Assignment 2.1: Building an Artificial Neural Network

Objective

Your task is to build an ANN classification model that can predict whether a user will click on an online ad based on the given factors.

Instructions

- Load the dataset into pandas dataframe.
- Perform data cleaning and preprocessing as necessary.
- Split the data into training and testing sets using an 80:20 ratio.
- Scale the data using StandardScaler.
- Build an ANN classification model.
- Experiment with different model architectures, activation functions, regularization techniques, learning rates, and batch sizes to optimize the model's performance.
- Evaluate the model's performance using accuracy, precision, recall, F1 score, and ROC AUC score as the metrics.
- Interpret the results and draw conclusions about the factors that are most important in predicting whether a user will click on an online ad.

Deliverables

- Convert your Jupyter Notebook or Python script to a single PDF file or MS Word document. Your deliverable should contain your implementations of the tasks above, as well as any additional comments or observations you may have. Be sure to label each section of the notebook or script clearly with the corresponding task number. Please ensure the PDF or MS Word document displays the code and output appropriately. Bonus

- For an extra 10 points, try using other classification models (e.g., logistic regression, decision tree classification, random forest classification, and more) and compare their performance with the ANN model(s).

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

```
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
```

In [ ]:
```
# importing the data

df = pd.read_csv('data.csv')

df.head()
```

Out[ ]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18

In [ ]:
```
# checking for nulls

df.isnull().sum()
```

Out[ ]:
```
Daily Time Spent on Site    0
Age                         0
Area Income                 0
Daily Internet Usage        0
Ad Topic Line               0
City                        0
Male                        0
Country                     0
Timestamp                   0
Clicked on Ad               0
dtype: int64
```

In [ ]:
```
df.dtypes
```

Out[ ]:
```
Daily Time Spent on Site    float64
Age                           int64
Area Income                 float64
Daily Internet Usage        float64
Ad Topic Line                object
City                         object
Male                          int64
Country                      object
Timestamp                    object
Clicked on Ad                 int64
dtype: object
```

In [ ]:
```
# getting the uniuque values of the 'Ad Topic Line' column

df['Ad Topic Line'].nunique()
```

Out[ ]:
```
1000
```

In [ ]:
```
# getting the unique values of the 'City' column

df['City'].nunique()
```

Out[ ]:
```
969
```

In [ ]:
```
# Dropping the timestamp column

df.drop('Timestamp', axis=1, inplace=True)
```

In [ ]:
```
# getting the dummies for the following columns, 'City', 'Country', 'Ad Topic Line'

df = pd.get_dummies(df, columns=['City', 'Country', 'Ad Topic Line'], drop_first=Tr
```

In [ ]:
```
df.head()
```

Out[ ]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad	City_Adamside	City_Adamsstad	City_Ala
0	68.95	35	61833.90	256.09	0	0	False	False	
1	80.23	31	68441.85	193.77	1	0	False	False	
2	69.47	26	59785.94	236.50	0	0	False	False	
3	74.15	29	54806.18	245.89	1	0	False	False	
4	68.37	35	73889.99	225.58	0	0	False	False	

5 rows × 2209 columns

In [ ]:
```
# now we will split the data into features and target target is clicked on ad

X = df.drop('Clicked on Ad', axis=1)
y = df['Clicked on Ad']

# now we will split the data into training and testing data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

In [ ]:
```
# Now we are going scale the data

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:
```
# We are now going to make our ANN model

# model type
model = tf.keras.models.Sequential()


# input layer
model.add(tf.keras.layers.Dense(units=100, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))


# hidden layers
model.add(tf.keras.layers.Dense(units=50, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(units=25, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
```

```
# output layer
model.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

# compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [ ]:

```
# Now we will train the model

model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_t
```

```
Epoch 1/100
22/22 [==============================] - 1s 22ms/step - loss: 0.9892 - accuracy: 0.5
029 - val_loss: 0.6920 - val_accuracy: 0.5233
Epoch 2/100
22/22 [==============================] - 0s 4ms/step - loss: 0.8008 - accuracy: 0.55
86 - val_loss: 0.6863 - val_accuracy: 0.5567
Epoch 3/100
22/22 [==============================] - 0s 4ms/step - loss: 0.7705 - accuracy: 0.54
57 - val_loss: 0.6857 - val_accuracy: 0.5533
Epoch 4/100
22/22 [==============================] - 0s 4ms/step - loss: 0.6669 - accuracy: 0.61
00 - val_loss: 0.6838 - val_accuracy: 0.5600
Epoch 5/100
22/22 [==============================] - 0s 3ms/step - loss: 0.6147 - accuracy: 0.66
86 - val_loss: 0.6794 - val_accuracy: 0.5800
Epoch 6/100
22/22 [==============================] - 0s 4ms/step - loss: 0.5945 - accuracy: 0.67
14 - val_loss: 0.6697 - val_accuracy: 0.6400
Epoch 7/100
22/22 [==============================] - 0s 5ms/step - loss: 0.5526 - accuracy: 0.69
00 - val_loss: 0.6520 - val_accuracy: 0.6900
Epoch 8/100
22/22 [==============================] - 0s 3ms/step - loss: 0.5010 - accuracy: 0.76
57 - val_loss: 0.6269 - val_accuracy: 0.6833
Epoch 9/100
22/22 [==============================] - 0s 4ms/step - loss: 0.4299 - accuracy: 0.80
57 - val_loss: 0.5948 - val_accuracy: 0.7333
Epoch 10/100
22/22 [==============================] - 0s 4ms/step - loss: 0.3364 - accuracy: 0.86
86 - val_loss: 0.5570 - val_accuracy: 0.7500
Epoch 11/100
22/22 [==============================] - 0s 3ms/step - loss: 0.2738 - accuracy: 0.89
14 - val_loss: 0.5211 - val_accuracy: 0.7800
Epoch 12/100
22/22 [==============================] - 0s 4ms/step - loss: 0.2158 - accuracy: 0.92
00 - val_loss: 0.4602 - val_accuracy: 0.8333
Epoch 13/100
22/22 [==============================] - 0s 4ms/step - loss: 0.1920 - accuracy: 0.93
29 - val_loss: 0.4314 - val_accuracy: 0.8267
Epoch 14/100
22/22 [==============================] - 0s 4ms/step - loss: 0.1423 - accuracy: 0.95
14 - val_loss: 0.3919 - val_accuracy: 0.8633
Epoch 15/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0981 - accuracy: 0.97
43 - val_loss: 0.3819 - val_accuracy: 0.8400
Epoch 16/100
22/22 [==============================] - 0s 4ms/step - loss: 0.1061 - accuracy: 0.96
57 - val_loss: 0.3746 - val_accuracy: 0.8333
Epoch 17/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0753 - accuracy: 0.97
71 - val_loss: 0.3952 - val_accuracy: 0.8200
Epoch 18/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0828 - accuracy: 0.96
71 - val_loss: 0.3743 - val_accuracy: 0.8333
Epoch 19/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0434 - accuracy: 0.99
```

