Name: Ritesh Chaudhary

Id: 2438464

Week-4 Workshop

Week4_Ritesh_Chaudhary_2438464.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Code + Text

Name:Ritesh Chaudhary Id: 2438464

```
[2] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
```

```
data = pd.read_csv('/content/drive/MyDrive/Concept of Ai Technology/Titanic-Dataset.csv')
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
[4] print("\nData Types:")
    print(data.dtypes)
```

```
Data Types:
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

```python
# Check for missing values in each column.
print("\nMissing Values:")
print(data.isnull().sum())
```

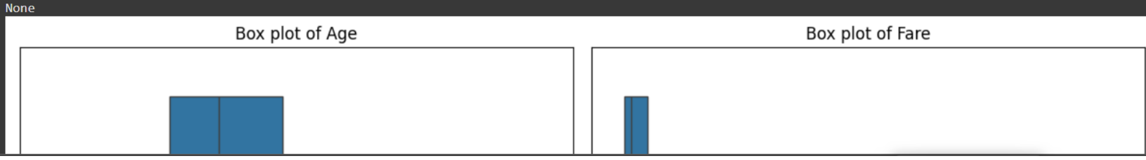
```
Missing Values:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

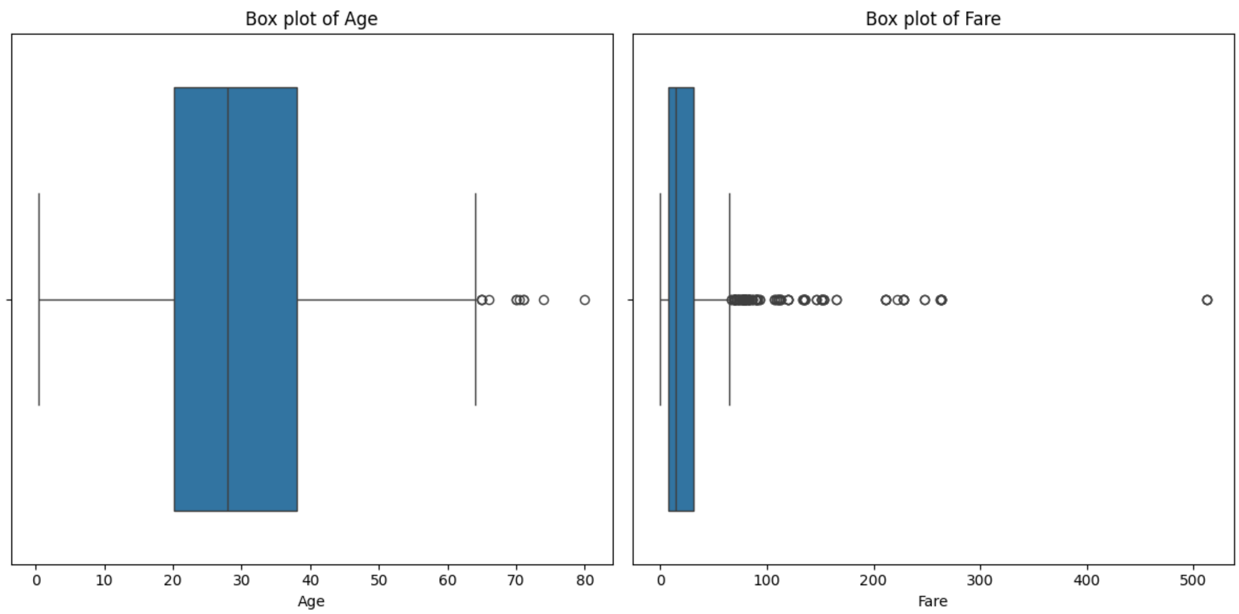
```python
# Summary statistics for numerical columns.
print("\nSummary Statistics:")
print(data.describe())
```

```
Summary Statistics:
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

```python
# Load the dataset
df = pd.read_csv('/content/drive/MyDrive/Concept of Ai Technology/Titanic-Dataset.csv')

# Check the first few rows of the dataset to understand its structure
print(df.head())

# Check the data types and look for numerical columns
print(df.info())

# We will plot box plots for relevant numerical columns: 'Age', 'Fare'
# Create a figure and a set of subplots
plt.figure(figsize=(12, 6))

