NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

ARTIFICIAL INTELLIGENCE LAB

NAME	Ayesha Imran
Class	CS-A
Oel	09
Course	Artificial Intelligence
Date	9-December-25
Submitted To	Lec. Ijlal Haider

IN LAB TASKS

TASK 01: For the given dataset "diabetes.csv", perform the following

Data Exploration and Preprocessing:

Understand the Dataset Structure:

- Load the dataset ;diabetes.csv and review its structure. Identify the number of rows and columns, data types of each feature, and the presence of missing or null values.
- Compute summary statistics (e.g., mean, median, min, max, standard deviation) for numerical features.
- Analyze the target variable (e.g., presence or absence of diabetes) to determine class distribution and check for any imbalances.

Visualize Feature Relationships:

- Create histograms or density plots to examine the distribution of numerical features (e.g., glucose levels, BMI, age).
- Use scatter plots or pair plots to explore relationships between features and their impact on the target variable.
- Visualize correlations between features using a heatmap to identify multicollinearity or highly correlated variables.

Handle Missing Values:

- Identify and handle missing or anomalous values, such as zero entries for features like BMI or Glucose, which may be biologically implausible.
- Use imputation techniques (e.g., mean, median, or predictive imputation) to address missing values.

Encode Categorical Variables:

- If categorical variables are present, encode them using techniques like one-hot encoding or label encoding to make them usable in machine learning models.

Normalize or Scale Numerical Features:

- Apply feature scaling methods such as MinMaxScaler or StandardScaler to ensure numerical features (e.g., Age, BMI, Glucose) are on a comparable
- scale, which is crucial for models like SVM and KNN.

Classification Model Implementation:

- Develop and train classification models to predict the target variable (diabetes
- diagnosis) using the following techniques:
- Logistic regression

- Decision tree
- Random forest
- Support vector machine (SVM)
- K-nearest neighbors

Performance Evaluation:

Evaluate the performance of each model using:

- Accuracy
- Precision
- Recall
- F1-Score

Generate detailed classification reports and confusion matrices for each model to

analyze errors and misclassifications. Comparison and Insights:

Compare the performance of all models across the evaluation metrics (accuracy,

precision, recall, F1-score).

- Highlight the advantages and limitations of each model in the context of this dataset.
- Identify key features that have the most significant impact on diabetes classification.
- Determine the best-performing model and explain why it performs better than others.

CODE:

```
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
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
import warnings
warnings.filterwarnings('ignore')

# ====================================
# 1. Load and Explore the Dataset
```

```
# ====================================
df = pd.read_csv('diabetes.csv')  # Change path if needed

print("Dataset Shape:", df.shape)
print("\nColumns and Data Types:")
print(df.dtypes)
print("\nMissing Values:")
print(df.isnull().sum())

# Summary statistics
print("\nSummary Statistics:")
print(df.describe().round(2))

# Target variable distribution
print("\nClass Distribution (Outcome):")
print(df['Outcome'].value_counts(normalize=True).round(3))

# ====================================
# 2. Handle Anomalous Zeros (Missing Values in Disguise)
# ====================================
# Columns where 0 is biologically implausible
cols_with_zeros = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

print("\nReplacing 0 values with NaN for columns where 0 is impossible:")
for col in cols_with_zeros:
    df[col] = df[col].replace(0, np.nan)

# Impute with median (robust to outliers)
for col in cols_with_zeros:
    df[col].fillna(df[col].median(), inplace=True)

print("\nMissing Values After Imputation:")
print(df.isnull().sum())

# ====================================
# 3. Visualizations
# ====================================
# Histograms of numerical features
df.hist(figsize=(12, 10), bins=20, color='skyblue', edgecolor='black')
plt.suptitle('Distribution of Numerical Features', y=1.02)
plt.show()

# Pairplot (subset to avoid too many plots)
sns.pairplot(df, hue='Outcome', vars=['Glucose', 'BMI', 'Age', 'Insulin',
'BloodPressure'],
```

```
            palette='viridis')
plt.suptitle('Feature Relationships by Diabetes Outcome', y=1.02)
plt.show()

# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()

# ====================================
# 4. Preprocessing
# ====================================
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# Train-test split (stratified to preserve class balance)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

# Feature scaling (important for SVM, KNN, Logistic Regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ====================================
# 5. Classification Models
# ====================================
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Decision Tree': DecisionTreeClassifier(max_depth=5, random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=200, max_depth=10,
random_state=42),
    'SVM': SVC(kernel='rbf', probability=True, random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=7)
}

results = []

for name, model in models.items():
    # Use scaled data for models that need it
    if name in ['Logistic Regression', 'SVM', 'KNN']:
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
    else:
```

```
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    results.append({
        'Model': name,
        'Accuracy': round(acc, 3),
        'Precision': round(prec, 3),
        'Recall': round(rec, 3),
        'F1-Score': round(f1, 3)
    })

    # Classification Report & Confusion Matrix
    print(f"\n=== {name} ===")
    print(classification_report(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No
Diabetes', 'Diabetes'])
    disp.plot(cmap='Blues')
    plt.title(f'Confusion Matrix - {name}')
    plt.show()

# ====================================
# 6. Results Table
# ====================================
results_df = pd.DataFrame(results)
print("\n=== Model Performance Comparison ===")
print(results_df.sort_values('F1-Score', ascending=False))

# ====================================
# 7. Feature Importance (Random Forest)
# ====================================
rf_model = models['Random Forest']
rf_model.fit(X_train, y_train)

feat_imp = pd.Series(rf_model.feature_importances_,
index=X.columns).sort_values(ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x=feat_imp.values, y=feat_imp.index, palette='viridis')
```

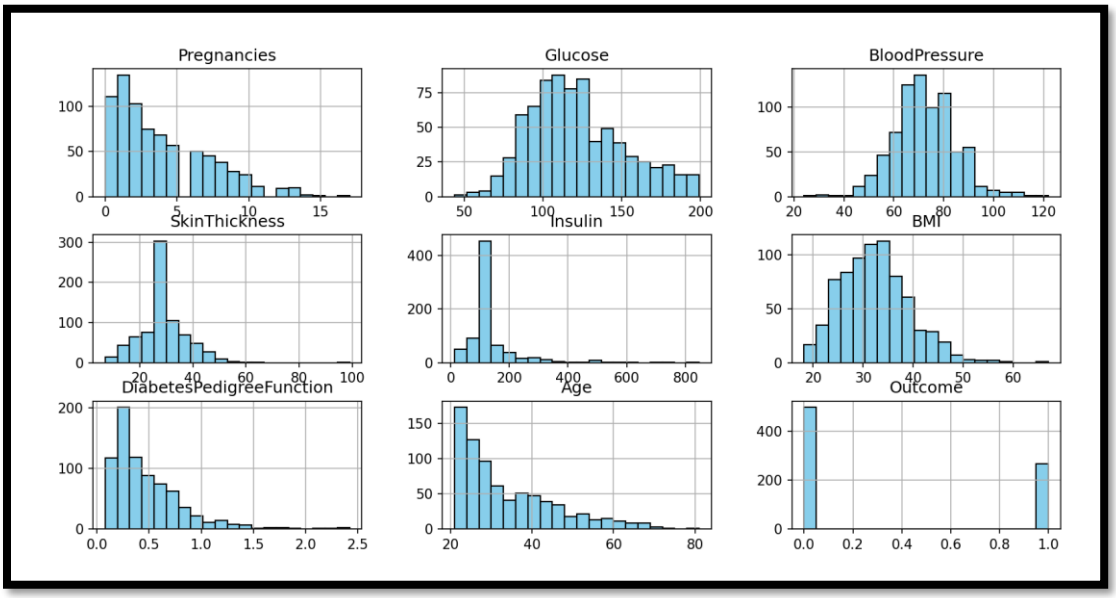
```
plt.title('Feature Importance - Random Forest')
plt.xlabel('Importance Score')
plt.show()

# ====================================
# 8. Key Insights
# ====================================
print("\n=== Key Insights ===")
print("Top 5 most important features:")
print(feat_imp.head(5))

best_model = results_df.loc[results_df['F1-Score'].idxmax()]
print(f"\nBest performing model: {best_model['Model']}")
print(f"Accuracy = {best_model['Accuracy']}, F1-Score = {best_model['F1-
Score']}")

print("\nWhy it performs well:")
print("- Random Forest usually wins because it handles non-linear relationships
and interactions well.")
print("- It is robust to outliers and less prone to overfitting than a single
Decision Tree.")
print("- It provides reliable feature importance.")
print("\nKey predictors of diabetes:")
print("- Glucose (highest importance) – strongly linked to diabetes")
print("- BMI, Age, and Insulin also play significant roles")
```

