Deakin University

SIG788- OnTrack Submission

Task 1.2 P

Submitted by

Neethu Sidhardhan
s223494027
Attempt # 1
Date: 19th March, 2023

Target Grade : (For Tasks with split focus)

Task Details –

1. What is the selected dataset and what is related problem for this dataset? You need to provide details of datasets, dataset description, what are the features, output (Class label) and discuss the problem needs to be solved by machine learning model. (Minimum 200 words)

2. You need to provide the screenshot of the built ML pipeline (Data ingestion, Data preparation, model training and evaluating the model). You need to provide explanation for cell by cell of the code.

3. What is the performance of the build model/ models? You need to provide discussion and justification of how the model is performing (discuss different matrices, accuracy confusion matrix) based on the selected dataset.

4. You need to compare the performance of the models and provide justifications which mode is performing better and why.

Solutions for the Task Details: -

1. **What is the selected dataset and what is related problem for this dataset? You need to provide details of datasets, dataset description, what are the features, output (Class label) and discuss the problem needs to be solved by machine learning model. (Minimum 200 words)**

We have selected dataset based on Travel Insurance. An Insurance firm providing travel insurance for specific destination is facing higher claim frequency. The insurance team decides to collect data from the past few years and analyse the data based on the below given attributes from which source this high claim request is coming. The dataset has list of customers who has claimed the insurance based on online and offline channels for their

travel destination to Asis, America and Europe. We have to predict based on the dataset, which attributes can lead towards higher insurance claim. With this dataset, we have done Machine Learning models like Decision Tree (CART) and Random Forest.

Attribute Information:
1. Target: Claim Status (Claimed)
2. Code of tour firm (Agency_Code)
3. Type of tour insurance firms (Type)
4. Distribution channel of tour insurance agencies (Channel)
5. Name of the tour insurance products (Product)
6. Duration of the tour (Duration)
7. Destination of the tour (Destination)
8. Amount of sales of tour insurance policies (Sales)
9. The commission received for tour insurance firm (Commission)
10. Age of insured (Age)

Data set: insurance_part2_data-1.csv

Data Set Detailing: The dataset has 3000 rows and 10 columns. Our targeted group is Claim status. Based on this we will analyse the data by using CART and Random Forest. Data set also included the details by which source the insurance was provided to each traveller. Age is also one of the attributes that can help us in analysing which age group of travellers are going for claims and which are more likely not to claim the insurance.

2. **You need to provide the screenshot of the built ML pipeline (Data ingestion, Data preparation, model training and evaluating the model). You need to provide explanation for cell by cell of the code**.

Import all libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
```

Figure 1: Import Libraries

Import Dataset

```
df = pd.read_csv("insurance_part2_data.csv")
```

```
df.head()
```

	Age	Agency_Code	Type	Claimed	Commision	Channel	Duration	Sales	Product Name	Destination
0	48	C2B	Airlines	No	0.70	Online	7	2.51	Customised Plan	ASIA
1	36	EPX	Travel Agency	No	0.00	Online	34	20.00	Customised Plan	ASIA
2	39	CWT	Travel Agency	No	5.94	Online	3	9.90	Customised Plan	Americas
3	36	EPX	Travel Agency	No	0.00	Online	4	26.00	Cancellation Plan	ASIA

Figure 2: Import Data Set

Data set consist of 10 rows and 3000 rows. The data has attributes like age, agency code, type of bookings, status of claims, commission, channel of booking, duration, sales, product name and travel destination.

We do not require the column “Agency Code” for processing the CART and Random Forest models. Hence, we will drop the column from the dataset

```
df=df.drop(["Agency_Code"],axis=True)
```

The data set do not have any missing values. The data type of the dataset is having a combination of Object variables, Integer and Float variables. For processing the CART and RF we need to convert the Object data type to categorical variables.

```
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Age           3000 non-null   int64  
 1   Type          3000 non-null   object 
 2   Claimed       3000 non-null   object 
 3   Commision     3000 non-null   float64
 4   Channel       3000 non-null   object 
 5   Duration      3000 non-null   int64  
 6   Sales         3000 non-null   float64
 7   Product Name  3000 non-null   object 
 8   Destination   3000 non-null   object 
dtypes: float64(2), int64(2), object(5)
```

Figure 3: Data Types

Checking for Duplicates:

```
print('Number of rows before discarding duplicates = %d' % (df.shape[0]))
df.drop_duplicates(subset = None, keep = 'first', inplace=True)
print('Number of rows after discarding duplicates = %d' % (df.shape[0]))
```

```
Number of rows before discarding duplicates = 3000
Number of rows after discarding duplicates = 2861
```

Figure 4: Duplicates

The dataset has 139 duplicates. After removing the duplicates the , the dataset has 2861 rows and 9 columns.

Covert the data type "Objects" to "Categorical" variable:

Attributes like Type, Claimed, Channel, Product Name and Destination are in object category. These variables have to be converted to categorical for better interpretation of the data.

