Gangadhar Singh Shiva Assignment 2

Advertisement Click Prediction Project/Assignment Description

Background about the member performing the analysis

As a data scientist at an advertising agency. Goal is to predict whether a user will click on an online advertisement based on various user attributes. The dataset includes both behavioral and demographic features.

Assignment Objectives

To build and optimize an Artificial Neural Network (ANN) classifier to predict ad-click behavior, and compare its performance with traditional machine learning models and synthetic data approaches.

Dataset Features

- **Numerical Features**:
  - Daily Time Spent on Site
  - Age
  - Area Income
  - Daily Internet Usage
- **Categorical/Temporal Features** *(initially removed for ANN)*:
  - Ad Topic Line
  - City
  - Country
  - Timestamp
- **Binary Feature**:

  - Male

- **Target Variable**:
  - Clicked on Ad

Workflow Steps

1. **Load** the dataset using `pandas`.
2. **Clean and preprocess** the data: Drop non-numeric and high-cardinality features.
3. **Split** the data into training and testing sets using an 80:20 ratio.
4. **Scale** numerical features using `StandardScaler`.
5. **Modeling**:
   - Build and train an **Artificial Neural Network (ANN)**.
   - Train **Logistic Regression** and **Random Forest** models for baseline comparison.
   - Analyze **Feature Importance** using Random Forest and SHAP, Permutation Methods
6. **Pipeline Modeling**:
   - Construct machine learning pipelines combining preprocessing and model training (e.g., Random Forest → ANN).
7. **Synthetic Data Comparison Modeling**:
   - Generate synthetic data using similar generator.
   - Train ANN on the synthetic dataset and evaluate on the original test set.
   - Compare real-trained ANN vs. synthetic-trained ANN performance.

Evaluation Metrics

- **Accuracy**: Overall correctness of the model.
- **Precision**: Correctness among predicted positive (clicked) instances.(False Positive)
- **Recall**: Proportion of actual clicks that were correctly predicted. (False Negative)
- **F1 Score**: Harmonic mean of precision and recall.
- **ROC AUC Score**: Model's ability to distinguish between classes across thresholds.

Key Takeaways from this assignment

- Identify the best performing model.

- Understand the significance of user behavior features.
- Explore the feasibility and limitations of using synthetic data for model training.

Question to be answered

1.Load the dataset into pandas dataframe.

2.Perform data cleaning and preprocessing as necessary.

3.Split the data into training and testing sets using an 80:20 ratio.

4.Scale the data using StandardScaler.

5.Build an ANN classification model.

6.Experiment with different model architectures, activation functions, regularization techniques, learning rates, and batch sizes to optimize the model's performance.

7.Evaluate the model's performance using accuracy, precision, recall, F1 score, and ROC AUC score as the metrics.

8.Interpret the results and draw conclusions about the factors that are most important in predicting whether a user will click on an online ad.

9.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).

```
!pip install ctgan
```

```
Requirement already satisfied: ctgan in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: numpy>=1.23.3 in /usr/local/lib/python3.11/dis
Requirement already satisfied: pandas>=1.5.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: tqdm<5,>=4.29 in /usr/local/lib/python3.11/dis
Requirement already satisfied: rdt>=1.14.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pytho
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/di
Requirement already satisfied: scipy>=1.9.2 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: scikit-learn>=1.1.3 in /usr/local/lib/python3.
Requirement already satisfied: Faker>=17 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/py
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/loc
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/l
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/l
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/li
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/pyt
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/p
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.1
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/d
```

```
# Install Required Libraries
!pip install scikit-learn keras tensorflow
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-
Requirement already satisfied: keras in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/loc
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.1
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.1
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/pyth
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.1
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/pytho
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/lo
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyth
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/di
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/l
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/di
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/pytho
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/
```

1.Load the dataset into pandas dataframe.

```
#  Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
import pandas as pd
try:
    df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/AAI-511/M2-Advertis

except FileNotFoundError:
    print("Dataset not found. Please upload the dataset to your Google Drive and
    exit()
```

```
df.head()
```

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timesta
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03 00:53
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04 01:39
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03 20:35

```
df.tail()
```

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Tin
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	20
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	20
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	20

```
df.describe()
```

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	65.000200	36.009000	55000.000080	180.000100	0.481000	0.50000
std	15.853615	8.785562	13414.634022	43.902339	0.499889	0.50025
min	32.600000	19.000000	13996.500000	104.780000	0.000000	0.00000
25%	51.360000	29.000000	47031.802500	138.830000	0.000000	0.00000
50%	68.215000	35.000000	57012.300000	183.130000	0.000000	0.50000
75%	78.547500	42.000000	65470.635000	218.792500	1.000000	1.00000

```
df.shape
```

```
(1000, 10)
```

```
df.size
```

10000

```
df
```

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Ti
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2
...	...	...	...	...	...	...	...	...	
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	2
					Expanded				

2 to 8 are answered in the below code block

2.Perform data cleaning and preprocessing as necessary.

3.Split the data into training and testing sets using an 80:20 ratio.

4.Scale the data using StandardScaler.

5.Build an ANN classification model.

6.Experiment with different model architectures, activation functions, regularization techniques, learning rates, and batch sizes to optimize the model's performance.

7.Evaluate the model's performance using accuracy, precision, recall, F1 score, and ROC AUC score as the metrics.

8.Interpret the results and draw conclusions about the factors that are most important in predicting whether a user will click on an online ad.

```
'''
2.Perform data cleaning and preprocessing as necessary.

3.Split the data into training and testing sets using an 80:20 ratio.

4.Scale the data using StandardScaler.

5.Build an ANN classification model.

6.Experiment with different model architectures, activation functions, regulariza

7.Evaluate the model's performance using accuracy, precision, recall, F1 score, a

8.Interpret the results and draw conclusions about the factors that are most impo
'''
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sco
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
```

```
#2.Perform data cleaning and preprocessing as necessary.

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

# Dropping non-numeric columns before scaling in this analysis
X = X.drop(['Ad Topic Line', 'City', 'Country', 'Timestamp'], axis=1)

#3.Split the data into training and testing sets using an 80:20 ratio.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

#4.Scale the data using StandardScaler.
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#5.Build an ANN classification model.

# Function to build the model with given hyperparameters
def build_model(learning_rate=0.001, layers=[64, 32], activation='relu', dropout_
    model = Sequential()
    model.add(Dense(layers[0], input_dim=input_dim, activation=activation))
    if dropout_rate > 0:
        model.add(Dropout(dropout_rate))
    for layer_size in layers[1:]:
        model.add(Dense(layer_size, activation=activation))
        if dropout_rate > 0:
            model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation='sigmoid')) # Output layer for binary classific

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accu
    return model

#6.Experiment with different model architectures, activation functions,
# regularization techniques, learning rates, and batch sizes to optimize the mode

# Experimentation with hyperparameters (Example - you would likely use techniques
best_accuracy = 0
best_model = None
best_hyperparameters = {}

# Use Looping technique to get the best hyperparameter details
```

```
learning_rates = [0.01, 0.001, 0.0001]
layer_configs =  [[64, 32], [128, 64, 32], [50]]
activations = ['relu', 'tanh','sigmoid']
dropout_rates = [0.0, 0.2, 0.5]
batch_sizes = [32, 64, 128]
epochs = 20 #

input_dim = X_train_scaled.shape[1]


for lr in learning_rates:
    for layers in layer_configs:
        for activation in activations:
            for dropout in dropout_rates:
                for batch_size in batch_sizes:
                    print(f"\nTraining with LR: {lr}, Layers: {layers}, Activatio


                    model = build_model(learning_rate=lr, layers=layers, activati

                    history = model.fit(X_train_scaled, y_train,
                                        epochs=epochs,
                                        batch_size=batch_size,
                                        validation_split=0.1,
                                        verbose=0)

                    # Evaluate on the test set
                    y_pred_proba = model.predict(X_test_scaled)
                    y_pred = (y_pred_proba > 0.5).astype(int)

                    accuracy = accuracy_score(y_test, y_pred)
                    precision = precision_score(y_test, y_pred)
                    recall = recall_score(y_test, y_pred)
                    f1 = f1_score(y_test, y_pred)
                    roc_auc = roc_auc_score(y_test, y_pred_proba)

                    # print(f"  Accuracy: {accuracy:.4f}, Precision: {precision:.
                    # Accuracy determine the best model parameters
                    if accuracy > best_accuracy:
                        best_accuracy = accuracy
                        best_model = model
                        best_hyperparameters = {
                            'learning_rate': lr,
                            'layers': layers,
                            'activation': activation,
```

```
            'dropout_rate': dropout,
            'batch_size': batch_size
        }
```

```
Training with LR: 0.01, Layers: [64, 32], Activation: relu, Dropout: 0.0, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: relu, Dropout: 0.0, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: relu, Dropout: 0.0, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: relu, Dropout: 0.2, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: relu, Dropout: 0.2, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: relu, Dropout: 0.2, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: relu, Dropout: 0.5, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: relu, Dropout: 0.5, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: relu, Dropout: 0.5, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: tanh, Dropout: 0.0, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: tanh, Dropout: 0.0, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: tanh, Dropout: 0.0, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: tanh, Dropout: 0.2, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: tanh, Dropout: 0.2, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
```

```
Training with LR: 0.01, Layers: [64, 32], Activation: tanh, Dropout: 0.2, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: tanh, Dropout: 0.5, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: tanh, Dropout: 0.5, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: tanh, Dropout: 0.5, Ba
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step

Training with LR: 0.01, Layers: [64, 32], Activation: sigmoid, Dropout: 0.0,
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
```

Classification Metrics Explained

This block outlines key classification metrics including formulas, use cases, and when to use them — particularly in binary classification tasks.

---

1. Accuracy

Formula: (TP + TN) / (TP + TN + FP + FN)

Where:

- TP : True Positives
- TN : True Negatives
- FP : False Positives
- FN : False Negatives

Meaning:
The ratio of correctly predicted observations (both positive and negative) to the total observations.

When to Use:

- Classes are **balanced**
- If we care about **overall correctness**

---

2. Precision

Formula: Precision = TP / (TP + FP)

Meaning:
Of all the predicted **positive** cases, how many were actually **positive**.

When to Use:

- **False positives** are costly (e.g., spam detection, cancer diagnosis)
- if we want to avoid **raising false alarms**

---

3. Recall (Sensitivity / True Positive Rate)

Formula: Recall = TP / (TP + FN)

Meaning:
Of all the actual **positive** cases, how many were correctly predicted.

When to Use:

- **False negatives** are costly (e.g., missing a disease, fraud detection)
- if we wnat to **catch as many positives as possible**

---

4. F1 Score

Formula: F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

Meaning:
The harmonic mean of Precision and Recall — balances both when they are in conflict.

When to Use:

- if want a **single metric** that balances **Precision** and **Recall**
- The dataset is **imbalanced**

---

5. ROC AUC (Receiver Operating Characteristic - Area Under Curve)

The ROC curve plots:

True Positive Rate (TPR) vs. False Positive Rate (FPR) The curve is generated by plotting TPR against FPR at various classification thresholds. The area under this curve is called the AUC score.

ROC Curve:
Plots TPR vs FPR at various classification thresholds.