# Box plot for 'Age'
plt.subplot(1, 2, 1)
sns.boxplot(x=df['Age'])
plt.title('Box plot of Age')

# Box plot for 'Fare'
plt.subplot(1, 2, 2)
sns.boxplot(x=df['Fare'])
plt.title('Box plot of Fare')

# Show the plots
plt.tight_layout()
plt.show()
```

```
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
```



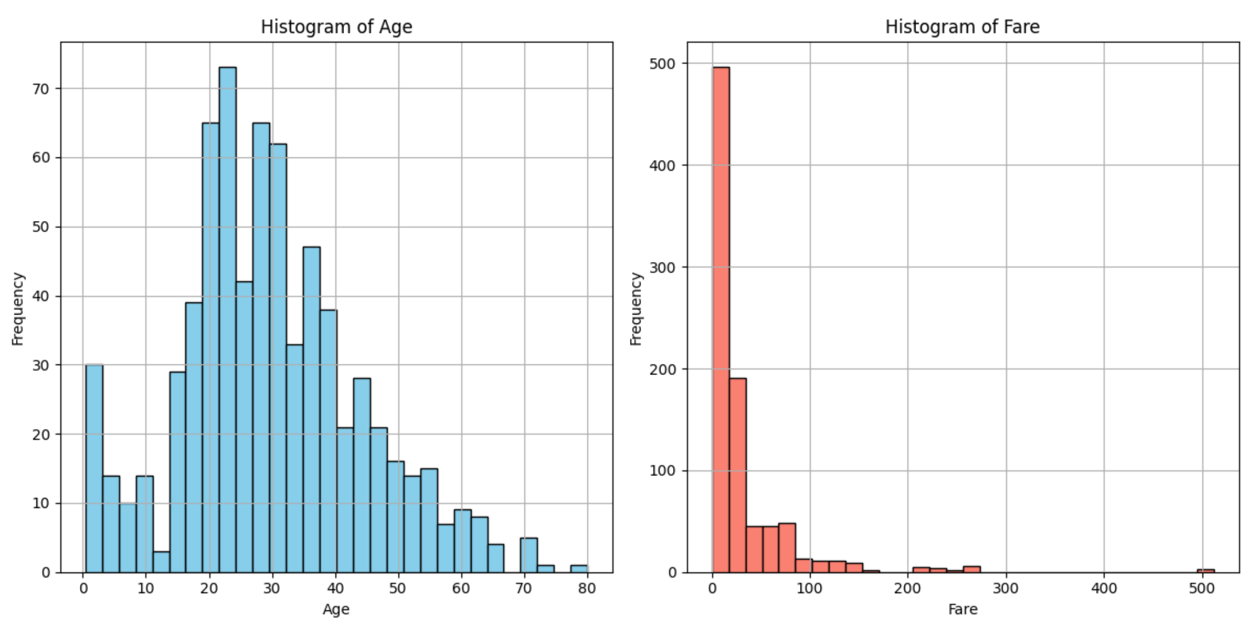


1s

```
# Build Histograms  appropriate columns


# Load the dataset
df = pd.read_csv('/content/drive/MyDrive/Concept of Ai Technology/Titanic-Dataset.csv')

# Check the first few rows of the dataset to understand its structure
print(df.head())

# Check the data types and look for numerical columns
print(df.info())

# Plot histograms for relevant numerical columns: 'Age' and 'Fare'
plt.figure(figsize=(12, 6))

# Histogram for 'Age'
plt.subplot(1, 2, 1)
df['Age'].dropna().hist(bins=30, color='skyblue', edgecolor='black')
plt.title('Histogram of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')

# Histogram for 'Fare'
plt.subplot(1, 2, 2)
df['Fare'].hist(bins=30, color='salmon', edgecolor='black')
plt.title('Histogram of Fare')
plt.xlabel('Fare')
```