```
feature: Type
['Airlines', 'Travel Agency']
Categories (2, object): ['Airlines', 'Travel Agency']
[0 1]


feature: Claimed
['No', 'Yes']
Categories (2, object): ['No', 'Yes']
[0 1]


feature: Channel
['Online', 'Offline']
Categories (2, object): ['Offline', 'Online']
[1 0]


feature: Product Name
['Customised Plan', 'Cancellation Plan', 'Bronze Plan', 'Silver Plan', 'Gold Plan']
Categories (5, object): ['Bronze Plan', 'Cancellation Plan', 'Customised Plan', 'Gold Plan', 'Silver Plan']
[2 1 0 4 3]


feature: Destination
['ASIA', 'Americas', 'EUROPE']
Categories (3, object): ['ASIA', 'Americas', 'EUROPE']
[0 1 2]
```

Figure 5: Object Datatype to Categorical

Data type post conversion of "Object" to "Categorical Variable" is as given below:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2861 entries, 0 to 2999
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Age           2861 non-null   int64
 1   Type          2861 non-null   int8
 2   Claimed       2861 non-null   int8
 3   Commision     2861 non-null   float64
 4   Channel       2861 non-null   int8
 5   Duration      2861 non-null   int64
 6   Sales         2861 non-null   float64
 7   Product Name  2861 non-null   int8
 8   Destination   2861 non-null   int8
dtypes: float64(2), int64(2), int8(5)
```

Exploratory Data Analysis (EDA):

Exploratory Data Analysis (EDA) is used analyse and interpret the data. It helps determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Age	2861.0	38.204124	10.678106	8.0	31.0	36.00	43.00	84.00
Type	2861.0	0.597344	0.490518	0.0	0.0	1.00	1.00	1.00
Claimed	2861.0	0.319469	0.466352	0.0	0.0	0.00	1.00	1.00
Commision	2861.0	15.080996	25.826834	0.0	0.0	5.63	17.82	210.21
Channel	2861.0	0.983922	0.125799	0.0	1.0	1.00	1.00	1.00
Duration	2861.0	72.120238	135.977200	-1.0	12.0	28.00	66.00	4580.00
Sales	2861.0	61.757878	71.399740	0.0	20.0	33.50	69.30	539.00
Product Name	2861.0	1.666550	1.277822	0.0	1.0	2.00	2.00	4.00
Destination	2861.0	0.261797	0.586239	0.0	0.0	0.00	0.00	2.00

Figure 6: Description of Dataset

We have considered mean, median, standard deviation, minimum and maximum of the dataset.

Correlation of dataset:

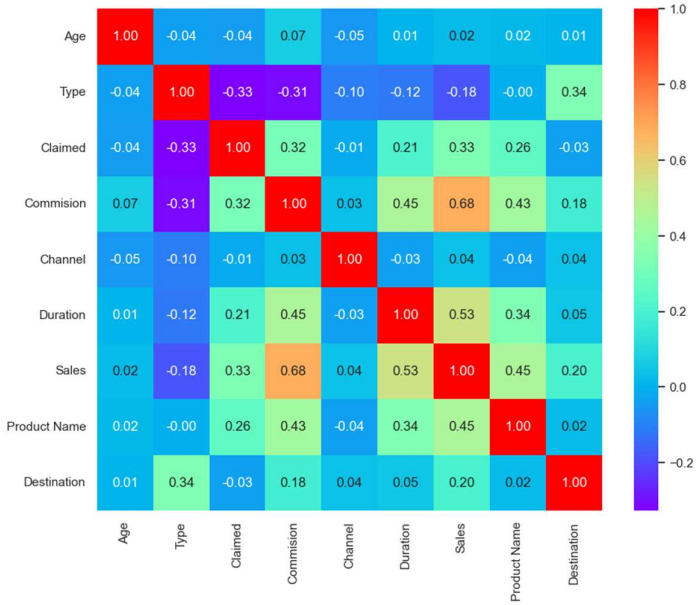

Figure 7: Correlation

We use Pearson Correlation coefficient, which measures the linear association between two variables. It has values between -1 and 1 where:

- -1 indicates perfectly negative correlation between the variables
- 0 indicates there is no correlation between the variables
- 1 indicates a perfectly positive correlation between the variables.

As per the above given correlation matrix, target variable Claim is highly correlated with Product Name, Sales, Duration and Commission. Age and Type are having negative correlation between Claim and Channel.

Outlier Detection:

An outlier is a value or point that differs substantially from the rest of the data. Outliers can look like this: Sometimes outliers might be errors that we want to exclude or an anomaly that we don't want to include in our analysis.

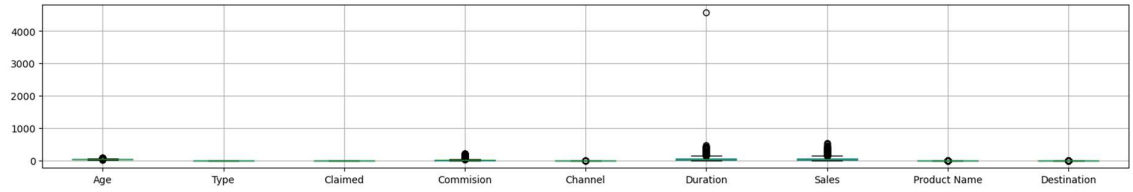

Figure 8: Outlier Detection

Box plot is used to identify the presence of outlier in a dataset. In our dataset, we have outliers in Age, Commission, Channel, Duration, Product Name and Destination.

