Imports

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, accuracy_s
from sklearn.preprocessing import StandardScaler,OneHotEncoder,LabelEncoder
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
```

Classification

Data analysis

```
classification_dataset = pd.read_csv('/content/drive/MyDrive/secondYear/5CS037 - Concepts
classification_df = pd.DataFrame(classification_dataset)
```

```
classification_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 13 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   Person ID                             400 non-null    int64  
 1   Gender                                400 non-null    object 
 2   Age                                   400 non-null    int64  
 3   Occupation                            400 non-null    object 
 4   Sleep Duration (hours)                400 non-null    float64
 5   Quality of Sleep (scale: 1-10)        400 non-null    float64
 6   Physical Activity Level (minutes/day) 400 non-null    int64  
 7   Stress Level (scale: 1-10)            400 non-null    int64  
 8   BMI Category                          400 non-null    object 
 9   Blood Pressure (systolic/diastolic)   400 non-null    object 
 10  Heart Rate (bpm)                      400 non-null    int64  
 11  Daily Steps                           400 non-null    int64  
 12  Sleep Disorder                        110 non-null    object 
dtypes: float64(2), int64(6), object(5)
memory usage: 40.8+ KB
```

```
classification_df.head(10)
```

	Person ID	Gender	Age	Occupation	Sleep Duration (hours)	Quality of Sleep (scale: 1-10)	Physical Activity Level (minutes/day)	Stress Level (scale: 1-10)	Categ
0	1	Male	29	Manual Labor	7.4	7.0	41	7	Ob
1	2	Female	43	Retired	4.2	4.9	41	5	Ob
2	3	Male	44	Retired	6.1	6.0	107	4	Underwe
3	4	Male	29	Office Worker	8.3	10.0	20	10	Ob
4	5	Male	67	Retired	9.1	9.5	19	4	Overwe
5	6	Female	47	Student	6.1	6.9	24	4	Nor
6	7	Male	22	Office Worker	5.1	6.1	26	6	Ob
7	8	Male	49	Office Worker	10.7	6.2	49	8	Ob
8	9	Male	25	Manual Labor	11.9	7.2	27	8	Underwe
9	10	Female	51	Retired	8.2	4.0	64	5	Overwe

Next steps: Generate code with classification_df View recommended plots New interactive sheet

Data Cleaning

```
classification_df = classification_df.dropna()
print("\nMissing Values:")
print(classification_df.isnull().sum())

print("\nNumber of duplicate rows:", classification_df.duplicated().sum())
df = classification_df.drop_duplicates()
print("Number of rows after removing duplicates:", len(classification_df))
```

```
Missing Values:
Person ID                                 0
Gender                                    0
Age                                       0
Occupation                                0
Sleep Duration (hours)                    0
Quality of Sleep (scale: 1-10)            0
Physical Activity Level (minutes/day)     0
Stress Level (scale: 1-10)                0
BMI Category                              0
Blood Pressure (systolic/diastolic)       0
Heart Rate (bpm)                          0
Daily Steps                               0
```