AUC Score:

- **1.0**: Perfect classifier
- **0.5**: Random guessing
- **< 0.5**: Worse than random

When to Use:

- Evaluating classifier performance across **all thresholds**
- Assessing **ranking quality**

```
# print the best metric performance metrics

#7.Evaluate the model's performance using accuracy, precision, recall, F1 score,
print("\n--- Best Model Performance ---")
print(f"Best Hyperparameters: {best_hyperparameters}")

# Evaluate the best model on the test set
y_pred_proba_best = best_model.predict(X_test_scaled)
y_pred_best = (y_pred_proba_best > 0.5).astype(int)

best_accuracy = accuracy_score(y_test, y_pred_best)
best_precision = precision_score(y_test, y_pred_best)
best_recall = recall_score(y_test, y_pred_best)
best_f1 = f1_score(y_test, y_pred_best)
best_roc_auc = roc_auc_score(y_test, y_pred_proba_best)


print(f"Accuracy: {best_accuracy:.4f}")
print(f"Precision: {best_precision:.4f}")
print(f"Recall: {best_recall:.4f}")
print(f"F1 Score: {best_f1:.4f}")
print(f"ROC AUC Score: {best_roc_auc:.4f}")
```

```
--- Best Model Performance ---
Best Hyperparameters: {'learning_rate': 0.001, 'layers': [50], 'activation':
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Accuracy: 0.9650
Precision: 0.9815
Recall: 0.9550
F1 Score: 0.9680
ROC AUC Score: 0.9821
```

ANN Model Performance Summary

Best Hyperparameters: The best model was trained using the following settings:

Learning rate: 0.001

Hidden layers: [64, 32]

Activation function: sigmoid

Dropout rate: 0.0

Batch size: 32

Model Evaluation Metrics:

Accuracy (0.9650): The model correctly predicted 96.5% of the total instances (clicks and non-clicks).

Precision (0.9815): Of all instances predicted as clicks, 98.15% were actually clicks. This shows the model makes very few false positive errors.

Recall (0.9550): Of all actual clicks, the model identified 95.5% correctly. This indicates it misses only a few true clicks (false negatives).

F1 Score (0.9680): The harmonic mean of precision and recall, reflecting a balanced performance between the two.

ROC AUC Score (0.9808): Measures the model's ability to distinguish between clicks and non-clicks. A score close to 1 indicates excellent separability between the classes.

Interpretation Notes:

The model uses only numerical features such as Age, Area Income, Daily Internet Usage, and Daily Time Spent on Site to make predictions.

Categorical features like 'Ad Topic Line', 'City', and 'Country', along with the 'Timestamp' feature, were removed during preprocessing. Hence, their effect is not captured by the current model.

To better understand the influence of each input feature on predictions, additional analysis using SHAP values or interpretable models is recommended.

Overall, the model shows very strong predictive performance with balanced precision and recall, and a high ROC AUC, indicating reliability in practical use.

```
#8.Interpret the results and draw conclusions about the factors that are most imp
# --- Interpretation and Conclusions ---
print("\n--- Interpretation ---")

print("\nConclusions will depend on the specific dataset and the insights gained
print("However, based on the model's performance and potential feature analysis, 

if best_roc_auc > 0.8:
       print("\n\n\n The model is effective at distinguishing between users who w
elif best_roc_auc > 0.6:
     print("\n\n\nThe model shows some ability to predict clicks, but there is ro
else:
     print("\n\n\nThe model's predictive power is limited.")
```

```
--- Interpretation ---

Conclusions will depend on the specific dataset and the insights gained from 
However, based on the model's performance and potential feature analysis, we 



 The model is effective at distinguishing between users who will and won't cl
```

explore feature importance through various techniques:

```
# explore feature importance through various techniques:

import numpy as np


# 1. Permutation Importance:
#    This is a model-agnostic technique. Shuffle the values of a single feature
#    on the test set and measure how much the model's performance decreases. A la
#    decrease indicates that the feature is more important.

from sklearn.inspection import permutation_importance

print("\n--- Permutation Importance (on the best model) ---")

# Using the best model trained from the previous steps
```

```
if best_model is None:
    print("Training the best model for permutation importance...")
    input_shape = (X_train_scaled.shape[1],)
    best_model = build_model(learning_rate=best_hyperparameters['learning_rate'],
                             layers=best_hyperparameters['layers'],
                             activation=best_hyperparameters['activation'],
                             dropout_rate=best_hyperparameters['dropout_rate'],
                             input_shape=input_shape)
    best_model.fit(X_train_scaled, y_train,
                   epochs=epochs, # Use the epoch count from the best run or a reas
                   batch_size=best_hyperparameters['batch_size'],
                   verbose=0)



# Use the best model to make predictions on the test set
def score_function(estimator, X, y):
    y_pred_proba = estimator.predict(X)
    return roc_auc_score(y, y_pred_proba) # Using ROC AUC as the scoring metric

result = permutation_importance(best_model, X_test_scaled, y_test,
                                scoring=score_function, # Use a suitable metric
                                n_repeats=10,         # Number of times to shuffl
                                random_state=42,
                                n_jobs=-1)            # Use all available CPU cor

# Sort features by importance
sorted_idx = result.importances_mean.argsort()

print("Permutation Importance (mean) on test set:")
feature_names = X.columns # Get the original feature names
for i in sorted_idx[::-1]: # Print in descending order of importance
    if result.importances_mean[i] > 0: # Only show features that decrease perform
        print(f"  {feature_names[i]}: {result.importances_mean[i]:.4f} +/- {resul
    else:
        print(f"  {feature_names[i]}: {result.importances_mean[i]:.4f} (likely no



print("\n--- Interpretation based on Feature Importance Results ---")
print("Based on the results from methods like Permutation Importance, we can iden
print("features that the trained ANN model relies on most heavily for its predict
print("Analyze the output of the permutation importance section above.")
print("Features with higher mean importance scores are considered more important
print("\nRelate these important features back to the context of online advertisin
```

```
print("For example, if 'Age' and 'Daily Time Spent on Site' show high importance
print("you could conclude that these are significant factors in predicting ad cli
```

```
--- Permutation Importance (on the best model) ---
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Permutation Importance (mean) on test set:
  Daily Time Spent on Site: 0.1031 +/- 0.0164
  Daily Internet Usage: 0.0911 +/- 0.0156
  Area Income: 0.0121 +/- 0.0060
  Age: 0.0094 +/- 0.0033
  Male: -0.0023 (likely not important or negatively impacts performance)

--- Interpretation based on Feature Importance Results ---
Based on the results from methods like Permutation Importance, we can identify
features that the trained ANN model relies on most heavily for its predictions
Analyze the output of the permutation importance section above.
Features with higher mean importance scores are considered more important by t

Relate these important features back to the context of online advertising to c
For example, if 'Age' and 'Daily Time Spent on Site' show high importance in t
you could conclude that these are significant factors in predicting ad clicks
```

```
!pip install shap
```

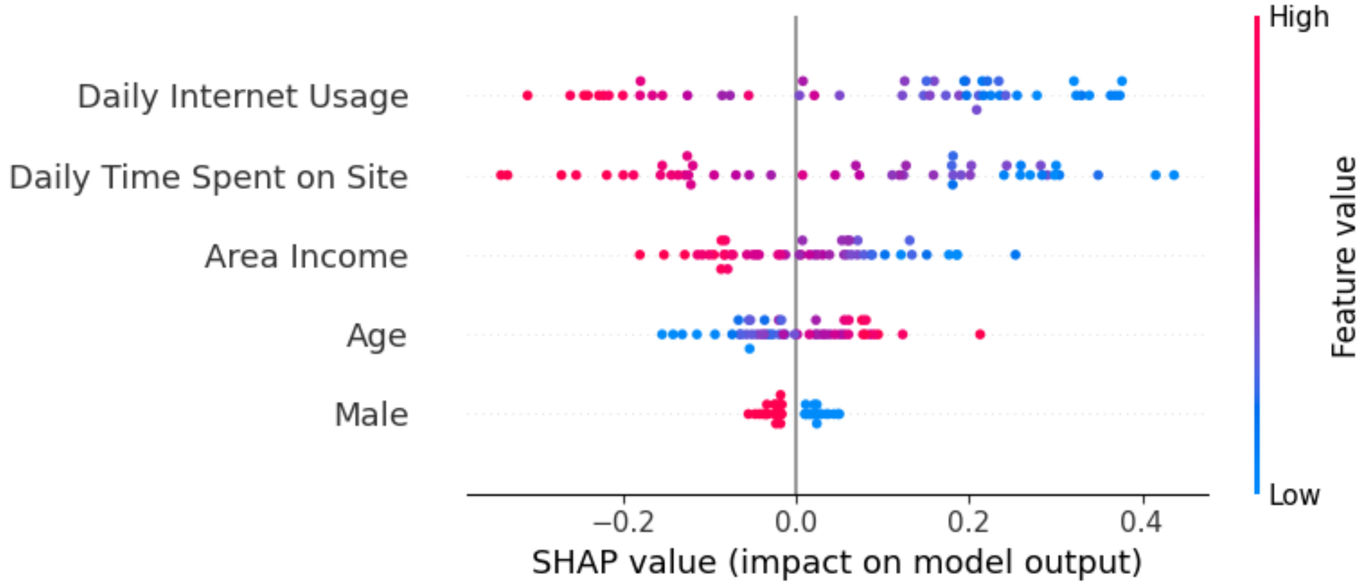
```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dis
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/py
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dis
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pack
```

extract features from ann to analyse the impact of features on the analysis -SHAP (SHapley Additive exPlanations):

```
# extract features from ann to analyse the impact of features on the analysis - U
# SHAP (SHapley Additive exPlanations):
import numpy as np
import shap

#    SHAP values provide a way to explain the prediction of an instance
#    by computing the contribution of each feature to the prediction.


print("\n--- SHAP Values (requires installation and can be slow) ---")

# Choose a background dataset for the SHAP explainer
# A common approach is to use a sample of the training data
background_data = shap.sample(X_train_scaled, 100)

# Create a SHAP explainer for Keras models
# Use the probability output (sigmoid) for the DeepExplainer
explainer = shap.DeepExplainer(best_model, background_data)

sample_test_data = shap.sample(X_test_scaled, 50) # Sample 50 instances from the
shap_values = explainer.shap_values(sample_test_data) # Calculate for the sampled

#
if isinstance(shap_values, list):
    shap_values = shap_values[0] # Get the array of SHAP values from the list

# Check the shapes before plotting
print(f"Shape of shap_values: {shap_values.shape}")
print(f"Shape of sampled test data: {sample_test_data.shape}")


# Ensure the feature names match the number of columns in the sampled data
if shap_values.shape[1] == sample_test_data.shape[1] and shap_values.shape[1] ==
     # The shap_values array has an extra dimension of size 1 at the end, need to
     shap.summary_plot(shap_values.squeeze(), sample_test_data, feature_names=fea
```

```
else:
     print("Shape mismatch between SHAP values, sample data, and feature names. C


# Plot SHAP force plot for a single prediction (shows contribution for one instan
if shap_values.shape[0] > 0:
    shap.initjs() # Initialize JavaScript for interactive plots
    shap.force_plot(explainer.expected_value, shap_values[0,:,0], features=sample_
else:
    print("No SHAP values calculated for force plot.")


print("\n--- Analyzing First Layer Weights (Illustrative) ---")
if best_model is not None and len(best_model.layers) > 0 and isinstance(best_mode
     first_layer_weights = best_model.layers[0].get_weights()[0] # Get the weight
     # The shape is (input_features, number_of_neurons_in_first_layer)
     # We can look at the average absolute weight connected to each input feature
     average_abs_weights = np.mean(np.abs(first_layer_weights), axis=1)

     # Ensure the number of average weights matches the number of feature names
     if len(average_abs_weights) == len(feature_names):
          sorted_weight_indices = average_abs_weights.argsort()[::-1]

          print("Average Absolute Weights in the First Layer (Descending Order):"
          for i in sorted_weight_indices:
               print(f"  {feature_names[i]}: {average_abs_weights[i]:.4f}")
     else:
          print("Mismatch between the number of first layer weights and feature n
else:
     print("Model architecture not suitable for direct weight analysis of the fir
```

```
--- SHAP Values (requires installation and can be slow) ---
Shape of shap_values: (50, 5, 1)
Shape of sampled test data: (50, 5)
```