Output:

```
Dataset Shape: (768, 9)

Columns and Data Types:
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object

Missing Values:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
Summary Statistics:
       Pregnancies  Glucose  ...     Age  Outcome
count       768.00   768.00  ...  768.00   768.00
mean          3.85   120.89  ...   33.24     0.35
std           3.37    31.97  ...   11.76     0.48
min           0.00     0.00  ...   21.00     0.00
25%           1.00    99.00  ...   24.00     0.00
50%           3.00   117.00  ...   29.00     0.00
75%           6.00   140.25  ...   41.00     1.00
max          17.00   199.00  ...   81.00     1.00

[8 rows x 9 columns]

Class Distribution (Outcome):
Outcome
0    0.651
1    0.349
Name: proportion, dtype: float64

Replacing 0 values with NaN for columns where 0 is impossible:

Missing Values After Imputation:
Pregnancies                 0
Glucose                     0
BloodPressure               0
25%           1.00    99.00  ...    24.00     0.00
50%           3.00   117.00  ...    29.00     0.00
75%           6.00   140.25  ...    41.00     1.00
max          17.00   199.00  ...    81.00     1.00

[8 rows x 9 columns]

Class Distribution (Outcome):
Outcome
0    0.651
1    0.349
Name: proportion, dtype: float64

Replacing 0 values with NaN for columns where 0 is impossible:

Missing Values After Imputation:
Pregnancies                 0
Glucose                     0
BloodPressure               0
[8 rows x 9 columns]

Class Distribution (Outcome):
Outcome
0    0.651
1    0.349
Name: proportion, dtype: float64

Replacing 0 values with NaN for columns where 0 is impossible:
```

```
Missing Values After Imputation:
Pregnancies                 0
Glucose                     0
BloodPressure               0

Replacing 0 values with NaN for columns where 0 is impossible:

Missing Values After Imputation:
Pregnancies                 0
Glucose                     0
BloodPressure               0
Replacing 0 values with NaN for columns where 0 is impossible:

Missing Values After Imputation:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
Missing Values After Imputation:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
Glucose                     0
```



POST LAB TASKS

TASK 02: For the given dataset “fashion-mnist.csv”, perform the following

Data Exploration and Preprocessing

Understand the Dataset Structure:

- Load the dataset fashion-mnist.csv and examine its structure, including the total number of rows and columns
- Identify data types for each column, presence of missing values, and provide summary statistics for numerical features
- Explore the distribution of the target variable and ensure the dataset is balanced or identify any class imbalances.

 Visualize Feature Relationships:

- Use appropriate visualization techniques (e.g., histograms, boxplots, pairplots) to analyze the distribution of features.
- Use heatmaps or correlation matrices to identify relationships between numerical features and their potential impact on the target variable.

 Handle Missing Values:

- Identify missing values in the dataset, and apply suitable imputation techniques (e.g., mean, median, or mode) to handle them.

 Encode Categorical Variables:

- Convert any categorical features into numerical representations using encoding techniques such as one-hot encoding or label encoding.

 Normalize or Scale Numerical Features:

- Apply normalization or standardization to ensure numerical features are on a
- comparable scale. Discuss the impact of this step on classification models.

Classification Model Implementation:

 Build and train classification models to predict the target class using the following techniques:

- Logistic regression
- Decision tree
- Random forest
- Support vector machine (SVM)
- K-nearest neighbors

Performance Evaluation:

Evaluate the performance of each model using:

- Accuracy
- Precision
- Recall
- F1-Score
- Generate a classification report and confusion matrix for each model to provide detailed insights into model predictions and errors Comparison and Insights:
- Compare the performance of all models based on the evaluation metrics.
- Discuss the strengths and weaknesses of each model, including their suitability for the
- Fashion-MNIST dataset.
- Identify which model performs best overall and explain why it outperforms others.
- Discuss the role of specific features (if identifiable) in driving classification
- performance.

CODE:

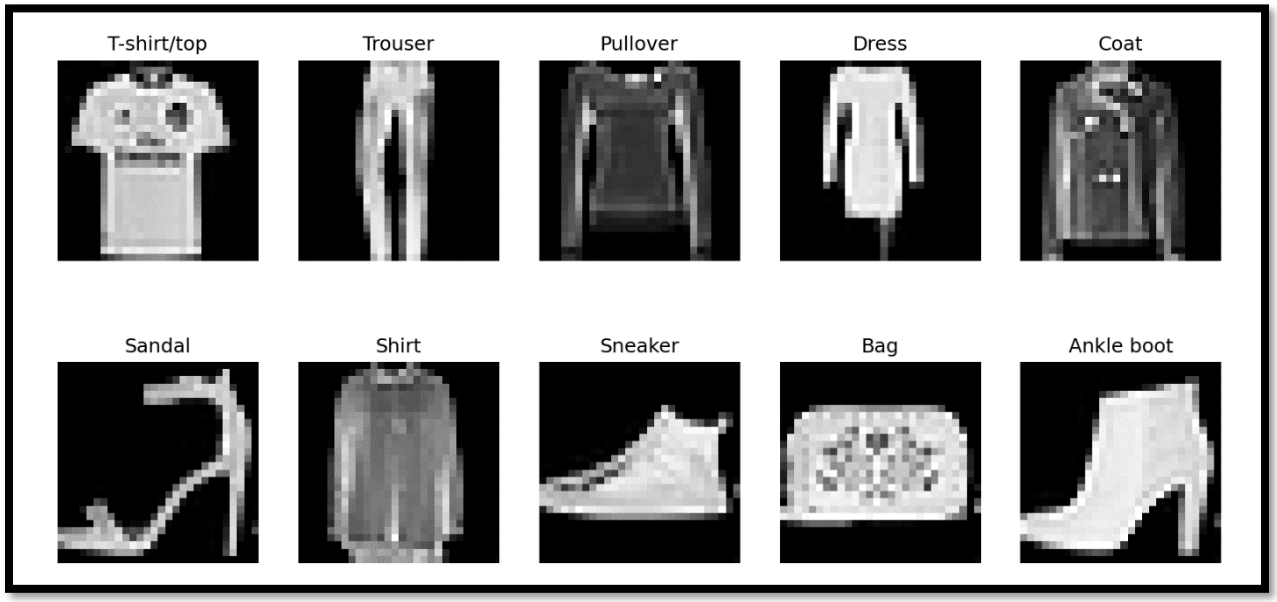
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