```python
# Show the plots
plt.tight_layout()
plt.show()
```

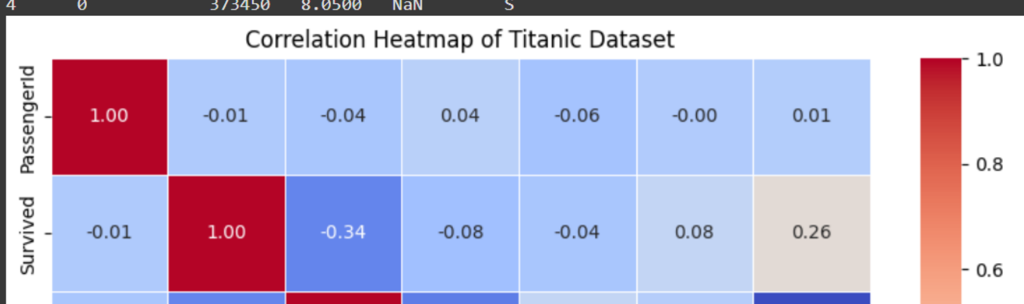
```
2       0  STON/O2. 3101282   7.9250   NaN         S
3       0            113803  53.1000  C123         S
4       0            373450   8.0500   NaN         S
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
```



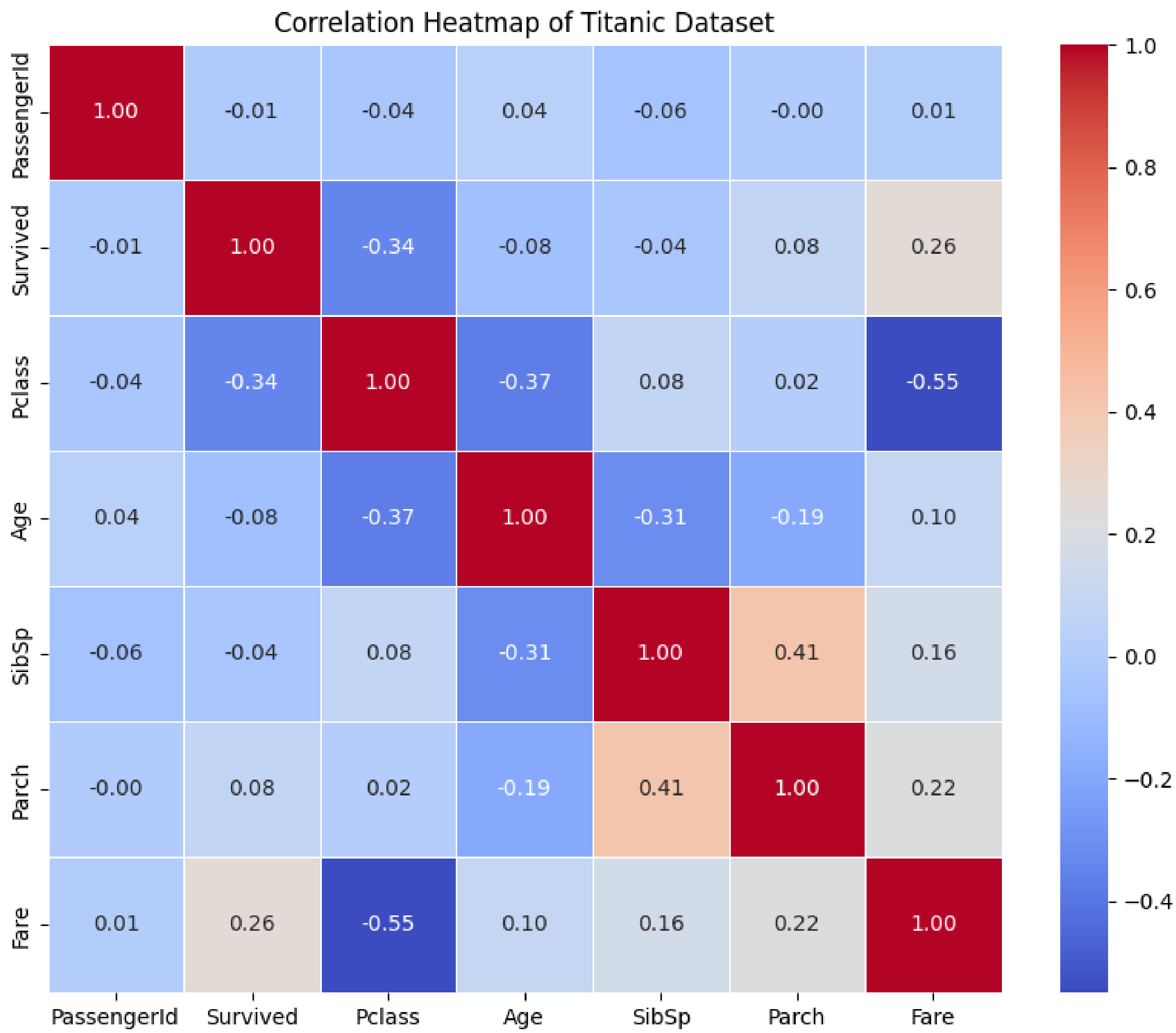
```
# Build Heatmaps for appropriate columns

# Load the dataset
df = pd.read_csv('/content/drive/MyDrive/Concept of Ai Technology/Titanic-Dataset.csv')

# Check the first few rows of the dataset to understand its structure
print(df.head())

# Select only numerical columns
numerical_columns = df.select_dtypes(include=['float64', 'int64'])

# Calculate the correlation matrix for numerical columns
correlation_matrix = numerical_columns.corr()

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

# Set title for the heatmap
plt.title('Correlation Heatmap of Titanic Dataset')

# Show the plot
plt.show()
```

```
1            2         1       1
2            3         1       3
3            4         1       1
```