```
Sleep Disorder                          0
dtype: int64

Number of duplicate rows: 0
Number of rows after removing duplicates: 110
```

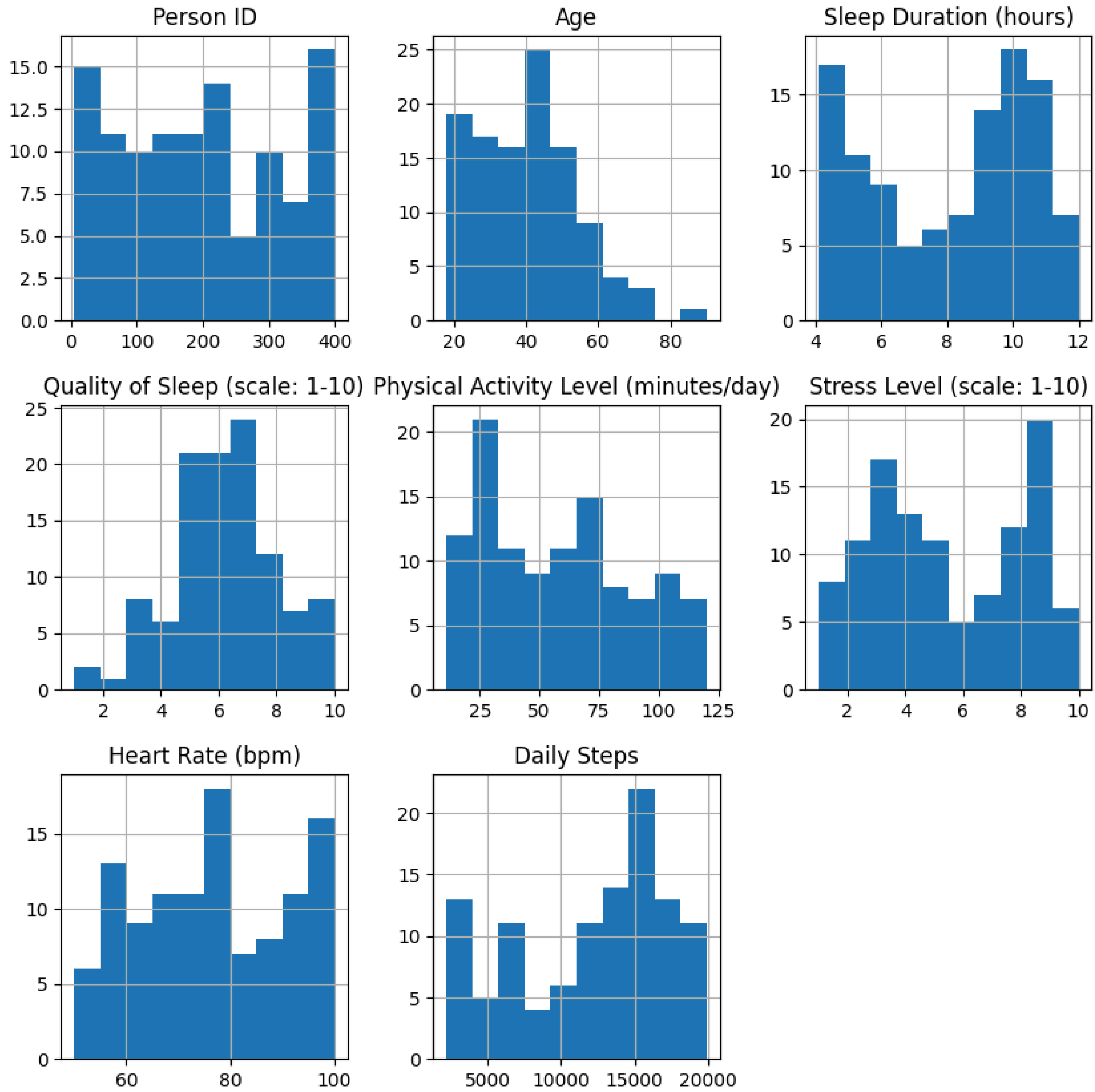
Basic Statistics

```
classification_df.describe()
```

	Person ID	Age	Sleep Duration (hours)	Quality of Sleep (scale: 1-10)	Physical Activity Level (minutes/day)	Stress Level (scale: 1-10)	He R (b
count	110.000000	110.000000	110.000000	110.000000	110.000000	110.000000	110.000
mean	198.409091	39.936364	8.114545	6.155455	57.727273	5.481818	76.063
std	122.082638	14.323444	2.505108	1.800794	30.912101	2.863025	14.396
min	5.000000	18.000000	4.100000	1.000000	11.000000	1.000000	50.000
25%	93.250000	30.000000	5.500000	5.100000	29.250000	3.000000	64.250
50%	191.500000	40.000000	8.850000	6.150000	55.500000	5.000000	77.000
75%	298.750000	47.750000	10.200000	7.175000	79.000000	8.000000	88.750
max	399.000000	90.000000	12.000000	10.000000	120.000000	10.000000	100.000

Data Visualization