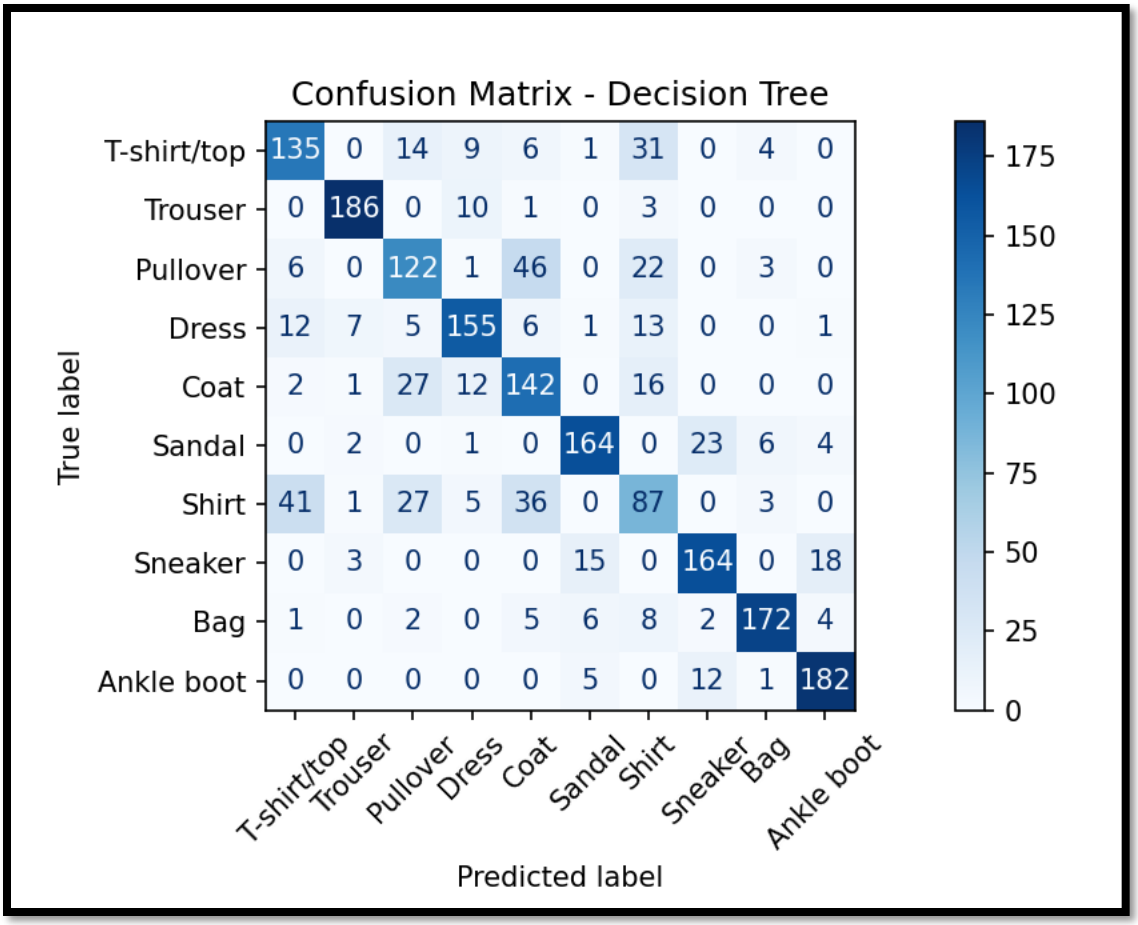
```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
import warnings
warnings.filterwarnings('ignore')

# 1. Load Data
df = pd.read_csv('fashion-mnist.csv')
print("Shape:", df.shape)
print("Missing Values:", df.isnull().sum().sum())
print("Class Distribution:\n", df['label'].value_counts().sort_index())

# Class names
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# 2. Visualizations
fig, axes = plt.subplots(2, 5, figsize=(12, 5))
for i, ax in enumerate(axes.flat):
    img = df[df['label'] == i].iloc[0, 1:].values.reshape(28, 28)
    ax.imshow(img, cmap='gray')
    ax.set_title(class_names[i])
    ax.axis('off')
plt.suptitle('Sample Images', y=1.02)
plt.show()

plt.figure(figsize=(10, 6))
sns.histplot(df.drop('label', axis=1).values.flatten(), bins=50, color='skyblue')
plt.title('Pixel Value Distribution')
plt.show()

# 3. Preprocessing
X = df.drop('label', axis=1) / 255.0
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. Models
```

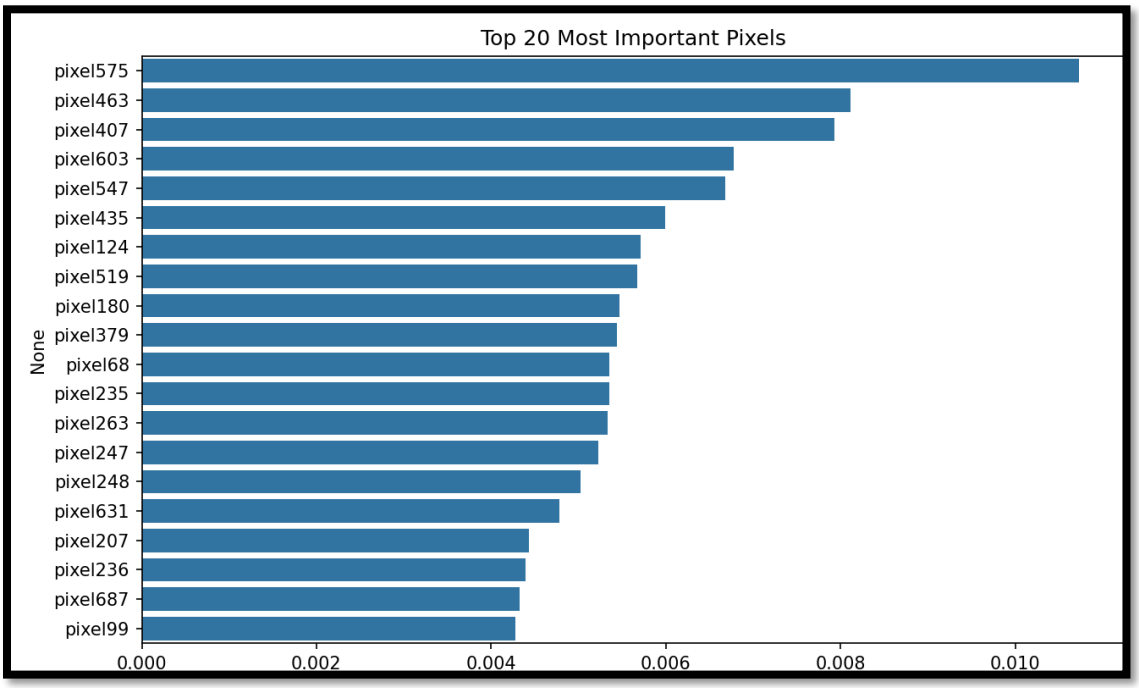
```
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Decision Tree': DecisionTreeClassifier(max_depth=10, random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=200, max_depth=15,
random_state=42),
    'SVM': SVC(kernel='rbf', C=1.0, random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=7)
}

results = []
for name, model in models.items():
    if name in ['Logistic Regression', 'SVM', 'KNN']:
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted')
    rec = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    results.append({
        'Model': name,
        'Accuracy': round(acc, 3),
        'Precision': round(prec, 3),
        'Recall': round(rec, 3),
        'F1-Score': round(f1, 3)
    })

    print(f"\n=== {name} ===")
    print(classification_report(y_test, y_pred, target_names=class_names))

    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(cm, display_labels=class_names)
    disp.plot(cmap='Blues', xticks_rotation=45)
    plt.title(f'Confusion Matrix - {name}')
    plt.show()

# 5. Results Table
results_df = pd.DataFrame(results)
print("\n=== Model Comparison ===")
print(results_df.sort_values('F1-Score', ascending=False))
```

```python
# 6. Feature Importance (Random Forest)
rf = models['Random Forest']
rf.fit(X_train, y_train)
top_pixels = pd.Series(rf.feature_importances_,
index=X.columns).sort_values(ascending=False).head(20)

plt.figure(figsize=(10, 6))
sns.barplot(x=top_pixels.values, y=top_pixels.index.astype(str))
plt.title('Top 20 Most Important Pixels')
plt.show()

# 7. Best Model
best = results_df.loc[results_df['F1-Score'].idxmax()]
print(f"\nBest Model: {best['Model']} (F1 = {best['F1-Score']})")
```