```
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```



0s completed at 9:07 PM


Correlation Heatmap of Titanic Dataset
PassengerId
Survived
Pclass
Age
SibSp
Parch
Fare
1.00 -0.01 -0.04 0.04 -0.06 -0.00 0.01
-0.01 1.00 -0.34 -0.08 -0.04 0.08 0.26
-0.04 -0.34 1.00 -0.37 0.08 0.02 -0.55
0.04 -0.08 -0.37 1.00 -0.31 -0.19 0.10
-0.06 -0.04 0.08 -0.31 1.00 0.41 0.16
-0.00 0.08 0.02 -0.19 0.41 1.00 0.22
0.01 0.26 -0.55 0.10 0.16 0.22 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

```python
#X = complete code
#y = complete code
# Importing necessary libraries

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer

# Load the Titanic dataset
df = pd.read_csv('/content/drive/MyDrive/Concept of Ai Technology/Titanic-Dataset.csv')

# Check for missing values and data types
print(df.info())

# Feature engineering: Handle missing values
# Impute missing numerical values (e.g., Age)
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns
imputer = SimpleImputer(strategy='mean')
df[numerical_columns] = imputer.fit_transform(df[numerical_columns])

# Handle categorical variables: Encoding the 'Sex' and 'Embarked' columns
df['Sex'] = LabelEncoder().fit_transform(df['Sex'])  # 'male' = 0, 'female' = 1
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])  # Fill missing values in 'Embarked'
df['Embarked'] = LabelEncoder().fit_transform(df['Embarked'])  # Encoding 'Embarked' values

# Drop columns that won't be useful for the model
df = df.drop(['Name', 'Ticket', 'Cabin', 'PassengerId'], axis=1)
```

```python
# Define the target variable y (Survived) and feature variables X
X = df.drop('Survived', axis=1)  # Features (all columns except 'Survived')
y = df['Survived']  # Target (Survived column)

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check the shapes of X and y
print(f"X_train shape: {X_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
```

```python
def train_test_split_scratch(X, y, test_size=0.3, random_seed=42):
    """
    Splits dataset into train and test sets.

    Arguments:
    X : pd.DataFrame
        Feature matrix (DataFrame).
    y : pd.Series
        Target array (Series).
    test_size : float
        Proportion of the dataset to include in the test split (0 < test_size < 1).
    random_seed : int
        Seed for reproducibility.

    Returns:
    X_train, X_test, y_train, y_test : np.ndarray
        Training and testing splits of features and target.
    """
    np.random.seed(random_seed)
    indices = np.arange(X.shape[0])
    np.random.shuffle(indices)  # Shuffle the indices

    test_split_size = int(len(X) * test_size)
    test_indices = indices[:test_split_size]
    train_indices = indices[test_split_size:]

    # Using .iloc[] for proper indexing with DataFrame
    X_train = X.iloc[train_indices]
```

```python
# Example usage (assuming X and y are already DataFrames/Series)
X_train, X_test, y_train, y_test = train_test_split_scratch(X, y, test_size=0.3)

# Printing the shapes of the resulting splits
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (624, 7)
Shape of X_test: (267, 7)
Shape of y_train: (624,)
Shape of y_test: (267,)
```

```python
def euclidean_distance(point1, point2):
    """
    Calculate the Euclidean distance between two points in n-dimensional space.

    Arguments:
    point1 : np.ndarray
        The first point as a numpy array.
    point2 : np.ndarray
        The second point as a numpy array.
```

[12]

```python
try:

    point1 = np.array([3, 4])
    point2 = np.array([0, 0])


    result = euclidean_distance(point1, point2)


    expected_result = 5.0
    assert np.isclose(result, expected_result), f"Expected {expected_result}, but got {result}"

    print("Test passed successfully!")
except ValueError as ve:
    print(f"ValueError: {ve}")
except AssertionError as ae:
    print(f"AssertionError: {ae}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Test passed successfully!
```

```python
[14] def compute_accuracy(y_true, y_pred):
```

```python
def compute_accuracy(y_true, y_pred):
    """
    Compute the accuracy of predictions.

    Arguments:
    y_true : np.ndarray
        The true labels.
    y_pred : np.ndarray
        The predicted labels.

    Returns:
    float
        The accuracy as a percentage (0 to 100).
    """
    #correct_predictions = complete code
    total_predictions = len(y_true)
    #accuracy = complete code
    return accuracy



try:

    predictions = knn_predict(X_test, X_train, y_train, k=3)
```

```python
[14] try:

    predictions = knn_predict(X_test, X_train, y_train, k=3)


    accuracy = compute_accuracy(y_test, predictions)


    print(f"Accuracy of the KNN model on the test set: {accuracy:.2f}%")
except Exception as e:
    print(f"An unexpected error occurred during prediction or accuracy computation: {e}")
```

```
An unexpected error occurred during prediction or accuracy computation: name 'knn_predict' is not defined
```

```python
# Sample data (replace with your actual data)
X_train = pd.DataFrame([[1, 2], [3, 4], [5, 6]])  # Features for training
y_train = np.array(['cat', 'dog', 'cat'])  # Categorical labels for training (example)
X_test = pd.DataFrame([[1, 2], [7, 8]])  # Features for testing

# Step 1: Encode categorical labels in y_train (and y_test if needed)
encoder = LabelEncoder()
y_train = encoder.fit_transform(y_train)  # Convert categorical labels to numeric
# If y_test contains categorical labels, you can similarly encode it
# y_test = encoder.transform(y_test)

# Step 2: Convert X_train and X_test to NumPy arrays if they are DataFrames
X_train = X_train.to_numpy()
X_test = X_test.to_numpy()

# Step 3: Check the types of X_train, X_test, y_train to ensure they are arrays
print(f"X_train type: {type(X_train)}")
print(f"X_test type: {type(X_test)}")
print(f"y_train type: {type(y_train)}")

# Your KNN functions
def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

def knn_predict_single(query, X_train, y_train, k=3):
    distances = [euclidean_distance(query, x) for x in X_train]
    sorted_indices = np.argsort(distances)
    nearest_indices = sorted_indices[:k]
```

```python
def knn_predict(X_test, X_train, y_train, k=3):
    predictions = [knn_predict_single(x, X_train, y_train, k) for x in X_test]
    return np.array(predictions)

# Step 4: Make predictions using the KNN algorithm
try:
    predictions = knn_predict(X_test, X_train, y_train, k=3)
    print("Predictions:", predictions)
    print("Actual labels:", y_train[:2])  # For comparison, we show actual labels for the first 2 test samples
    assert predictions.shape == y_train[:2].shape, "Shape mismatch"
    print("Test case passed successfully!")
except AssertionError as ae:
    print(f"AssertionError: {ae}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
X_train type: <class 'numpy.ndarray'>
X_test type: <class 'numpy.ndarray'>
y_train type: <class 'numpy.ndarray'>
Predictions: [0 0]
Actual labels: [0 1]
Test case passed successfully!
```