```
df.hist(figsize=(10, 10))
plt.show()
```



Build a model from Scratch

```
X = classification_df.drop(columns=["Sleep Disorder"])
Y = df["Sleep Disorder"]
```

```
if X.shape[0] == Y.shape[0]:
  print("Progress Further")
else:
  print("X and Y are not created correctly")

cat_columns = X.select_dtypes(include=['object']).columns
```

```
encoder_X = OneHotEncoder(drop='first', sparse_output=False)
X_encoded = encoder_X.fit_transform(X[cat_columns])

# Convert encoded features to DataFrame and concatenate with numerical features
X_encoded_df = pd.DataFrame(X_encoded, columns=encoder_X.get_feature_names_out(cat_columr
X = pd.concat([X.drop(columns=cat_columns), X_encoded_df], axis=1)

# Encode y (Sleep Disorder) using LabelEncoder (0: No, 1: Yes)
encoder_y = LabelEncoder()
y_encoded = encoder_y.fit_transform(Y)
```

```
Progress Further
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X_encoded, y_encoded, test_size=0.2,
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# if X_train.shape[0] == Y_train.shape[0]:
#   print("Progress Further")
# else:
#   print("x_train and y_train are not created correctly")
```

```
def logistic_function(x):
  """
  Computes the logistic function applied to any value of x.
  Arguments:
    x: scalar or numpy array of any size.
  Returns:
    y: logistic function applied to x.
  """
  import numpy as np
  y = 1 / (1 + np.exp(-x))
  return y
```

```
def log_loss(y_true, y_pred):
  """
  Computes log loss for true target value y ={0 or 1} and predicted target value y' inbet
  Arguments:
    y_true (scalar): true target value {0 or 1}.
    y_pred (scalar): predicted taget value {0-1}.
  Returns:
    loss (float): loss/error value
  """
  import numpy as np
  loss = -y_true * np.log(y_pred) - (1 - y_true) * np.log(1 - y_pred)
  return loss


def cost_function(y_true, y_pred):
    """
    Computes log loss for inputs true value (0 or 1) and predicted value (between 0 and 1
    Args:
      y_true    (array_like, shape (m,)): array of true values (0 or 1)
      y_pred (array_like, shape (m,)): array of predicted values (probability of y_pred b
    Returns:
      cost (float): nonnegative cost corresponding to y_true and y_pred
    """
    assert len(y_true) == len(y_pred), "Length of true values and length of predicted val
    n = len(y_true)
    loss_vec = sum(log_loss(y_true, y_pred))
    cost = loss_vec/n
    return cost


def costfunction_logreg(X, y, w, b):
    """
    Computes the cost function, given data and model parameters
    Args:
      X (ndarray, shape (m,n))  : data on features, m observations with n features
      y (array_like, shape (m,)): array of true values of target (0 or 1)
      w (array_like, shape (n,)): weight parameters of the model
      b (float)                 : bias parameter of the model
    Returns:
      cost (float): nonnegative cost corresponding to y and y_dash
    """
    m, n = X.shape
    assert len(y) == m, "Number of feature observations and number of target observations
    assert len(w) == n, "Number of features and number of weight parameters do not match"
    z = X.dot(w) + b
    y_pred = 1/(1+np.exp(-z))
    cost = cost_function(y, y_pred)
    return cost


# Function to compute gradients of the cost function with respect to model parameters - u
def compute_gradient(X, y, w, b):
    """
    Computes gradients of the cost function with respect to model parameters
    Args:
      X (ndarray, shape (m,n))  : data on features, m observations with n features
      y (array_like, shape (m,)): array of true values of target (0 or 1)
```