Outlier Correction:

```
def remove_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

```
lratio,uratio=remove_outlier(df['Age'])
df['Age']=np.where(df['Age']>uratio,uratio,df['Age'])
df['Age']=np.where(df['Age']<lratio,lratio,df['Age'])

lratio,uratio=remove_outlier(df['Commision'])
df['Commision']=np.where(df['Commision']>uratio,uratio,df['Commision'])
df['Commision']=np.where(df['Commision']<lratio,lratio,df['Commision'])

lratio,uratio=remove_outlier(df['Duration'])
df['Duration']=np.where(df['Duration']>uratio,uratio,df['Duration'])
df['Duration']=np.where(df['Duration']<lratio,lratio,df['Duration'])

lratio,uratio=remove_outlier(df['Sales'])
df['Sales']=np.where(df['Sales']>uratio,uratio,df['Sales'])
df['Sales']=np.where(df['Sales']<lratio,lratio,df['Sales'])

lratio,uratio=remove_outlier(df['Product Name'])
df['Product Name']=np.where(df['Product Name']>uratio,uratio,df['Product Name'])
df['Product Name']=np.where(df['Product Name']<lratio,lratio,df['Product Name'])

lratio,uratio=remove_outlier(df['Destination'])
df['Destination']=np.where(df['Destination']>uratio,uratio,df['Destination'])
df['Destination']=np.where(df['Destination']<lratio,lratio,df['Destination'])
```

Figure 9: Outlier Treatment

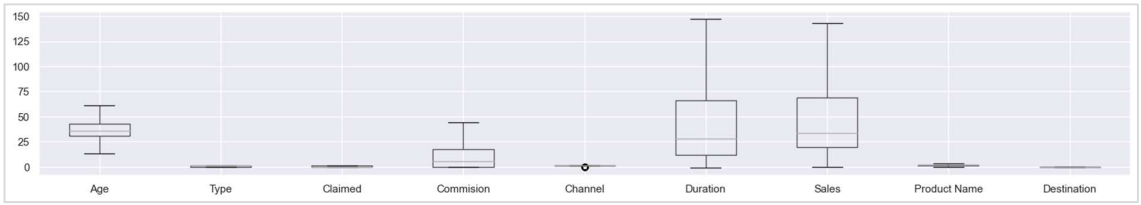

Figure 10: After treatment of Outlier

There are two methods to treat outlier. They are Interquartile Range Method (IQR) and Z Score. Here we have used IQR method for treating the outliers in the data.

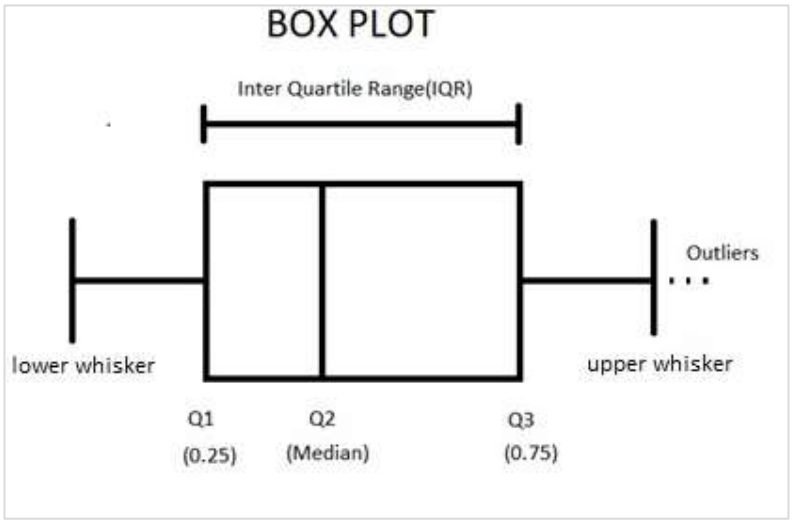

Figure 11: IQR Method

Proportion of 1's and 0's

```
plt.figure(figsize=(7,6))
df['Claimed'].value_counts().plot(kind='bar')
plt.legend()
plt.grid()
print("Percentage of 0's",round(df.Claimed.value_counts().values[0]/df.Claimed.count()*100,2),'%')
print("Percentage of 1's",round(df.Claimed.value_counts().values[1]/df.Claimed.count()*100,2),'%')
```

```
Percentage of 0's 68.05 %
Percentage of 1's 31.95 %
```

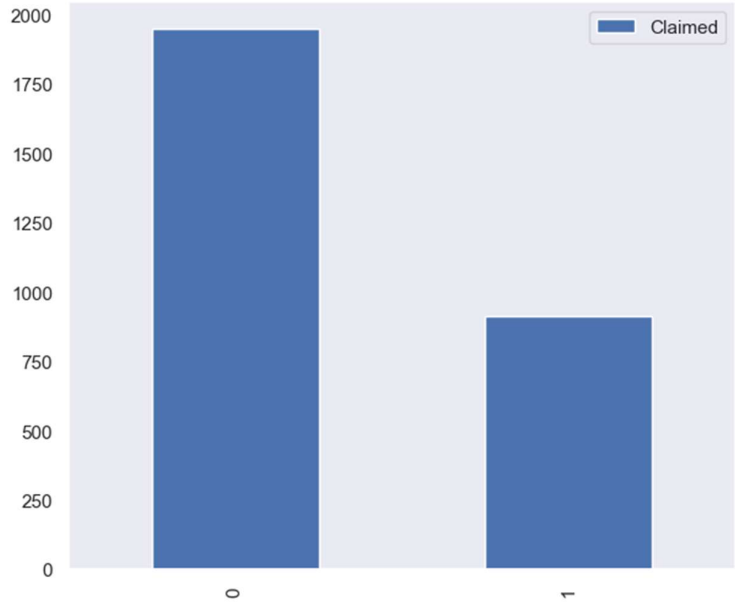

Figure 12: Percentage of Claim Yes and No

Here 0 indicates “Not Claimed” and 1 indicates “Claimed”. 68% of the claims are not claimed by the travellers with the insurance company where 32% has claimed their travel insurance.

Model Train and Test Split:
The data set is split into train and test. In order to split the dataset, we will use the library “train test split” from sklearn package. Here data is split as train and test based on 70:30 ratio. 70% of the data is train set and 30% of data is test set. Target variable is considered as Y and rest of the data is considered as X. Based on, we have created X.