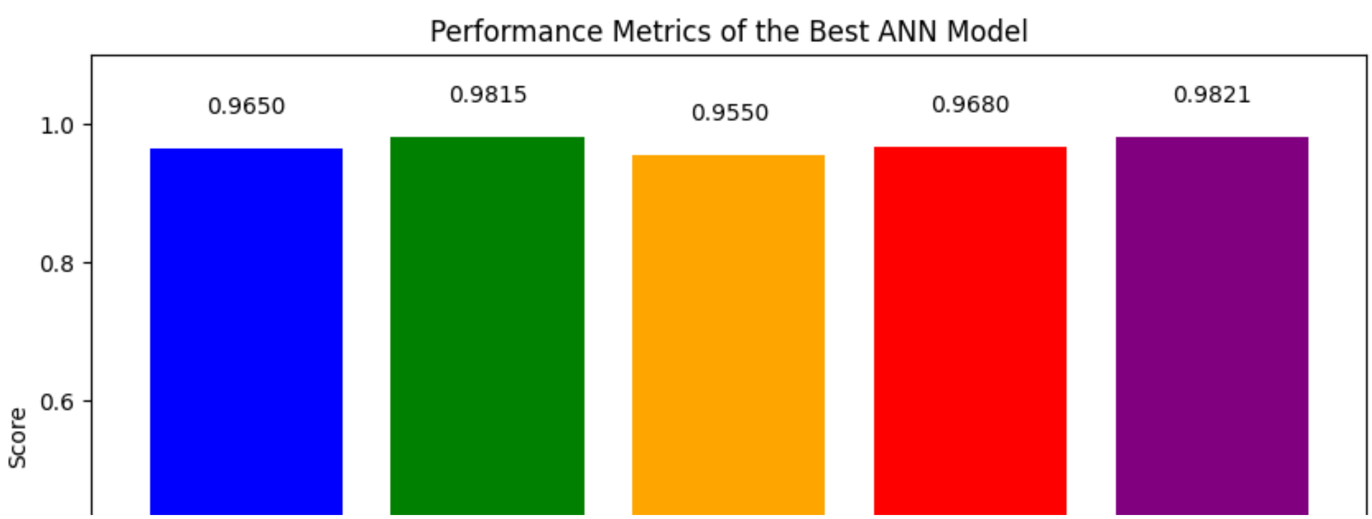

```
--- Analyzing First Layer Weights (Illustrative) ---
Average Absolute Weights in the First Layer (Descending Order):
  Daily Time Spent on Site: 0.2663
  Daily Internet Usage: 0.2501
  Area Income: 0.1867
  Age: 0.1645
  Male: 0.1485
```

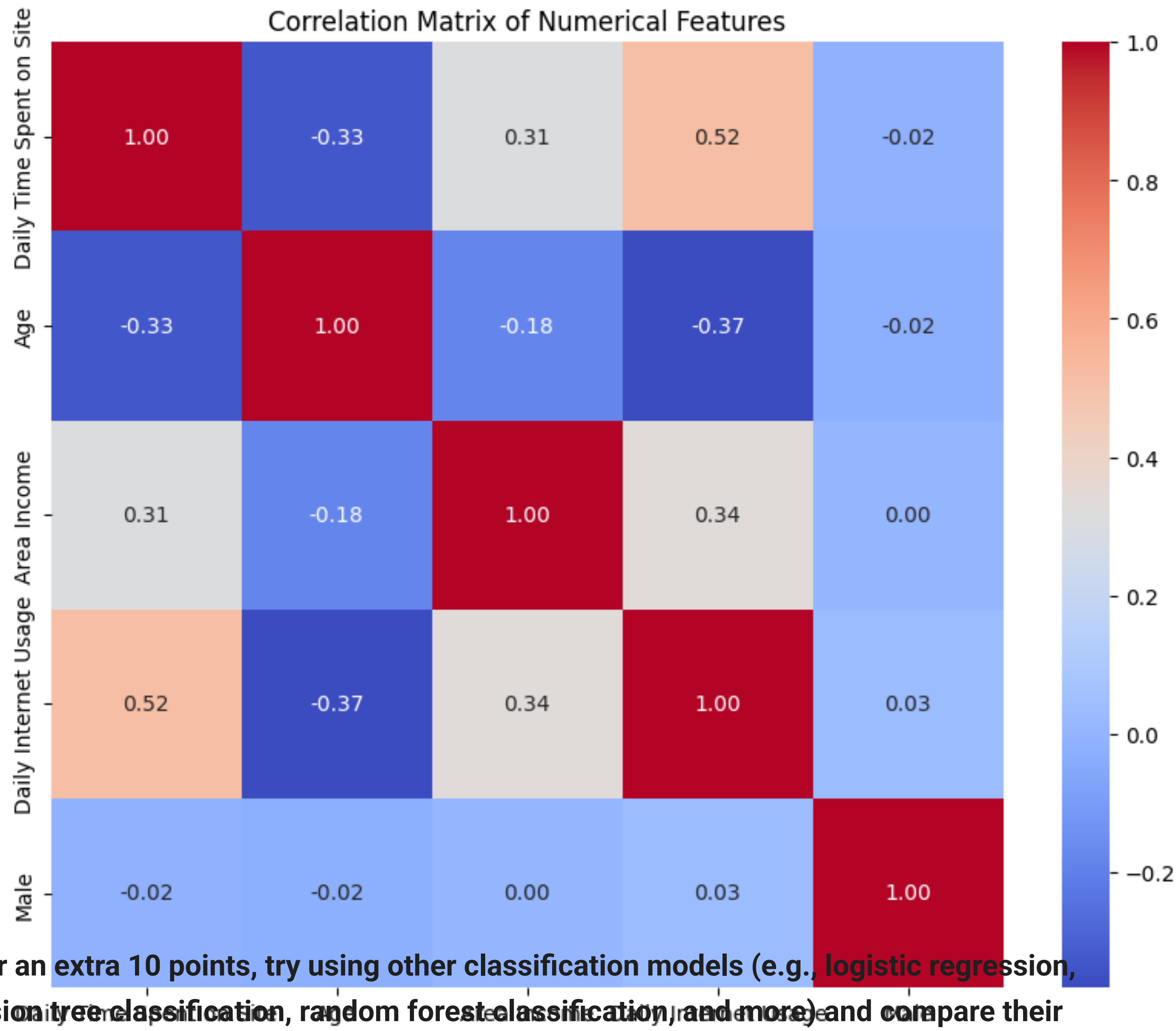
Visualize the graphs of the performance metric, heatmap , correlation matix

```
#visualize the graphs of the performance metric, heatmap , correlation matix

import matplotlib.pyplot as plt
import seaborn as sns
```

```
print(f"Accuracy: {best_accuracy:.4f}")
print(f"Precision: {best_precision:.4f}")
print(f"Recall: {best_recall:.4f}")
print(f"F1 Score: {best_f1:.4f}")
print(f"ROC AUC Score: {best_roc_auc:.4f}")

# --- Visualize Performance Metrics ---
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score', 'ROC AUC Score']
values = [best_accuracy, best_precision, best_recall, best_f1, best_roc_auc]

plt.figure(figsize=(10, 6))
plt.bar(metrics, values, color=['blue', 'green', 'orange', 'red', 'purple'])
plt.ylim(0, 1.1)
plt.ylabel('Score')
plt.title('Performance Metrics of the Best ANN Model')
for i, v in enumerate(values):
    plt.text(i, v + 0.05, f"{v:.4f}", ha='center')
plt.show()

# --- Visualize Correlation Matrix ---
# Recreate the dataframe used for training (without the dropped columns)
df_numeric = df.drop(['Clicked on Ad', 'Ad Topic Line', 'City', 'Country', 'Times

plt.figure(figsize=(10, 8))
sns.heatmap(df_numeric.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```