```
      w (array_like, shape (n,)): weight parameters of the model
      b (float)                 : bias parameter of the model
    Returns:
      grad_w (array_like, shape (n,)): gradients of the cost function with respect to the
      grad_b (float)                 : gradient of the cost function with respect to the
    """
    m, n = X.shape
    assert len(y) == m, "Number of feature observations and number of target observations
    assert len(w) == n, "Number of features and number of weight parameters do not match"
    y_pred = logistic_function(np.dot(X, w) + b)
    grad_w = (1/m)* X.T.dot(y_pred - y)
    grad_b = (1/m)* np.sum(y_pred - y)
    return grad_w, grad_b


# Gradient descent algorithm for logistic regression
def gradient_descent(X, y, w, b, alpha, n_iter, show_cost = True, show_params = False):
    """
    Implements batch gradient descent algorithm to learn and update model parameters
    with prespecified number of interations and learning rate
    Args:
      X (ndarray, shape (m,n))  : data on features, m observations with n features
      y (array_like, shape (m,)): true values of target (0 or 1)
      w (array_like, shape (n,)): initial value of weight parameters
      b (scalar)                : initial value of bias parameter
      cost_func                 : function to compute cost
      grad_func                 : function to compute gradients of cost with respect to m
      alpha (float)             : learning rate
      n_iter (int)              : number of iterations
    Returns:
      w (array_like, shape (n,)): updated values of weight parameters
      b (scalar)                : updated value of bias parameter
    """
    from tqdm.contrib import itertools
    import math
    import tqdm
    from time import sleep
    m, n = X.shape
    assert len(y) == m, "Number of feature observations and number of target observations
    assert len(w) == n, "Number of features and number of weight parameters do not match"
    cost_history, params_history = [], []
    for i, j in itertools.product(range(n_iter), range(1)):
        grad_w, grad_b = compute_gradient(X, y, w, b)
        w += -alpha * grad_w
        b += -alpha * grad_b
        cost =  costfunction_logreg(X, y, w, b)
        cost_history.append(cost)
        params_history.append([w, b])


    return w, b, cost_history, params_history


from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
y_train = le.fit_transform(Y_train)
```

```
y_test = le.transform(Y_test)



from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(X_train)
x_test_scaled = scaler.transform(X_test)


n_features = x_train_scaled.shape[1]
w = np.zeros(n_features)
b = 0


learning_rate = 0.01
num_iterations = 1000

w_optimized, b_optimized, cost_history, params_history = gradient_descent(
    x_train_scaled, y_train, w, b, learning_rate, num_iterations
)

print("Optimized sleep (w):", w_optimized)
print("Optimized bias (b):", b_optimized)
```

```
100%                                    1000/1000 [00:00<00:00, 5913.63it/s]
Optimized sleep (w): [-0.19479009 -0.06508698  0.12502176  0.01654822  0.22307072  0.
 -0.02446211 -0.13816797 -0.12779584 -0.13816797 -0.12057242 -0.1846115
 -0.14091333  0.14701722  0.31536915  0.34551741  0.35166494  0.
  0.          0.33049145  0.14401933 -0.13816797 -0.1140801   0.33665631
 -0.12779584  0.          0.33214722 -0.19611589  0.33104121  0.48809731
 -0.11533451 -0.1972188   0.32492552  0.32465787  0.         -0.11358532
 -0.15564407 -0.12879035  0.         -0.11533451 -0.18533449 -0.11533451
  0.19435003 -0.10861468 -0.13332837  0.         -0.12779584 -0.11279367
  0.32492552 -0.20840024  0.         -0.11279367 -0.14240933  0.32985993
  0.         -0.17568488 -0.13372293 -0.11959905  0.          0.
 -0.12057242  0.         -0.13945582 -0.11358532 -0.13319224 -0.14856097
 -0.13319224 -0.14947338  0.32185983 -0.21532449 -0.12779584  0.31458896
  0.         -0.18732691  0.11404501  0.          0.30223445  0.32185983
 -0.16357093 -0.14856097 -0.17014506  0.         -0.17014506  0.
 -0.11533451 -0.11832607 -0.16481833 -0.11279367 -0.11358532  0.
  0.29557431  0.32185983 -0.16357093 -0.14091333  0.29676722  0.34551741]
Optimized bias (b): -0.8965976993140091
```

Predictions