```
00 - val_loss: 0.3657 - val_accuracy: 0.8433
Epoch 20/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0359 - accuracy: 0.99
14 - val_loss: 0.3489 - val_accuracy: 0.8700
Epoch 21/100
22/22 [==============================] - 0s 6ms/step - loss: 0.0447 - accuracy: 0.98
57 - val_loss: 0.3484 - val_accuracy: 0.8700
Epoch 22/100
22/22 [==============================] - 0s 6ms/step - loss: 0.0283 - accuracy: 0.99
43 - val_loss: 0.3451 - val_accuracy: 0.8767
Epoch 23/100
22/22 [==============================] - 0s 6ms/step - loss: 0.0419 - accuracy: 0.98
86 - val_loss: 0.3557 - val_accuracy: 0.8667
Epoch 24/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0267 - accuracy: 0.99
14 - val_loss: 0.3771 - val_accuracy: 0.8600
Epoch 25/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0227 - accuracy: 0.99
71 - val_loss: 0.3765 - val_accuracy: 0.8633
Epoch 26/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0260 - accuracy: 0.99
29 - val_loss: 0.3822 - val_accuracy: 0.8667
Epoch 27/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0197 - accuracy: 0.99
57 - val_loss: 0.3999 - val_accuracy: 0.8567
Epoch 28/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0217 - accuracy: 0.99
43 - val_loss: 0.3804 - val_accuracy: 0.8700
Epoch 29/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0221 - accuracy: 0.99
43 - val_loss: 0.3795 - val_accuracy: 0.8700
Epoch 30/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0139 - accuracy: 0.99
71 - val_loss: 0.3901 - val_accuracy: 0.8700
Epoch 31/100
22/22 [==============================] - 0s 6ms/step - loss: 0.0110 - accuracy: 0.99
86 - val_loss: 0.4033 - val_accuracy: 0.8700
Epoch 32/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0161 - accuracy: 0.99
57 - val_loss: 0.4191 - val_accuracy: 0.8600
Epoch 33/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0152 - accuracy: 0.99
43 - val_loss: 0.4228 - val_accuracy: 0.8567
Epoch 34/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0095 - accuracy: 0.99
86 - val_loss: 0.4401 - val_accuracy: 0.8533
Epoch 35/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0121 - accuracy: 0.99
57 - val_loss: 0.4703 - val_accuracy: 0.8433
Epoch 36/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0122 - accuracy: 0.99
43 - val_loss: 0.4691 - val_accuracy: 0.8433
Epoch 37/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0287 - accuracy: 0.99
57 - val_loss: 0.4354 - val_accuracy: 0.8467
Epoch 38/100
```

```
22/22 [==============================] - 0s 3ms/step - loss: 0.0069 - accuracy: 1.00
00 - val_loss: 0.4513 - val_accuracy: 0.8467
Epoch 39/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0100 - accuracy: 0.99
86 - val_loss: 0.4370 - val_accuracy: 0.8500
Epoch 40/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0111 - accuracy: 0.99
57 - val_loss: 0.4124 - val_accuracy: 0.8800
Epoch 41/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0110 - accuracy: 0.99
57 - val_loss: 0.4077 - val_accuracy: 0.8833
Epoch 42/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0065 - accuracy: 1.00
00 - val_loss: 0.4111 - val_accuracy: 0.8667
Epoch 43/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0057 - accuracy: 1.00
00 - val_loss: 0.4227 - val_accuracy: 0.8700
Epoch 44/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0117 - accuracy: 0.99
71 - val_loss: 0.4447 - val_accuracy: 0.8633
Epoch 45/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0044 - accuracy: 1.00
00 - val_loss: 0.4639 - val_accuracy: 0.8633
Epoch 46/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0036 - accuracy: 0.99
86 - val_loss: 0.4807 - val_accuracy: 0.8600
Epoch 47/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0079 - accuracy: 0.99
71 - val_loss: 0.4875 - val_accuracy: 0.8600
Epoch 48/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0036 - accuracy: 1.00
00 - val_loss: 0.4997 - val_accuracy: 0.8567
Epoch 49/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0034 - accuracy: 0.99
86 - val_loss: 0.5168 - val_accuracy: 0.8533
Epoch 50/100
22/22 [==============================] - 0s 6ms/step - loss: 0.0044 - accuracy: 1.00
00 - val_loss: 0.5327 - val_accuracy: 0.8500
Epoch 51/100
22/22 [==============================] - 0s 7ms/step - loss: 0.0052 - accuracy: 0.99
71 - val_loss: 0.5602 - val_accuracy: 0.8500
Epoch 52/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0022 - accuracy: 1.00
00 - val_loss: 0.5844 - val_accuracy: 0.8467
Epoch 53/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0054 - accuracy: 0.99
71 - val_loss: 0.6049 - val_accuracy: 0.8467
Epoch 54/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0050 - accuracy: 1.00
00 - val_loss: 0.6084 - val_accuracy: 0.8500
Epoch 55/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0041 - accuracy: 1.00
00 - val_loss: 0.6114 - val_accuracy: 0.8500
Epoch 56/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0058 - accuracy: 0.99
86 - val_loss: 0.5626 - val_accuracy: 0.8600
```

```
Epoch 57/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0035 - accuracy: 1.00
00 - val_loss: 0.5454 - val_accuracy: 0.8667
Epoch 58/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0068 - accuracy: 0.99
71 - val_loss: 0.5674 - val_accuracy: 0.8567
Epoch 59/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0024 - accuracy: 1.00
00 - val_loss: 0.5809 - val_accuracy: 0.8533
Epoch 60/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0022 - accuracy: 1.00
00 - val_loss: 0.5831 - val_accuracy: 0.8567
Epoch 61/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0019 - accuracy: 1.00
00 - val_loss: 0.5823 - val_accuracy: 0.8567
Epoch 62/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0014 - accuracy: 1.00
00 - val_loss: 0.5780 - val_accuracy: 0.8633
Epoch 63/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0025 - accuracy: 1.00
00 - val_loss: 0.5781 - val_accuracy: 0.8633
Epoch 64/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0045 - accuracy: 0.99
71 - val_loss: 0.5461 - val_accuracy: 0.8800
Epoch 65/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0014 - accuracy: 1.00
00 - val_loss: 0.5150 - val_accuracy: 0.8867
Epoch 66/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0050 - accuracy: 0.99
86 - val_loss: 0.5474 - val_accuracy: 0.8833
Epoch 67/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0058 - accuracy: 0.99
71 - val_loss: 0.5660 - val_accuracy: 0.8800
Epoch 68/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0018 - accuracy: 1.00
00 - val_loss: 0.5986 - val_accuracy: 0.8667
Epoch 69/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0020 - accuracy: 1.00
00 - val_loss: 0.5953 - val_accuracy: 0.8667
Epoch 70/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0019 - accuracy: 1.00
00 - val_loss: 0.5949 - val_accuracy: 0.8633
Epoch 71/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0022 - accuracy: 0.99
86 - val_loss: 0.5914 - val_accuracy: 0.8667
Epoch 72/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0024 - accuracy: 1.00
00 - val_loss: 0.5880 - val_accuracy: 0.8700
Epoch 73/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0037 - accuracy: 0.99
86 - val_loss: 0.5919 - val_accuracy: 0.8800
Epoch 74/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0021 - accuracy: 1.00
00 - val_loss: 0.5969 - val_accuracy: 0.8767
Epoch 75/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0012 - accuracy: 1.00
```