+ Code + Text

[16]
```python
import numpy as np
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
```

```python
import numpy as np
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Assuming X and y are your feature matrix and labels
encoder = LabelEncoder()

# Encode the labels if they are categorical (strings)
y_train_encoded = encoder.fit_transform(y_train)
y_test_encoded = encoder.transform(y_test)

def euclidean_distance(point1, point2):
    """
    Calculate the Euclidean distance between two points in n-dimensional space.
    """
    return np.sqrt(np.sum((point1 - point2) ** 2))

def knn_predict_single(query, X_train, y_train, k=3):
    """
    Predict the class label for a single query using the K-nearest neighbors algorithm.
    """
    distances = [euclidean_distance(query, x) for x in X_train]
    sorted_indices = np.argsort(distances)
    nearest_indices = sorted_indices[:k]
    nearest_labels = y_train[nearest_indices]
    prediction = np.bincount(nearest_labels).argmax()
    return prediction
```

```python
def knn_predict(X_test, X_train, y_train, k=3):
    """
    Predict the class labels for all test samples using the K-nearest neighbors algorithm.
    """
    predictions = [knn_predict_single(x, X_train, y_train, k) for x in X_test]
    return np.array(predictions)

def compute_accuracy(y_true, y_pred):
    """
    Compute the accuracy by comparing true and predicted labels.
    """
    return np.mean(y_true == y_pred) * 100  # Multiply by 100 to get percentage

def experiment_knn_k_values(X_train, y_train, X_test, y_test, k_values):
    """
    Run KNN predictions for different values of k and plot the accuracies.
    """
    accuracies = {}

    for k in k_values:
        # Predict the labels for the test set using the current k
        predictions = knn_predict(X_test, X_train, y_train, k=k)

        # Ensure that y_test and predictions have the same length
        if y_test.shape != predictions.shape:
            print(f"Warning: Shape mismatch! y_test shape: {y_test.shape}, predictions shape: {predictions.shape}")
            continue  # Skip this k value if there's a shape mismatch
```

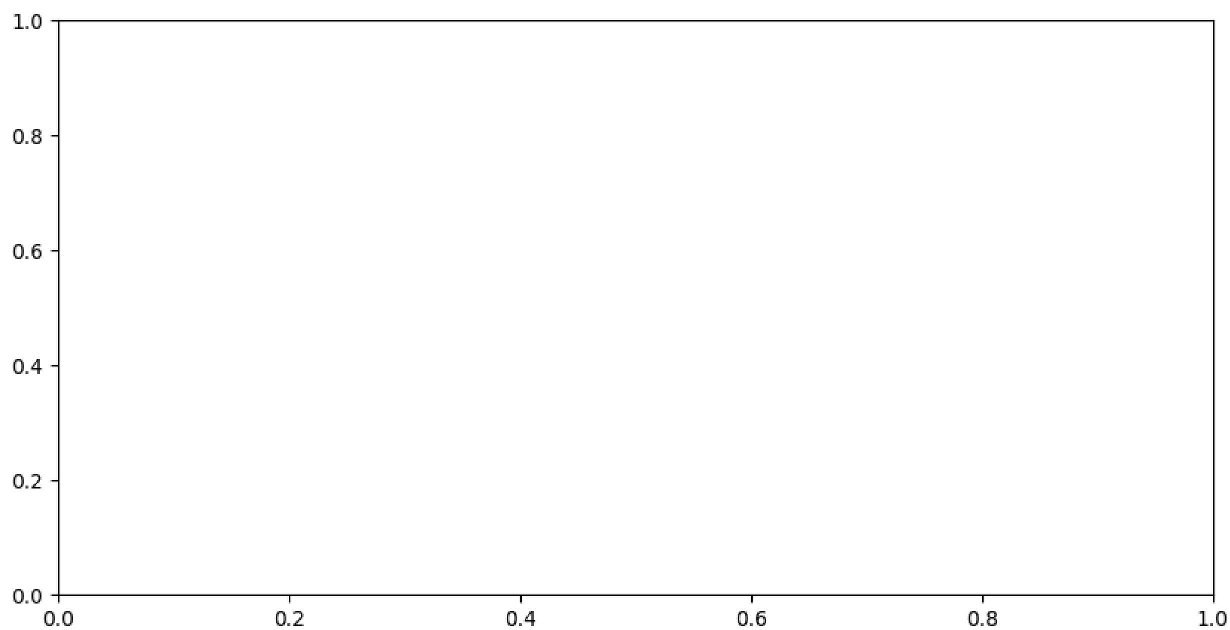
0s completed at 9:07 PM

```python
    for k in k_values:
        # Predict the labels for the test set using the current k
        predictions = knn_predict(X_test, X_train, y_train, k=k)

        # Ensure that y_test and predictions have the same length
        if y_test.shape != predictions.shape:
            print(f"Warning: Shape mismatch! y_test shape: {y_test.shape}, predictions shape: {predictions.shape}")
            continue  # Skip this k value if there's a shape mismatch

        # Calculate accuracy
        accuracy = compute_accuracy(y_test, predictions)
        accuracies[k] = accuracy

        print(f"Accuracy for k={k}: {accuracy:.2f}%")

    # Plotting the accuracies
    plt.figure(figsize=(10, 5))
    plt.plot(k_values, list(accuracies.values()), marker='o')
    plt.xlabel('k (Number of Neighbors)')
    plt.ylabel('Accuracy (%)')
    plt.title('Accuracy of KNN with Different Values of k')
    plt.grid(True)
    plt.show()

    return accuracies

# Example call
k_values = range(1, 21)  # Experimenting with k values from 1 to 20
```