```
y_pred_prob = logistic_function(np.dot(x_test_scaled, w_optimized) + b_optimized)
y_pred = (y_pred_prob >= 0.5).astype(int)
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:")
```

```
print(classification_report(y_test, y_pred))
print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}")
```

```
Accuracy: 0.7727272727272727

Classification Report:
              precision    recall  f1-score   support

           0       0.77      1.00      0.87        17
           1       0.00      0.00      0.00         5

    accuracy                           0.77        22
   macro avg       0.39      0.50      0.44        22
weighted avg       0.60      0.77      0.67        22

Confusion Matrix:
[[17  0]
 [ 5  0]]
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: Unde
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: Unde
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: Unde
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
tree_model = DecisionTreeClassifier(random_state=42)
tree_model.fit(x_train_scaled, y_train)

y_pred_tree = tree_model.predict(x_test_scaled)

print("Decision Tree Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_tree)}")
print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_pred_tree)}")
print(f"Classification Report:\n{classification_report(y_test, y_pred_tree)}")
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, confusion_matr
```

```
Decision Tree Evaluation:
Accuracy: 0.7727272727272727
Confusion Matrix:
[[17  0]
 [ 5  0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.77      1.00      0.87        17
           1       0.00      0.00      0.00         5

    accuracy                           0.77        22
   macro avg       0.39      0.50      0.44        22
weighted avg       0.60      0.77      0.67        22

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: Unde
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: Unde
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: Unde
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(x_train_scaled, y_train)
y_pred_rf = rf_classifier.predict(x_test_scaled)
f1_rf = f1_score(y_test, y_pred_rf, average='weighted')
print(f"Random Forest F1 Score: {f1_rf:.4f}")
print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_pred_rf)}")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf)}")
print(f"Classification Report:\n{classification_report(y_test, y_pred_rf)}")
```

```
Random Forest F1 Score: 0.6737
Confusion Matrix:
[[17  0]
 [ 5  0]]
Accuracy: 0.7727272727272727
Classification Report:
              precision    recall  f1-score   support

           0       0.77      1.00      0.87        17
           1       0.00      0.00      0.00         5

    accuracy                           0.77        22
   macro avg       0.39      0.50      0.44        22
weighted avg       0.60      0.77      0.67        22

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: Unde
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: Unde
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: Unde
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Hyper parameter and Cross Validation

```
from sklearn.model_selection import GridSearchCV

tree_params = {
    "criterion": ["gini", "entropy"],
    "max_depth": [3, 5, 10, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}
grid_tree = GridSearchCV(DecisionTreeClassifier(random_state=42), tree_params, cv=5, scori
grid_tree.fit(x_train_scaled, y_train)

# Best hyperparameters for Decision Tree
print("\nBest Parameters for Decision Tree:")
print(grid_tree.best_params_)

# Hyperparameter tuning for Random Forest
```

```
rf_params = {
    "n_estimators": [50, 100, 150],
    "max_depth": [None, 10, 20],
    "min_samples_split": [2, 5, 10]
}
grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), rf_params, cv=5, scoring="
grid_rf.fit(x_train_scaled, y_train)

# Best hyperparameters for Random Forest
print("\nBest Parameters for Random Forest:")
print(grid_rf.best_params_)
```

```
Best Parameters for Decision Tree:
{'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 2}

Best Parameters for Random Forest:
{'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 50}
```

```
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier

# Initialize RFE with Decision Tree Classifier
tree_clf = DecisionTreeClassifier(random_state=42)
rfe = RFE(estimator=tree_clf, n_features_to_select=6)

# Fit RFE
rfe.fit(x_train_scaled, y_train)

# Select only the important features (fixing index error)
selected_features_tree = x_train_scaled[:, rfe.support_]  # Apply mask along columns

# Print results
print("Selected Features for Decision Tree:", selected_features_tree.shape)
print(f"Selected Features Mask: {rfe.support_}")
print(f"Feature Ranking: {rfe.ranking_}")
```

```
Selected Features for Decision Tree: (88, 6)
Selected Features Mask: [False  True False  True False False False False False False
 False False False  True False False False False False False False False
 False False False False False  True False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False  True False False False False False False False False False
 False False False False False False False False False False  True False]
Feature Ranking: [88  1 26  1  2 20 18 30 23 32 31 33 22 15 12  1 13 39 49 10 16 56 4
 58 67  6 68  4  1 71 75 11  8 77 80 83 81 82 90 85 65  9 89 79 55 72 91
  5 87 86 84 21  7 78 76 74 70 66 64 41 42 28 24 35 27 38 45 25 29 40 57
 36 37  1 50 47 43 17 51 44 53 54 63 61 73 52 62 69 60 34 59 19 46  1  3]
```