```
00 - val_loss: 0.5956 - val_accuracy: 0.8767
Epoch 76/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0010 - accuracy: 1.00
00 - val_loss: 0.5915 - val_accuracy: 0.8767
Epoch 77/100
22/22 [==============================] - 0s 7ms/step - loss: 8.4449e-04 - accuracy:
1.0000 - val_loss: 0.6106 - val_accuracy: 0.8733
Epoch 78/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0037 - accuracy: 0.99
86 - val_loss: 0.6014 - val_accuracy: 0.8767
Epoch 79/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0014 - accuracy: 1.00
00 - val_loss: 0.5634 - val_accuracy: 0.8867
Epoch 80/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0086 - accuracy: 0.99
57 - val_loss: 0.5687 - val_accuracy: 0.8833
Epoch 81/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0015 - accuracy: 1.00
00 - val_loss: 0.5770 - val_accuracy: 0.8833
Epoch 82/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0024 - accuracy: 1.00
00 - val_loss: 0.5747 - val_accuracy: 0.8833
Epoch 83/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0079 - accuracy: 0.99
71 - val_loss: 0.6650 - val_accuracy: 0.8700
Epoch 84/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0066 - accuracy: 0.99
86 - val_loss: 0.6170 - val_accuracy: 0.8733
Epoch 85/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0022 - accuracy: 0.99
86 - val_loss: 0.5321 - val_accuracy: 0.8967
Epoch 86/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0032 - accuracy: 1.00
00 - val_loss: 0.5412 - val_accuracy: 0.8933
Epoch 87/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0045 - accuracy: 0.99
86 - val_loss: 0.5428 - val_accuracy: 0.8967
Epoch 88/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0012 - accuracy: 1.00
00 - val_loss: 0.5379 - val_accuracy: 0.8967
Epoch 89/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0012 - accuracy: 1.00
00 - val_loss: 0.5394 - val_accuracy: 0.9000
Epoch 90/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0015 - accuracy: 0.99
86 - val_loss: 0.4867 - val_accuracy: 0.8933
Epoch 91/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0024 - accuracy: 1.00
00 - val_loss: 0.4804 - val_accuracy: 0.8900
Epoch 92/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0013 - accuracy: 1.00
00 - val_loss: 0.4827 - val_accuracy: 0.8900
Epoch 93/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0016 - accuracy: 1.00
00 - val_loss: 0.4864 - val_accuracy: 0.8900
Epoch 94/100
```

```
22/22 [==============================] - 0s 4ms/step - loss: 0.0017 - accuracy: 1.00
00 - val_loss: 0.4907 - val_accuracy: 0.8900
Epoch 95/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0026 - accuracy: 0.99
86 - val_loss: 0.5123 - val_accuracy: 0.8967
Epoch 96/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0011 - accuracy: 1.00
00 - val_loss: 0.5566 - val_accuracy: 0.8933
Epoch 97/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0010 - accuracy: 1.00
00 - val_loss: 0.5717 - val_accuracy: 0.8933
Epoch 98/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0015 - accuracy: 1.00
00 - val_loss: 0.5992 - val_accuracy: 0.8833
Epoch 99/100
22/22 [==============================] - 0s 3ms/step - loss: 5.5841e-04 - accuracy:
1.0000 - val_loss: 0.6107 - val_accuracy: 0.8833
Epoch 100/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0026 - accuracy: 0.99
86 - val_loss: 0.6279 - val_accuracy: 0.8767
```

Out[ ]: <keras.callbacks.History at 0x29be89153a0>

In [ ]:

```python
# Now we are going to eval the model, using accuracy, precision, recall, F1 score,

# get predictions
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)  # Convert probabilities to class predictio


# Check the shape of y_pred_prob to determine if it's binary or multiclass
if y_pred_prob.shape[1] == 1:
    # Binary classification
    y_pred = (y_pred_prob > 0.5).astype("int32").flatten()
    roc_auc = roc_auc_score(y_test, y_pred_prob)
else:
    # Multiclass classification
    y_pred = np.argmax(y_pred_prob, axis=1)  # Convert probabilities to class predi
    roc_auc = roc_auc_score(y_test, y_pred_prob, multi_class='ovr')

# Calculate evaluation metrics
f1 = f1_score(y_test, y_pred, average='weighted')  # Use 'weighted' for multiclass
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# Print evaluation metrics
print(f"F1 Score: {f1}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"ROC AUC Score: {roc_auc}")
```

```
10/10 [==============================] - 0s 1ms/step
10/10 [==============================] - 0s 1ms/step
F1 Score: 0.8747109974424552
Accuracy: 0.8766666666666667
Precision: 0.8925867527862209
Recall: 0.8766666666666667
ROC AUC Score: 0.9538773328582246
```

Conext behind the model, and the interpretation of the results

F1 Score: 0.8747109974424552

- The F1 score is a measure precision and recall dataset. The f1 score of 87% shows that it has a good blanance between the

Accuracy: 0.8766666666666667

- The accuracy of the model is 0.8767, which means that the model is able to correctly predict whether a user will click on an online ad 87.67% of the time. This is a good accuracy score, but it is not perfect. There is still room for improvement in the model's performance, bu the accuracy should never be extremely high as it may indicate overfitting, So I am happy with the accuracy of the model.

Precision: 0.8925867527862209

- The precision of the model is 0.8926, which means that the model is able to correctly identify 89.26% of the users who will click on an online ad. This is a good precision score, but it is not perfect. There is still room for improvement in the model's performance. So with the percision being at 89%, between the two prediction metrics, I would say that the precision is more important as it is more important to correctly identify the users who will click on an online ad.

Recall: 0.8766666666666667

- Recall is the true positive rate. A very high recall means that the model is able to correctly identify most of the users who will click on an online ad.

ROC AUC Score: 0.9538773328582246

- An ROC AUC score of 0.9539 indicates that the model has a very high ability to distinguish between the positive and negative classes. with that being said, a model with an ROC AUC score of 0.5 is considered to be random, so the model does do somewhat well in distinguishing between the positive and negative classes.

In [ ]:
```
# Making the model again, with another hidden layer

# model type
```

```
model = tf.keras.models.Sequential()


# input layer
model.add(tf.keras.layers.Dense(units=100, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))


# hidden layers
model.add(tf.keras.layers.Dense(units=50, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))

# New layer
model.add(tf.keras.layers.Dense(units=50, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(units=25, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))