```
X = df.drop("Claimed", axis=1)
y = df.pop("Claimed")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0)
```

Figure 13: Train-Test Spilt

CART Model:
A **decision tree** is a flowchart-like structure in which each internal node represents a “test” on an attribute (e.g., whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes) In order to build the decision tree model, we will be importing the library from skleran as Decision Tree Classifier. To construct the tree, we will be using another package from sklearn named tree.

```
dtree = DecisionTreeClassifier(random_state=0)
```

```
dtree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(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=False,
                       random_state=0, splitter='best')
```

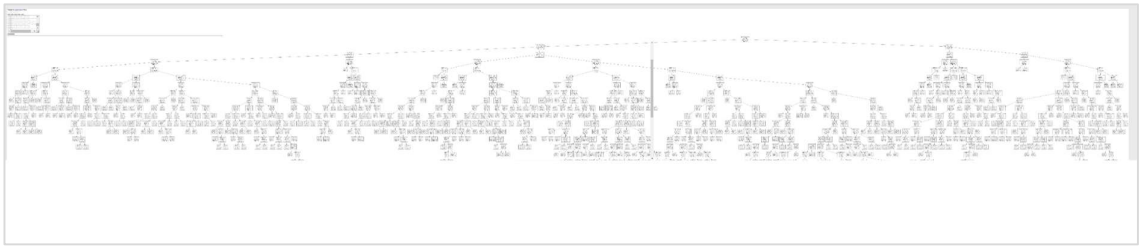

Figure 14: Decision Tree

We will import the tree in order to view the depth of the tree. We have to identify how many splits have happened for the whole dataset. If the tree has grown further, then we have to use the pruning technique. The model uses Gini Index as the criterion.

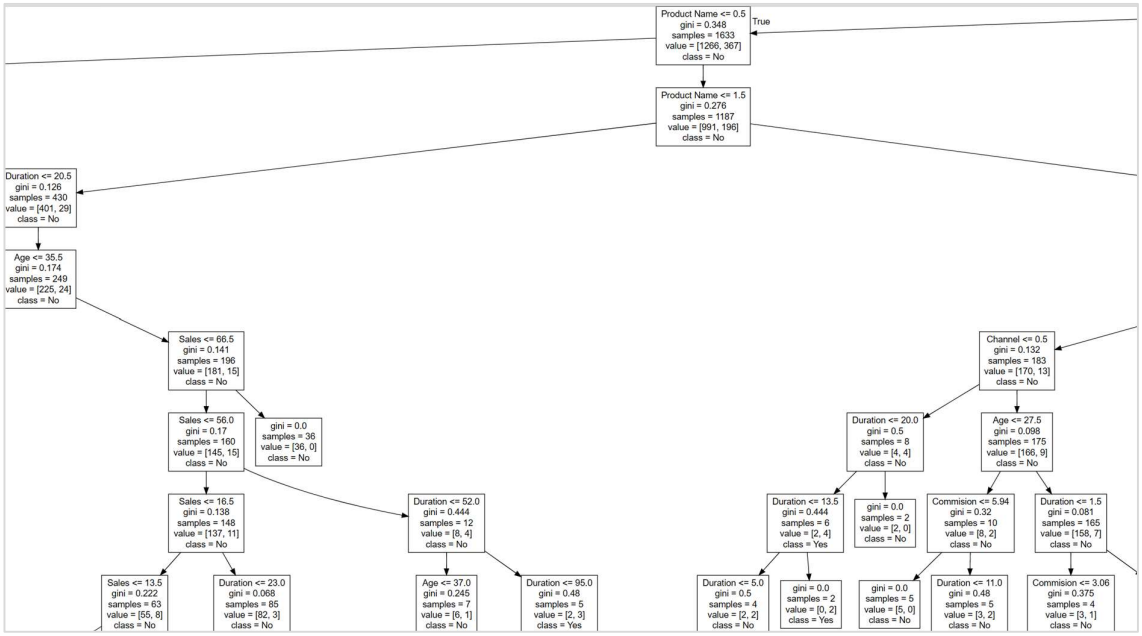

Figure 15: Snip from tree

Important Features in the Tree

	Imp
Age	0.180133
Type	0.005029
Commision	0.098749
Channel	0.008854
Duration	0.254734
Sales	0.225695
Product Name	0.226805
Destination	0.000000

The important feature in Decision tree is split between Sales, Product Name , Duration and Age. Based on Gini Index the optimal split of node is determined. A perfect split has a Gini index of 0, while a worst split has a Gini index of 0.5 in case of two classes. Gini index is one of the criteria for calculating information gain, which is the difference between the impurity of the parent node and the weighted average impurity of the child nodes.

We have not used any pruning techniques for this graph. Hence, we have to regularise the graphs by using pruning techniques.

```
y_predict = dtree.predict(X_test)
```

```
reg_dtree = DecisionTreeClassifier(criterion = 'gini', max_depth = 6,min_samples_leaf=10,min_samples_split=150,random_state=0)
reg_dtree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=10, min_samples_split=150,
                       random_state=0)
```

```
Insurance_tree_regularized = open('C:/Users/Dell/OneDrive/Documents/Deakin/Insurance_tree_regularized.dot','w')
dot_data = tree.export_graphviz(reg_dtree, out_file= Insurance_tree_regularized , feature_names = list(X_train), class_names = li

Insurance_tree_regularized.close()

print (pd.DataFrame(dtree.feature_importances_, columns = ["Imp"], index = X_train.columns))
```