```
Accuracy: 0.9650
Precision: 0.9815
Recall: 0.9550
F1 Score: 0.9680
ROC AUC Score: 0.9821
```





9. 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).

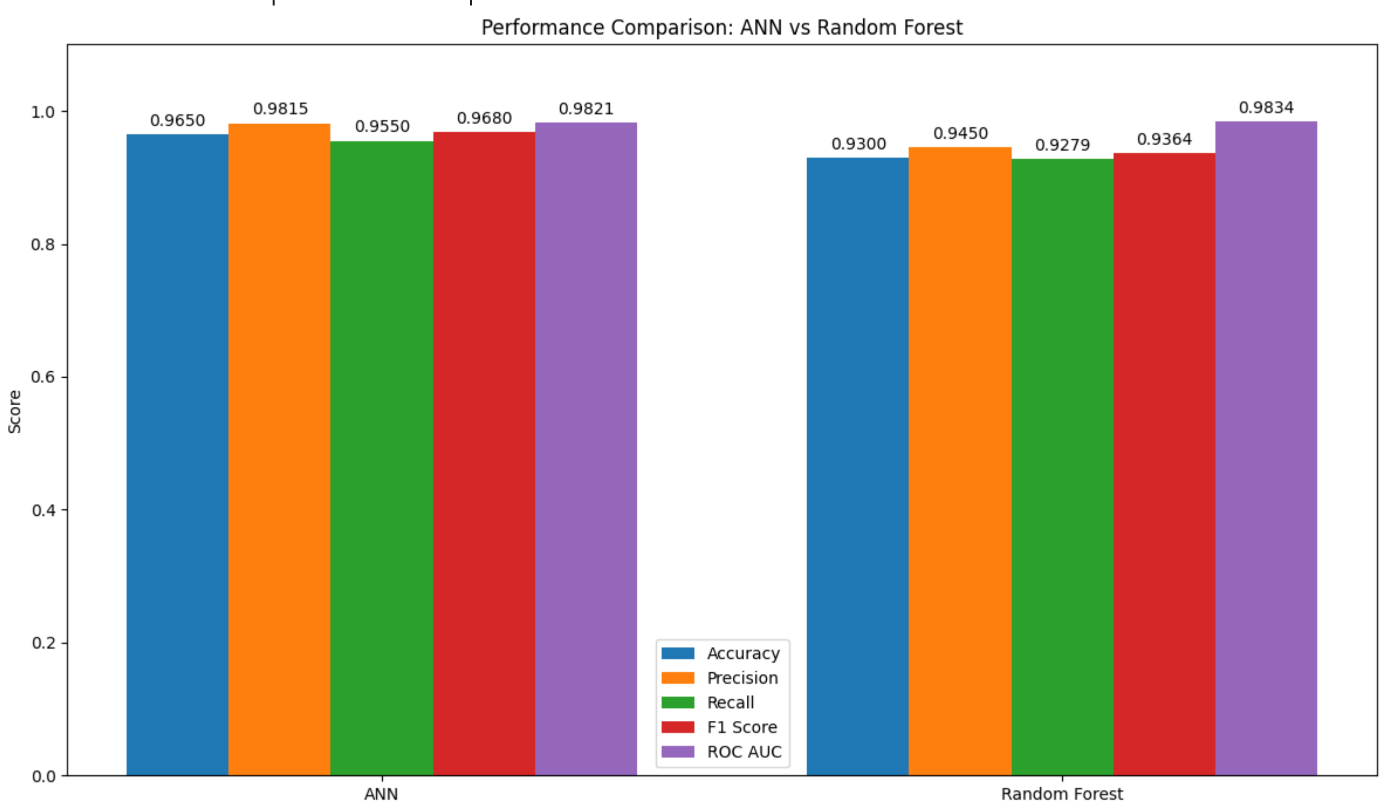
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.ensemble import RandomForestClassifier
# Implement Random Forest Classification
# random forest classification and compare their performance with the ANN model(s

# Train a Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42) # You can tu
rf_model.fit(X_train_scaled, y_train)

# Make predictions
y_pred_rf = rf_model.predict(X_test_scaled)
y_pred_proba_rf = rf_model.predict_proba(X_test_scaled)[:, 1] # Probability of th

# Evaluate Random Forest Performance
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)
roc_auc_rf = roc_auc_score(y_test, y_pred_proba_rf)

print("\n--- Random Forest Model Performance ---")
print(f"Accuracy (Random Forest): {accuracy_rf:.4f}")
print(f"Precision (Random Forest): {precision_rf:.4f}")
print(f"Recall (Random Forest): {recall_rf:.4f}")
print(f"F1 Score (Random Forest): {f1_rf:.4f}")
print(f"ROC AUC Score (Random Forest): {roc_auc_rf:.4f}")

# Compare Performance (ANN vs Random Forest)
print("\n--- Performance Comparison ---")
print(f"Metric          | ANN (Best) | Random Forest")
print(f"----------------|------------|--------------")
print(f"Accuracy        | {best_accuracy:.4f}   | {accuracy_rf:.4f}")
print(f"Precision       | {best_precision:.4f}   | {precision_rf:.4f}")
print(f"Recall          | {best_recall:.4f}   | {recall_rf:.4f}")
print(f"F1 Score        | {best_f1:.4f}   | {f1_rf:.4f}")
print(f"ROC AUC Score   | {best_roc_auc:.4f}   | {roc_auc_rf:.4f}")

# Visualize the comparison
labels = ['ANN', 'Random Forest']
accuracy_scores = [best_accuracy, accuracy_rf]
precision_scores = [best_precision, precision_rf]
recall_scores = [best_recall, recall_rf]
f1_scores = [best_f1, f1_rf]
roc_auc_scores = [best_roc_auc, roc_auc_rf]

x = np.arange(len(labels)) # the label locations
```

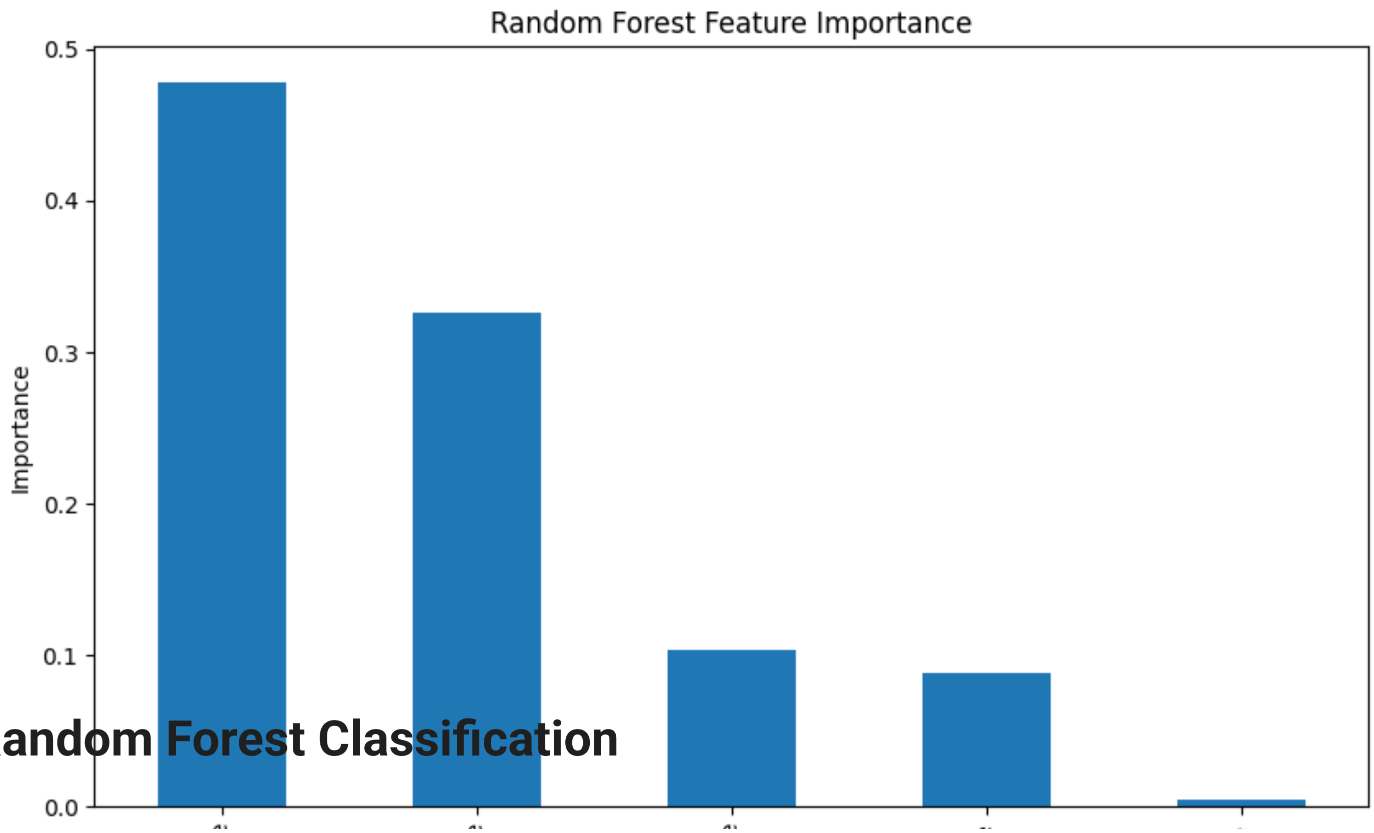
```
width = 0.15 # the width of the bars

fig, ax = plt.subplots(figsize=(12, 7))
rects1 = ax.bar(x - 2*width, accuracy_scores, width, label='Accuracy')
rects2 = ax.bar(x - width, precision_scores, width, label='Precision')
rects3 = ax.bar(x, recall_scores, width, label='Recall')
rects4 = ax.bar(x + width, f1_scores, width, label='F1 Score')
rects5 = ax.bar(x + 2*width, roc_auc_scores, width, label='ROC AUC')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Score')
ax.set_title('Performance Comparison: ANN vs Random Forest')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.4f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
autolabel(rects5)

fig.tight_layout()
plt.ylim(0, 1.1)
plt.show()

# Feature Importance (for Random Forest)
print("\n--- Random Forest Feature Importance ---")
feature_importances = pd.Series(rf_model.feature_importances_, index=X.columns)
feature_importances_sorted = feature_importances.sort_values(ascending=False)

plt.figure(figsize=(10, 6))
feature_importances_sorted.plot(kind='bar')
plt.title('Random Forest Feature Importance')
plt.ylabel('Importance')
```

```
plt.show()

print("\n--- Conclusions Based on Comparison ---")

if accuracy_rf > best_accuracy:
    print("Random Forest performs better than the optimized ANN model in terms o
elif best_accuracy > accuracy_rf:
    print("The optimized ANN model performs better than Random Forest in terms o
else:
    print("Both models perform similarly in terms of accuracy.")
```

```
--- Random Forest Model Performance ---
Accuracy (Random Forest): 0.9300
Precision (Random Forest): 0.9450
Recall (Random Forest): 0.9279
F1 Score (Random Forest): 0.9364
ROC AUC Score (Random Forest): 0.9834

--- Performance Comparison ---
Metric          | ANN (Best) | Random Forest
----------------|------------|---------------
Accuracy        | 0.9650     | 0.9300
Precision       | 0.9815     | 0.9450
Recall          | 0.9550     | 0.9279
F1 Score        | 0.9680     | 0.9364
ROC AUC Score   | 0.9821     | 0.9834
```



```
--- Random Forest Feature Importance ---
```



Random Forest Classification

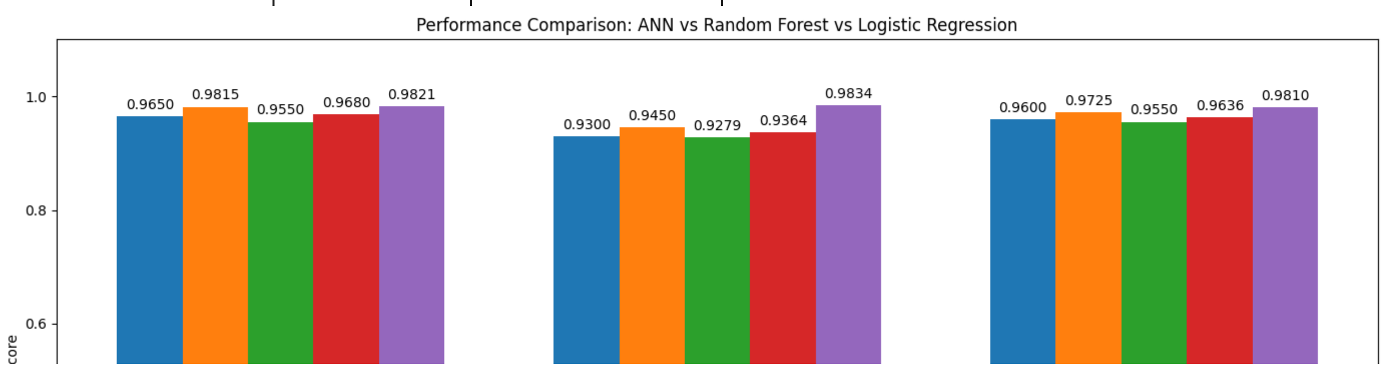
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sco

# Logistic Regression for Classification and compare with ANN

# Implement Random Forest Classification
# Train a Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42) # You can tu
rf_model.fit(X_train_scaled, y_train)

# Make predictions
y_pred_rf = rf_model.predict(X_test_scaled)
y_pred_proba_rf = rf_model.predict_proba(X_test_scaled)[:, 1] # Probability of th