# output layer
model.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

# compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [ ]:

```
# Now we will train the model

model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_t
```

```
Epoch 1/100
22/22 [==============================] - 1s 13ms/step - loss: 0.9835 - accuracy: 0.4
929 - val_loss: 0.6903 - val_accuracy: 0.5433
Epoch 2/100
22/22 [==============================] - 0s 3ms/step - loss: 0.8611 - accuracy: 0.48
00 - val_loss: 0.6914 - val_accuracy: 0.5400
Epoch 3/100
22/22 [==============================] - 0s 3ms/step - loss: 0.7624 - accuracy: 0.54
29 - val_loss: 0.6915 - val_accuracy: 0.5300
Epoch 4/100
22/22 [==============================] - 0s 3ms/step - loss: 0.7790 - accuracy: 0.46
71 - val_loss: 0.6916 - val_accuracy: 0.5233
Epoch 5/100
22/22 [==============================] - 0s 3ms/step - loss: 0.7478 - accuracy: 0.52
57 - val_loss: 0.6919 - val_accuracy: 0.5600
Epoch 6/100
22/22 [==============================] - 0s 4ms/step - loss: 0.7194 - accuracy: 0.52
14 - val_loss: 0.6921 - val_accuracy: 0.5067
Epoch 7/100
22/22 [==============================] - 0s 3ms/step - loss: 0.7136 - accuracy: 0.53
43 - val_loss: 0.6920 - val_accuracy: 0.5100
Epoch 8/100
22/22 [==============================] - 0s 3ms/step - loss: 0.6934 - accuracy: 0.56
86 - val_loss: 0.6912 - val_accuracy: 0.5300
Epoch 9/100
22/22 [==============================] - 0s 4ms/step - loss: 0.7008 - accuracy: 0.55
86 - val_loss: 0.6907 - val_accuracy: 0.5300
Epoch 10/100
22/22 [==============================] - 0s 4ms/step - loss: 0.6778 - accuracy: 0.59
29 - val_loss: 0.6896 - val_accuracy: 0.5433
Epoch 11/100
22/22 [==============================] - 0s 4ms/step - loss: 0.6895 - accuracy: 0.57
43 - val_loss: 0.6873 - val_accuracy: 0.5833
Epoch 12/100
22/22 [==============================] - 0s 4ms/step - loss: 0.6342 - accuracy: 0.63
00 - val_loss: 0.6836 - val_accuracy: 0.6600
Epoch 13/100
22/22 [==============================] - 0s 4ms/step - loss: 0.6479 - accuracy: 0.61
86 - val_loss: 0.6794 - val_accuracy: 0.6267
Epoch 14/100
22/22 [==============================] - 0s 3ms/step - loss: 0.6116 - accuracy: 0.66
14 - val_loss: 0.6695 - val_accuracy: 0.6500
Epoch 15/100
22/22 [==============================] - 0s 4ms/step - loss: 0.5762 - accuracy: 0.69
14 - val_loss: 0.6478 - val_accuracy: 0.7300
Epoch 16/100
22/22 [==============================] - 0s 3ms/step - loss: 0.5221 - accuracy: 0.71
71 - val_loss: 0.6173 - val_accuracy: 0.8067
Epoch 17/100
22/22 [==============================] - 0s 3ms/step - loss: 0.4567 - accuracy: 0.75
29 - val_loss: 0.5799 - val_accuracy: 0.8600
Epoch 18/100
22/22 [==============================] - 0s 3ms/step - loss: 0.4059 - accuracy: 0.81
14 - val_loss: 0.5275 - val_accuracy: 0.8533
Epoch 19/100
22/22 [==============================] - 0s 4ms/step - loss: 0.3663 - accuracy: 0.83
```

```
57 - val_loss: 0.4626 - val_accuracy: 0.8900
Epoch 20/100
22/22 [==============================] - 0s 3ms/step - loss: 0.2692 - accuracy: 0.88
71 - val_loss: 0.4031 - val_accuracy: 0.9067
Epoch 21/100
22/22 [==============================] - 0s 4ms/step - loss: 0.2403 - accuracy: 0.90
57 - val_loss: 0.3468 - val_accuracy: 0.9200
Epoch 22/100
22/22 [==============================] - 0s 3ms/step - loss: 0.1890 - accuracy: 0.92
57 - val_loss: 0.3263 - val_accuracy: 0.9200
Epoch 23/100
22/22 [==============================] - 0s 3ms/step - loss: 0.1254 - accuracy: 0.95
71 - val_loss: 0.3177 - val_accuracy: 0.8967
Epoch 24/100
22/22 [==============================] - 0s 3ms/step - loss: 0.1367 - accuracy: 0.95
00 - val_loss: 0.3079 - val_accuracy: 0.8967
Epoch 25/100
22/22 [==============================] - 0s 4ms/step - loss: 0.1055 - accuracy: 0.96
57 - val_loss: 0.2942 - val_accuracy: 0.8967
Epoch 26/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0896 - accuracy: 0.97
43 - val_loss: 0.2690 - val_accuracy: 0.9000
Epoch 27/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0657 - accuracy: 0.98
00 - val_loss: 0.2642 - val_accuracy: 0.9100
Epoch 28/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0818 - accuracy: 0.97
29 - val_loss: 0.2589 - val_accuracy: 0.9033
Epoch 29/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0508 - accuracy: 0.98
29 - val_loss: 0.2404 - val_accuracy: 0.9100
Epoch 30/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0505 - accuracy: 0.98
57 - val_loss: 0.2502 - val_accuracy: 0.9067
Epoch 31/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0476 - accuracy: 0.98
86 - val_loss: 0.2461 - val_accuracy: 0.9067
Epoch 32/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0438 - accuracy: 0.99
14 - val_loss: 0.2286 - val_accuracy: 0.9167
Epoch 33/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0318 - accuracy: 0.99
00 - val_loss: 0.2210 - val_accuracy: 0.9167
Epoch 34/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0449 - accuracy: 0.98
71 - val_loss: 0.2324 - val_accuracy: 0.9133
Epoch 35/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0448 - accuracy: 0.98
57 - val_loss: 0.2360 - val_accuracy: 0.9167
Epoch 36/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0376 - accuracy: 0.98
86 - val_loss: 0.2427 - val_accuracy: 0.9167
Epoch 37/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0319 - accuracy: 0.99
14 - val_loss: 0.2397 - val_accuracy: 0.9167
Epoch 38/100
```

```
22/22 [==============================] - 0s 4ms/step - loss: 0.0373 - accuracy: 0.98
86 - val_loss: 0.2670 - val_accuracy: 0.9100
Epoch 39/100
22/22 [==============================] - 0s 6ms/step - loss: 0.0169 - accuracy: 0.99
43 - val_loss: 0.2661 - val_accuracy: 0.9067
Epoch 40/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0292 - accuracy: 0.98
86 - val_loss: 0.2860 - val_accuracy: 0.9000
Epoch 41/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0303 - accuracy: 0.99
00 - val_loss: 0.2781 - val_accuracy: 0.8967
Epoch 42/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0120 - accuracy: 0.99
43 - val_loss: 0.2736 - val_accuracy: 0.9000
Epoch 43/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0205 - accuracy: 0.99
57 - val_loss: 0.2696 - val_accuracy: 0.9133
Epoch 44/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0164 - accuracy: 0.99
86 - val_loss: 0.2490 - val_accuracy: 0.9100
Epoch 45/100
22/22 [==============================] - 0s 7ms/step - loss: 0.0097 - accuracy: 0.99
57 - val_loss: 0.2498 - val_accuracy: 0.9133
Epoch 46/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0162 - accuracy: 0.99
57 - val_loss: 0.2589 - val_accuracy: 0.9100
Epoch 47/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0110 - accuracy: 0.99
71 - val_loss: 0.2761 - val_accuracy: 0.9100
Epoch 48/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0150 - accuracy: 0.99
43 - val_loss: 0.2888 - val_accuracy: 0.9100
