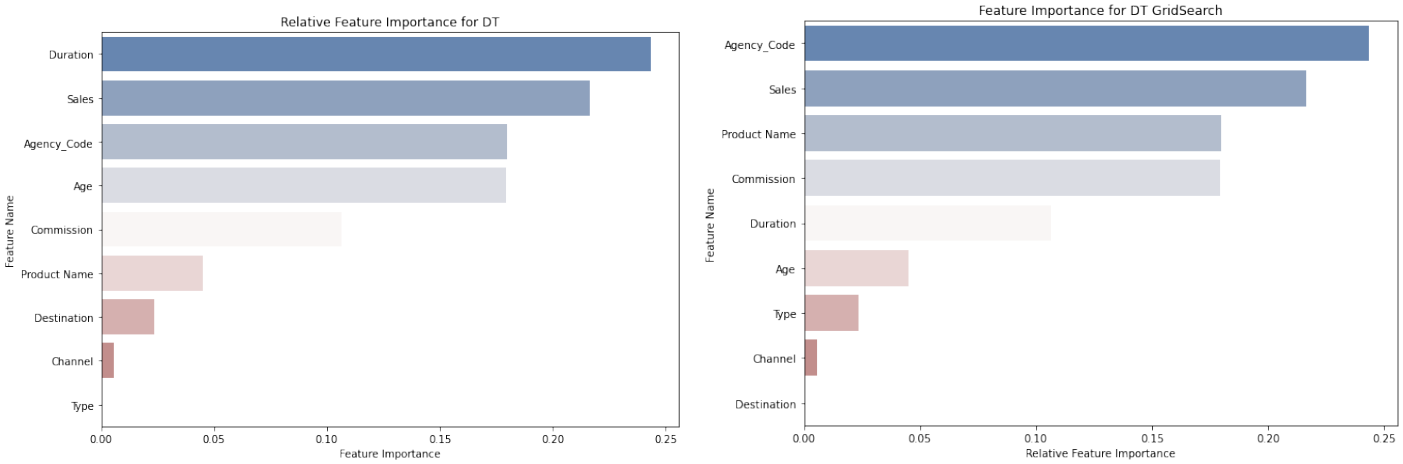


Fig. Top features influencing Claim, according to different techniques

Since comparatively Decision Tree did not perform very poorly, these feature importances can still be considered.

* * *