0s completed at 9:07 PM

```python
    accuracies = experiment_knn_k_values(X_train, y_train_encoded, X_test, y_test_encoded, k_values)
    print("Experiment completed. Check the plot for the accuracy trend.")
except Exception as e:
    print(f"An unexpected error occurred during the experiment: {e}")
```

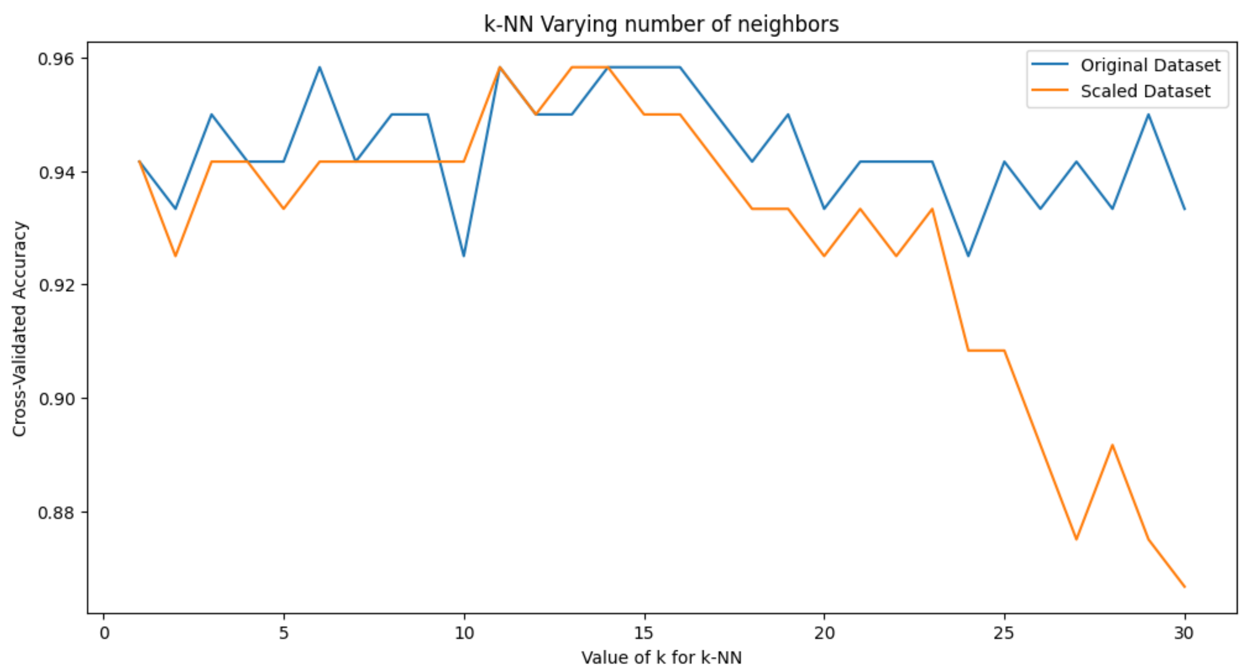
```
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
Warning: Shape mismatch! y_test shape: (267,), predictions shape: (2,)
An unexpected error occurred during the experiment: x and y must have same first dimension, but have shapes (20,) and (0,)
```



```python
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import time

# Load your dataset
# For demonstration, let's create a sample dataset
# Replace this with your actual dataset loading code
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Range of k values to test
k_range = range(1, 31)
accuracy_original = []
accuracy_scaled = []
time_original = []
time_scaled = []
```

```python
    knn = KNeighborsClassifier(n_neighbors=k)

    # Original dataset
    start_time = time.time()
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    end_time = time.time()
    accuracy_original.append(scores.mean())
    time_original.append(end_time - start_time)

    # Scaled dataset
    start_time = time.time()
    scores = cross_val_score(knn, X_train_scaled, y_train, cv=10, scoring='accuracy')
    end_time = time.time()
    accuracy_scaled.append(scores.mean())
    time_scaled.append(end_time - start_time)

# Plot k vs. Accuracy
plt.figure(figsize=(12, 6))
plt.plot(k_range, accuracy_original, label='Original Dataset')
plt.plot(k_range, accuracy_scaled, label='Scaled Dataset')
plt.xlabel('Value of k for k-NN')
plt.ylabel('Cross-Validated Accuracy')
plt.title('k-NN Varying number of neighbors')
plt.legend()
plt.show()

# Plot k vs. Time Taken
plt.figure(figsize=(12, 6))
```