Epoch 49/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0105 - accuracy: 0.99
71 - val_loss: 0.2948 - val_accuracy: 0.9067
Epoch 50/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0184 - accuracy: 0.99
43 - val_loss: 0.2836 - val_accuracy: 0.9067
Epoch 51/100
22/22 [==============================] - 0s 10ms/step - loss: 0.0069 - accuracy: 0.9
971 - val_loss: 0.2830 - val_accuracy: 0.9100
Epoch 52/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0286 - accuracy: 0.99
29 - val_loss: 0.2567 - val_accuracy: 0.9233
Epoch 53/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0093 - accuracy: 0.99
57 - val_loss: 0.2506 - val_accuracy: 0.9233
Epoch 54/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0066 - accuracy: 1.00
00 - val_loss: 0.2566 - val_accuracy: 0.9233
Epoch 55/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0122 - accuracy: 0.99
71 - val_loss: 0.2633 - val_accuracy: 0.9233
Epoch 56/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0075 - accuracy: 0.99
71 - val_loss: 0.2758 - val_accuracy: 0.9233
```

```
Epoch 57/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0088 - accuracy: 0.99
71 - val_loss: 0.2753 - val_accuracy: 0.9233
Epoch 58/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0035 - accuracy: 1.00
00 - val_loss: 0.2810 - val_accuracy: 0.9233
Epoch 59/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0133 - accuracy: 0.99
57 - val_loss: 0.2981 - val_accuracy: 0.9133
Epoch 60/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0051 - accuracy: 1.00
00 - val_loss: 0.3165 - val_accuracy: 0.9100
Epoch 61/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0120 - accuracy: 0.99
71 - val_loss: 0.3408 - val_accuracy: 0.9067
Epoch 62/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0075 - accuracy: 0.99
57 - val_loss: 0.3542 - val_accuracy: 0.9067
Epoch 63/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0080 - accuracy: 0.99
71 - val_loss: 0.3640 - val_accuracy: 0.9033
Epoch 64/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0097 - accuracy: 0.99
57 - val_loss: 0.3766 - val_accuracy: 0.9033
Epoch 65/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0041 - accuracy: 0.99
86 - val_loss: 0.3815 - val_accuracy: 0.9033
Epoch 66/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0220 - accuracy: 0.99
43 - val_loss: 0.3705 - val_accuracy: 0.9033
Epoch 67/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0036 - accuracy: 1.00
00 - val_loss: 0.3476 - val_accuracy: 0.9100
Epoch 68/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0125 - accuracy: 0.99
57 - val_loss: 0.3279 - val_accuracy: 0.9133
Epoch 69/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0044 - accuracy: 0.99
86 - val_loss: 0.3368 - val_accuracy: 0.9133
Epoch 70/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0031 - accuracy: 0.99
86 - val_loss: 0.3376 - val_accuracy: 0.9133
Epoch 71/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0038 - accuracy: 0.99
86 - val_loss: 0.3392 - val_accuracy: 0.9133
Epoch 72/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0087 - accuracy: 0.99
43 - val_loss: 0.3522 - val_accuracy: 0.9133
Epoch 73/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0045 - accuracy: 0.99
86 - val_loss: 0.3651 - val_accuracy: 0.9100
Epoch 74/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0047 - accuracy: 0.99
71 - val_loss: 0.3694 - val_accuracy: 0.9067
Epoch 75/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0050 - accuracy: 0.99
```

```
71 - val_loss: 0.3747 - val_accuracy: 0.9067
Epoch 76/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0109 - accuracy: 0.99
71 - val_loss: 0.3555 - val_accuracy: 0.9133
Epoch 77/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0058 - accuracy: 0.99
86 - val_loss: 0.3614 - val_accuracy: 0.9100
Epoch 78/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0028 - accuracy: 1.00
00 - val_loss: 0.3867 - val_accuracy: 0.9033
Epoch 79/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0037 - accuracy: 1.00
00 - val_loss: 0.3932 - val_accuracy: 0.9033
Epoch 80/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0036 - accuracy: 1.00
00 - val_loss: 0.3947 - val_accuracy: 0.9067
Epoch 81/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0027 - accuracy: 1.00
00 - val_loss: 0.4145 - val_accuracy: 0.9033
Epoch 82/100
22/22 [==============================] - 0s 4ms/step - loss: 4.4513e-04 - accuracy:
1.0000 - val_loss: 0.4254 - val_accuracy: 0.9033
Epoch 83/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0061 - accuracy: 0.99
86 - val_loss: 0.4255 - val_accuracy: 0.9067
Epoch 84/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0063 - accuracy: 0.99
86 - val_loss: 0.4080 - val_accuracy: 0.9067
Epoch 85/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0046 - accuracy: 0.99
86 - val_loss: 0.3727 - val_accuracy: 0.9100
Epoch 86/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0112 - accuracy: 0.99
57 - val_loss: 0.3759 - val_accuracy: 0.9133
Epoch 87/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0035 - accuracy: 0.99
86 - val_loss: 0.3862 - val_accuracy: 0.9100
Epoch 88/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0043 - accuracy: 0.99
71 - val_loss: 0.3922 - val_accuracy: 0.9100
Epoch 89/100
22/22 [==============================] - 0s 7ms/step - loss: 0.0042 - accuracy: 0.99
86 - val_loss: 0.4012 - val_accuracy: 0.9100
Epoch 90/100
22/22 [==============================] - 0s 6ms/step - loss: 0.0022 - accuracy: 0.99
86 - val_loss: 0.4101 - val_accuracy: 0.9100
Epoch 91/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0035 - accuracy: 0.99
86 - val_loss: 0.4132 - val_accuracy: 0.9100
Epoch 92/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0145 - accuracy: 0.99
57 - val_loss: 0.3977 - val_accuracy: 0.9167
Epoch 93/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0020 - accuracy: 0.99
86 - val_loss: 0.3823 - val_accuracy: 0.9200
Epoch 94/100
```

```
22/22 [==============================] - 0s 4ms/step - loss: 0.0024 - accuracy: 1.00
00 - val_loss: 0.3871 - val_accuracy: 0.9233
Epoch 95/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0234 - accuracy: 0.99
57 - val_loss: 0.3615 - val_accuracy: 0.9233
Epoch 96/100
22/22 [==============================] - 0s 5ms/step - loss: 0.0085 - accuracy: 0.99
71 - val_loss: 0.3583 - val_accuracy: 0.9267
Epoch 97/100
22/22 [==============================] - 0s 7ms/step - loss: 0.0018 - accuracy: 1.00
00 - val_loss: 0.3564 - val_accuracy: 0.9233
Epoch 98/100
22/22 [==============================] - 0s 6ms/step - loss: 0.0071 - accuracy: 0.99
71 - val_loss: 0.3395 - val_accuracy: 0.9200
Epoch 99/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0020 - accuracy: 1.00
00 - val_loss: 0.3423 - val_accuracy: 0.9200
Epoch 100/100
22/22 [==============================] - 0s 4ms/step - loss: 0.0064 - accuracy: 0.99
86 - val_loss: 0.3546 - val_accuracy: 0.9167
```