```
                 Imp
Age         0.180133
Type        0.005029
Commision   0.098749
Channel     0.008854
Duration    0.254734
Sales       0.225695
Product Name  0.226805
Destination   0.000000
```

Figure 16: Pruning the CART

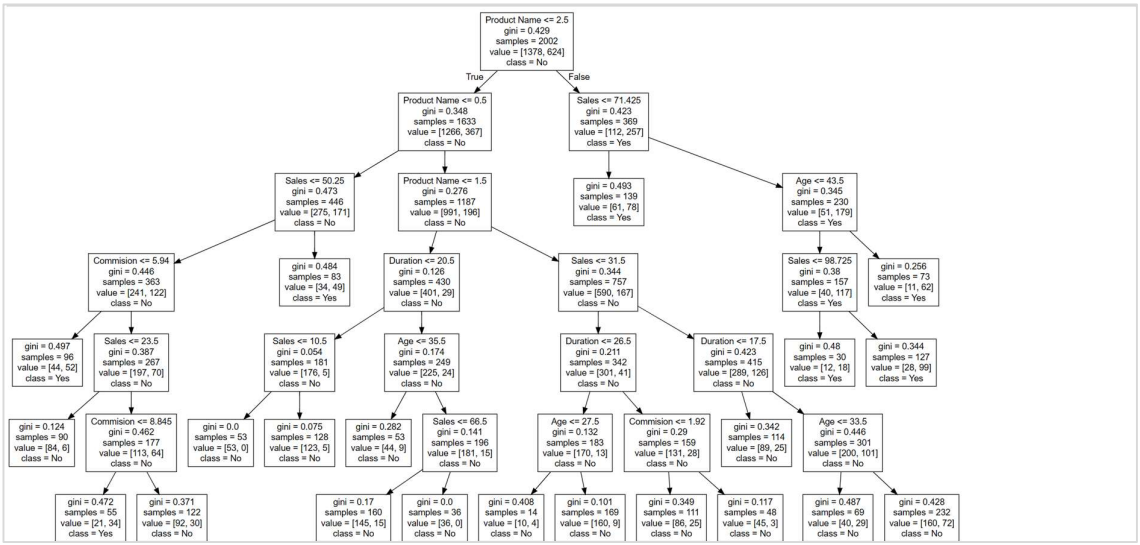

Figure 17: Decision Tree after Pruning

Using Gini Index, the tree is split between 150 samples with a maximum depth of 6 nodes and Minimum split of 10 leaves. Above tree is split between Product Name and Sales.

Random Forest:

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds.

Import the library from sklearn package for Random Forest Classifier.

We will split the data into test and train with the target variable “Claimed”. Here we will decide how many numbers of trees to need to be constructed. In this case we are constructing 350 trees for this dataset.

```
param_grid = {
    'max_depth': [7],#10
    'max_features': [4],#6
    'min_samples_leaf': [50],#150
    'min_samples_split': [150],#550
    'n_estimators': [350]#50
}

rfcl = RandomForestClassifier()

grid_search = GridSearchCV(estimator = rfcl, param_grid = param_grid, cv = 5)
```

```
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [7], 'max_features': [4],
                         'min_samples_leaf': [50], 'min_samples_split': [150],
                         'n_estimators': [350]})
```

Figure 18: Random Forest -Grid Search

We will check the Out of Bag data points for this dataset. The OOB Score is 73.87%

```
rfcl.oob_score_
```

```
0.7387612387612388
```

The Maximum level of the trees considering for the data set is 7

The total number of independent variables for this tree is 4

There will be 50 observations to be presented in the terminal notes.

The minimum sample splits will be 150

We will update all these in a Grid Search CV package to construct the Random Forest classifier.

```
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              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,
                                              n_estimators='warn', n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [7], 'max_features': [4],
                         'min_samples_leaf': [50], 'min_samples_split': [150],
                         'n_estimators': [350]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

Figure 19: Best Fit Grid

Train and Test Split of Random Forest:

```
ytrain_predict = best_grid.predict(X_train)
ytest_predict = best_grid.predict(X_test)
```

Figure 20: RF Train-Test Spilt

Based on the best grid, the data is split into Train and Test.

```
best_grid
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=7, max_features=4, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=50, min_samples_split=150,
                       min_weight_fraction_leaf=0.0, n_estimators=350,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

3. **What is the performance of the build model/ models? You need to provide discussion and justification of how the model is performing (discuss different matrices, accuracy confusion matrix) based on the selected dataset**

Decision Tree – Model Building and Model Performance

After finalizing the tree, the next step is to predict the Train and test data values.

```
ytrain_predict = reg_dtree.predict(X_train)
ytest_predict = reg_dtree.predict(X_test)
```

The classification report will share us the performance of the model. The performance of the model is split as below:

- Confusion Matrix
- Accuracy of the model
- Sensitivity of the model
- Precision of the model
- Specificity of the model.

	precision	recall	f1-score	support
0	0.76	0.76	0.76	569
1	0.52	0.53	0.53	290
accuracy			0.68	859
macro avg	0.64	0.64	0.64	859
weighted avg	0.68	0.68	0.68	859

- **Confusion Matrix**: summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion

matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

Figure 21: Confusion Matrix

Confusion Matrix for Decision Tree (Train)

```
confusion_matrix(y_train, ytrain_predict)
```

```
array([[1167,  211],
       [ 232,  392]], dtype=int64)
```

Figure 22: Confusion Matrix - Train Dataset

The True positive is 1167 and True Negative is 392.

- **Accuracy**: It gives you the overall accuracy of the model, meaning the fraction of the total samples that were correctly classified by the classifier. To calculate accuracy, use the following formula: (TP+TN)/(TP+TN+FP+FN). Accuracy of Train data is 78%.
- **Precision** is a metric that tells us about the quality of positive predictions. Precision is calculated by using the formula: (TP)/(TP+FP). Precision of Train data is 63%.
- **Recall** tells about how well the model identifies true positive. Recall is calculated by using the formula: (TP)/(TP+FN). Recall on train data set is 64%.
- **F1 Score** is the weighted average of precision and recall. F1- Score for train data is 65%.

Confusion Matrix for Test data is as below:

```
confusion_matrix(y_test, ytest_predict)
```

```
array([[481,  88],
       [117, 173]], dtype=int64)
```

Figure 23:Confusion Matrix Test-Data

True positive is 481 and True Negative is 173.

Classification Report:

	precision	recall	f1-score	support
0	0.80	0.85	0.82	569
1	0.66	0.60	0.63	290
accuracy			0.76	859
macro avg	0.73	0.72	0.73	859
weighted avg	0.76	0.76	0.76	859

Figure 24: Classification Report

```
cart_metrics=classification_report(y_test, ytest_predict,output_dict=True)
df=pd.DataFrame(cart_metrics).transpose()
cart_test_precision=round(df.loc["1"][1],2)
cart_test_recall=round(df.loc["1"][2],2)
cart_test_f1=round(df.loc["1"][0],2)
print ('cart_test_precision ',cart_test_precision)
print ('cart_test_recall ',cart_test_recall)
print ('cart_test_f1 ',cart_test_f1)
```

```
cart_test_precision  0.6
cart_test_recall  0.63
cart_test_f1  0.66
```

Figure 25: Classification Report Test data

Accuracy of test data is 76%
Precision of test data is 60%
Recall for test data is 63%
F1 Score is 66%

ROC Curve and AUC Score for CART:
A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The higher the area under the ROC curve (AUC), the better the classifier. A classifier with an AUC higher than 0.5 is better than a random classifier. If AUC is lower than 0.5, then something is wrong with your model. A perfect classifier would have an AUC of 1.

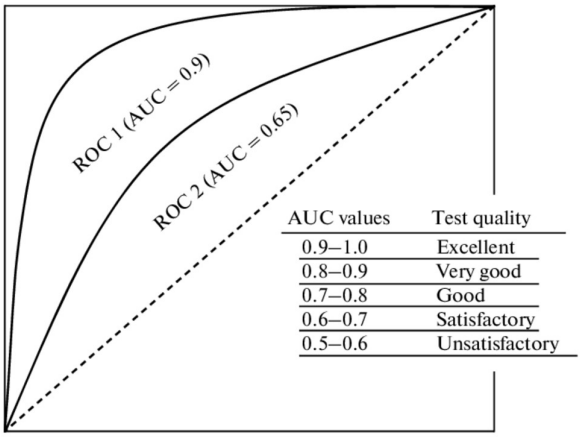

Figure 26:ROC (Picture Courtesy: Internet)

The below illustrated is the ROC curve for the CART Model:

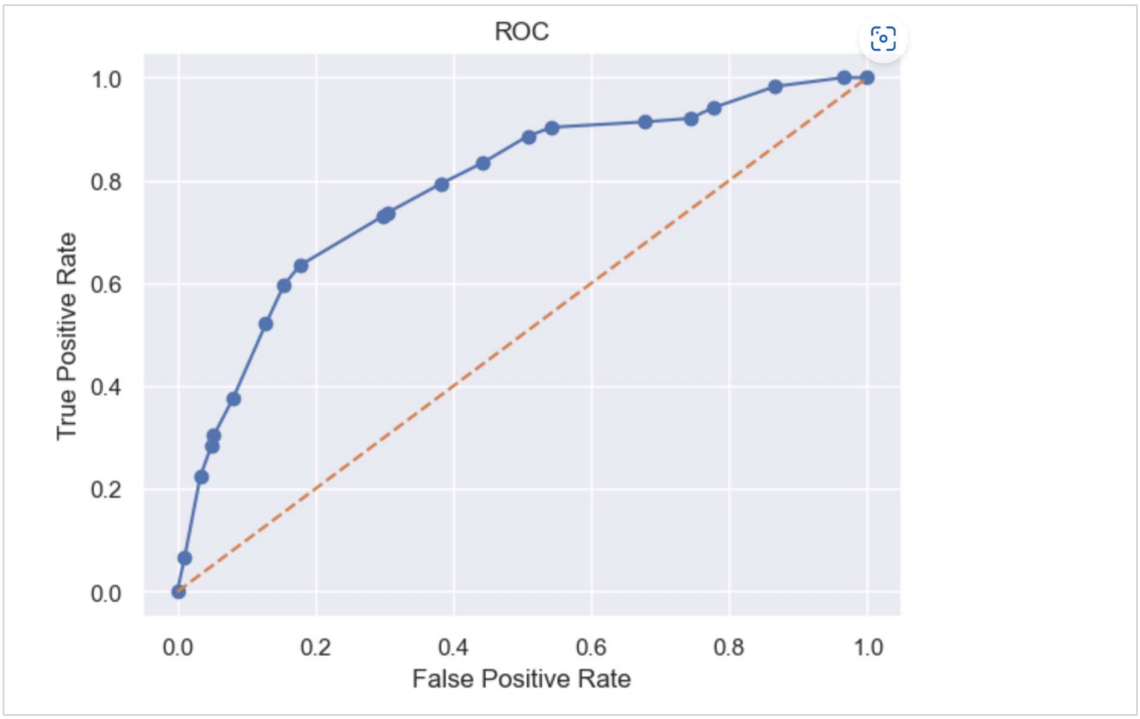

Figure 27: ROC for CART

Training and Test results are almost similar. Hence, we conclude by saying model is "Good". AUC for Train is 83% and Test is 78%

Random Forest – Model Building and Model Performance

The library "GridSearchCV" helps in implementing "fit" and "score" method. This function also has "score samples", "predict", " decision_function", "transform" and "inverse transform function". The parameters of the estimator use these methods to optimise by cross validated grid search over a parameter grid.