OUTPUT:




Confusion Matrix - Decision Tree
T-shirt/top 135 0 14 9 6 1 31 0 4 0
Trouser 0 186 0 10 1 0 3 0 0 0
Pullover 6 0 122 1 46 0 22 0 3 0
Dress 12 7 5 155 6 1 13 0 0 1
Coat 2 1 27 12 142 0 16 0 0 0
Sandal 0 2 0 1 0 164 0 23 6 4
Shirt 41 1 27 5 36 0 87 0 3 0
Sneaker 0 3 0 0 0 15 0 164 0 18
Bag 1 0 2 0 5 6 8 2 172 4
Ankle boot 0 0 0 0 0 5 0 12 1 182
True label
T-shirt/top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot
Predicted label
175
150
125
100
75
50
25
0


Top 20 Most Important Pixels
None
pixel575
pixel463
pixel407
pixel603
pixel547
pixel435
pixel124
pixel519
pixel180
pixel379
pixel68
pixel235
pixel263
pixel247
pixel248
pixel631
pixel207
pixel236
pixel687
pixel99
0.000
0.002
0.004
0.006
0.008
0.010

Task 2: For the give dataset “Obesity Classification.csv”, bout the obesity classification of individuals:

Data Exploration and Preprocessing:

 Understand the Dataset Structure:

- Load the dataset "Obesity Classification.csv" and examine its structure, including the number of rows and columns.
- Identify the data types for each column and the presence of missing values.
- Generate summary statistics (e.g., mean, median, standard deviation) for numerical features and frequency counts for categorical variables.
- Explore the distribution of the target variable (e.g., obesity classification categories) to check for class imbalances.

 Visualize Relationships Between Features:

- Create visualizations such as histograms, bar charts, or pie charts to understand the distribution of features.
- Use scatter plots, box plots, or violin plots to examine relationships between independent variables (e.g., age, dietary habits, physical activity) and the target variable.
- Generate a correlation matrix or heatmap to identify potential multicollinearity among numerical features.

 Handle Missing Values:

- Identify columns with missing values and apply appropriate imputation techniques (e.g., mean, median, mode, or predictive imputation) to fill them.

 Encode Categorical Variables:

- Convert categorical features (e.g., gender, dietary preferences) into numerical format using techniques such as one-hot encoding or label encoding.

 Normalize or Scale Numerical Features:

- Apply feature scaling methods like MinMaxScaler or StandardScaler to numerical features to ensure they are on a comparable scale.

Classification Model Implementation:

 Develop classification models to predict the obesity classification of individuals using the following techniques:

- Logistic regression
- Decision tree
- Random forest
- Support vector machine (SVM)
- K-nearest neighbors

Performance Evaluation:

 Evaluate the performance of each model using:

- Accuracy
- Precision
- Recall
- F1-Score

 Generate detailed performance reports for each model, including classification reports and confusion matrices.

Comparison and Insights:

 Compare the results of all models based on the evaluation metrics (accuracy,

precision, recall, F1-score).

 Highlight strengths and weaknesses of each model, considering factors like training

time, interpretability, and overfitting tendencies.

 Identify key features that significantly impact obesity classification (e.g., dietary

habits, physical activity levels).

 Determine the best-performing model and explain why it outperformed others.

CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, classification_report, confusion_matrix

df = pd.read_csv('Obesity Classification.csv')

print("Dataset Shape:", df.shape)
print("\nData Types:\n", df.dtypes)
print("\nMissing Values:\n", df.isnull().sum())
print("\nSummary Statistics:\n", df.describe())

categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
for col in categorical_cols:
    print(f"\nValue Counts for {col}:\n", df[col].value_counts())

target = 'Label'  # Change to 'ObesityCategory' if needed
print(f"\nTarget Distribution:\n", df[target].value_counts(normalize=True))

numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()

df[numerical_cols].hist(bins=20, figsize=(12, 10))
plt.suptitle('Histograms of Numerical Features')
plt.show()

for col in categorical_cols:
    if col != target:
        sns.countplot(x=col, data=df)
        plt.title(f'Bar Chart of {col}')
        plt.show()

df[target].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title('Pie Chart of Target Variable')
plt.show()

for col in numerical_cols:
    if col not in ['ID']:
        sns.boxplot(x=target, y=col, data=df)
        plt.title(f'Box Plot of {col} by {target}')
        plt.show()

sns.pairplot(df, hue=target, vars=numerical_cols)
plt.suptitle('Pairplot of Features Colored by Target')
plt.show()
```

```
corr = df[numerical_cols].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()

for col in numerical_cols:
    df[col].fillna(df[col].mean(), inplace=True)
for col in categorical_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)

print("\nMissing Values After Imputation:\n", df.isnull().sum())

df = pd.get_dummies(df, columns=['Gender'], drop_first=True)

le = LabelEncoder()
df[target] = le.fit_transform(df[target])

scaler = StandardScaler()
scale_cols = [col for col in numerical_cols if col not in ['ID']]
df[scale_cols] = scaler.fit_transform(df[scale_cols])

if 'ID' in df.columns:
    df.drop('ID', axis=1, inplace=True)