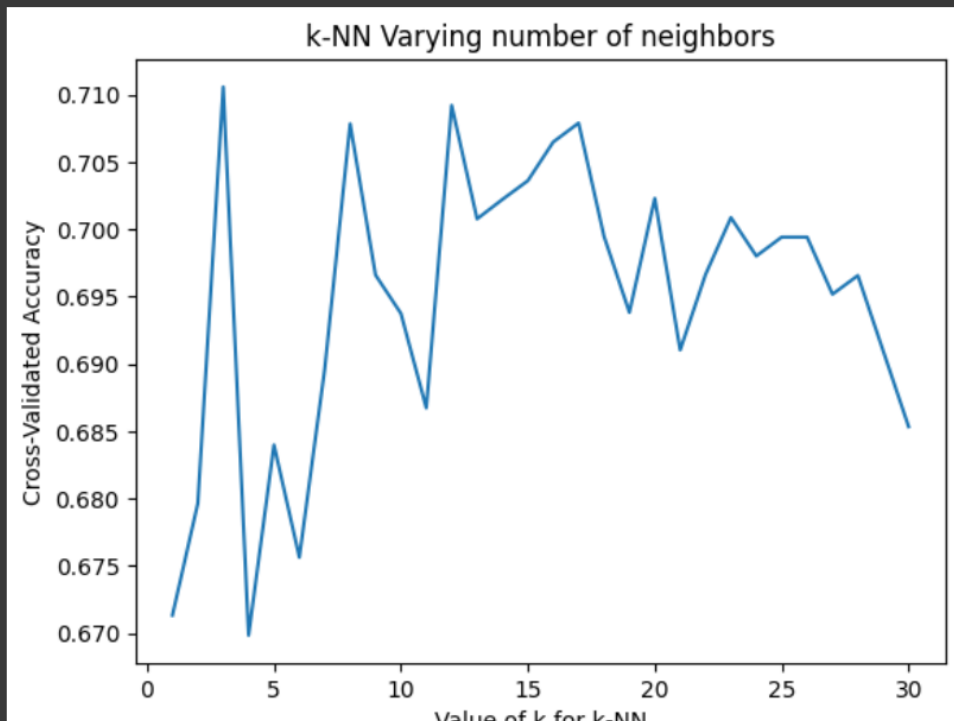
```python
plt.title('k-NN Varying number of neighbors')
plt.legend()
plt.show()

# Plot k vs. Time Taken
plt.figure(figsize=(12, 6))
plt.plot(k_range, time_original, label='Original Dataset')
plt.plot(k_range, time_scaled, label='Scaled Dataset')
plt.xlabel('Value of k for k-NN')
plt.ylabel('Time Taken (seconds)')
plt.title('k-NN Varying number of neighbors - Time Taken')
plt.legend()
plt.show()

# Find the optimal k for both datasets
optimal_k_original = k_range[accuracy_original.index(max(accuracy_original))
optimal_k_scaled = k_range[accuracy_scaled.index(max(accuracy_scaled))]
print("Optimal k for original dataset:", optimal_k_original)
print("Optimal k for scaled dataset:", optimal_k_scaled)
```



```python
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Range of k values to test
k_range = range(1, 31)
k_scores = []

# Perform cross-validation for each k
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())

# Plot the results
plt.plot(k_range, k_scores)
plt.xlabel('Value of k for k-NN')
plt.ylabel('Cross-Validated Accuracy')
plt.title('k-NN Varying number of neighbors')
plt.show()

# Find the optimal k
optimal_k = k_range[k_scores.index(max(k_scores))]
print("Optimal k:", optimal_k)
```