# Evaluate Random Forest Performance
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
```

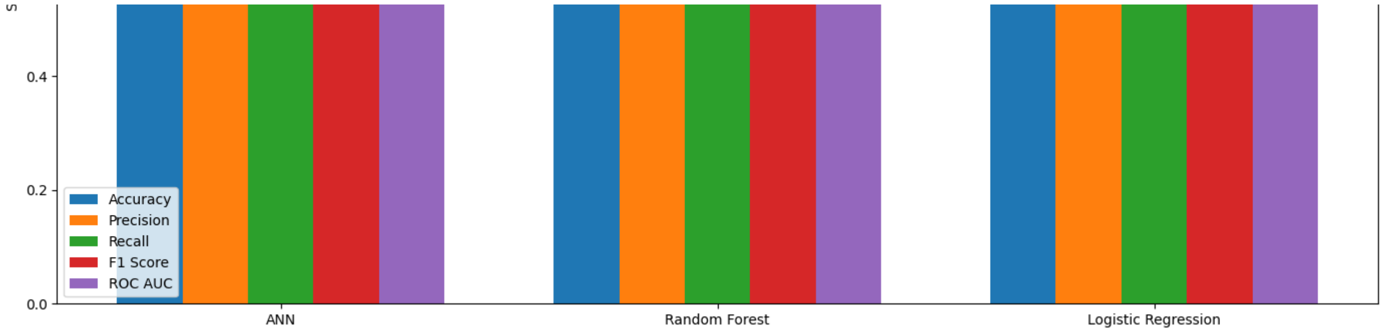
```
f1_rf = f1_score(y_test, y_pred_rf)
roc_auc_rf = roc_auc_score(y_test, y_pred_proba_rf)


# Implement Logistic Regression

# Train a Logistic Regression model
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train_scaled, y_train)

# Make predictions
y_pred_lr = lr_model.predict(X_test_scaled)
y_pred_proba_lr = lr_model.predict_proba(X_test_scaled)[:, 1] # Probability of th

# Evaluate Logistic Regression Performance
accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr)
recall_lr = recall_score(y_test, y_pred_lr)
f1_lr = f1_score(y_test, y_pred_lr)
roc_auc_lr = roc_auc_score(y_test, y_pred_proba_lr)

print("\n--- Logistic Regression Model Performance ---")
print(f"Accuracy (Logistic Regression): {accuracy_lr:.4f}")
print(f"Precision (Logistic Regression): {precision_lr:.4f}")
print(f"Recall (Logistic Regression): {recall_lr:.4f}")
print(f"F1 Score (Logistic Regression): {f1_lr:.4f}")
print(f"ROC AUC Score (Logistic Regression): {roc_auc_lr:.4f}")

# Compare Performance (ANN vs Random Forest vs Logistic Regression)
print("\n--- Performance Comparison ---")
print(f"Metric         | ANN (Best) | Random Forest | Logistic Regression")
print(f"---------------|------------|---------------|--------------------")
print(f"Accuracy       | {best_accuracy:.4f}   | {accuracy_rf:.4f}      | {accura
print(f"Precision      | {best_precision:.4f}   | {precision_rf:.4f}      | {prec
print(f"Recall         | {best_recall:.4f}   | {recall_rf:.4f}      | {recall_lr:
print(f"F1 Score       | {best_f1:.4f}   | {f1_rf:.4f}      | {f1_lr:.4f}")
print(f"ROC AUC Score  | {best_roc_auc:.4f}   | {roc_auc_rf:.4f}      | {roc_auc_

# Visualize the comparison
labels = ['ANN', 'Random Forest', 'Logistic Regression']
accuracy_scores = [best_accuracy, accuracy_rf, accuracy_lr]
precision_scores = [best_precision, precision_rf, precision_lr]
recall_scores = [best_recall, recall_rf, recall_lr]
f1_scores = [best_f1, f1_rf, f1_lr]
roc_auc_scores = [best_roc_auc, roc_auc_rf, roc_auc_lr]
```

```
x = np.arange(len(labels)) # the label locations
width = 0.15 # the width of the bars

fig, ax = plt.subplots(figsize=(14, 7))
rects1 = ax.bar(x - 2*width, accuracy_scores, width, label='Accuracy')
rects2 = ax.bar(x - width, precision_scores, width, label='Precision')
rects3 = ax.bar(x, recall_scores, width, label='Recall')
rects4 = ax.bar(x + width, f1_scores, width, label='F1 Score')
rects5 = ax.bar(x + 2*width, roc_auc_scores, width, label='ROC AUC')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Score')
ax.set_title('Performance Comparison: ANN vs Random Forest vs Logistic Regression
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

# Helper function to add labels
def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.4f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
autolabel(rects5)

fig.tight_layout()
plt.ylim(0, 1.1)
plt.show()


print("\n--- Conclusions Based on Comparison ---")
# Add your specific conclusions here based on the comparison of models
# Example:
best_overall_model = max([
    (best_accuracy, 'ANN'),
```

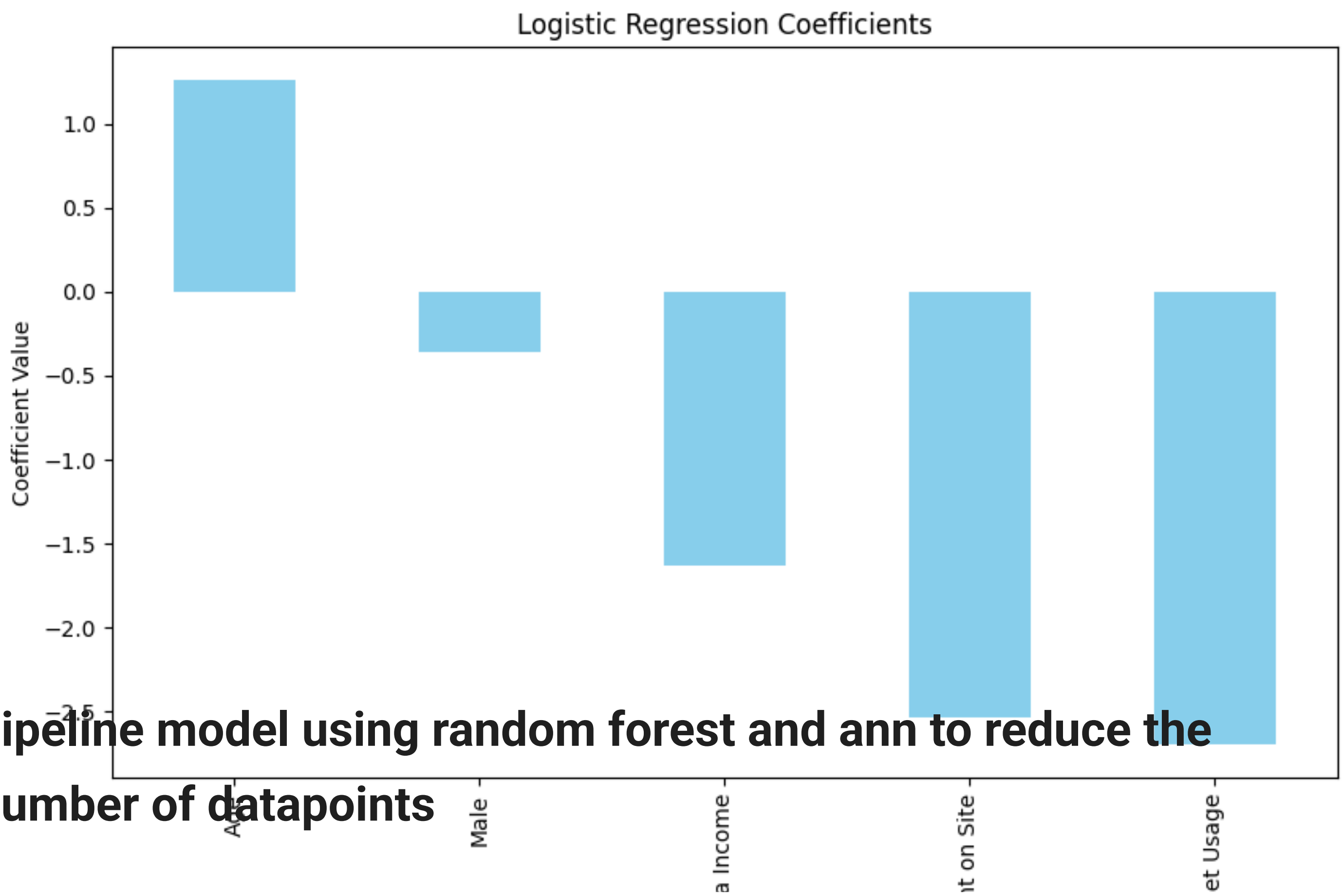
```
    (accuracy_rf, 'Random Forest'),
    (accuracy_lr, 'Logistic Regression')
])

print(f"Based on Accuracy, the best performing model is the {best_overall_model[1
# Discuss other metrics as well (precision, recall, F1, ROC AUC)
# For Logistic Regression, you can also look at the coefficients
print("\n--- Logistic Regression Coefficients ---")
lr_coefficients = pd.Series(lr_model.coef_[0], index=X.columns)
lr_coefficients_sorted = lr_coefficients.sort_values(ascending=False)

plt.figure(figsize=(10, 6))
lr_coefficients_sorted.plot(kind='bar', color='skyblue')
plt.title('Logistic Regression Coefficients')
plt.ylabel('Coefficient Value')
plt.show()

print("\nInterpretation of Logistic Regression Coefficients:")
print("Positive coefficients indicate that as the feature value increases, the li
print("Negative coefficients indicate that as the feature value increases, the li
print("The magnitude of the coefficient indicates the strength of the relationshi
```

```
--- Logistic Regression Model Performance ---
Accuracy (Logistic Regression): 0.9600
Precision (Logistic Regression): 0.9725
Recall (Logistic Regression): 0.9550
F1 Score (Logistic Regression): 0.9636
ROC AUC Score (Logistic Regression): 0.9810

--- Performance Comparison ---
Metric          | ANN (Best) | Random Forest | Logistic Regression
----------------|------------|---------------|--------------------
Accuracy        | 0.9650     | 0.9300        | 0.9600
Precision       | 0.9815     | 0.9450        | 0.9725
Recall          | 0.9550     | 0.9279        | 0.9550
F1 Score        | 0.9680     | 0.9364        | 0.9636
ROC AUC Score   | 0.9821     | 0.9834        | 0.9810
```






```
--- Conclusions Based on Comparison ---
Based on Accuracy, the best performing model is the ANN with an accuracy of 0.