```
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {
    'max_depth': [7],#10
    'max_features': [4],#6
    'min_samples_leaf': [50],#150
    'min_samples_split': [150],#550
    'n_estimators': [350]#50
}

rfcl = RandomForestClassifier()

grid_search = GridSearchCV(estimator = rfcl, param_grid = param_grid, cv = 5)
```

```
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [7], 'max_features': [4],
                         'min_samples_leaf': [50], 'min_samples_split': [150],
                         'n_estimators': [350]})
```

```
grid_search.best_params_
```

```
{'max_depth': 7,
 'max_features': 4,
 'min_samples_leaf': 50,
 'min_samples_split': 150,
 'n_estimators': 350}
```

Figure 28: Grid search_param

Based on the best fit grid search from cross validation, the models are split to Train and Test.

```
ytrain_predict = best_grid.predict(X_train)
ytest_predict = best_grid.predict(X_test)
```

Confusion Matrix – Train data

```
array([[1257,  121],
       [ 342,  282]], dtype=int64)
```

The confusion matrix for random forest for train data shows 1257 true positives and 282 false positive. The accuracy of the train model is 77%.

Classification Report of Train data

	precision	recall	f1-score	support
0	0.79	0.91	0.84	1378
1	0.70	0.45	0.55	624
accuracy			0.77	2002
macro avg	0.74	0.68	0.70	2002
weighted avg	0.76	0.77	0.75	2002

Accuracy of test data is 77%
Precision of test data is 45%
Recall for test data is 55%
F1 Score is 70%

AUC Curve for Train set

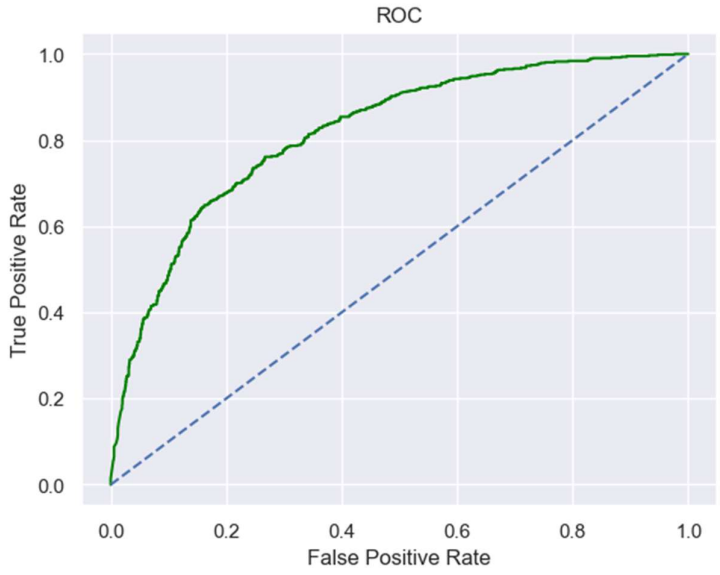

Figure 29: AUC – Train

Area Under Curve is 68%

Confusion Matrix – Test Data

```
array([[515,  54],
       [170, 120]], dtype=int64)
```

True positive of test data is 515 and False positive is 120. Accuracy of Test data is 74%

Confusion Matrix:

	precision	recall	f1-score	support
0	0.75	0.91	0.82	569
1	0.69	0.41	0.52	290
accuracy			0.74	859
macro avg	0.72	0.66	0.67	859
weighted avg	0.73	0.74	0.72	859

Precision of test data is 41%
Recall of test data is 52%
F1 score is 69%

AUC- Test Data

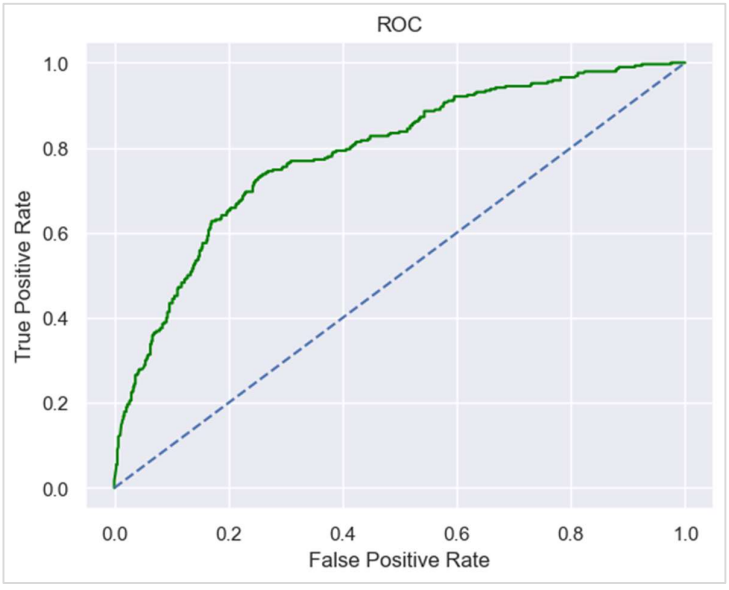

Figure 30: AUC Test data

The AUC value and ROC curve for the Test model shows 65.8%. wider the model the stronger the model will be. Here the model is an average one

4. **You need to compare the performance of the models and provide justifications which mode is performing better and why.**

Comparing the performance of Decision Tree and Random Forest for both Train and Test data is as given below:

```
index=['Accuracy', 'AUC', 'Recall','Precision','F1 Score']
data = pd.DataFrame({'CART Train':[cart_train_acc,cart_train_auc,cart_train_recall,cart_train_precision,cart_train_f1],
        'CART Test':[cart_test_acc,cart_test_auc,cart_test_recall,cart_test_precision,cart_test_f1],
       'Random Forest Train':[rf_train_acc,rf_train_auc,rf_train_recall,rf_train_precision,rf_train_f1],
        'Random Forest Test':[rf_test_acc,rf_test_auc,rf_test_recall,rf_test_precision,rf_test_f1]},index=index)
round(data,2)
```

	CART Train	CART Test	Random Forest Train	Random Forest Test
Accuracy	0.78	0.76	0.77	0.74
AUC	0.83	0.78	0.68	0.66
Recall	0.64	0.63	0.55	0.52
Precision	0.63	0.60	0.45	0.41
F1 Score	0.65	0.66	0.70	0.69

Figure 31: Comparison of CART & RF

- Comparing the accuracy of both the models, Decision Tree model proves to be better.
- Area under the curve for Decision Tree is very good.
- Recall for Decision Tree proves to be better than Random Forest.
- Precision for Decision Tree is better compared with Random Forest,
- F1 Score for Random Forest is better compared with CART.

ROC for Training and Test Data

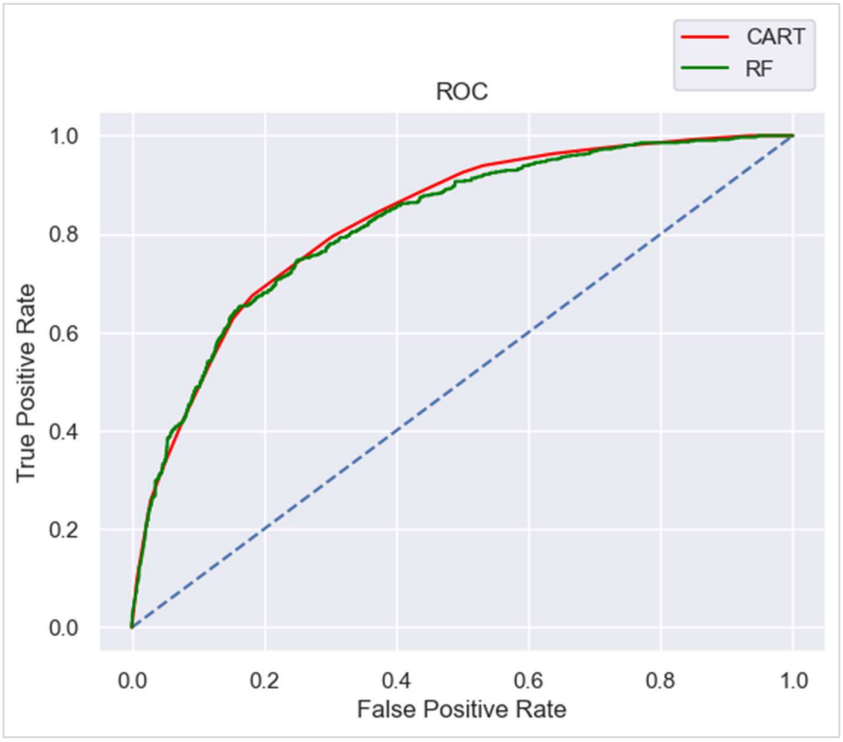

Figure 32: ROC for 2 Models

The graphs shows that ROC curve for Training shows both Random Forest and CART are graphed equally on the curve. For the training data we opt for the CART or Decision tree and Random Forest as the best model.

ROC for Test Data

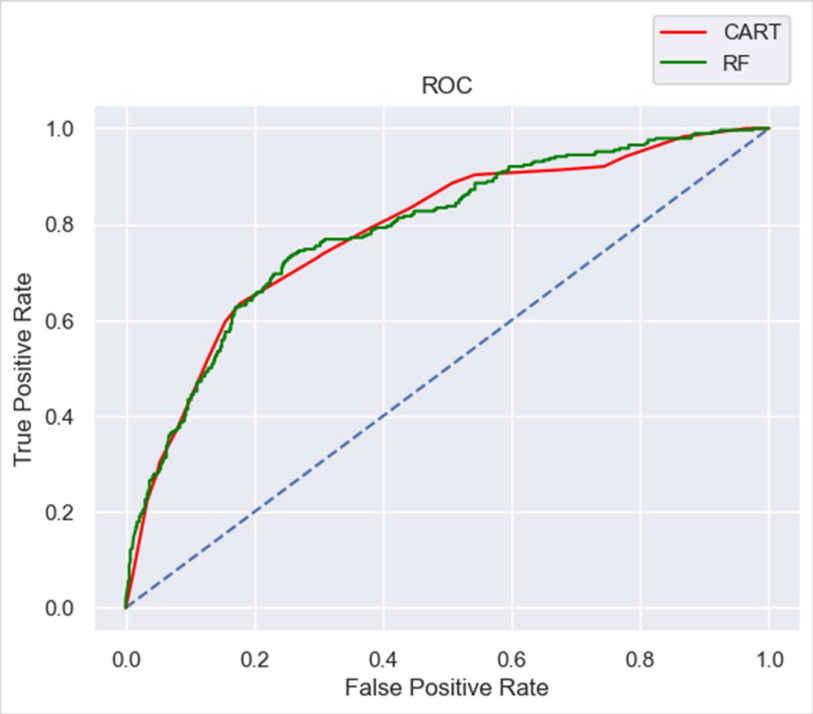

Figure 33: ROC for Test Data

The graphs show that ROC curve for Test shows CART as better model. Both the models showing stronger ROC curve.