✓ 0s completed at 9:07 PM

```
[ ] # Find the optimal k
    optimal_k = k_range[k_scores.index(max(k_scores))]
    print("Optimal k:", optimal_k)
```



✓ 0s completed at 9:07 PM

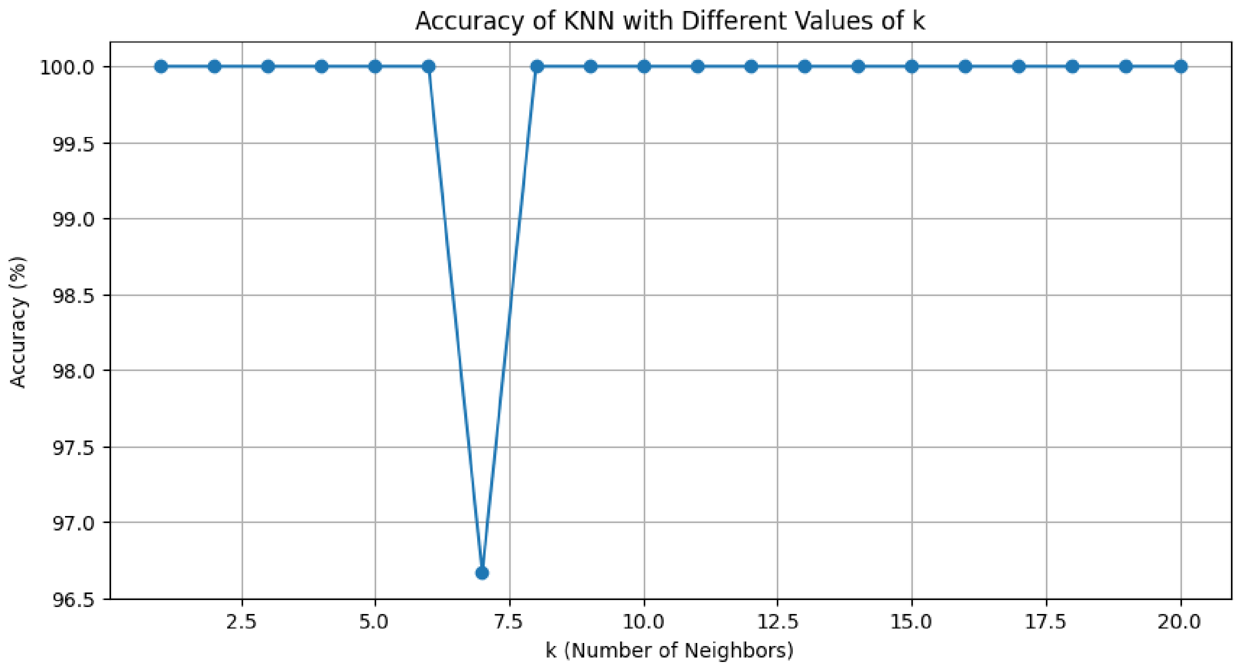
```python
import matplotlib.pyplot as plt
def experiment_knn_k_values(X_train, y_train, X_test, y_test, k_values):
    """
    Run KNN predictions for different values of k and plot the accuracies.

    Arguments:
    X_train : np.ndarray
        The training feature matrix.
    y_train : np.ndarray
        The training labels.
    X_test : np.ndarray
        The test feature matrix.
    y_test : np.ndarray
        The test labels.
    k_values : list of int
        A list of k values to experiment with.

    Returns:
    dict
        A dictionary with k values as keys and their corresponding accuracies as values.
    """
    accuracies = {}

    for k in k_values:
        predictions = knn_predict(X_test, X_train, y_train, k=k)

        accuracy = compute_accuracy(y_test, predictions)
```

✓ 0s completed at 9:07 PM

```python
    plt.figure(figsize=(10, 5))
    plt.plot(k_values, list(accuracies.values()), marker='o')
    plt.xlabel('k (Number of Neighbors)')
    plt.ylabel('Accuracy (%)')
    plt.title('Accuracy of KNN with Different Values of k')
    plt.grid(True)
    plt.show()

    return accuracies


k_values = range(1, 21)


try:
    accuracies = experiment_knn_k_values(X_train, y_train, X_test, y_test, k_values
    print("Experiment completed. Check the plot for the accuracy trend.")
except Exception as e:
    print(f"An unexpected error occurred during the experiment: {e}")
```

```
Accuracy for k=1: 100.00%
Accuracy for k=2: 100.00%
Accuracy for k=3: 100.00%
Accuracy for k=4: 100.00%
Accuracy for k=5: 100.00%
Accuracy for k=6: 100.00%
Accuracy for k=7: 96.67%
Accuracy for k=8: 100.00%
Accuracy for k=9: 100.00%
Accuracy for k=10: 100.00%
```

✓ 0s completed at 9:07 P


Accuracy of KNN with Different Values of k
Accuracy (%)
100.0
99.5
99.0
98.5
98.0
97.5
97.0
96.5
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
k (Number of Neighbors)