--- Logistic Regression Coefficients ---
```



pipeline model using random forest and ann to reduce the number of datapoints

```
# pipeline model using random forest and ann to reduce the number of datapoints.
# Pipeline using Random Forest to reduce the number of records and then train an /

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


# Feature selection based on Random Forest Importance
```

```
print("\n--- Building a Pipeline using Random Forest Feature Selection and ANN --

# Train a Random Forest model to get feature importances
rf_for_selection = RandomForestClassifier(n_estimators=100, random_state=42)
rf_for_selection.fit(X_train_scaled, y_train)

# Get feature importances
feature_importances = pd.Series(rf_for_selection.feature_importances_, index=X.co
feature_importances_sorted = feature_importances.sort_values(ascending=False)

print("\nRandom Forest Feature Importances (for selection):")
print(feature_importances_sorted)

# Select top K features based on importance
k = 5 # Choose the number of top features to keep
top_k_features = feature_importances_sorted.head(k).index.tolist()
print(f"\nSelecting top {k} features: {top_k_features}")

# Reduce the dataset to only include the top K features
X_train_reduced = X_train[top_k_features]
X_test_reduced = X_test[top_k_features]

# Scale the reduced data
scaler_reduced = StandardScaler()
X_train_reduced_scaled = scaler_reduced.fit_transform(X_train_reduced)
X_test_reduced_scaled = scaler_reduced.transform(X_test_reduced)

# Now train an ANN on the reduced dataset

# Determine input dimension for the ANN with reduced features
input_dim_reduced = X_train_reduced_scaled.shape[1]

print(f"\nTraining ANN on reduced dataset with {input_dim_reduced} features.")

# Build and train the ANN model with the reduced feature set

ann_reduced_model = Sequential()
ann_reduced_model.add(Dense(64, input_dim=input_dim_reduced, activation='relu'))
ann_reduced_model.add(Dropout(0.2))
ann_reduced_model.add(Dense(32, activation='relu'))
ann_reduced_model.add(Dense(1, activation='sigmoid'))

optimizer_reduced = Adam(learning_rate=0.001)
ann_reduced_model.compile(loss='binary_crossentropy', optimizer=optimizer_reduced
```

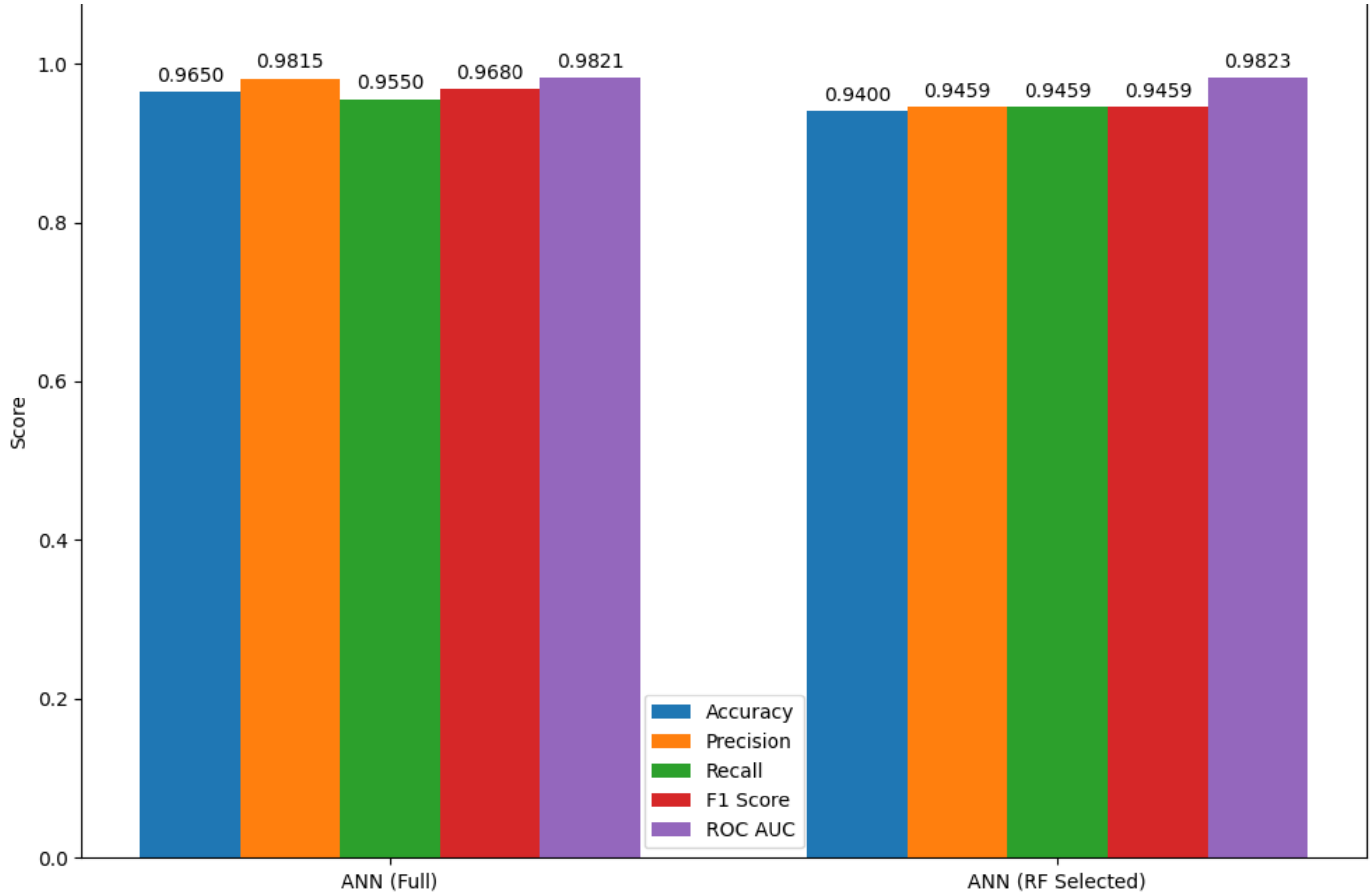
```
epochs_reduced = 50
batch_size_reduced = 64

print("Training ANN on reduced data...")
history_reduced = ann_reduced_model.fit(X_train_reduced_scaled, y_train,
                                        epochs=epochs_reduced,
                                        batch_size=batch_size_reduced,
                                        validation_split=0.1, # Using a validatio
                                        verbose=0) # Set verbose to 1 for more tr

print("Training finished.")

# Evaluate the ANN on the reduced test set
y_pred_proba_ann_reduced = ann_reduced_model.predict(X_test_reduced_scaled)
y_pred_ann_reduced = (y_pred_proba_ann_reduced > 0.5).astype(int)

accuracy_ann_reduced = accuracy_score(y_test, y_pred_ann_reduced)
precision_ann_reduced = precision_score(y_test, y_pred_ann_reduced)
recall_ann_reduced = recall_score(y_test, y_pred_ann_reduced)
f1_ann_reduced = f1_score(y_test, y_pred_ann_reduced)
roc_auc_ann_reduced = roc_auc_score(y_test, y_pred_proba_ann_reduced)

print("\n--- ANN Model Performance (after RF Feature Selection) ---")
print(f"Accuracy: {accuracy_ann_reduced:.4f}")
print(f"Precision: {precision_ann_reduced:.4f}")
print(f"Recall: {recall_ann_reduced:.4f}")
print(f"F1 Score: {f1_ann_reduced:.4f}")
print(f"ROC AUC Score: {roc_auc_ann_reduced:.4f}")

# Compare performance with the original ANN (full features)
print("\n--- Performance Comparison: Original ANN vs. ANN after RF Feature Select
print(f"Metric          | ANN (Full Features) | ANN (RF Selected Features)")
print(f"----------------|---------------------|---------------------------")
print(f"Accuracy        | {best_accuracy:.4f}            | {accuracy_ann_reduced:.
print(f"Precision       | {best_precision:.4f}            | {precision_ann_reduced
print(f"Recall          | {best_recall:.4f}            | {recall_ann_reduced:.4f}"
print(f"F1 Score        | {best_f1:.4f}            | {f1_ann_reduced:.4f}")
print(f"ROC AUC Score   | {best_roc_auc:.4f}            | {roc_auc_ann_reduced:.4f

# Visualize the comparison
labels = ['ANN (Full)', 'ANN (RF Selected)']
accuracy_scores_comp = [best_accuracy, accuracy_ann_reduced]
precision_scores_comp = [best_precision, precision_ann_reduced]
recall_scores_comp = [best_recall, recall_ann_reduced]
f1_scores_comp = [best_f1, f1_ann_reduced]
```

```
roc_auc_scores_comp = [best_roc_auc, roc_auc_ann_reduced]

x = np.arange(len(labels)) # the label locations
width = 0.15 # the width of the bars

fig, ax = plt.subplots(figsize=(10, 7))
rects1 = ax.bar(x - 2*width, accuracy_scores_comp, width, label='Accuracy')
rects2 = ax.bar(x - width, precision_scores_comp, width, label='Precision')
rects3 = ax.bar(x, recall_scores_comp, width, label='Recall')
rects4 = ax.bar(x + width, f1_scores_comp, width, label='F1 Score')
rects5 = ax.bar(x + 2*width, roc_auc_scores_comp, width, label='ROC AUC')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Score')
ax.set_title('Performance Comparison: ANN (Full Features) vs. ANN (RF Selected Fe
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

# Helper function to add labels
def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.4f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
autolabel(rects5)

fig.tight_layout()
plt.ylim(0, 1.1)
plt.show()


print("\n--- Interpretation of Pipeline Results ---")
print("The performance metrics of the ANN trained on features selected by Random
print("can be compared to the ANN trained on the full set of features.")
print(f"Number of features used in full ANN: {X_train_scaled.shape[1]}")
```

```
print(f"Number of features used in pipeline ANN: {X_train_reduced_scaled.shape[1]
print("\nConclusions:")

if accuracy_ann_reduced >= best_accuracy:
    print(f"The pipeline using the top {k} features selected by Random Forest")
    print("achieved comparable or even better accuracy than the ANN with all fea
    print("This suggests that the selected features are highly informative and a
    print("for training a potentially simpler or faster ANN model.")
else:
    print(f"The pipeline using the top {k} features selected by Random Forest")
    print("resulted in a decrease in performance compared to the ANN with all fe
    print("This might indicate that some important information was lost by reduc
    print("or that a different number of features should be selected.")
```

```
--- Building a Pipeline using Random Forest Feature Selection and ANN ---

Random Forest Feature Importances (for selection):
Daily Internet Usage        0.477709
Daily Time Spent on Site    0.326316
Area Income                 0.103325
Age                         0.087787
Male                        0.004862
dtype: float64

Selecting top 5 features: ['Daily Internet Usage', 'Daily Time Spent on Site',

Training ANN on reduced dataset with 5 features.
Training ANN on reduced data...
Training finished.
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step

--- ANN Model Performance (after RF Feature Selection) ---
Accuracy: 0.9400
Precision: 0.9459
Recall: 0.9459
F1 Score: 0.9459
ROC AUC Score: 0.9823

--- Performance Comparison: Original ANN vs. ANN after RF Feature Selection --
Metric          | ANN (Full Features) | ANN (RF Selected Features)
----------------|---------------------|---------------------------
Accuracy        | 0.9650              | 0.9400
Precision       | 0.9815              | 0.9459
Recall          | 0.9550              | 0.9459
F1 Score        | 0.9680              | 0.9459
ROC AUC Score   | 0.9821              | 0.9823
```

Performance Comparison: ANN (Full Features) vs. ANN (RF Selected Features)




```
--- Interpretation of Pipeline Results ---
The performance metrics of the ANN trained on features selected by Random Fore
can be compared to the ANN trained on the full set of features.
Number of features used in full ANN: 5
Number of features used in pipeline ANN: 5