X = df.drop(target, axis=1)
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
    'K-Nearest Neighbors': KNeighborsClassifier()
}

results = []
for name, model in models.items():
    start_time = time.time()
    model.fit(X_train, y_train)
    train_time = time.time() - start_time

    y_pred = model.predict(X_test)
```

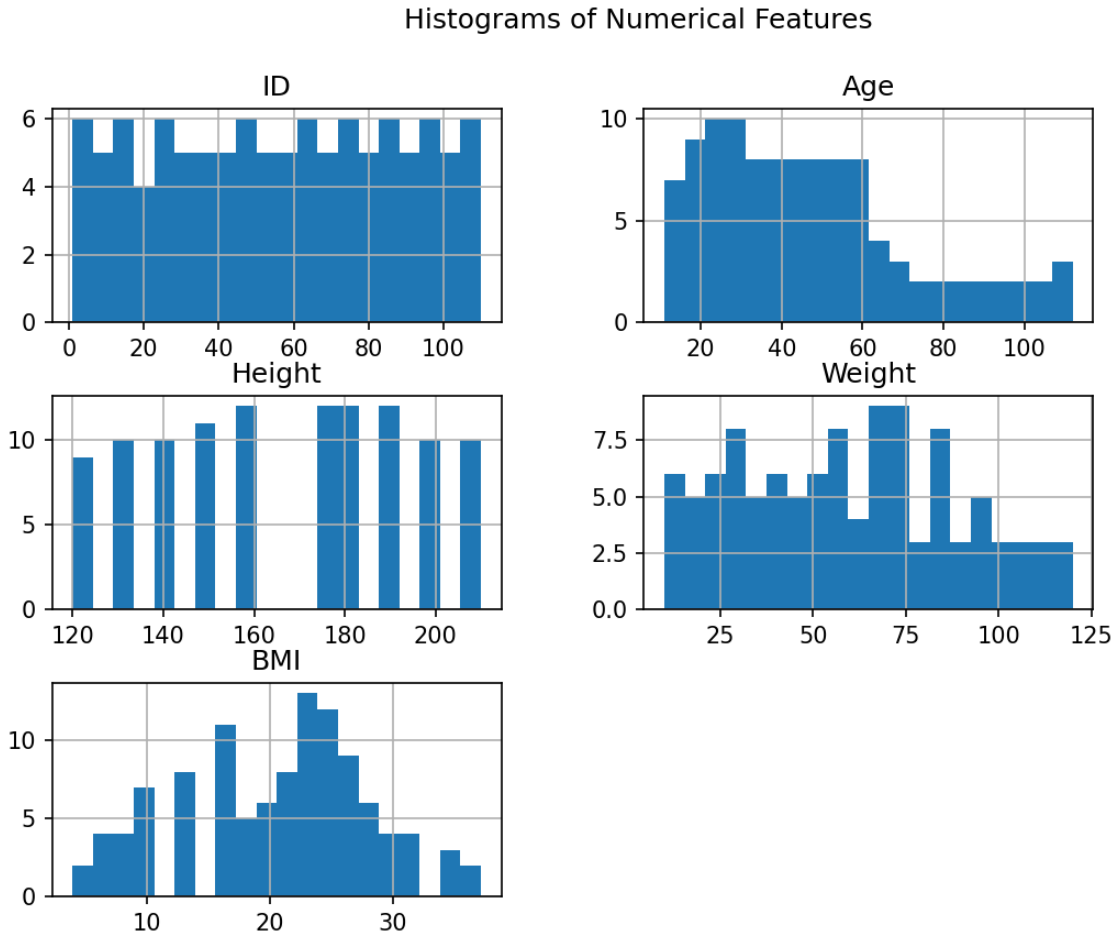
```
    y_train_pred = model.predict(X_train)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted')
    rec = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    results.append({
        'Model': name,
        'Accuracy': acc,
        'Precision': prec,
        'Recall': rec,
        'F1-Score': f1,
        'Training Time (s)': train_time
    })

    print(f"\n{name} Classification Report:\n", classification_report(y_test,
y_pred))
    cm = confusion_matrix(y_test, y_pred)
    print(f"{name} Confusion Matrix:\n", cm)

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix for {name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

results_df = pd.DataFrame(results)
print("\nModel Comparison:\n", results_df)

if 'Random Forest' in models:
    rf_model = models['Random Forest']
    importances = pd.DataFrame({'Feature': X.columns, 'Importance':
rf_model.feature_importances_})
    importances = importances.sort_values('Importance', ascending=False)
    print("\nKey Features from Random Forest:\n", importances)

    sns.barplot(x='Importance', y='Feature', data=importances)
    plt.title('Feature Importances from Random Forest')
    plt.show()

best_model = results_df.loc[results_df['F1-Score'].idxmax()]
print(f"\nBest Performing Model: {best_model['Model']} with F1-Score:
{best_model['F1-Score']:.4f}")
```

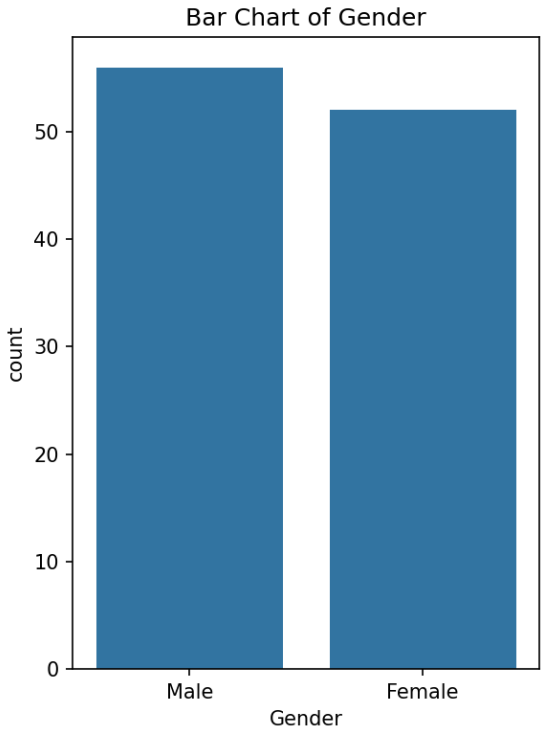
OUTPUT:

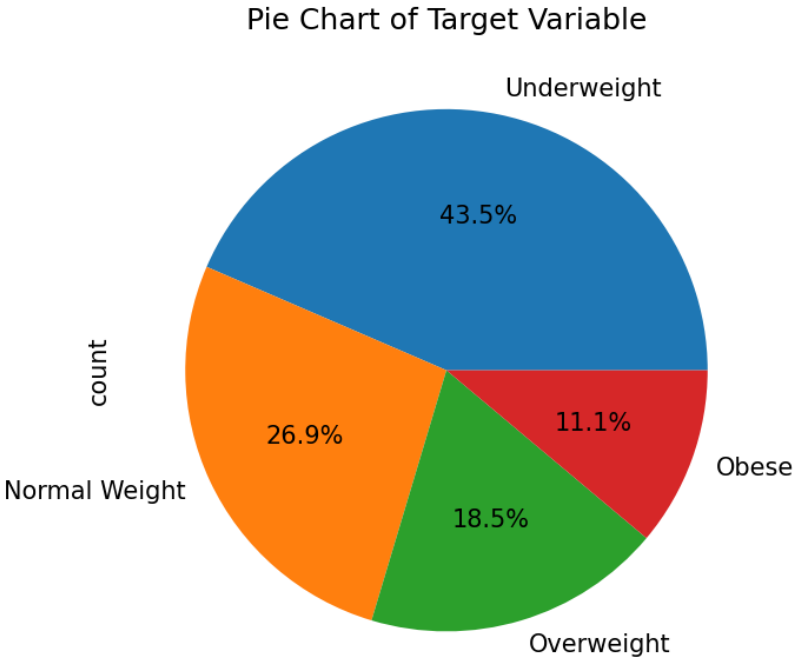
```
Dataset Shape: (108, 7)

