# Training The Model In Multiple Algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

# Decision Tree Model

A function named decisionTree is created and train and test data are passed as the parameters. Inside the function, DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

We are going to use x_train and y_train obtained above in train_test_split section to train our **Decision Tree Classifier** model. We're using the fit method and passing the parameters as shown below.

```
from sklearn.tree import DecisionTreeClassifier
classifier=DecisionTreeClassifier(random_state=0)
classifier.fit(x_train,y_train)
```

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

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
decisiontree=classifier.predict(x_test)
```

```
decisiontree
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=uint8)
```

```
from sklearn.metrics import accuracy_score
desacc=accuracy_score(y_test,decisiontree)
```

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=10,criterion='entropy')
```

```
rfc.fit(x_train,y_train)
```

```
C:\Users\GASCCS23\AppData\Local\Temp\ipykernel_784\4070307935.py:1: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  rfc.fit(x_train,y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

## Random Forest Model:

A function named random Forest is created and train and test data are passed as the parameters. Inside the function, Random Forest Classifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with. predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=10,criterion='entropy')
```

```
rfc.fit(x_train,y_train)
```

```
C:\Users\GASCCS23\AppData\Local\Temp\ipykernel_784\4070307935.py:1: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  rfc.fit(x_train,y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

## ANN Model:

Building and training an Artificial Neural Network (ANN) using the Keras library with TensorFlow as the backend. The ANN is initialised as an instance of the Sequential class, which is a linear stack of layers. Then, the input layer and two hidden layers are added to the model using the Dense class, where the number of units and activation function are specified. The output layer is also added using the Dense class with a sigmoid activation function. The model is then compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric. Finally, the model is fit to the training data with a batch size of 100, 20% validation split, and 100 epochs.

```
y_predit=rfc.predict(x_test)
```

```
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
classification=Sequential()
classification.add(Dense(30,activation='relu'))
classification.add(Dense(128,activation='relu'))
classification.add(Dense(64,activation='relu'))
classification.add(Dense(32,activation='relu'))
classification.add(Dense(1,activation='sigmoid'))
```

```
classification.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
classification.fit(x_train,y_train,batch_size=4,validation_split=0.2,epochs=100)
```

```
1797/1797 [==============================] - 2s 1ms/step - loss: 0.0351 - accuracy: 0.9871 - val_loss: 0.0694 - val_accuracy:
0.9772
Epoch 96/100
1797/1797 [==============================] - 2s 1ms/step - loss: 0.0418 - accuracy: 0.9819 - val_loss: 0.0715 - val_accuracy:
0.9750
Epoch 97/100
1797/1797 [==============================] - 2s 1ms/step - loss: 0.0391 - accuracy: 0.9840 - val_loss: 0.0644 - val_accuracy:
0.9755
Epoch 98/100
1797/1797 [==============================] - 3s 1ms/step - loss: 0.0356 - accuracy: 0.9840 - val_loss: 0.0675 - val_accuracy:
0.9783
Epoch 99/100
1797/1797 [==============================] - 3s 1ms/step - loss: 0.0362 - accuracy: 0.9840 - val_loss: 0.0549 - val_accuracy:
0.9766
Epoch 100/100
1797/1797 [==============================] - 3s 1ms/step - loss: 0.0371 - accuracy: 0.9853 - val_loss: 0.0826 - val_accuracy:
0.9683

<keras.callbacks.History at 0x249ee8d5390>
```

# Test The Model:

In ANN we first have to save the model to the test the inputs.

```
y_pred=classifier.predict([[129,99,1,0,0,1]])

print(y_pred)
(y_pred)
```

```
[0]

array([0], dtype=uint8)
```

```
y_pred=rfc.predict([[129,99,1,0,0,1]])

print(y_pred)
(y_pred)
```

```
[0]

array([0], dtype=uint8)
```

```
classification.save('flight.h5')
```

```
y_pred=classification.predict(x_test)
```

```
71/71 [==============================] - 0s 872us/step
```

```
y_pred
```

```
array([[0.        ],
       [0.00100767],
       [0.        ],
       ...,
       [0.        ],
       [0.        ],
       [0.9852775 ]], dtype=float32)
```

```
y_pred=(y_pred>0.5)
y_pred
```

```
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [ True]])
```

```
def predict_exit(sample_value):
    sample_value=np.array(sample_value)
    sample_value=sample_value.reshape(1,-1)
    sample_value=sc.transform(sample_value)
    return classifier.predict(sample_value)
```

This code defines a function named "predict_exit" which takes in a sample_value as an input. The function then converts the input

sample_value from a list to a numpy array. It reshapes the sample_value array as it contains only one record. Then, it applies feature scaling to the reshaped sample_value array using a scaler object 'sc' that should have been previously defined and fitted. Finally, the function returns the prediction of the classifier on the scaled sample_value.

```
def predict_exit(sample_value):

  # Convert list to numpy array
    sample_value = np.array(sample_value)

  # Reshape because sample_value contains only 1 record
    sample_value = sample_value.reshape(1, -1)

  # Feature Scaling
    sample_value = sc.transform(sample_value)

    return classifier.predict(sample_value)
```

```
test=classification.predict([[1,1,121.000000,36.0,0,0,1,0,1,1,1,1,1,1,1,1]])
if test==1:
    print('Prediction: Chance of delay')
else:
    print('Prediction: No chance of delay.')
```

```
Prediction: No chance of delay.
```