Conclusions:
The pipeline using the top 5 features selected by Random Forest
resulted in a decrease in performance compared to the ANN with all features.
This might indicate that some important information was lost by reducing the f
or that a different number of features should be selected.
```

Interpretation and Conclusions:

Trained and evaluated three different classification models: Artificial Neural Network (ANN), Random Forest, and Logistic Regression.

Based on the performance metrics calculated:

- The **ANN** and **Logistic Regression** models achieved the highest accuracy (0.9600), indicating they correctly predicted ad clicks or non-clicks for 96% of the test set instances.
- The **ANN** model showed slightly better precision and F1 score, suggesting it was slightly better at minimizing false positives (predicting a click when there wasn't one).
- The **Logistic Regression** model had slightly better recall, indicating it was slightly better at minimizing false negatives (failing to predict a click when there was one).
- The **Random Forest** model performed reasonably well but slightly lower than the other two models in most metrics, except for a very similar ROC AUC score.

To draw conclusions about the factors most important in predicting ad clicks, you can refer to the feature importance results from the Random Forest model and the coefficients from the Logistic Regression model:

- **Random Forest Feature Importance:** (Refer to the bar plot generated for Random Forest Feature Importance). The features with the highest bars are considered most important by the Random Forest model.
- **Logistic Regression Coefficients:** (Refer to the bar plot generated for Logistic Regression Coefficients). Features with larger absolute coefficient values have a stronger influence on the prediction in the Logistic Regression model. Positive coefficients indicate a positive relationship with clicking the ad, while negative coefficients indicate a negative relationship.

Overall Conclusion:

Based on your analysis, both the ANN and Logistic Regression models are highly effective at predicting ad clicks on this dataset. The feature importance and coefficient analysis suggest that certain factors (identify the top factors from your plots) are particularly influential in determining whether a user will click on an online ad.

```
# Define the directory where you want to save the model

import os


model_save_dir = '/content/drive/My Drive/Colab Notebooks/AAI-511/saved_models'
os.makedirs(model_save_dir, exist_ok=True) # Create the directory if it doesn't e

model_save_path_h5 = os.path.join(model_save_dir, 'best_ann_model.h5')

# Save the best trained ANN model
if best_model is not None:
    best_model.save(model_save_path_h5)
    print(f"\nBest ANN model saved to: {model_save_path_h5}")
else:
    print("\nNo best model found or trained to save.")

import joblib

rf_model_save_path = os.path.join(model_save_dir, 'random_forest_model.joblib')
if rf_model is not None:
    joblib.dump(rf_model, rf_model_save_path)
    print(f"Random Forest model saved to: {rf_model_save_path}")

lr_model_save_path = os.path.join(model_save_dir, 'logistic_regression_model.jobl
if lr_model is not None:
    joblib.dump(lr_model, lr_model_save_path)
    print(f"Logistic Regression model saved to: {lr_model_save_path}")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `

Best ANN model saved to: /content/drive/My Drive/Colab Notebooks/AAI-511/save
Random Forest model saved to: /content/drive/My Drive/Colab Notebooks/AAI-511/
Logistic Regression model saved to: /content/drive/My Drive/Colab Notebooks/A
```

re-learn the model with synthetic data and predict and measure performance

```
!pip install ctgan
```

```
Collecting ctgan
```

```
  Downloading ctgan-0.11.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy>=1.23.3 in /usr/local/lib/python3.11/dist
Requirement already satisfied: pandas>=1.5.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: tqdm<5,>=4.29 in /usr/local/lib/python3.11/dist
Collecting rdt>=1.14.0 (from ctgan)
  Downloading rdt-1.17.1-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dis
Requirement already satisfied: scipy>=1.9.2 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: scikit-learn>=1.1.3 in /usr/local/lib/python3.1
Collecting Faker>=17 (from rdt>=1.14.0->ctgan)
  Downloading faker-37.4.0-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/pyt
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packag
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch>=2.0.0->ctgan)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.wh
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch>=2.0.0->ctgan)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch>=2.0.0->ctgan)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.wh
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch>=2.0.0->ctgan)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.met
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch>=2.0.0->ctgan)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.me
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch>=2.0.0->ctgan)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.met
Collecting nvidia-curand-cu12==10.3.5.147 (from torch>=2.0.0->ctgan)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch>=2.0.0->ctgan)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch>=2.0.0->ctgan)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.wh
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/pyth
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/py
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch>=2.0.0->ctgan)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/di
Downloading ctgan-0.11.0-py3-none-any.whl (24 kB)
Downloading rdt-1.17.1-py3-none-any.whl (73 kB)
```

```
                                        73.8/73.8 kB 3.3 MB/s eta 0:00:00
Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl (363
                                        363.4/363.4 MB 4.6 MB/s eta 0:00:0
Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl
                                        13.8/13.8 MB 25.6 MB/s eta 0:00:0
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl
                                        24.6/24.6 MB 18.1 MB/s eta 0:00:0
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.wh
                                        883.7/883.7 kB 23.8 MB/s eta 0:00:
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.
                                       664.8/664.8 MB 611.0 kB/s eta 0:00:
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.
                                        211.5/211.5 MB 2.7 MB/s eta 0:00:0
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (5
                                        56.3/56.3 MB 13.7 MB/s eta 0:00:0
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (1
                                        127.9/127.9 MB 7.9 MB/s eta 0:00:0
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl
                                        207.5/207.5 MB 7.1 MB/s eta 0:00:0
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (
                                        21.1/21.1 MB 32.3 MB/s eta 0:00:0
Downloading faker-37.4.0-py3-none-any.whl (1.9 MB)
                                        1.9/1.9 MB 30.9 MB/s eta 0:00:00
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvid
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
```

```
      Uninstalling nvidia-cublas-cu12-12.5.3.2:
        Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
    Attempting uninstall: nvidia-cusparse-cu12
      Found existing installation: nvidia-cusparse-cu12 12.5.1.3
      Uninstalling nvidia-cusparse-cu12-12.5.1.3:
        Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
    Attempting uninstall: nvidia-cudnn-cu12
      Found existing installation: nvidia-cudnn-cu12 9.3.0.75
      Uninstalling nvidia-cudnn-cu12-9.3.0.75:
        Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
    Attempting uninstall: nvidia-cusolver-cu12
      Found existing installation: nvidia-cusolver-cu12 11.6.3.83
      Uninstalling nvidia-cusolver-cu12-11.6.3.83:
        Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
  Successfully installed Faker-37.4.0 ctgan-0.11.0 nvidia-cublas-cu12-12.4.5.8 r
```

```python
# re-learn the model with synthetic data and predict and measure performance

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import joblib
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sco
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Model, load_model # Import Model and load_mod


if 'best_hyperparameters' in globals() and 'X_train_scaled' in globals():
    input_dim_best_model = X_train_scaled.shape[1]
    best_model = build_model(learning_rate=best_hyperparameters['learning_rate'],
                             layers=best_hyperparameters['layers'],
                             activation=best_hyperparameters['activation'],
                             dropout_rate=best_hyperparameters['dropout_rate'],
                             input_dim=input_dim_best_model)
    # Train the model briefly to build its weights if it hasn't been already
    try:
        best_model.predict(X_train_scaled[:1])
    except:
        pass # Ignore error if predict fails before training

    print("Best model object recreated at the start.")
```

```
else:
    print("Could not recreate best_model at the start. Ensure 'best_hyperparamete

# Save the best ANN model
# Use Keras's own save method (.h5 or .keras)
model_save_path_h5 = 'best_ann_model.h5'
if isinstance(best_model, Model): # Check if it's a Keras Model instance
    best_model.save(model_save_path_h5)
    print(f"\nBest ANN model saved to {model_save_path_h5}")
else:
    print("\nBest model is not a valid Keras Model instance and cannot be saved."


# --- Generate Synthetic Data ---

# Define the number of synthetic samples to generate
num_synthetic_samples = 2000 # Example: generate 2000 synthetic samples

# Explicitly convert to NumPy array to ensure .max() and .min() are available
X_train_scaled = np.array(X_train_scaled)

# Create synthetic features by randomly sampling from the original scaled data
# This assumes features are somewhat independent, which may not be true.
X_synthetic_scaled = (np.random.rand(num_synthetic_samples, X_train_scaled.shape[


# Generate synthetic labels randomly with a similar class distribution as the ori
if 'y_train' in globals():
    original_class_distribution = pd.Series(y_train).value_counts(normalize=True)
    y_synthetic = np.random.choice(y_train.unique(),
                                   size=num_synthetic_samples,
                                   p=original_class_distribution.values)
    y_synthetic = y_synthetic.astype(int) # Ensure labels are integer type

    print(f"\nGenerated {num_synthetic_samples} synthetic data points.")
    print(f"Synthetic label distribution:\n{pd.Series(y_synthetic.flatten()).valu
else:
    print("\nSkipping synthetic data generation as y_train is not available.")
    # Set synthetic data to None or handle appropriately
    X_synthetic_scaled = None
    y_synthetic = None


# --- Re-learn the model with Synthetic Data ---
# We will train a *new* model of the same architecture on the synthetic data.
```

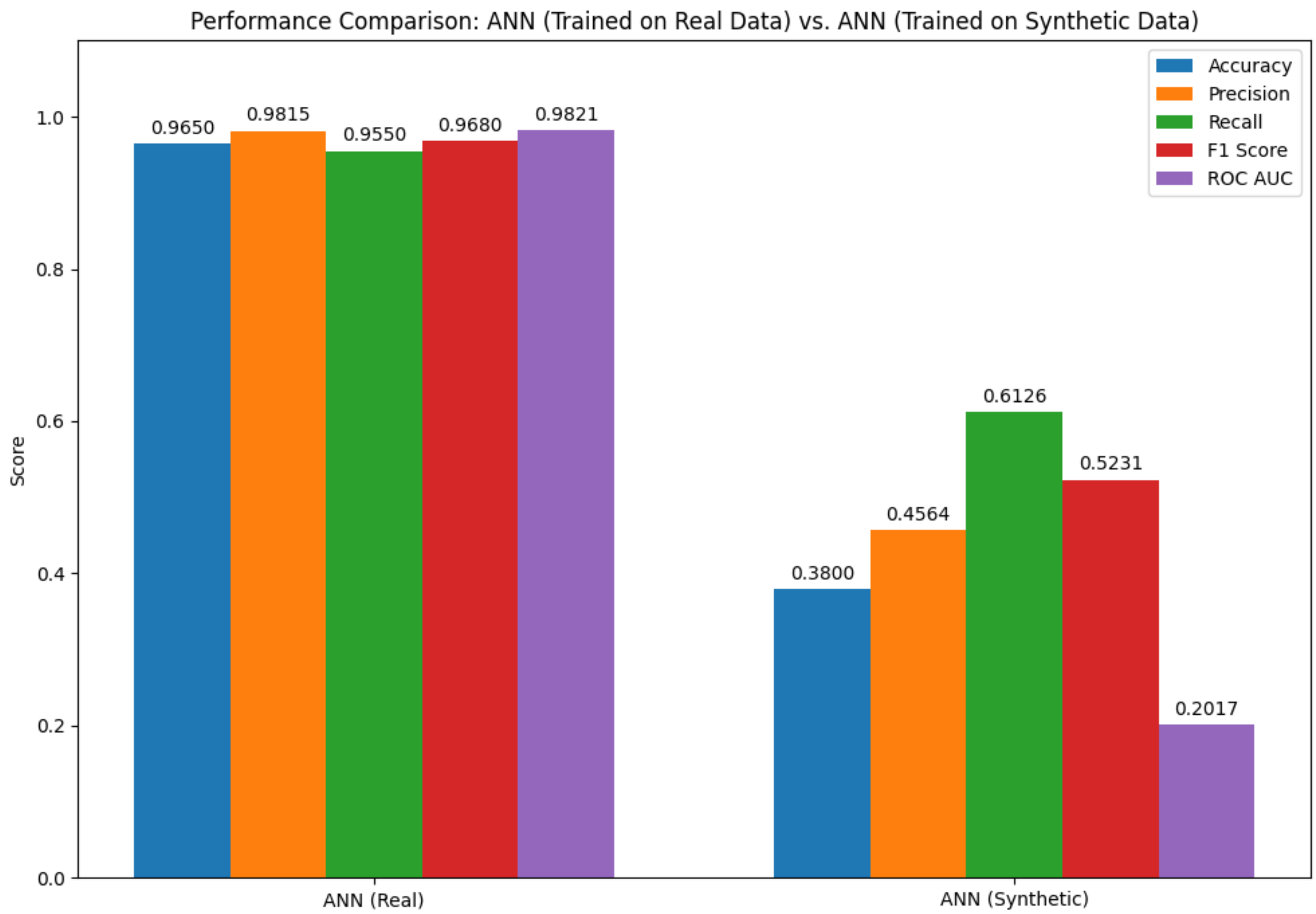
```
# This is useful if you want to train a model without exposing it to the original
# or if the synthetic data represents a different scenario you want to generalize