Data Types:
 ID          int64
Age          int64
Gender      object
Height       int64
Weight       int64
BMI        float64
Label       object
dtype: object

Missing Values:
 ID         0
Age         0
Gender      0
Height      0
Weight      0
BMI         0
Label       0
dtype: int64
```

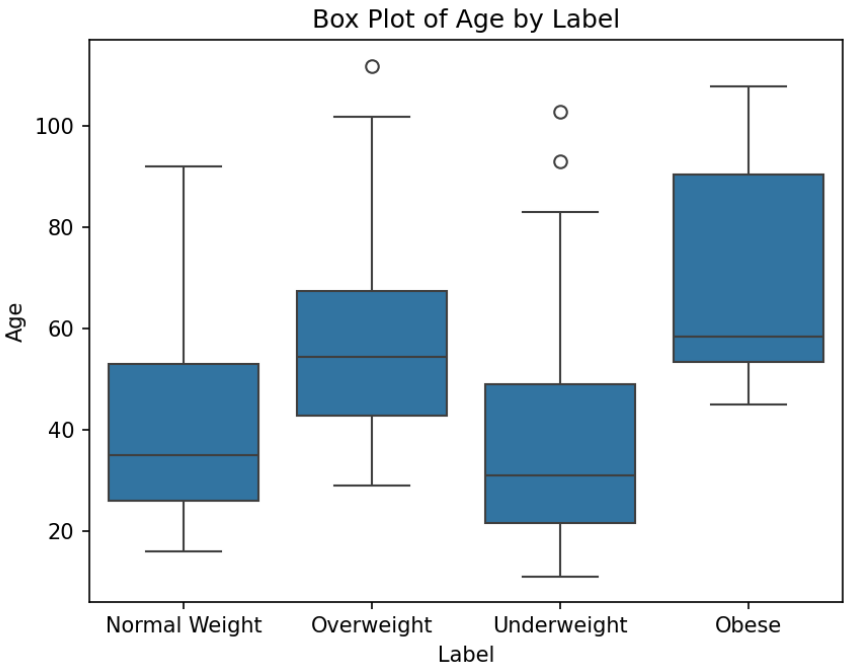
```
Summary Statistics:
                ID         Age      Height      Weight         BMI
count   108.000000  108.000000  108.000000  108.000000  108.000000
mean     56.046296   46.555556  166.574074   59.490741   20.549074
std      31.917939   24.720620   27.873615   28.856233    7.583818
min       1.000000   11.000000  120.000000   10.000000    3.900000
25%      28.750000   27.000000  140.000000   35.000000   16.700000
50%      56.500000   42.500000  175.000000   55.000000   21.200000
75%      83.250000   59.250000  190.000000   85.000000   26.100000
max     110.000000  112.000000  210.000000  120.000000   37.200000

Value Counts for Gender:
 Gender
Male       56
Female     52
Name: count, dtype: int64