Out[ ]: <keras.callbacks.History at 0x29be433df40>

In [ ]:
```python
# get predictions
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)  # Convert probabilities to class predictio


# Check the shape of y_pred_prob to determine if it's binary or multiclass
if y_pred_prob.shape[1] == 1:
    # Binary classification
    y_pred = (y_pred_prob > 0.5).astype("int32").flatten()
    roc_auc = roc_auc_score(y_test, y_pred_prob)
else:
    # Multiclass classification
    y_pred = np.argmax(y_pred_prob, axis=1)  # Convert probabilities to class predi
    roc_auc = roc_auc_score(y_test, y_pred_prob, multi_class='ovr')

# Calculate evaluation metrics
f1 = f1_score(y_test, y_pred, average='weighted')  # Use 'weighted' for multiclass
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# Print evaluation metrics
print(f"F1 Score: {f1}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"ROC AUC Score: {roc_auc}")
```

```
10/10 [==============================] - 0s 3ms/step
F1 Score: 0.9166972633577738
Accuracy: 0.9166666666666666
Precision: 0.9169112850619698
Recall: 0.9166666666666666
ROC AUC Score: 0.9744777515478152
```

That actually made the model worse, everything went down, other than the ROC AUC score, which went up, only by .01 though.

Making a Decision Tree Classifier model, and a Random Forest Classifier model for the bonus points.

In [ ]:
```python
# Now we are going to reload the data and make a tree classifier and random forest

df = pd.read_csv('data.csv')
```

In [ ]:
```python
# importing the necessary libraries for the tree classifier and random forest class

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```python
# now we are going to qiuckly clean the data

df.drop('Timestamp', axis=1, inplace=True)

df = pd.get_dummies(df, columns=['City', 'Country', 'Ad Topic Line'], drop_first=Tr

X = df.drop('Clicked on Ad', axis=1)
y = df['Clicked on Ad']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

In [ ]:
```python
# Now we are going to build the tree classifier

tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)

# get predictions
y_pred = tree.predict(X_test)

# Calculate evaluation metrics
f1_tc = f1_score(y_test, y_pred, average='weighted')  # Use 'weighted' for multicla
accuracy_tc = accuracy_score(y_test, y_pred)
precision_tc = precision_score(y_test, y_pred, average='weighted')
recall_tc = recall_score(y_test, y_pred, average='weighted')
roc_auc_tc = roc_auc_score(y_test, tree.predict_proba(X_test)[:, 1])
```

In [ ]:
```python
# Print evaluation metrics
print(f"F1 Score: {f1_tc}")
print(f"Accuracy: {accuracy_tc}")
```

```
print(f"Precision: {precision_tc}")
print(f"Recall: {recall_tc}")
print(f"ROC AUC Score: {roc_auc_tc}")
```

```
F1 Score: 0.9532958651144118
Accuracy: 0.9533333333333334
Precision: 0.9535990586412857
Recall: 0.9533333333333334
ROC AUC Score: 0.9526079016524877
```

In [ ]:
```
# Now we are going to build the random forest classifier

rf = RandomForestClassifier()
rf.fit(X_train, y_train)

# get predictions
y_pred = rf.predict(X_test)

# Calculate evaluation metrics
f1_rf = f1_score(y_test, y_pred, average='weighted')  # Use 'weighted' for multicla
accuracy_rf = accuracy_score(y_test, y_pred)
precision_rf = precision_score(y_test, y_pred, average='weighted')
recall_rf = recall_score(y_test, y_pred, average='weighted')
roc_auc_rf = roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])
```

In [ ]:
```
# printing for the random forest classifier
print(f"F1 Score: {f1_rf}")
print(f"Accuracy: {accuracy_rf}")
print(f"Precision: {precision_rf}")
print(f"Recall: {recall_rf}")
print(f"ROC AUC Score: {roc_auc_rf}")
```

```
F1 Score: 0.9633272069061541
Accuracy: 0.9633333333333334
Precision: 0.9633425447197957
Recall: 0.9633333333333334
ROC AUC Score: 0.9957908333704512
```

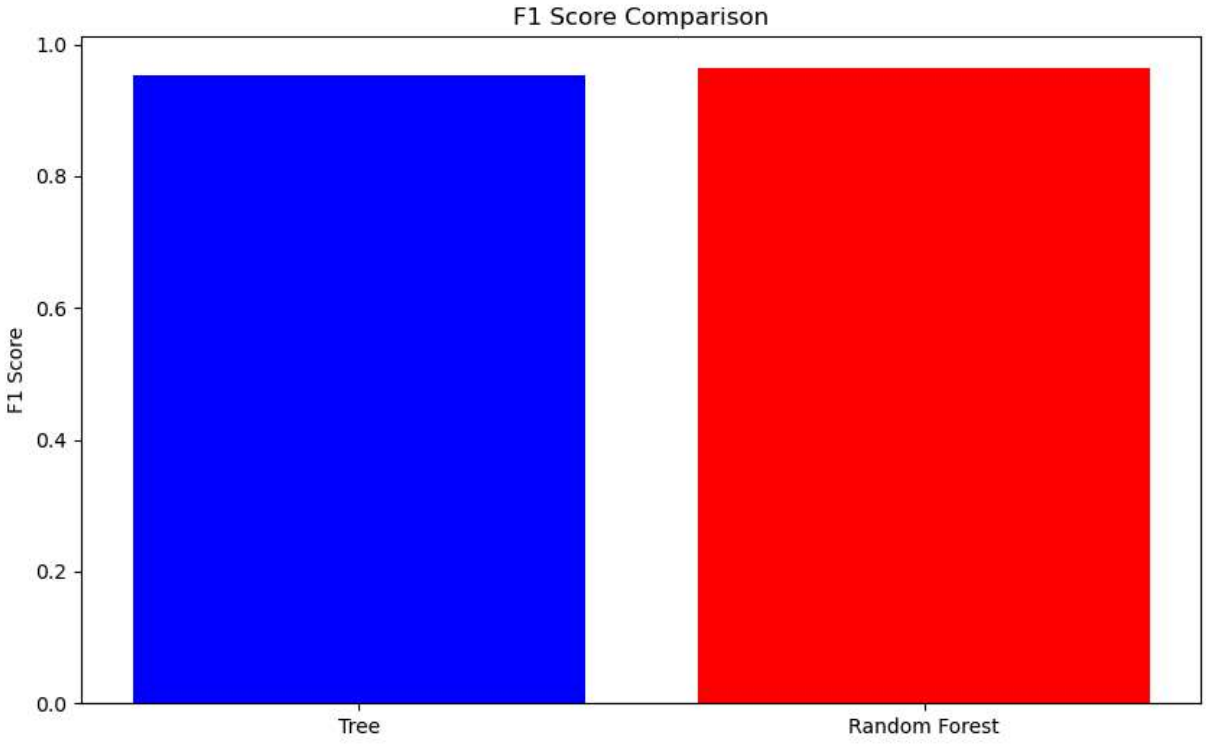
In [ ]:
```
# importing matplotlib for the visualization