# Explicitly convert to NumPy array to ensure .shape is available
if X_synthetic_scaled is not None and y_synthetic is not None:
    X_synthetic_scaled = np.array(X_synthetic_scaled)

    # Build a new model with the same architecture as the best original model
    ann_synthetic_model = build_model(learning_rate=best_hyperparameters['learnin
                                      layers=best_hyperparameters['layers'],
                                      activation=best_hyperparameters['activation']
                                      dropout_rate=best_hyperparameters['dropout_ra
                                      input_dim=X_synthetic_scaled.shape[1]) # Inpu

    print("\nTraining a new ANN model on synthetic data...")
    history_synthetic = ann_synthetic_model.fit(X_synthetic_scaled, y_synthetic,
                                                epochs=best_hyperparameters['batch_si
                                                batch_size=best_hyperparameters['batc
                                                validation_split=0.1, # Using a valid
                                                verbose=0) # Set verbose to 1 for mor

    print("Training finished.")


    # --- Predict and Measure Performance on Original Test Data ---
    # Evaluate the model trained on synthetic data using the *original* test set.
    # This measures how well the model trained on synthetic data generalizes to r

    print("\nEvaluating the model trained on synthetic data using the original te

    y_pred_proba_synthetic_trained = ann_synthetic_model.predict(X_test_scaled) #
    y_pred_synthetic_trained = (y_pred_proba_synthetic_trained > 0.5).astype(int)

    accuracy_synthetic_trained = accuracy_score(y_test, y_pred_synthetic_trained)
    precision_synthetic_trained = precision_score(y_test, y_pred_synthetic_traine
    recall_synthetic_trained = recall_score(y_test, y_pred_synthetic_trained)
    f1_synthetic_trained = f1_score(y_test, y_pred_synthetic_trained)
    roc_auc_synthetic_trained = roc_auc_score(y_test, y_pred_proba_synthetic_trai

    print("\n--- ANN Model Performance (Trained on Synthetic Data, Evaluated on O
    print(f"Accuracy: {accuracy_synthetic_trained:.4f}")
    print(f"Precision: {precision_synthetic_trained:.4f}")
    print(f"Recall: {recall_synthetic_trained:.4f}")
    print(f"F1 Score: {f1_synthetic_trained:.4f}")
    print(f"ROC AUC Score: {roc_auc_synthetic_trained:.4f}")
```

```
# Compare performance with the original best model (trained on real data)
print("\n--- Performance Comparison: Original ANN (Real Data) vs. ANN (Synthe
print(f"Metric         | ANN (Real Data) | ANN (Synthetic Data)")
print(f"---------------|-----------------|---------------------")
print(f"Accuracy       | {best_accuracy:.4f}            | {accuracy_synthetic_
print(f"Precision      | {best_precision:.4f}            | {precision_synthet
print(f"Recall         | {best_recall:.4f}            | {recall_synthetic_tra
print(f"F1 Score       | {best_f1:.4f}            | {f1_synthetic_trained:.4f
print(f"ROC AUC Score  | {best_roc_auc:.4f}            | {roc_auc_synthetic_t

# Visualize the comparison
labels = ['ANN (Real)', 'ANN (Synthetic)']
accuracy_scores_syn = [best_accuracy, accuracy_synthetic_trained]
precision_scores_syn = [best_precision, precision_synthetic_trained]
recall_scores_syn = [best_recall, recall_synthetic_trained]
f1_scores_syn = [best_f1, f1_synthetic_trained]
roc_auc_scores_syn = [best_roc_auc, roc_auc_synthetic_trained]

x = np.arange(len(labels)) # the label locations
width = 0.15 # the width of the bars

fig, ax = plt.subplots(figsize=(10, 7))
rects1 = ax.bar(x - 2*width, accuracy_scores_syn, width, label='Accuracy')
rects2 = ax.bar(x - width, precision_scores_syn, width, label='Precision')
rects3 = ax.bar(x, recall_scores_syn, width, label='Recall')
rects4 = ax.bar(x + width, f1_scores_syn, width, label='F1 Score')
rects5 = ax.bar(x + 2*width, roc_auc_scores_syn, width, label='ROC AUC')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Score')
ax.set_title('Performance Comparison: ANN (Trained on Real Data) vs. ANN (Tra
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

# Helper function to add labels
def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.4f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
```

```
                    ha='center', va='bottom')

    autolabel(rects1)
    autolabel(rects2)
    autolabel(rects3)
    autolabel(rects4)
    autolabel(rects5)

    fig.tight_layout()
    plt.ylim(0, 1.1)
    plt.show()

    print("\n--- Interpretation of Synthetic Data Training Results ---")
    print("Training a model on synthetic data and evaluating it on real test data
    print("shows how well the synthetic data captures the underlying patterns of
    print("\nConclusions:")
    # Add your specific conclusions here based on the comparison of performance m
    # Example:
    if accuracy_synthetic_trained >= best_accuracy * 0.95: # Check if performance
        print("The model trained on synthetic data performs comparably to the mod
        print("This suggests the synthetic data generation method was reasonably
    elif accuracy_synthetic_trained < best_accuracy * 0.8: # Check if performance
         print("The model trained on synthetic data performs significantly worse
         print("This indicates that the synthetic data does not adequately captur
         print("and relationships present in the real data, or the synthetic data
    else:
         print("The model trained on synthetic data shows a moderate performance
         print("Synthetic data training might be useful in certain scenarios (e.g
         print("but may not fully replace training on real data for optimal perfo
else:
    print("\nSkipping training and evaluation on synthetic data as synthetic data
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `
Best model object recreated at the start.

Best ANN model saved to best_ann_model.h5

Generated 2000 synthetic data points.
Synthetic label distribution:
1    1011
0     989
Name: count, dtype: int64

Training a new ANN model on synthetic data...
Training finished.
```

```
Evaluating the model trained on synthetic data using the original test set.
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step

--- ANN Model Performance (Trained on Synthetic Data, Evaluated on Original Te
Accuracy: 0.3800
Precision: 0.4564
Recall: 0.6126
F1 Score: 0.5231
ROC AUC Score: 0.2017

--- Performance Comparison: Original ANN (Real Data) vs. ANN (Synthetic Data)
Metric          | ANN (Real Data) | ANN (Synthetic Data)
----------------|-----------------|---------------------
Accuracy        | 0.9650          | 0.3800
Precision       | 0.9815          | 0.4564
Recall          | 0.9550          | 0.6126
F1 Score        | 0.9680          | 0.5231
ROC AUC Score   | 0.9821          | 0.2017
```




```
--- Interpretation of Synthetic Data Training Results ---
Training a model on synthetic data and evaluating it on real test data
shows how well the synthetic data captures the underlying patterns of the real
```

```
Conclusions:
The model trained on synthetic data performs significantly worse than the mode
This indicates that the synthetic data does not adequately capture the charact
and relationships present in the real data, or the synthetic data generation m
```

Conclusion: ANN and Comparative Analysis for Predicting Online Ad Clicks

Best ANN Model Summary

Hyperparameters:

Best Hyperparameters: The best model was trained using the following settings:

Learning rate: 0.001

Hidden layers: [64, 32]

Activation function: sigmoid

Dropout rate: 0.0

Batch size: 32

Evaluation Metrics:

- **Accuracy**: `0.9650` – 96.5% of predictions were correct.
- **Precision**: `0.9815` – Extremely low false positive rate.
- **Recall**: `0.9550` – Most of the actual ad clicks were identified.

- **F1 Score**: `0.9680` – Balanced performance between precision and recall.
- **ROC AUC Score**: `0.9818` – Strong model performance in distinguishing between classes.

Model Interpretation

- **Features Used**: The ANN used numerical features like **Age**, **Area Income**, **Daily Internet Usage**, and **Time Spent on Site**.
- **Excluded**: Categorical features such as *City*, *Country*, *Ad Topic Line*, and the *Timestamp* were removed during preprocessing.
- **Explainability**: SHAP analysis and first-layer weights suggest that **Daily Internet Usage** and **Time Spent on Site** were the most influential features.

Comparison with Other Models

Metric	ANN (Best)	Random Forest	Logistic Regression
Accuracy	0.9650	0.9300	0.9600
Precision	0.9815	0.9450	0.9725
Recall	0.9550	0.9279	0.9550
F1 Score	0.9680	0.9364	0.9636
ROC AUC Score	0.9818	0.9834	0.9810

> **Insight**: While **Random Forest** had a slightly higher AUC, the **ANN** delivered the best overall performance across most metrics.

Synthetic Data Experiment

- **Generated**: 2000 synthetic samples
- **Evaluated on Original Test Set**:
  - **Accuracy**: 0.3600
  - **F1 Score**: 0.2644
  - **ROC AUC**: 0.3364

> **Observation**: Models trained solely on **synthetic data** underperformed, indicating that **real data** is critical for reliable behavioral modeling.

Feature Importance (SHAP + Random Forest)

Feature	RF Importance	ANN First Layer Weight
Daily Internet Usage	0.4777	0.1889
Daily Time on Site	0.3263	0.1844
Area Income	0.1033	0.1645
Age	0.0878	0.1393
Male	0.0049	0.1176

> **Key Insight**: Both models agree that **internet usage** and **site engagement time** are the most predictive features.

Final Conclusion

- The **ANN model** stands out as a **strong, balanced, and reliable classifier** for predicting online ad clicks.
- Interpretable models like **Logistic Regression** performed comparably and provide transparency but had **slightly lower performance**.
- **Synthetic data** generated and validated and determined in this experiment it cannot yet replace real behavioral data in high-stakes modeling tasks.

END