Value Counts for Label:
 Label
Underweight       47
Normal Weight     29
Overweight        20
Obese             12
Name: count, dtype: int64
```

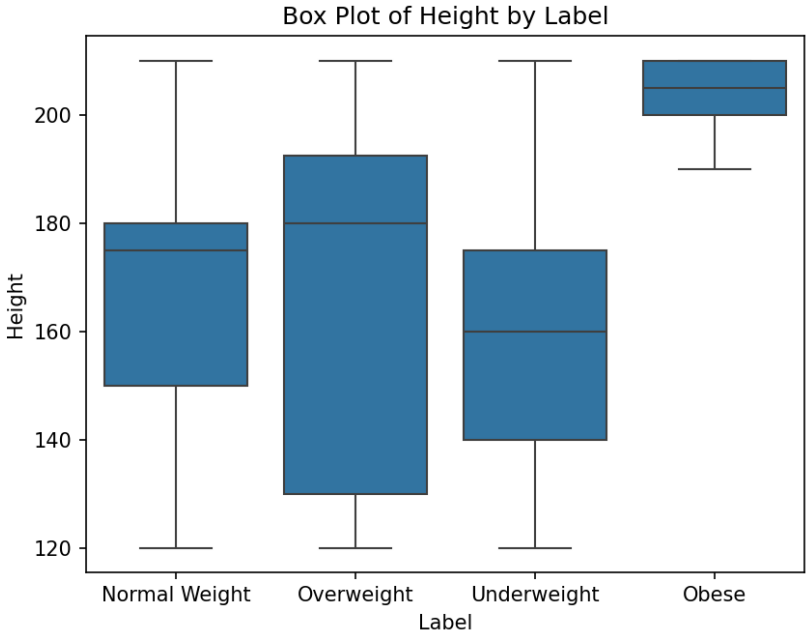
```
Target Distribution:
 Label
Underweight       0.435185
Normal Weight     0.268519
Overweight        0.185185
Obese             0.111111
Name: proportion, dtype: float64
```

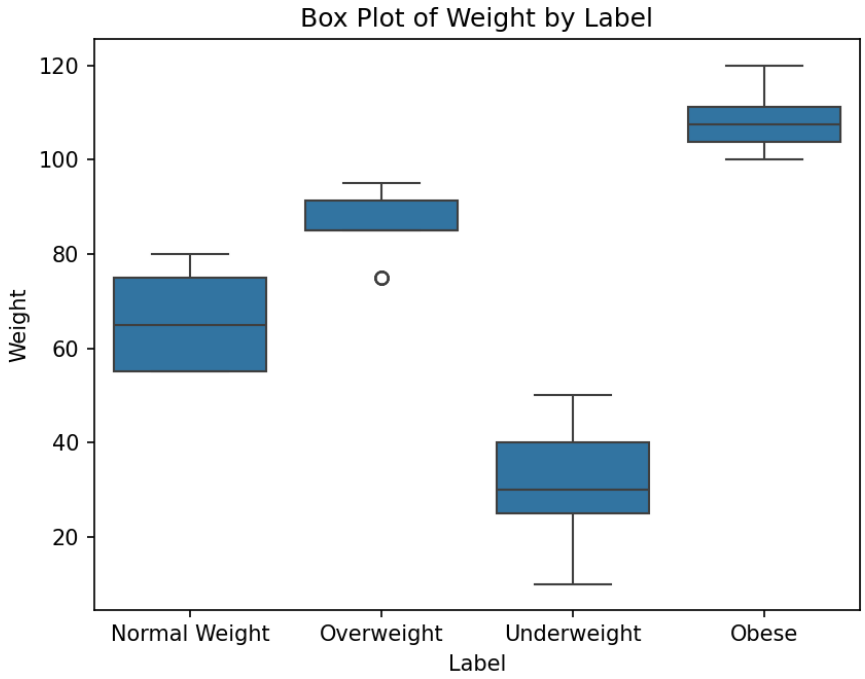


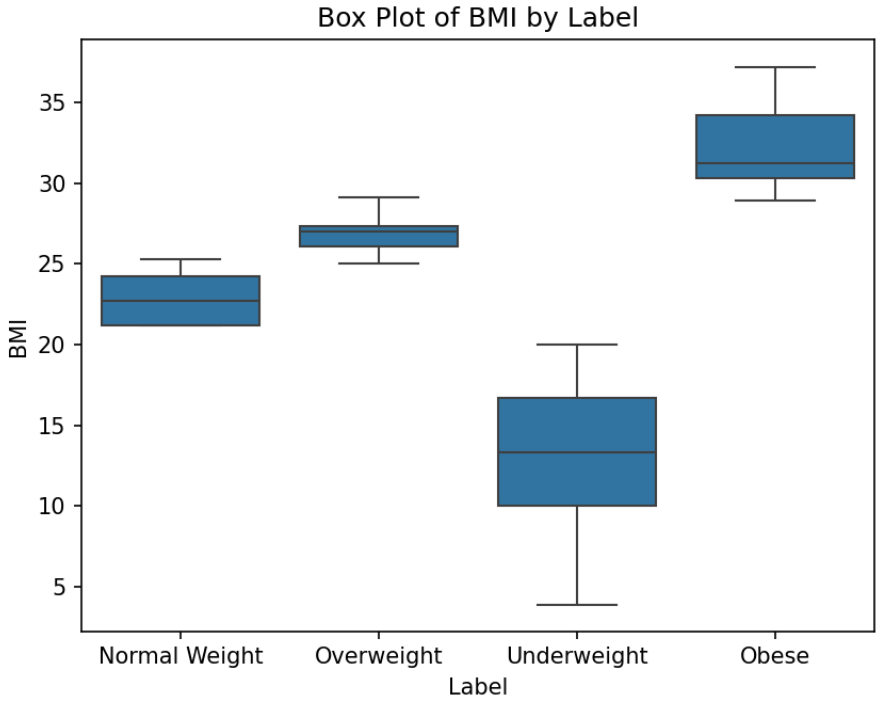

Bar Chart of Gender
count
50
40
30
20
10
0
Male
Female
Gender


Pie Chart of Target Variable
Underweight
43.5%
count
26.9%
Normal Weight
11.1%
Obese
18.5%
Overweight

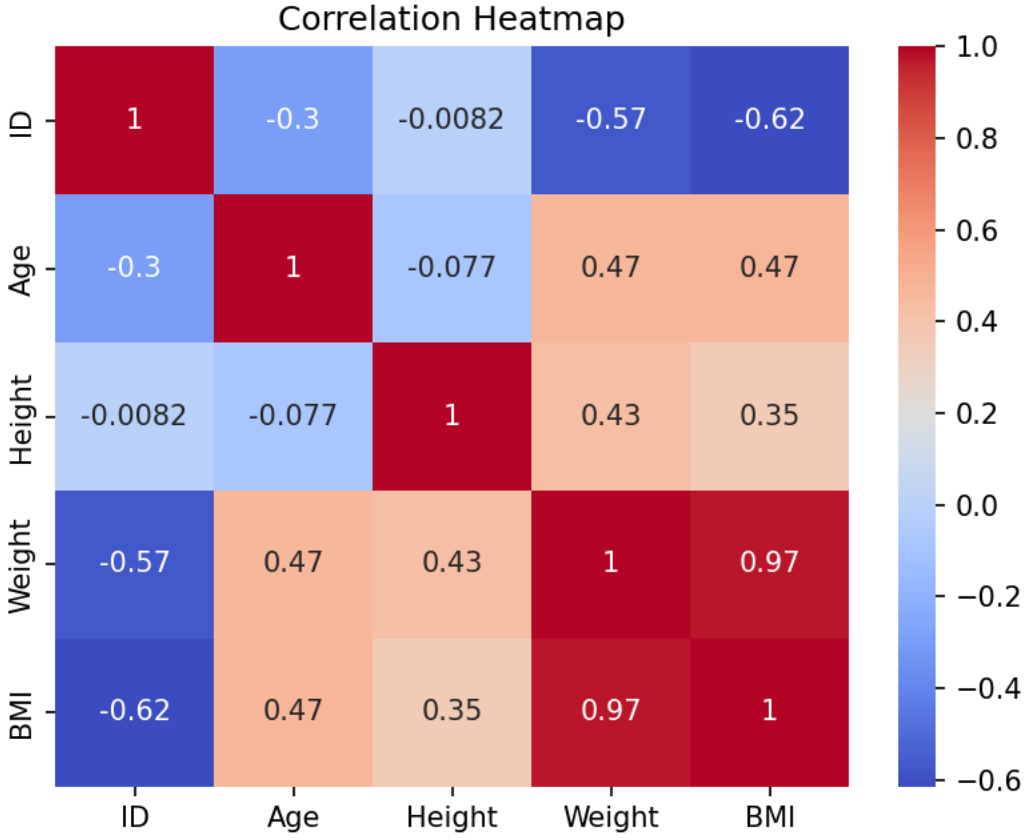

Box Plot of Age by Label
Age
100
80
60
40
20
Normal Weight
Overweight
Underweight
Obese
Label

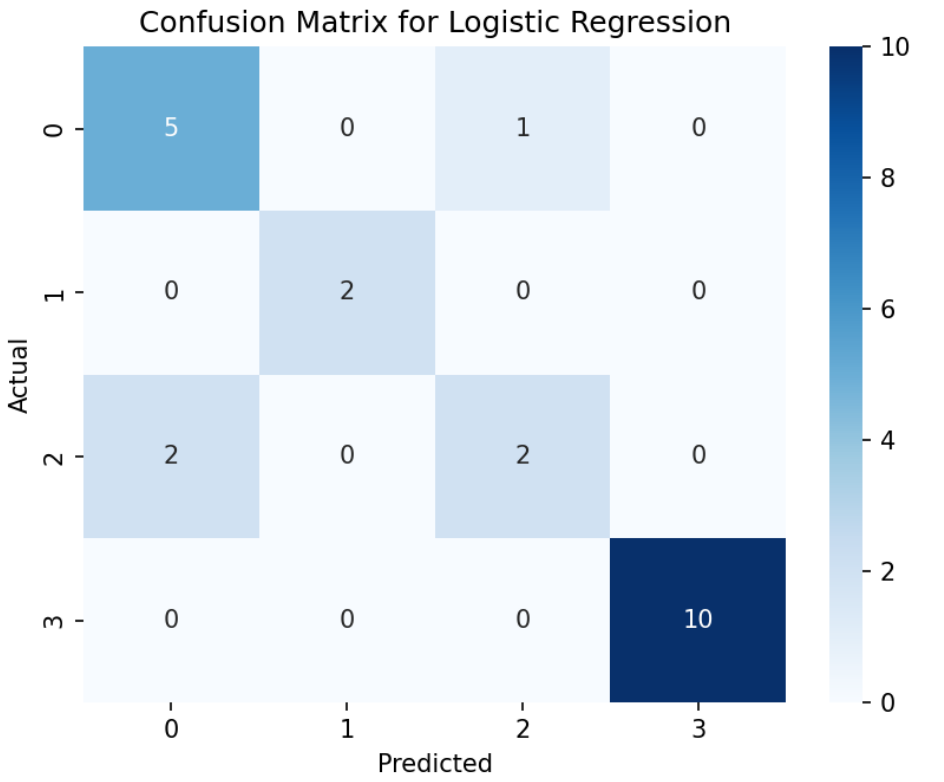

Box Plot of Height by Label
Height
200
180
160
140
120
Normal Weight
Overweight
Underweight
Obese
Label

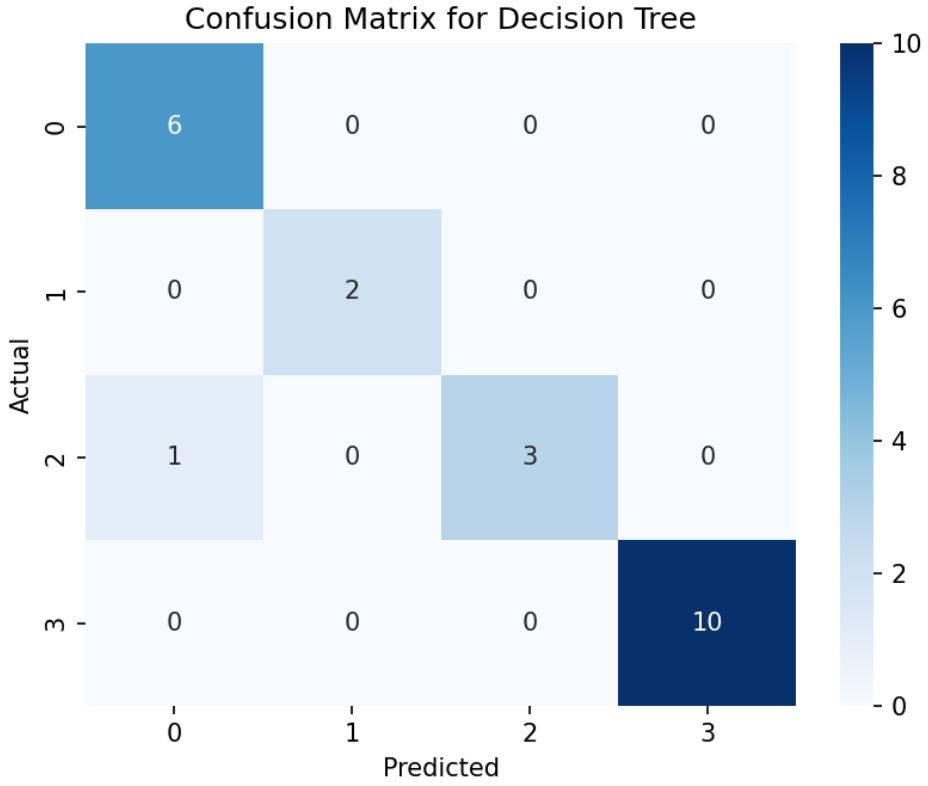

Box Plot of Weight by Label
Weight
120
100
80
60
40
20
Normal Weight
Overweight