Feature Selection

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

# Ensure X_train is a DataFrame (assuming original feature names exist)
feature_names = [f'Feature {i}' for i in range(x_train_scaled.shape[1])]  # Create generi
X_train_df = pd.DataFrame(x_train_scaled, columns=feature_names)  # Convert NumPy array t

# Initialize and train RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(x_train_scaled, y_train.ravel())  # Ensure y_train is 1D

# Get feature importances
feature_importances = pd.Series(rf.feature_importances_, index=X_train_df.columns)  # Now

# Sort features by importance
feature_importances = feature_importances.sort_values(ascending=False)
print("Feature Importances:\n", feature_importances)

# Select top 6 important features
top_features = list(feature_importances.index[:6])  # Convert to list for better handling
print("Selected Features:", top_features)
```

```
Feature Importances:
 Feature 29    0.059927
Feature 0     0.042571
Feature 6     0.033082
Feature 1     0.033056
Feature 26    0.032713
                ...
Feature 72    0.000000
Feature 58    0.000000
Feature 59    0.000000
Feature 61    0.000000
Feature 54    0.000000
Length: 96, dtype: float64
Selected Features: ['Feature 29', 'Feature 0', 'Feature 6', 'Feature 1', 'Feature 26
```

Final Model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_s

# Generate feature names based on dataset shape
feature_names = [f'Feature {i}' for i in range(x_train_scaled.shape[1])]

# Convert scaled data back to DataFrame
x_train_scaled_df = pd.DataFrame(x_train_scaled, columns=feature_names)
x_test_scaled_df = pd.DataFrame(x_test_scaled, columns=feature_names)

# Train a RandomForestClassifier to get feature importances
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(x_train_scaled, y_train.ravel())
```

```
# Get feature importance values
feature_importances = pd.Series(rf.feature_importances_, index=feature_names)
feature_importances = feature_importances.sort_values(ascending=False)

# Select the top 6 features
selected_features_rf = list(feature_importances.index[:6])
print("Selected Features:", selected_features_rf)

# Subset dataset using selected features
x_train_rf = x_train_scaled_df[selected_features_rf]
x_test_rf = x_test_scaled_df[selected_features_rf]

# Train the final RandomForest model with selected features
rf_final = RandomForestClassifier(n_estimators=50, max_depth=10, min_samples_split=5, rar
rf_final.fit(x_train_rf, y_train.ravel())
y_pred_rf = rf_final.predict(x_test_rf)

# Evaluate the final model
f1_rf = f1_score(y_test, y_pred_rf, average='weighted')
print("\nRandom Forest Evaluation:")
print(f"Random Forest F1 Score: {f1_rf:.4f}")
print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_pred_rf)}")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf)}")
print(f"Classification Report:\n{classification_report(y_test, y_pred_rf)}")
```

```
Selected Features: ['Feature 29', 'Feature 0', 'Feature 6', 'Feature 1', 'Feature 26

Random Forest Evaluation:
Random Forest F1 Score: 0.6737
Confusion Matrix:
[[17  0]
 [ 5  0]]
Accuracy: 0.7727272727272727
Classification Report:
              precision    recall  f1-score   support
```