import matplotlib.pyplot as plt
```

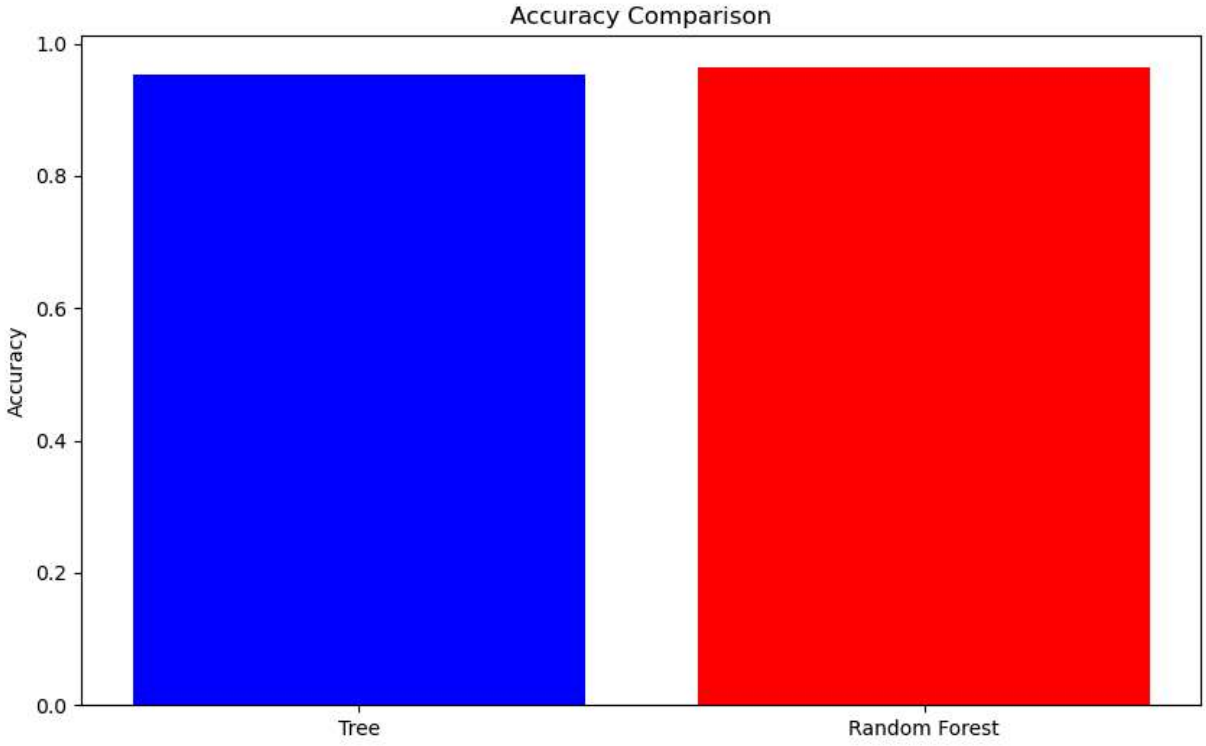
In [ ]:
```
# Now comparing the two tree models to eachother