Underweight
Obese
Label

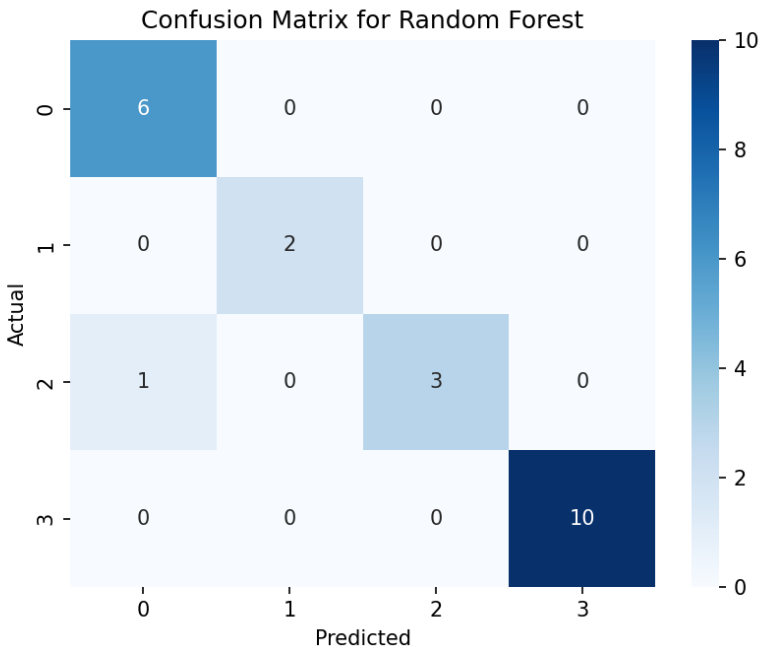

Box Plot of BMI by Label
BMI
35
30
25
20
15
10
5
Normal Weight
Overweight
Underweight
Obese
Label

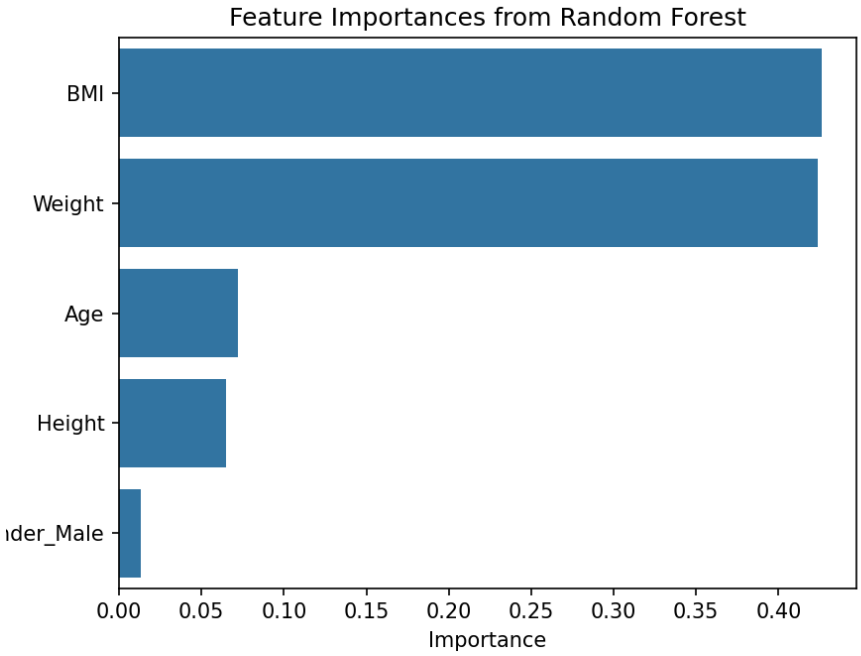

Pairplot of Features Colored by Target
ID
Age
Height
Weight
BMI
Label
Normal Weight
Overweight
Underweight
Obese


Correlation Heatmap
ID
Age
Height
Weight
BMI
1
-0.3
-0.0082
-0.57
-0.62
-0.3
1
-0.077
0.47
0.47
-0.0082
-0.077
1
0.43
0.35
-0.57
0.47
0.43
1
0.97
-0.62
0.47
0.35
0.97
1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


Confusion Matrix for Logistic Regression
Actual
Predicted
0
1
2
3
5
0
1
0
0
2
0
0
2
0
2
0
0
0
0
10
10
8
6
4
2
0


Confusion Matrix for Decision Tree
Actual
Predicted
0
1
2
3
6
0
0
0
0
2
0
0
1
0
3
0
0
0
0
10
10
8
6
4
2
0


Confusion Matrix for Random Forest
Actual
Predicted
0
1
2
3
6
0
0
0
0
2
0
0
1
0
3
0
0
0
0
10
10
8
6
4
2
0


Feature Importances from Random Forest
BMI
Weight
Age
Height
ıder_Male
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Importance

END