print(f"F1 Score: {f1_tc} vs {f1_rf}")
print(f"Accuracy: {accuracy_tc} vs {accuracy_rf}")
print(f"Precision: {precision_tc} vs {precision_rf}")
print(f"Recall: {recall_tc} vs {recall_rf}")
print(f"ROC AUC Score: {roc_auc_tc} vs {roc_auc_rf}")
```

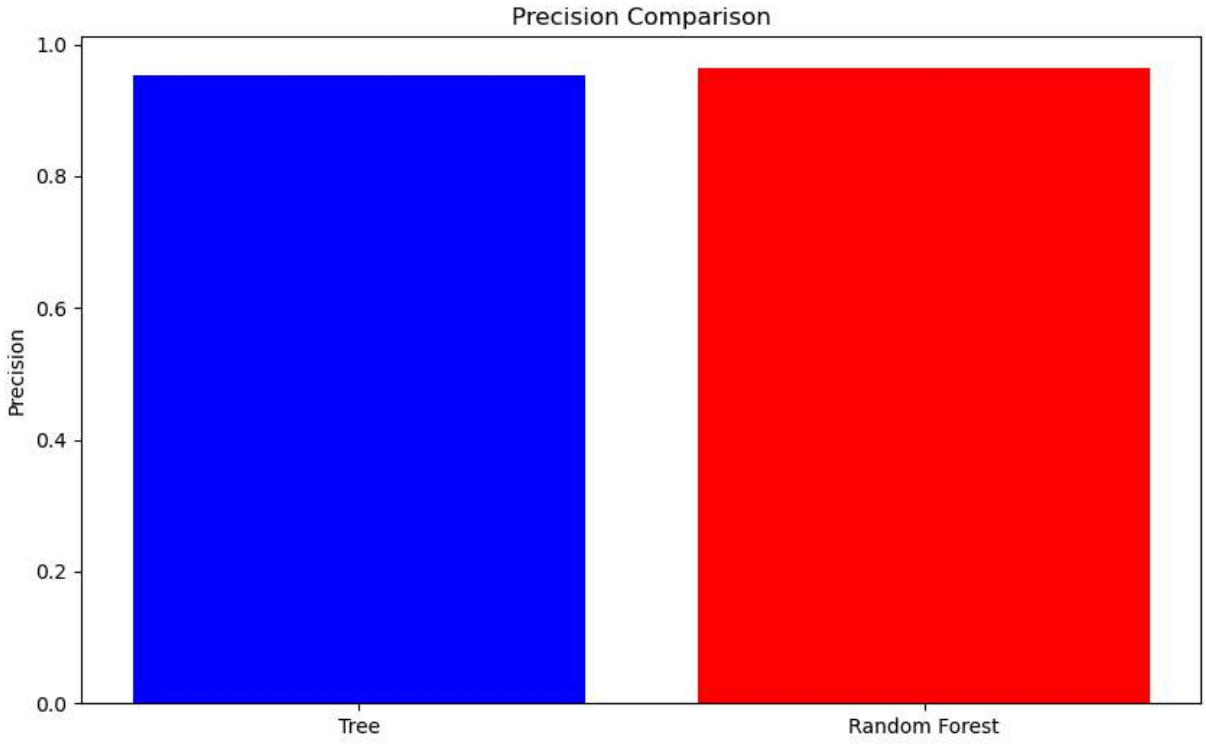
```
F1 Score: 0.9532958651144118 vs 0.9633272069061541
Accuracy: 0.9533333333333334 vs 0.9633333333333334
Precision: 0.9535990586412857 vs 0.9633425447197957
Recall: 0.9533333333333334 vs 0.9633333333333334
ROC AUC Score: 0.9526079016524877 vs 0.9957908333704512
```

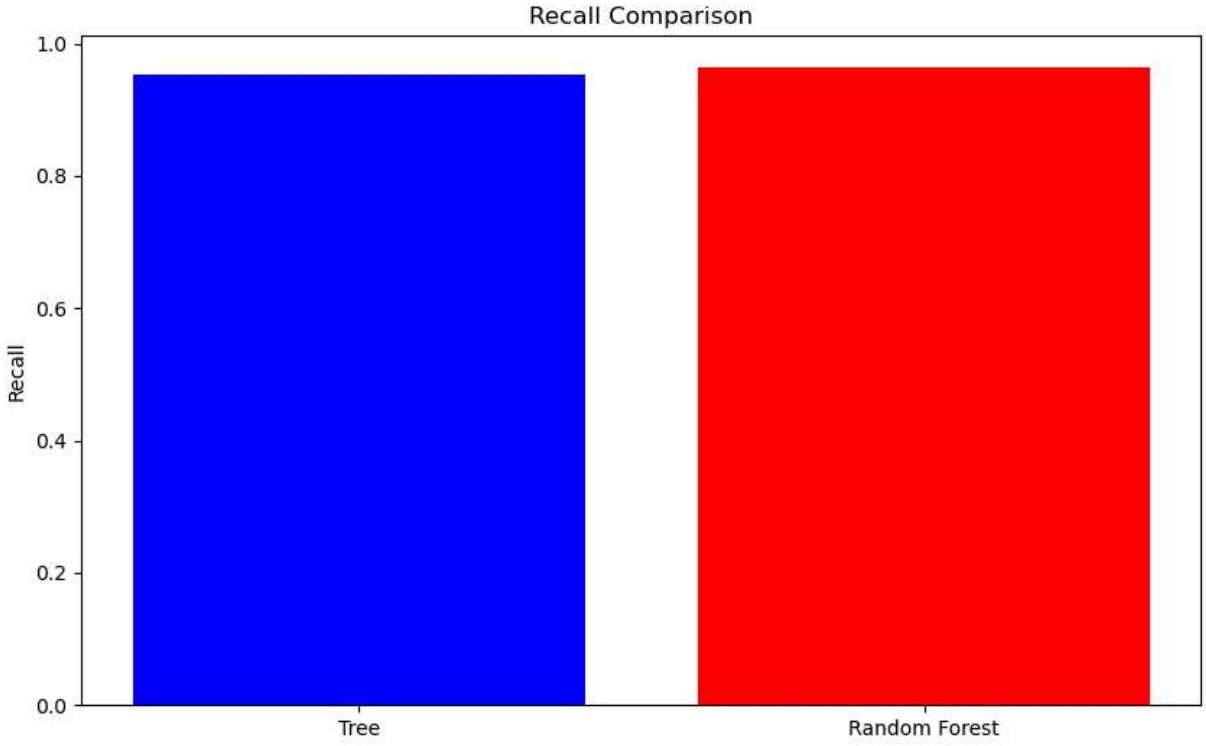
In [ ]:

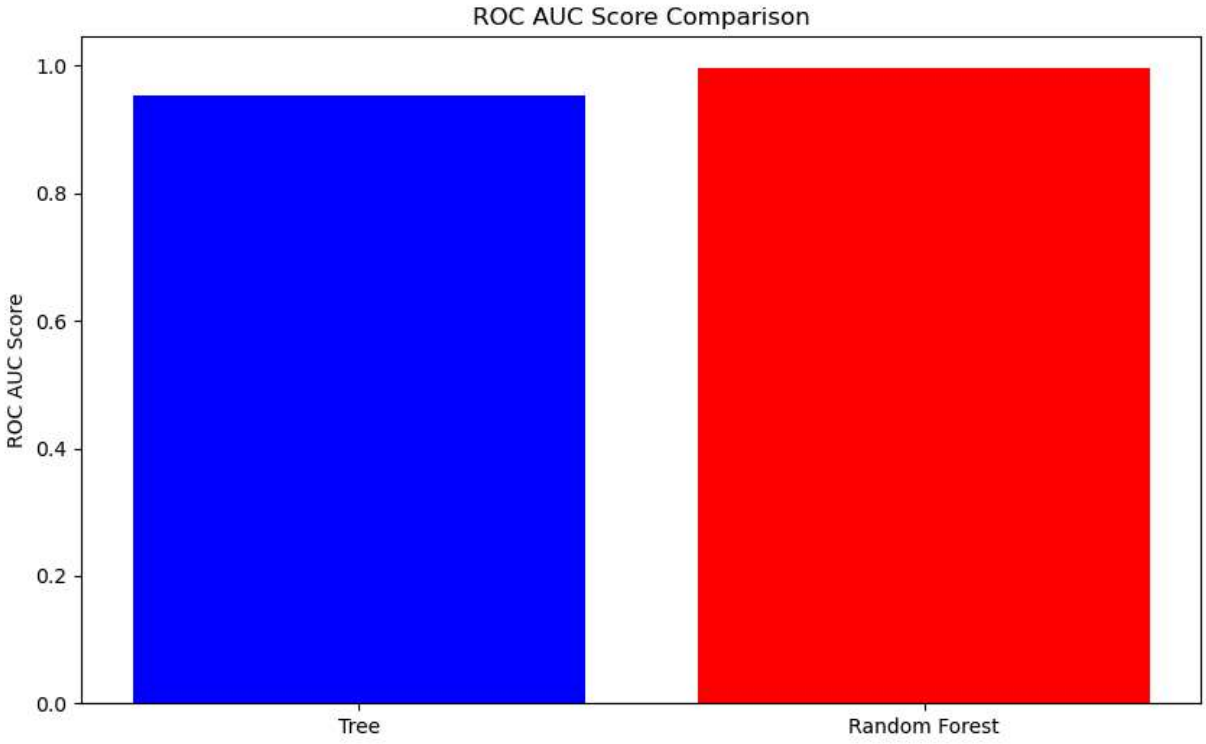
```
# Compparing the test of the models to eachother creating individual plots for each

# F1 Score
plt.figure(figsize=(10, 6))
plt.bar(["Tree", "Random Forest"], [f1_tc, f1_rf], color=['blue', 'red'])
plt.title("F1 Score Comparison")
plt.ylabel("F1 Score")
plt.show()

# Accuracy
plt.figure(figsize=(10, 6))
plt.bar(["Tree", "Random Forest"], [accuracy_tc, accuracy_rf], color=['blue', 'red'
plt.title("Accuracy Comparison")
plt.ylabel("Accuracy")
plt.show()

# Precision
plt.figure(figsize=(10, 6))
plt.bar(["Tree", "Random Forest"], [precision_tc, precision_rf], color=['blue', 're
plt.title("Precision Comparison")
plt.ylabel("Precision")
plt.show()

# Recall
plt.figure(figsize=(10, 6))
plt.bar(["Tree", "Random Forest"], [recall_tc, recall_rf], color=['blue', 'red'])
plt.title("Recall Comparison")
plt.ylabel("Recall")
plt.show()

# ROC AUC Score
plt.figure(figsize=(10, 6))
plt.bar(["Tree", "Random Forest"], [roc_auc_tc, roc_auc_rf], color=['blue', 'red'])
plt.title("ROC AUC Score Comparison")
plt.ylabel("ROC AUC Score")
plt.show()
```


F1 Score Comparison
F1 Score
1.0
0.8
0.6
0.4
0.2
0.0
Tree
Random Forest


Accuracy Comparison
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Tree
Random Forest


Precision Comparison
Precision
1.0
0.8
0.6
0.4
0.2
0.0
Tree
Random Forest


Recall Comparison
Recall
1.0
0.8
0.6
0.4
0.2
0.0
Tree
Random Forest


ROC AUC Score Comparison
ROC AUC Score
1.0
0.8
0.6
0.4
0.2
0.0
Tree
Random Forest

In [ ]: