Week 1 & 2: NumPy Operations and Tasks

Create NumPy arrays from Python Data Structures, Intrinsic NumPy objects and Random Functions.

```
# 1. Create NumPy arrays
# a. From Python data structures
import numpy as np

list_data = [1, 2, 3, 4, 5]
array_from_list = np.array(list_data)

# b. From intrinsic NumPy objects
zeros_array = np.zeros((3, 3))
ones_array = np.ones((2, 4))

# c. Using random functions
random_array = np.random.rand(3, 3)
print("Array from List:\n", array_from_list)
print("Zeros Array:\n", zeros_array)
print("Random Array:\n", random_array)
```

```
Array from List:
 [1 2 3 4 5]
Zeros Array:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
Random Array:
 [[0.66504087 0.4886642  0.72990495]
 [0.41879849 0.49981234 0.98057609]
 [0.90091381 0.90376282 0.12051952]]
```

Manipulation of NumPy arrays- Indexing, Slicing, Reshaping, Joining and Splitting.

```
# Indexing
print("First element:", array_from_list[0])

# Slicing
print("Slice of array:", array_from_list[1:4])

# Reshaping
reshaped_array = np.arange(1, 13).reshape(3, 4)
print("Reshaped Array:\n", reshaped_array)

# Joining
joined_array = np.concatenate((array_from_list, array_from_list))
```

```
print("Joined Array:\n", joined_array)

# Splitting
split_array = np.split(joined_array, 2)
print("Split Array:\n", split_array)
```

```
First element: 1
Slice of array: [2 3 4]
Reshaped Array:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
Joined Array:
 [1 2 3 4 5 1 2 3 4 5]
Split Array:
 [array([1, 2, 3, 4, 5]), array([1, 2, 3, 4, 5])]
```

Computation on NumPy arrays using Universal Functions and Mathematical methods.

```
squared_array = np.square(array_from_list)
mean_value = np.mean(array_from_list)
print("Squared Array:\n", squared_array)
print("Mean Value:", mean_value)
```

```
Squared Array:
 [ 1  4  9 16 25]
Mean Value: 3.0
```

Import a CSV file and perform various Statistical and Comparison operations on rows/columns.

```
import pandas as pd
data = {
'A': [10, 20, 30],
'B': [5, 15, 25],
'C': [8, 18, 28]
}
df = pd.DataFrame(data)
csv_path = "sample.csv"
df.to_csv(csv_path, index=False)

# Load the CSV file
csv_data = pd.read_csv(csv_path)
print("CSV Data:\n", csv_data)

# Statistical operations
column_sum = csv_data['A'].sum()
row_mean = csv_data.iloc[0].mean()
print("Sum of Column A:", column_sum)
print("Mean of Row 0:", row_mean)
```

```
# Comparison operations
greater_than_15 = csv_data > 15
print("Elements greater than 15:\n", greater_than_15)
```

```
CSV Data:
     A   B   C
0  105   8
1  20  15  18
2  30  25  28
Sum of Column A: 60
Mean of Row 0: 7.666666666666667
Elements greater than 15:
        A      B      C
0  False  False  False
1   True  False   True
2   True   True   True
```

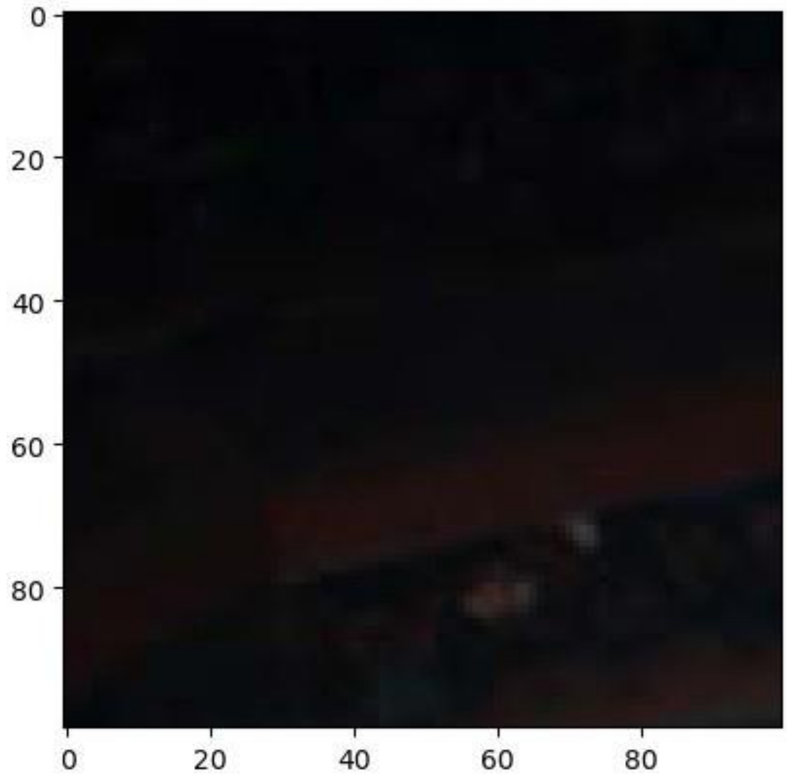
Load an image file and do crop and flip operation using NumPy Indexing

```
from PIL import Image
import matplotlib.pyplot as plt
image_path = "C:\\Users\\B11202016\\Desktop\\nba.jpg" # Replace with your image path
image = Image.open(image_path)
# Convert image to NumPy array
image_array = np.array(image)
# Crop operation
cropped_image = image_array[50:150, 50:150]
# Flip operation
flipped_image = np.flipud(image_array)

# Convert arrays back to images and save
cropped_image_pil = Image.fromarray(cropped_image)
flipped_image_pil = Image.fromarray(flipped_image)
cropped_image_pil.save("cropped_image.jpg")
plt.imshow(cropped_image_pil)
flipped_image_pil.save("flipped_image.jpg")
plt.imshow(flipped_image_pil)
print("Image operations completed. Cropped and flipped images saved.")
```

```
Image operations completed. Cropped and flipped images saved.
```


```
#image  croping
top,bottom,left,right=100,400,200,500
pic2=flipped_image_pil.crop((top,bottom,left,right))
plt.imshow(pic2)
```

```
<matplotlib.image.AxesImage at 0x20236a25670>
```


0
20
40
60
80
0
20
40
60
80

Week 3, 4, 5: Pandas Operations and Tasks

Create Pandas Series and Data Frame from various inputs.

```
# a. From a list
series_from_list = pd.Series([10, 20, 30, 40])
print("Series from List:\n", series_from_list)

# b. From a dictionary
df_from_dict = pd.DataFrame({
'Name': ['Alice', 'Bob', 'Charlie'],
'Age': [25, 30, 35],
'Score': [85.5, 92.0, 78.0]
})
print("DataFrame from Dictionary:\n", df_from_dict)
```

```
Series from List:
 0    10
1    20
2    30
3    40
dtype: int64
DataFrame from Dictionary:
       Name  Age  Score
```

```
0    Alice   25   85.5
1      Bob   30   92.0
2  Charlie   35   78.0
```

Import any CSV file to Pandas Data Frame and perform the following:

(a) Visualize the first and last 10 records.

(b) Get the shape, index and column details.

(c) Select/Delete the records (rows)/columns based on conditions.

(d) Perform ranking and sorting operations.

(e) Do required statistical operations on the given columns.

(f) Find the count and uniqueness of the given categorical values.

(g) Rename single/multiple columns.

```
# Create a sample CSV
data = {
'ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
'Category': ['A', 'B', 'A', 'C', 'B', 'C', 'A', 'A', 'B', 'C'],
'Value': [50, 60, 45, 70, 80, 75, 85, 90, 55, 65]
}
csv_file = "sample_week3.csv"
pd.DataFrame(data).to_csv(csv_file, index=False)

# Load the CSV
df = pd.read_csv(csv_file)

# a. Visualize the first and last 10 records
print("First 10 records:\n", df.head(10))
print("Last 10 records:\n", df.tail(10))

# b. Get the shape, index, and column details
print("Shape of DataFrame:", df.shape)
print("Index:", df.index)
print("Columns:", df.columns)

# c. Select/Delete records based on conditions

# Select rows where Value > 60
selected_rows  = df[df['Value'] >  60]
print("Rows where Value > 60:\n", selected_rows)

# Delete rows where Category == 'B'
filtered_df = df[df['Category'] != 'B']
print("DataFrame after deleting rows where Category == 'B':\n",
filtered_df)
```

```python
# d. Perform ranking and sorting
df['Value_Rank'] = df['Value'].rank(ascending=False)
sorted_df = df.sort_values(by='Value', ascending=False)
print("Ranked DataFrame:\n", df)
print("Sorted DataFrame:\n", sorted_df)

# e. Statistical operations
mean_value = df['Value'].mean()
sum_value = df['Value'].sum()
print("Mean of Value column:", mean_value)
print("Sum of Value column:", sum_value)

# f. Count and uniqueness of categorical values
category_counts = df['Category'].value_counts()
unique_categories = df['Category'].unique()
print("Category Counts:\n", category_counts)
print("Unique Categories:\n", unique_categories)

# g. Rename single/multiple columns
df.rename(columns={'Value': 'Score_Value', 'Category': 'Group'},
inplace=True)
print("Renamed DataFrame:\n", df)
```

```
First 10 records:
    ID Category  Value
0   1        A     50
1   2        B     60
2   3        A     45
3   4        C     70
4   5        B     80
5   6        C     75
6   7        A     85
7   8        A     90
8   9        B     55
9  10        C     65
Last 10 records:
    ID Category  Value
0   1        A     50
1   2        B     60
2   3        A     45
3   4        C     70
4   5        B     80
5   6        C     75
6   7        A     85
7   8        A     90
8   9        B     55
9  10        C     65
Shape of DataFrame: (10, 3)
Index: RangeIndex(start=0, stop=10, step=1)
```

```
Columns: Index(['ID', 'Category', 'Value'], dtype='object')
Rows where Value > 60:
    ID Category  Value
3   4        C     70
4   5        B     80
5   6        C     75
6   7        A     85
7   8        A     90
9  10        C     65
DataFrame after deleting rows where Category == 'B':
    ID Category  Value
0   1        A     50
2   3        A     45
3   4        C     70
5   6        C     75
6   7        A     85
7   8        A     90
9  10        C     65
Ranked DataFrame:
    ID Category  Value  Value_Rank
0   1        A     50         9.0
1   2        B     60         7.0
2   3        A     45        10.0
3   4        C     70         5.0
4   5        B     80         3.0
5   6        C     75         4.0
6   7        A     85         2.0
7   8        A     90         1.0
8   9        B     55         8.0
9  10        C     65         6.0
Sorted DataFrame:
    ID Category  Value  Value_Rank
7   8        A     90         1.0
6   7        A     85         2.0
4   5        B     80         3.0
5   6        C     75         4.0
3   4        C     70         5.0
9  10        C     65         6.0
1   2        B     60         7.0
8   9        B     55         8.0
0   1        A     50         9.0
2   3        A     45        10.0
Mean of Value column: 67.5
Sum of Value column: 675
Category Counts:
 Category
A    4
B    3
C    3
```

```
Name: count, dtype: int64
Unique Categories:
 ['A' 'B' 'C']
Renamed DataFrame:
   ID Group  Score_Value  Value_Rank
0   1     A           50         9.0
1   2     B           60         7.0
2   3     A           45        10.0
3   4     C           70         5.0
4   5     B           80         3.0
5   6     C           75         4.0
6   7     A           85         2.0
7   8     A           90         1.0
8   9     B           55         8.0
9  10     C           65         6.0
```

Week 6, 7, 8: Linear Regression and Residual Analysis

Develop a model on residual analysis of simple linear regression.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from scipy.stats import probplot

# Generate sample data for demonstration
np.random.seed(42)

X = np.random.rand(100, 1) * 10 # Predictor variable
y = 5 * X + np.random.normal(0, 2, (100, 1)) # Response variable with
noise

# Split into training data
X = X.flatten() # Flatten for compatibility
y = y.flatten() # Flatten for compatibility

# 1. Develop a model on residual analysis of simple linear regression
model = LinearRegression()
model.fit(X.reshape(-1, 1), y)

# Predictions
y_pred = model.predict(X.reshape(-1, 1))
```

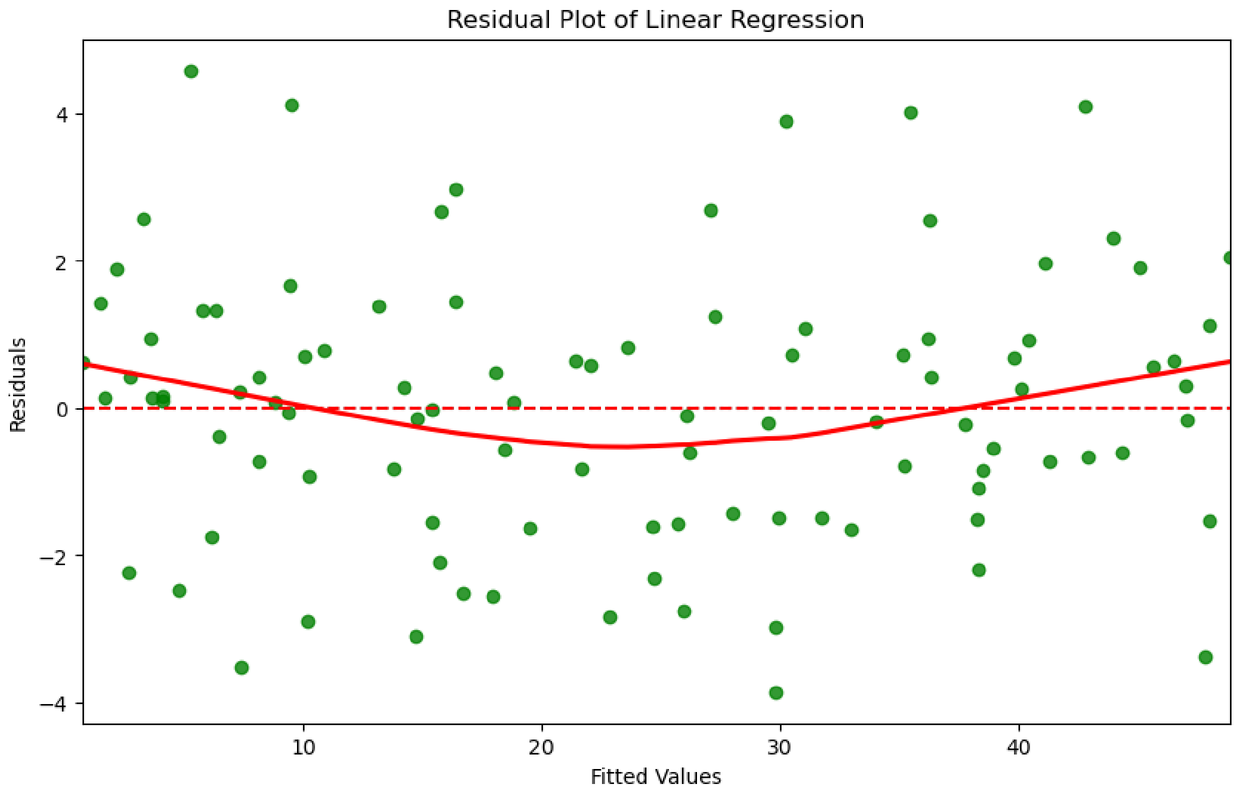
```python
# Residuals
residuals = y - y_pred

print("predictions")
print(y_pred)
print("residuals")
print(residuals)
```

```
predictions
[18.81279122 47.09168167 36.35678697 29.81262224  8.08765795
8.08647412
  3.28096234 42.9425104  29.93318972 35.18271558  1.44048863
48.03380776
 41.28685468 10.85189217  9.35424417  9.43176884 15.36253959
26.18547598
 21.63024974 14.7238506  30.46020989  7.27661426 14.76878417
18.41139775
 22.81431399 38.96698461 10.2302721  25.66905178 29.50616804
2.70999964
 30.2487692   8.79959365  3.622954   47.00192485 47.82385046
40.10670081
 15.38077426  5.22398397 34.01265959 22.03307634  6.41988423
24.73369965
  2.11799653 45.06005007 13.13123118 32.94708652 15.7291133
25.95536667
 27.26298078  9.50293283 48.0178456  38.47406284 46.54122647
44.34872452
 29.77539445 45.67619788  4.77344445 10.04912009  2.64996817
16.3975525
 19.50664998 13.74812586 41.10500513 17.93980739 14.21858547
27.06596221
  7.34681719 39.80238396  4.08917172 48.86705076 38.33231584
10.18324809
  0.70122033 40.45340907 35.12307134 36.21019476 38.28449074
4.0643374
 18.02385285  6.11709831 42.79169991 31.02194707 16.67081745
3.54966497
 15.69334571 16.39033725 36.23959446 31.72180217 43.97499611
23.60671501
  6.29993215 35.43656996 37.76986755 27.97793174 38.26961118
24.66590414
 26.08615666 21.41409941  1.67777458  5.72555249]
residuals
[ 0.08830886 -0.15398105  0.42643167 -3.85483586 -0.7260697
0.42747704
  2.57900635 -0.67024355 -1.49442634 -0.78260077  1.41954032
1.11918707
 -0.72424304  0.79159823 -0.06884071  1.67574663 -1.55453363 -
0.60297869
 -0.81721511 -3.08942349  0.7246754   0.22018931 -0.15132483 -
```

```
0.56247985
 -2.84155626 -0.54947719 -0.93201203 -1.5618844  -0.20801102
0.42062271
  3.9008452   0.07576816  0.14472643  0.29346018 -3.37979123
0.26013885
 -0.02962538  4.58610596 -0.18573019  0.57764303 -0.38739603 -
2.31221022
  1.88707516  1.9098361   1.38983179 -1.63974721  2.66202912 -
2.75566774
  1.24624737  4.1207012  -1.51968686 -0.85001714  0.63302334 -
0.61430831
 -2.98172237  0.55463982 -2.47342678  0.69720789 -2.22745219
2.96883285
 -1.63929208 -0.82479731  1.95890476 -2.56386969  0.28305989
2.68312745
 -3.51557241  0.67673279  0.15812605  2.04094181 -2.1939788  -
2.88837724
  0.61876866  0.9136317   0.72078155  0.93306006 -1.08102285
0.10240258
  0.48557852 -1.75234817  4.0950204   1.09062511 -2.5085232
0.94135976
 -2.09359296  1.44299806  2.55790562 -1.48529324  2.31239327
0.8295931
  1.32390047  4.01925537 -0.22139135 -1.42154419 -1.50028104 -
1.6077449
 -0.10371861  0.64525546  0.14656335  1.32338536]
```

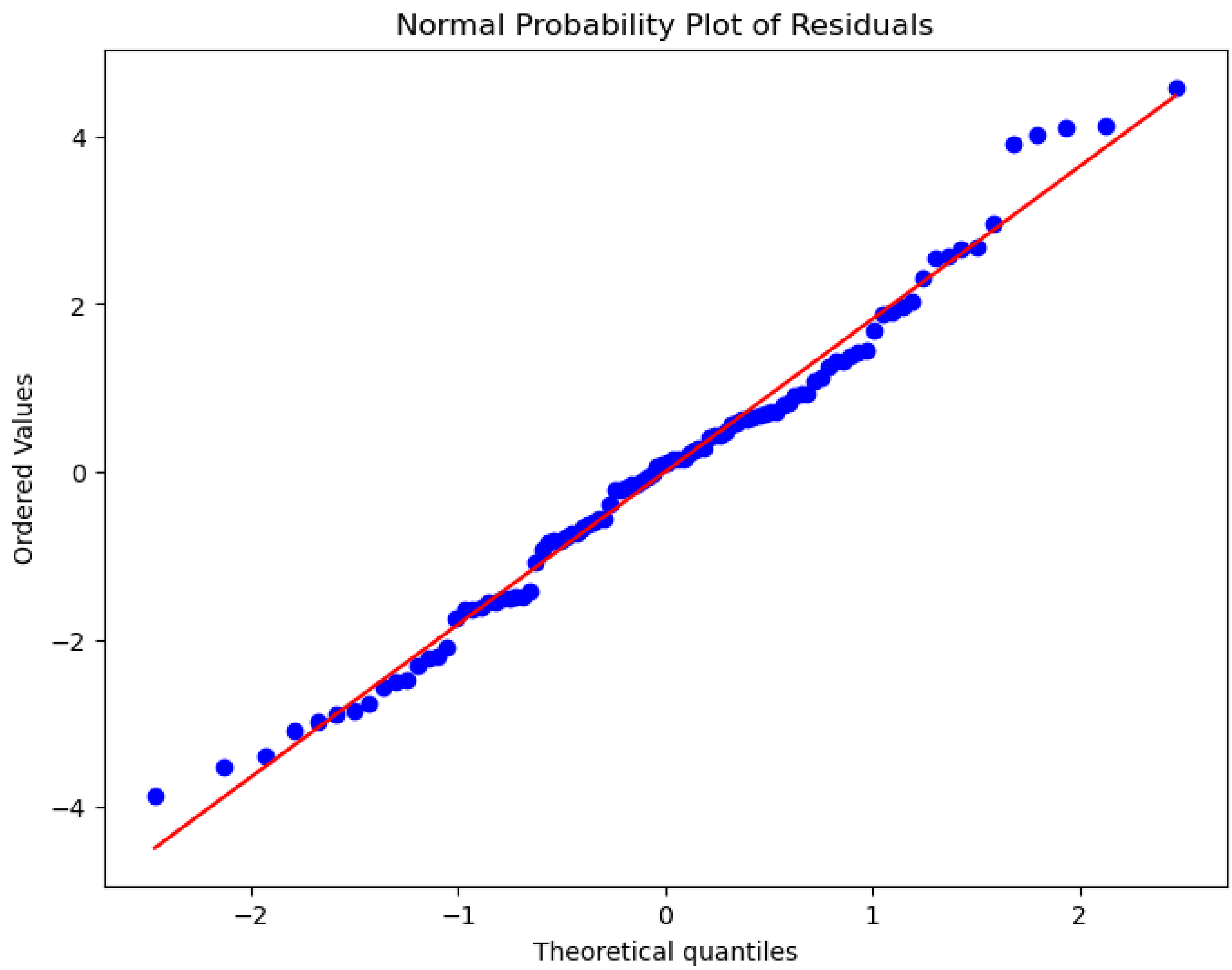
Residual plots of linear regression

```
plt.figure(figsize=(10, 6))
sns.residplot(x=y_pred, y=residuals, lowess=True, color='g',
line_kws={"color": "red"})
plt.axhline(y=0, color='r', linestyle='--')
plt.title("Residual Plot of Linear Regression")
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.show()
```



Normal probability plots

```
plt.figure(figsize=(8, 6))
probplot(residuals, dist="norm", plot=plt)

plt.title("Normal Probability Plot of Residuals")
plt.show()
```

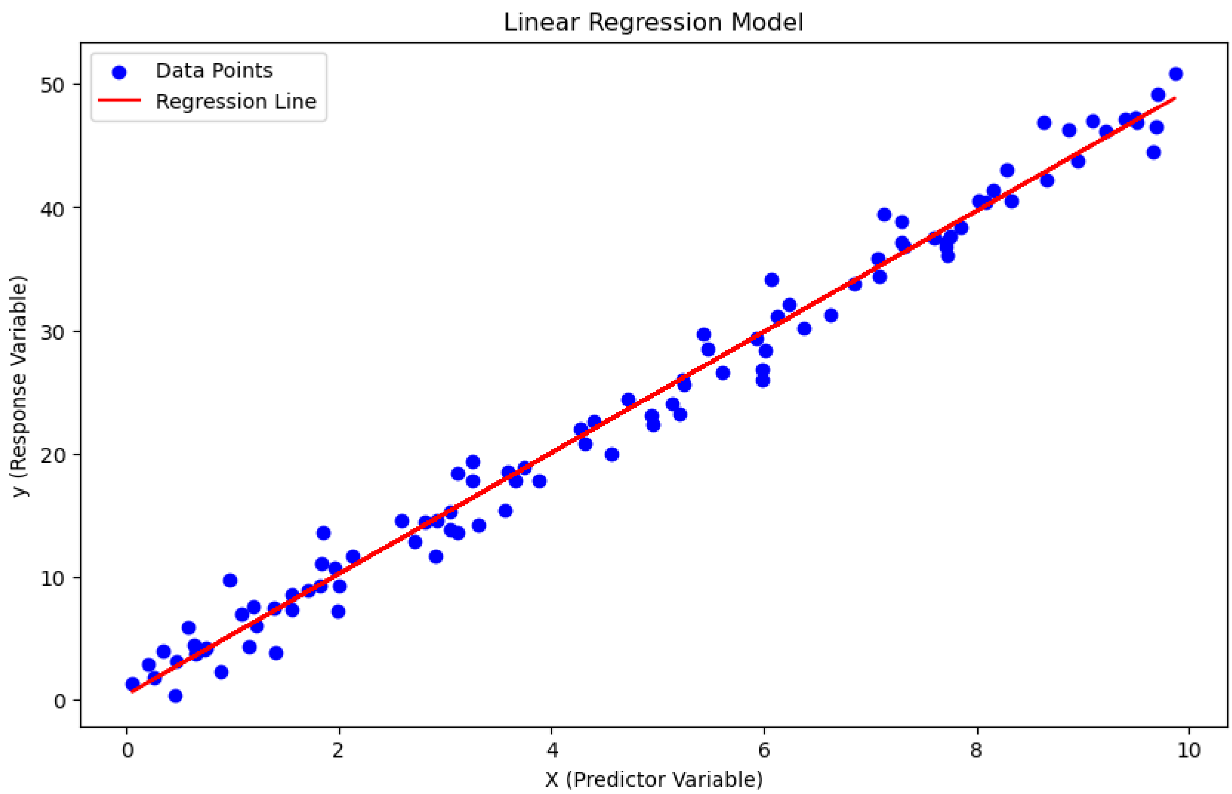


Empirical model of linear regression analysis

```
# Print coefficients and intercept
print("Intercept:", model.intercept_)
print("Slope:", model.coef_[0])

# Empirical model equation
print(f"Empirical model: y = {model.intercept_:.2f} +
{model.coef_[0]:.2f} * X")

# Mean Squared Error
mse = mean_squared_error(y, y_pred)
print("Mean Squared Error:", mse)

# Visualize the regression line
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.title("Linear Regression Model")
plt.xlabel("X (Predictor Variable)")
```

```
plt.ylabel("y (Response Variable)")
plt.legend()
plt.show()
```

```
Intercept: 0.43019231509349254
Slope: 4.9080453545753935
Empirical model: y = 0.43 + 4.91 * X
Mean Squared Error: 3.226338255868214
```



Week 9, 10, 11 Tasks

Task 1: Pandas DataFrame Operations

```
#Step 1: Importing a CSV File
import pandas as pd
# Import CSV file
df = pd.read_csv('iris.csv')
# Display the first few rows of the DataFrame
print(df.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
```

```
2            4.7          3.2           1.3          0.2  setosa
3            4.6          3.1           1.5          0.2  setosa
4            5.0          3.6           1.4          0.2  setosa
```

```python
#Step 2: Handling Missing Data
#(a) Detecting and Dropping/ Filling Missing Values
# Detect missing values
print(df.isnull().sum())

# Drop rows with missing values
df_dropped = df.dropna()
print(df_dropped)

# Fill missing values with a specific value or method
df_filled = df.fillna(method='ffill') # Forward fill
print(df_filled)
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
     sepal_length  sepal_width  petal_length  petal_width    species
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]
     sepal_length  sepal_width  petal_length  petal_width    species
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica
```

```
[150 rows x 5 columns]

C:\Users\B11202016\AppData\Local\Temp\ipykernel_5692\3643678138.py:11:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will
raise in a future version. Use obj.ffill() or obj.bfill() instead.
  df_filled = df.fillna(method='ffill') # Forward fill
```

```
#Step 3: Data Transformation Using apply() and map()
# Using apply() for row-wise or column-wise operations
#  Transform data using apply () and map() method.

import pandas as pd
# Sample DataFrame
data = {
    'A': [1, 2, 3],
    'B': [10, 20, 30],
    'C': [100, 200, 300]
}
df = pd.DataFrame(data)

# Define a function
def multiply_by_2(x):
    return x * 2

# Apply the function to the DataFrame
df_applied = df.apply(multiply_by_2)
print("DataFrame after apply():")
print(df_applied)
```

```
DataFrame after apply():
   A   B    C
0  2  20  200
1  4  40  400
2  6  60  600
```

```
data = {
    'A': [1, 2, 3],
    'B': [10, 20, 30],
    'C': [100, 200, 300]
}
df = pd.DataFrame(data)

df['A'] = df['A'].map(multiply_by_2)
print(df)
```

```
   A   B    C
0  2  10  100
1  4  20  200
2  6  30  300
```

(c) Detect and filter outliers.

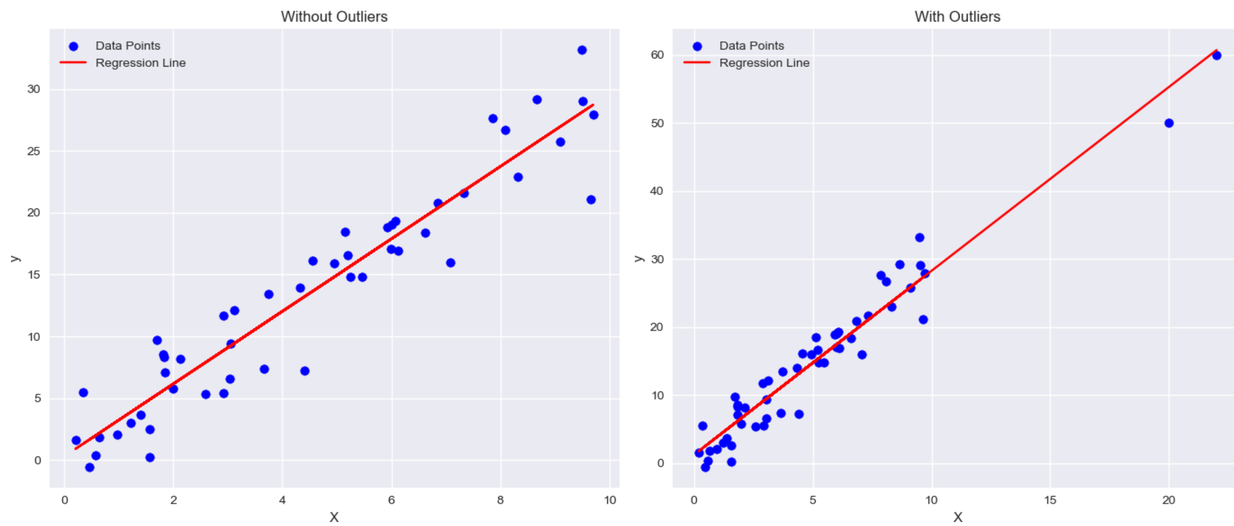
```
import pandas as pd

# Sample DataFrame
data = {
    'A': [1, 2, 3, 4, 5, 100, 6, 7, 8, 9]
}
df = pd.DataFrame(data)

# Function to detect and filter out outliers using IQR
def remove_outliers(df, column_name):
    Q1 = df[column_name].quantile(0.25)
    Q3 = df[column_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Filter out outliers
    filtered_df = df[(df[column_name] >= lower_bound) &
(df[column_name] <= upper_bound)]

    return filtered_df

# Detect and filter out outliers from column 'A'
filtered_df = remove_outliers(df, 'A')
print("Filtered DataFrame (outliers removed):")
print(filtered_df)
```

```
Filtered DataFrame (outliers removed):
   A
0  1
1  2
2  3
3  4
4  5
6  6
7  7
8  8
9  9
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Generate a simple linear dataset
np.random.seed(42)
X = np.random.rand(50, 1) * 10  # Random values between 0 and 10
```

```
y = 3 * X + np.random.randn(50, 1) * 3  # y = 3x + noise

# Add outliers
X_outliers = np.append(X, [[20], [22]], axis=0) # Adding extreme X
values
y_outliers = np.append(y, [[50], [60]], axis=0) # Adding extreme Y
values

# Create Linear Regression models
model_no_outliers = LinearRegression()
model_with_outliers = LinearRegression()

# Fit models
model_no_outliers.fit(X, y)
model_with_outliers.fit(X_outliers, y_outliers)

# Predictions
y_pred_no_outliers = model_no_outliers.predict(X)
y_pred_with_outliers = model_with_outliers.predict(X_outliers)

# Mean Squared Error (MSE)
mse_no_outliers = mean_squared_error(y, y_pred_no_outliers)
mse_with_outliers = mean_squared_error(y_outliers,
y_pred_with_outliers)

# Plotting
plt.figure(figsize=(14, 6))

# Without Outliers
plt.subplot(1, 2, 1)
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X, y_pred_no_outliers, color='red', label='Regression Line')
plt.title("Without Outliers")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.grid(True)

# With Outliers
plt.subplot(1, 2, 2)
plt.scatter(X_outliers, y_outliers, color='blue', label='Data Points')
plt.plot(X_outliers, y_pred_with_outliers, color='red',
label='Regression Line')
plt.title("With Outliers")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.grid(True)

plt.tight_layout()
```

```
plt.show()

# Print Mean Squared Errors
print(f"Mean Squared Error (Without Outliers): {mse_no_outliers:.2f}")
print(f"Mean Squared Error (With Outliers): {mse_with_outliers:.2f}")
```




```
Mean Squared Error (Without Outliers): 7.41
Mean Squared Error (With Outliers): 8.09
```

(d) Perform Vectorized String operations on Pandas Series. 2. Implement regularized Linear regression

```
import pandas as pd

# Sample data with strings
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva']}
df = pd.DataFrame(data)

# Convert strings to uppercase
df['Name_upper'] = df['Name'].str.upper()

# Convert strings to lowercase
df['Name_lower'] = df['Name'].str.lower()

# Replace a substring (e.g., replace 'a' with 'X')
df['Name_replaced'] = df['Name'].str.replace('a', 'X')

# Check if the string contains the letter 'e'
df['Has_e'] = df['Name'].str.contains('e')

# Extract the first 3 characters of each name
df['Name_first3'] = df['Name'].str[:3]
```

```
# Count the occurrences of the letter 'a' in each name
df['Count_a'] = df['Name'].str.count('a')

print("DataFrame after Vectorized String Operations:")
print(df)

DataFrame after Vectorized String Operations:
      Name Name_upper Name_lower Name_replaced Has_e Name_first3
Count_a
0    Alice      ALICE      alice         Alice   True         Ali
0
1      Bob        BOB        bob           Bob  False         Bob
0
2  Charlie    CHARLIE    charlie       ChXrlie   True         Cha
1
3    David      DAVID      david         DXvid  False         Dav
1
4      Eva        EVA        eva           EvX  False         Eva
1
```

e) Develop a model on logistic regression on any data set for prediction

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.datasets import load_iris

# Load the Iris dataset from sklearn
iris = load_iris()
X = iris.data # Features (sepal_length, sepal_width, petal_length,
petal_width)
y = iris.target  # Target labels (species)

# Convert to DataFrame for easier handling
df = pd.DataFrame(X, columns=iris.feature_names)
df['species'] = iris.target

# Display the first few rows
print("First 5 rows of the dataset:")
print(df.head())

# Split the data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize the features (important for logistic regression)
scaler = StandardScaler()
```

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

# Initialize the Logistic Regression model
logreg = LogisticRegression(multi_class='ovr', max_iter=200)

# Train the model
logreg.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = logreg.predict(X_test_scaled)

# Model Evaluation
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Predict on new data (example)
new_data = np.array([[5.4, 3.9, 1.7, 0.4],  # Example flower 1
                     [6.7, 3.1, 4.7, 1.5]])  # Example flower 2

new_data_scaled = scaler.transform(new_data)
predictions = logreg.predict(new_data_scaled)

# Print the species predictions (0 -> Setosa, 1 -> Versicolor, 2 ->
Virginica)
print("\nPredictions for new data:")
print(predictions)

First 5 rows of the dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)  \
0                5.1               3.5                1.4
0.2
1                4.9               3.0                1.4
0.2
2                4.7               3.2                1.3
0.2
3                4.6               3.1                1.5
0.2
4                5.0               3.6                1.4
0.2

   species
0        0
1        0
2        0
3        0
```

```
4         0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30


Confusion Matrix:


 [ 0  0 11]]

Predictions for new data:
[0 1]
```

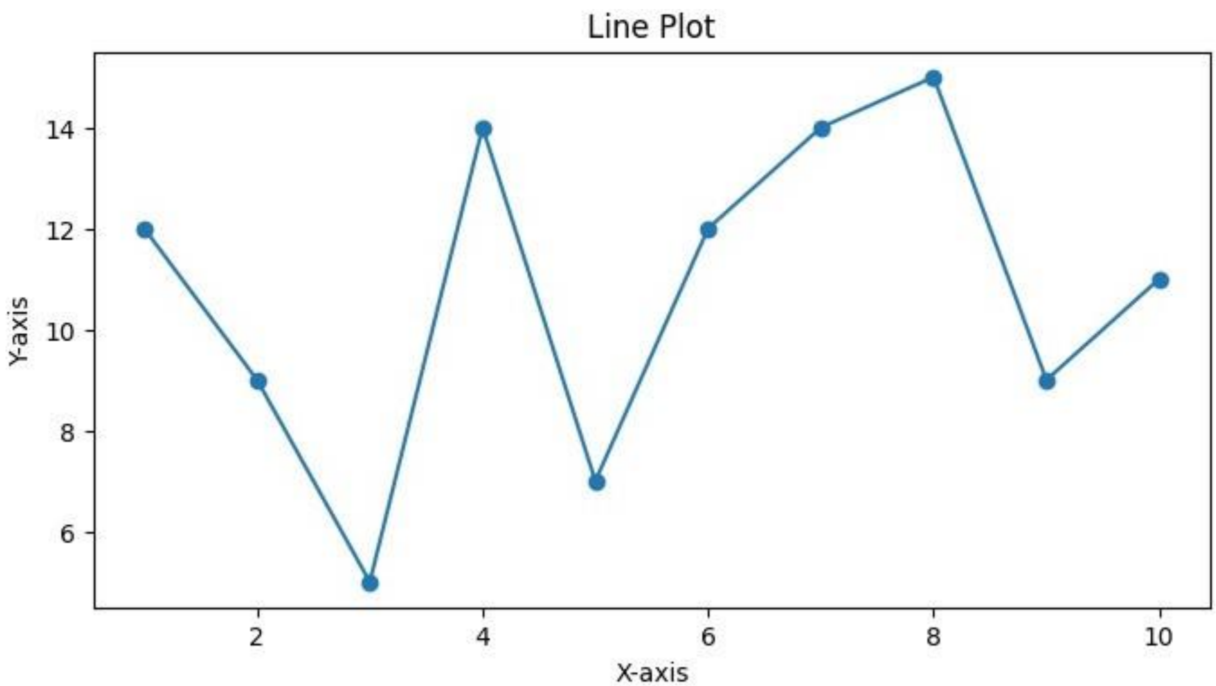
WEEK 12, 13 & 14:

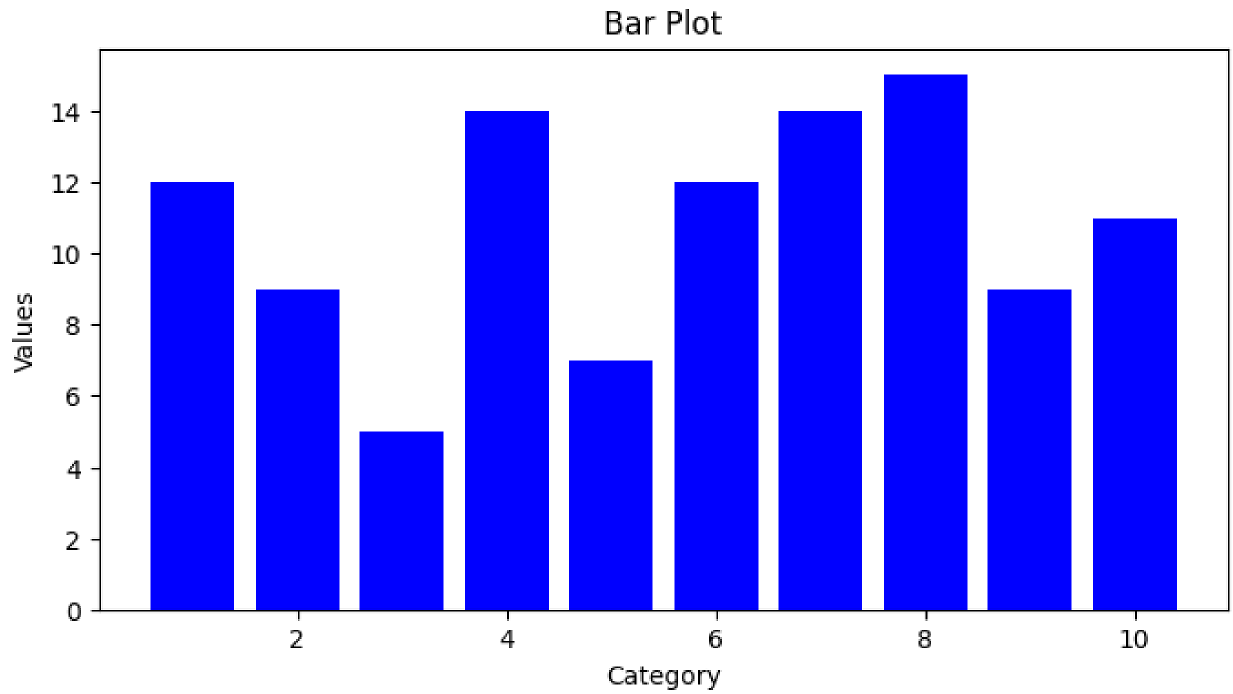
#Visualize data using Line Plots, Bar Plots, Histograms, Density Plots and Scatter Plots.

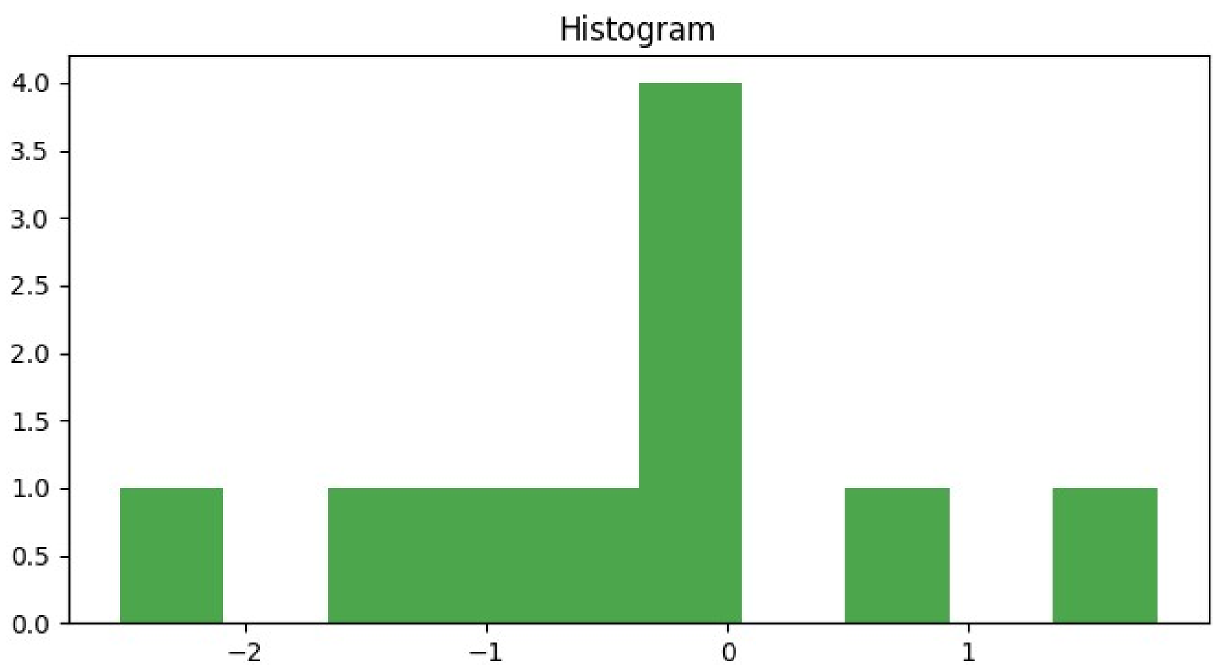
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from scipy.stats import skew

# Task 1: Visualize data using Line Plots, Bar Plots, Histograms,
Density Plots, and Scatter Plots
# Generating a sample dataset for demonstration
data = pd.DataFrame({
    'A': np.arange(1, 11),
    'B': np.random.randint(5, 20, 10),
    'C': np.random.randn(10)
})

# Line Plot
plt.figure(figsize=(8, 4))
plt.plot(data['A'], data['B'], marker='o')
plt.title('Line Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()

# Bar Plot
plt.figure(figsize=(8, 4))
plt.bar(data['A'], data['B'], color='blue')
plt.title('Bar Plot')
plt.xlabel('Category')
plt.ylabel('Values')
plt.show()

# Histogram
plt.figure(figsize=(8, 4))
plt.hist(data['C'], bins=10, color='green', alpha=0.7)
plt.title('Histogram')
plt.show()

# Density Plot
plt.figure(figsize=(8, 4))
sns.kdeplot(data['C'], fill=True, color='red')
plt.title('Density Plot')
plt.show()

# Scatter Plot
plt.figure(figsize=(8, 4))
```

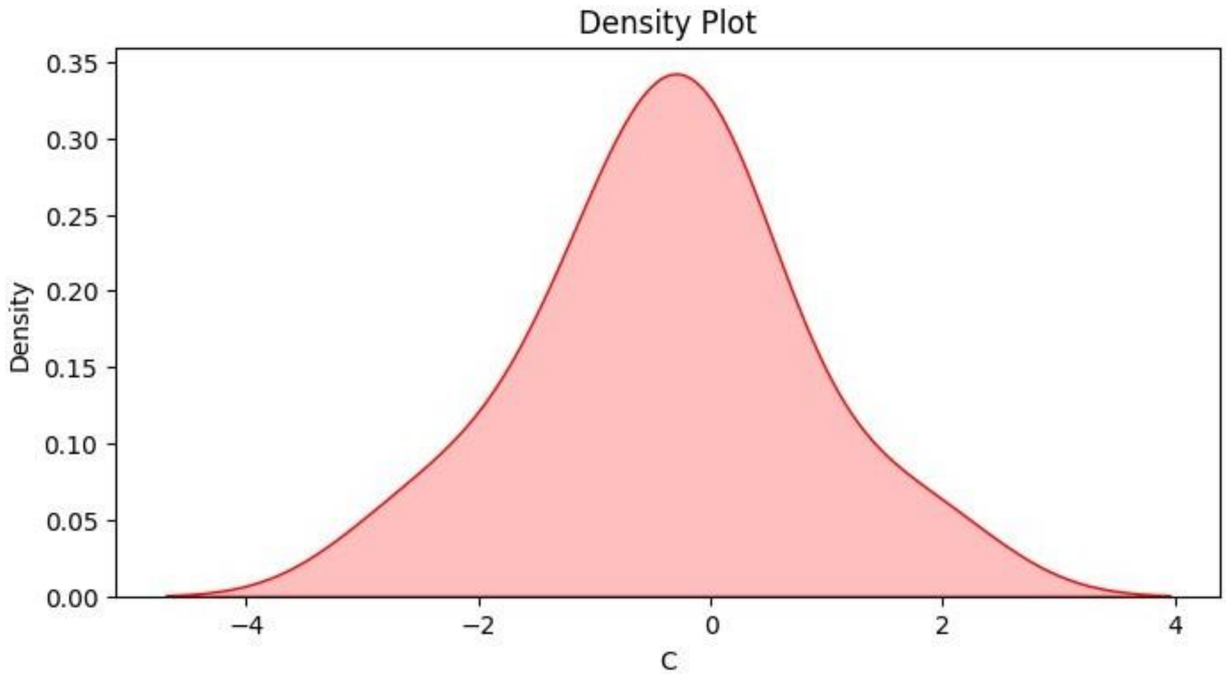
```python
plt.scatter(data['A'], data['C'], c='purple', alpha=0.8)
plt.title('Scatter Plot')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()

# Task 2: Download the House Pricing dataset and map values to 23
Aesthetics
# Assuming the dataset is loaded into `house_data`
# house_data = pd.read_csv('house_pricing.csv')

# Example of mapping to aesthetic dimensions using seaborn
# sns.scatterplot(data=house_data, x='GrLivArea', y='SalePrice',
hue='OverallQual', size='LotArea')

# Task 3: Use different color scales on the Rainfall Prediction
dataset
# Assuming `rainfall_data` dataset
# rainfall_data = pd.read_csv('rainfall_prediction.csv')

# Example visualization with color scales
# sns.heatmap(rainfall_data.corr(), cmap='coolwarm', annot=True)

# Task 4: Create different bar plots for variables in any dataset
# Sample demonstration with `data`
data.plot(kind='bar', figsize=(10, 6), title='Bar Plot for All
Columns')
plt.show()

# Task 5: Show an example of skewed data and removal of skewness
skewed_data = np.random.exponential(size=1000) # Generating skewed
data
plt.hist(skewed_data, bins=30, alpha=0.7)
plt.title('Skewed Data')
plt.show()

# Remove skewness using log transformation
transformed_data = np.log1p(skewed_data)
plt.hist(transformed_data, bins=30, alpha=0.7)
plt.title('Transformed (Less Skewed) Data')
plt.show()

# Task 6: Time Series Visualization
# Generating example time series data
time_data = pd.DataFrame({
    'Date': pd.date_range(start='2023-01-01', periods=100, freq='D'),
    'Sales': np.random.randint(100, 500, 100)
})
time_data.set_index('Date', inplace=True)
time_data['Sales'].plot(figsize=(12, 6), title='Time Series
```

```
Visualization')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.show()

# Task 7: Build a Scatterplot and suggest dimension reduction
# Sample scatterplot
plt.figure(figsize=(8, 4))
plt.scatter(data['A'], data['B'], c='orange', alpha=0.7)
plt.title('Scatter Plot')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# Dimensionality reduction using PCA
pca = PCA(n_components=1)
reduced_data = pca.fit_transform(data[['A', 'B']])
print("Reduced Dimension Data:\n", reduced_data)
```

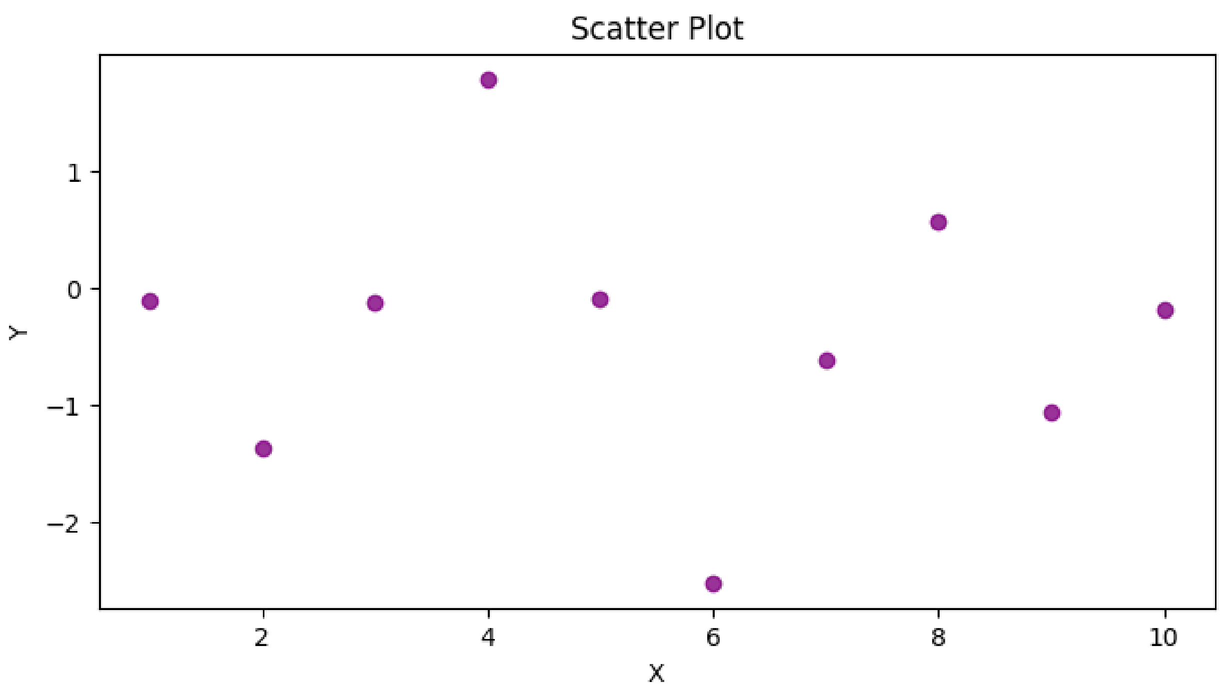



Bar Plot
Values
Category
0
2
4
6
8
10
12
14
2
4
6
8
10

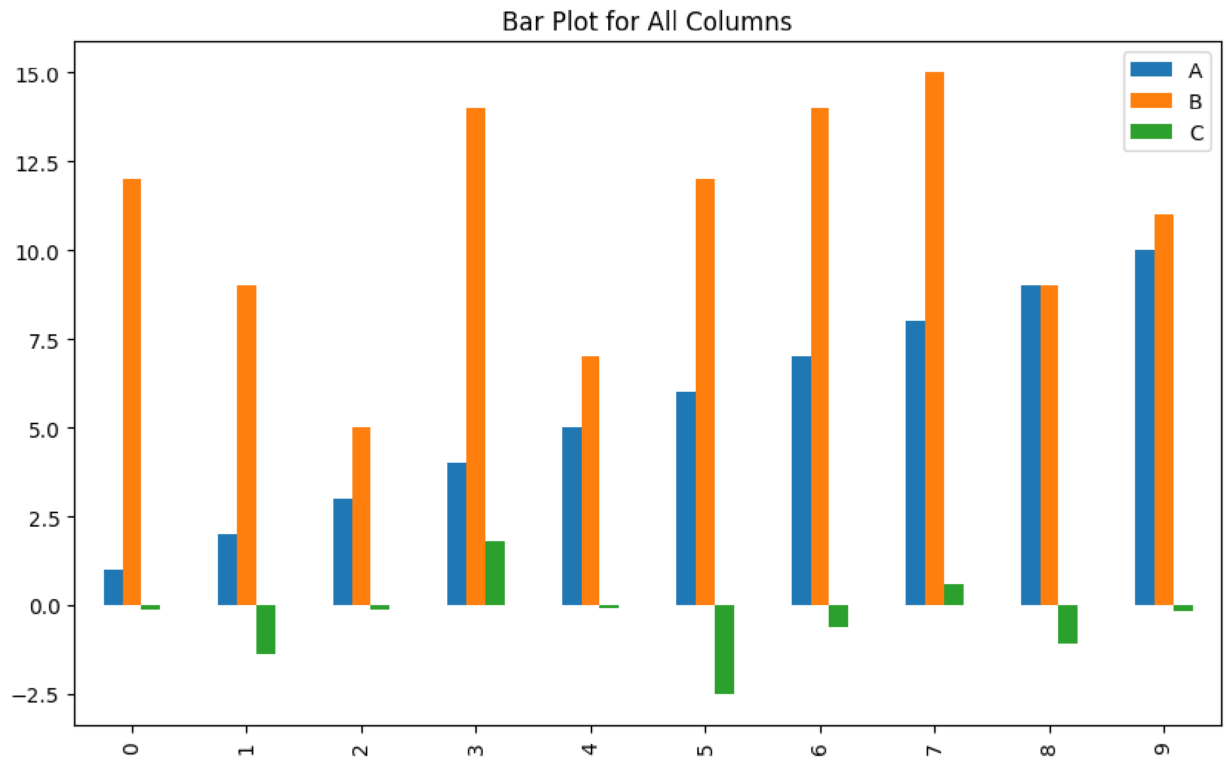

Histogram
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
−2
−1
0
1

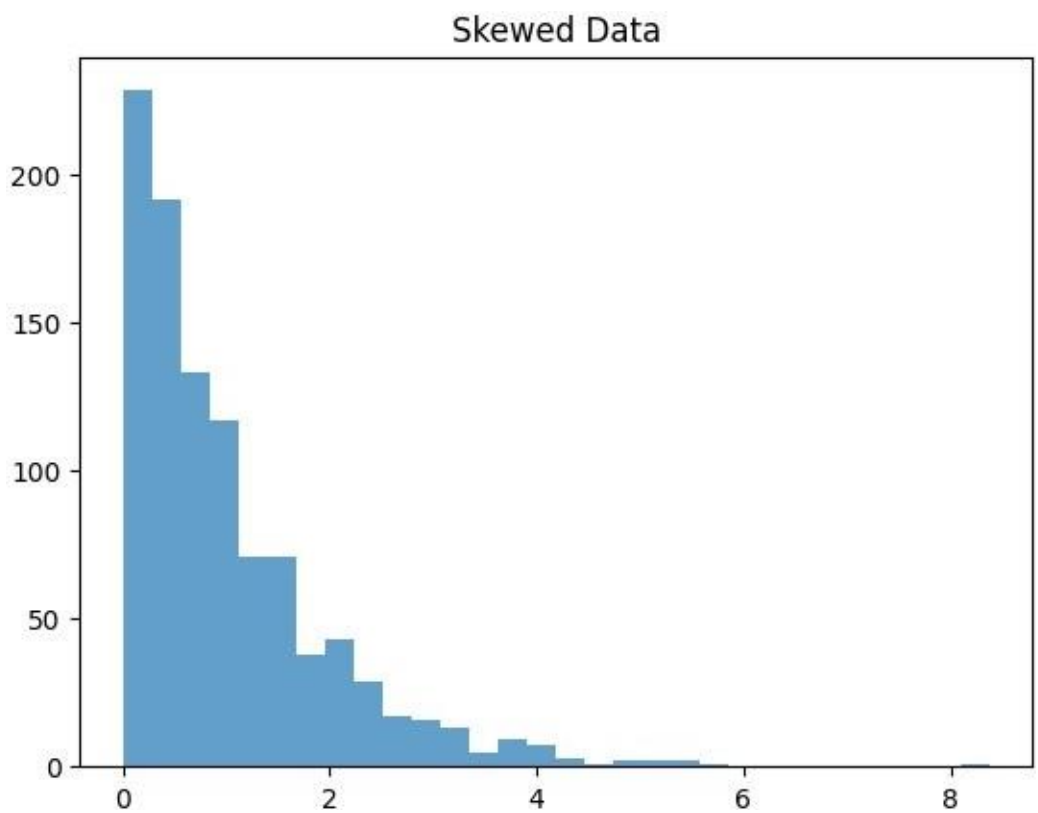


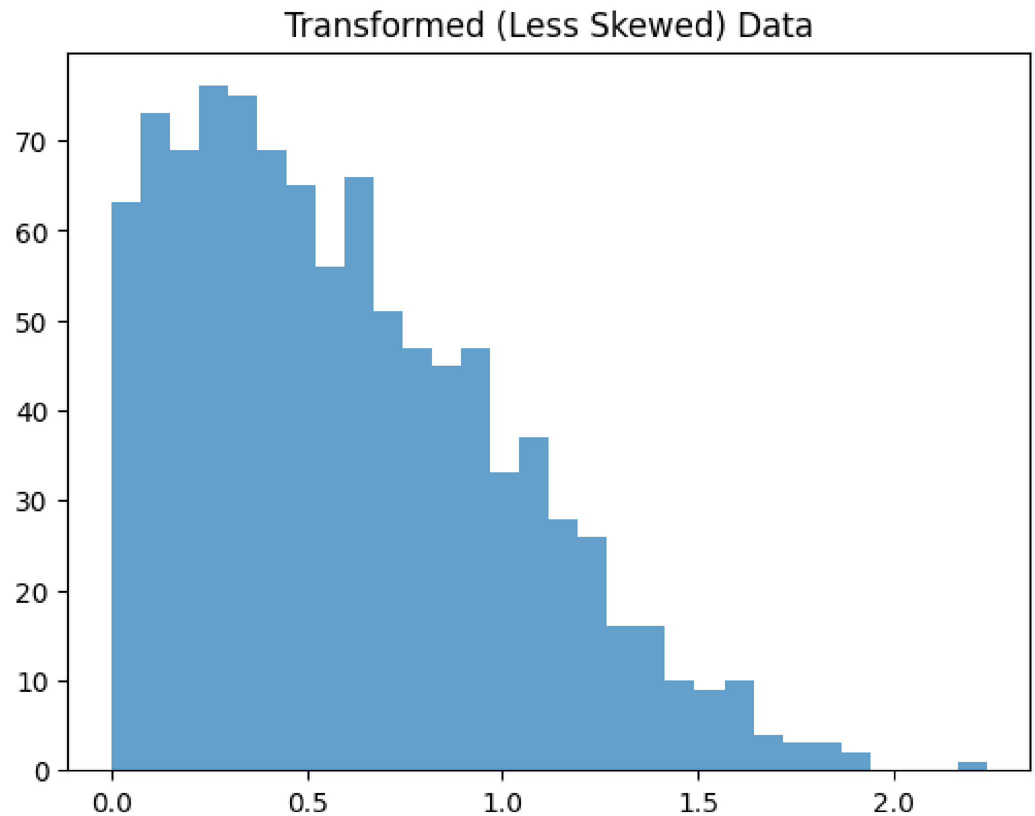
Density Plot
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
−4
−2
0
2
4
C

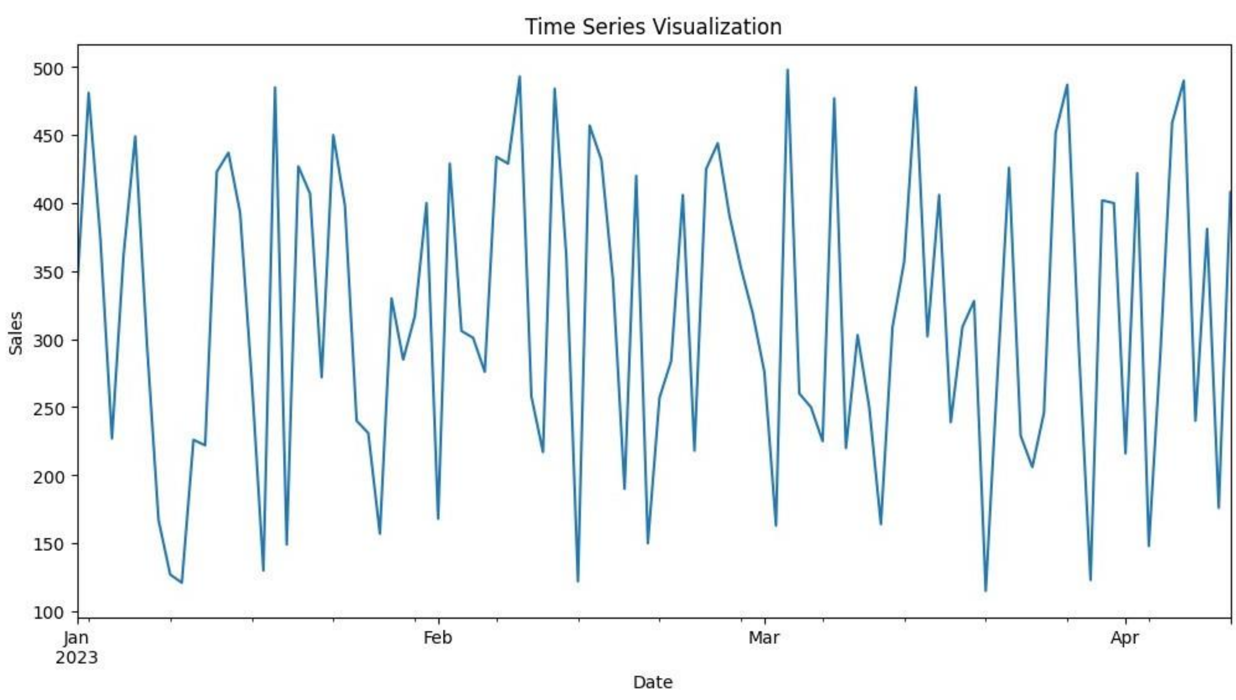


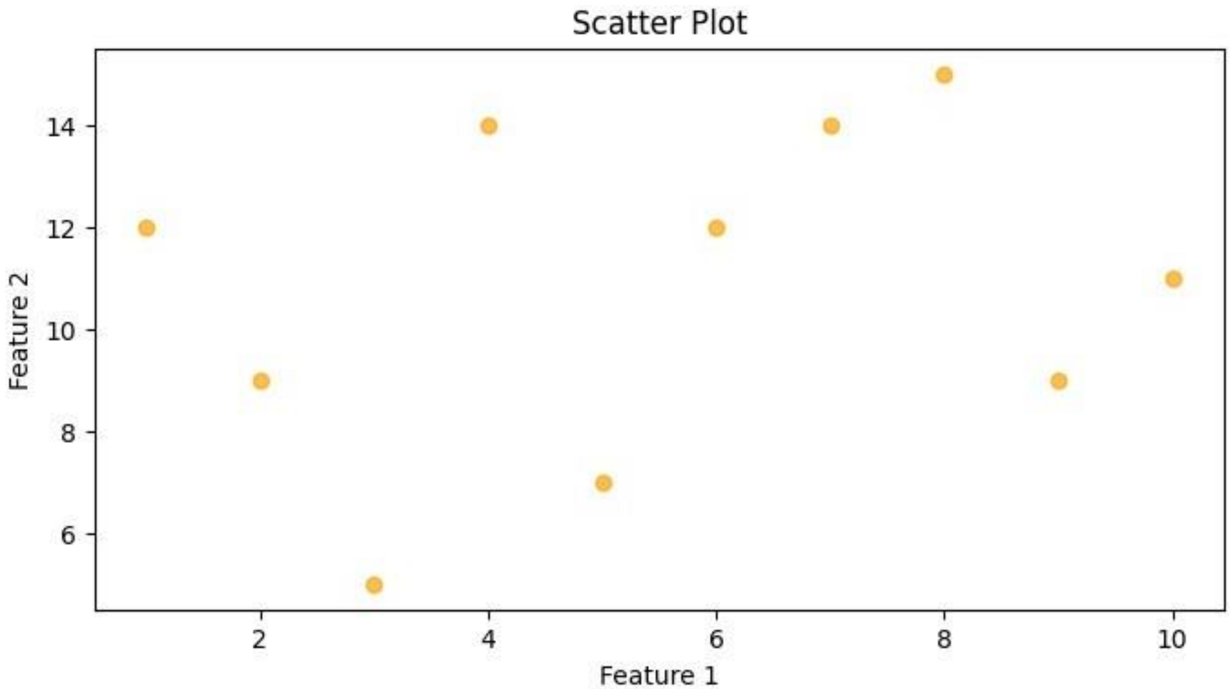
Scatter Plot
1
0
−1
−2
Y
2
4
6
8
10
X


Bar Plot for All Columns
A
B
C
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
0
1
2
3
4
5
6
7
8
9


Skewed Data
200
150
100
50
0
0
2
4
6
8


Transformed (Less Skewed) Data
70
60
50
40
30
20
10
0
0.0
0.5
1.0
1.5
2.0


Time Series Visualization
Sales
500
450
400
350
300
250
200
150
100
Jan
2023
Feb
Mar
Apr
Date



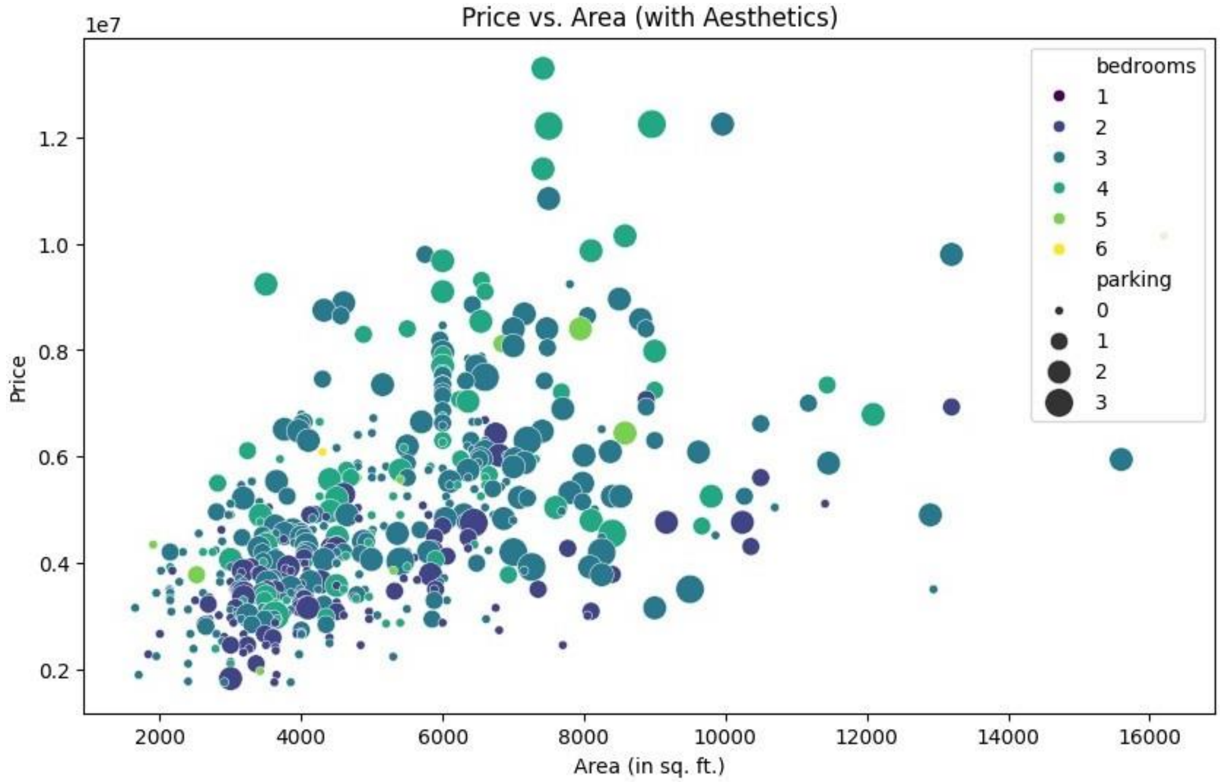

```
Reduced Dimension Data:
 [[-1.7537202 ]
 [-3.54544829]
 [-6.13526554]
 [ 1.65007624]
 [-3.33400848]
 [ 1.25897668]
 [ 3.45769436]
 [ 4.85832289]
 [ 0.67232733]
 [ 2.87104502]]
```

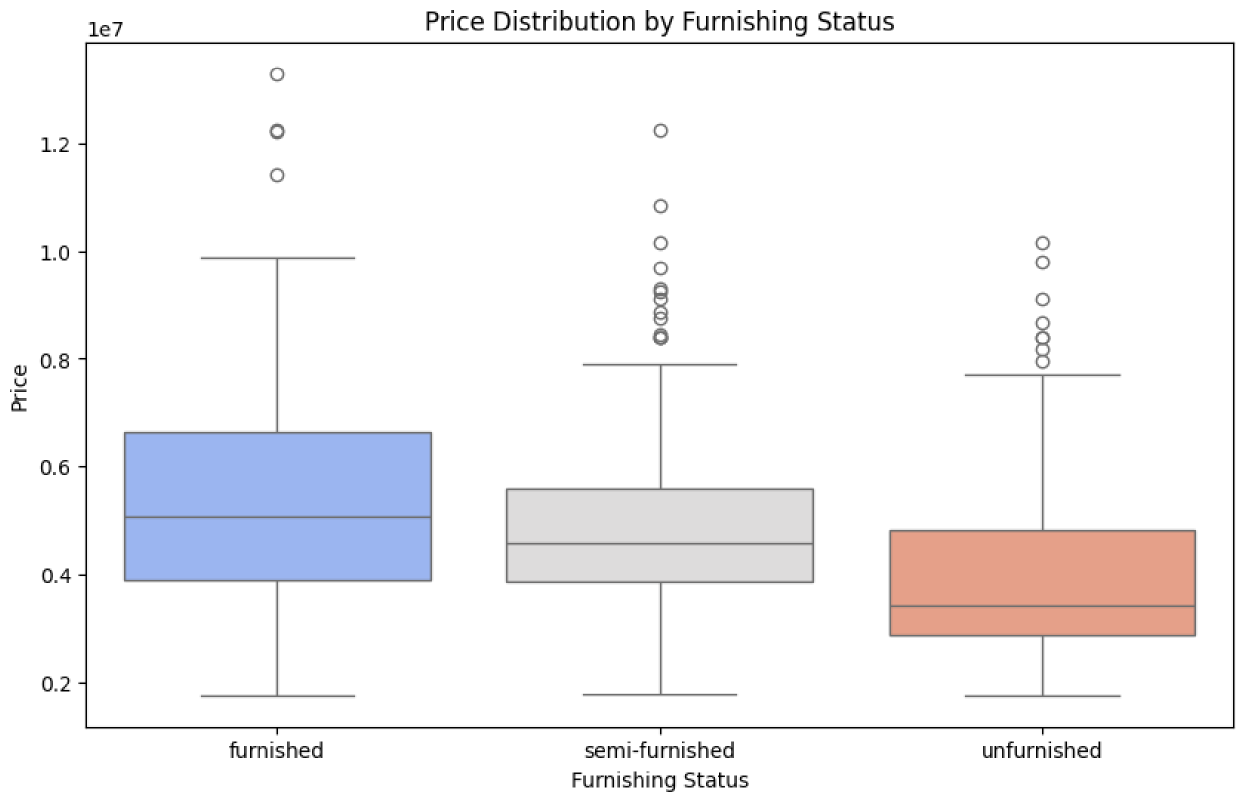
Download the House Pricing dataset from Kaggle and map the values to 23 Aesthetics

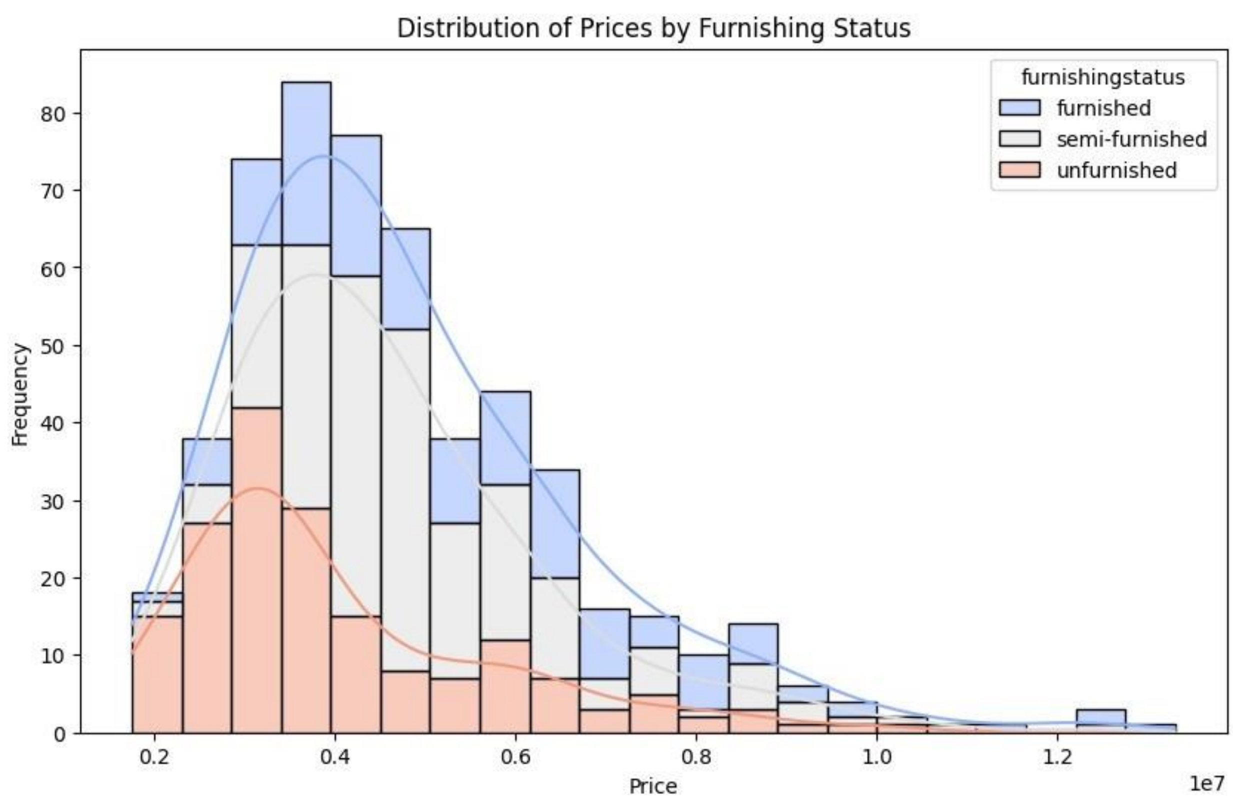
```
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

# Step 1: Load the dataset (assuming it's in a CSV format)
# Replace 'your_dataset.csv' with your actual file path
df = pd.read_csv('Housing.csv')

# Step 2: Explore the data
# Display the first few rows of the dataset to understand its
structure
df.head(3)
```

```
# Step 3: Data Preprocessing (if needed)
# Check for missing values or data types


# Step 4: Correlation Heatmap (for numerical columns)
# Compute the correlation matrix for numerical features
# corr_matrix = df.corr()

# # # Plot the heatmap
# plt.figure(figsize=(12, 8))
# sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt='.2f',
linewidths=0.5)
# plt.title('Correlation Heatmap of Housing Features')
# plt.show()

# Step 5: Scatter Plot (Mapping Aesthetics)
# Example: Scatter plot with price vs. area, using color for the
number of bedrooms and size for parking
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='area', y='price', hue='bedrooms',
size='parking', sizes=(20, 200), palette="viridis")
plt.title('Price vs. Area (with Aesthetics)')
plt.xlabel('Area (in sq. ft.)')
plt.ylabel('Price')
plt.show()

# Step 6: Boxplot to visualize categorical features (e.g., Furnishing
Status vs. Price)
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='furnishingstatus', y='price',
palette='coolwarm')
plt.title('Price Distribution by Furnishing Status')
plt.xlabel('Furnishing Status')
plt.ylabel('Price')
plt.show()

# Step 7: Interactive Plot with Plotly
# Example: Interactive scatter plot showing Price vs. Area, colored by
number of Bedrooms
fig = px.scatter(df, x="area", y="price", color="bedrooms",
size="parking", hover_data=["furnishingstatus", "stories"])
fig.update_layout(
    title="Interactive Plot: Price vs. Area",
    xaxis_title="Area (in sq. ft.)",
    yaxis_title="Price",
    template="plotly_dark"
)
fig.show()
```

```
# Step 8: Additional Exploration - Distribution of Prices by
Furnishing Status
plt.figure(figsize=(10, 6))
sns.histplot(df, x="price", hue="furnishingstatus", multiple="stack",
kde=True, palette="coolwarm")
plt.title('Distribution of Prices by Furnishing Status')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

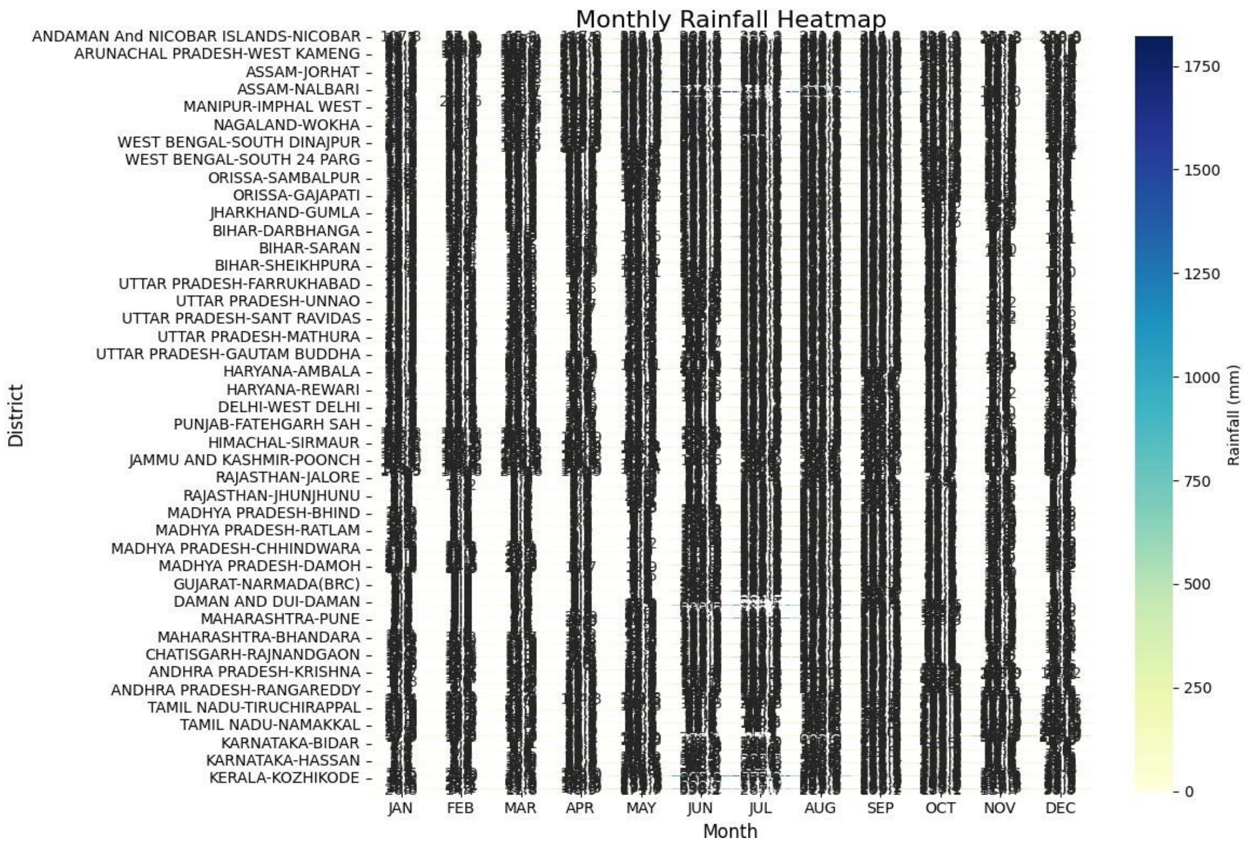






. Use different Color scales on the Rainfall Prediction dataset.

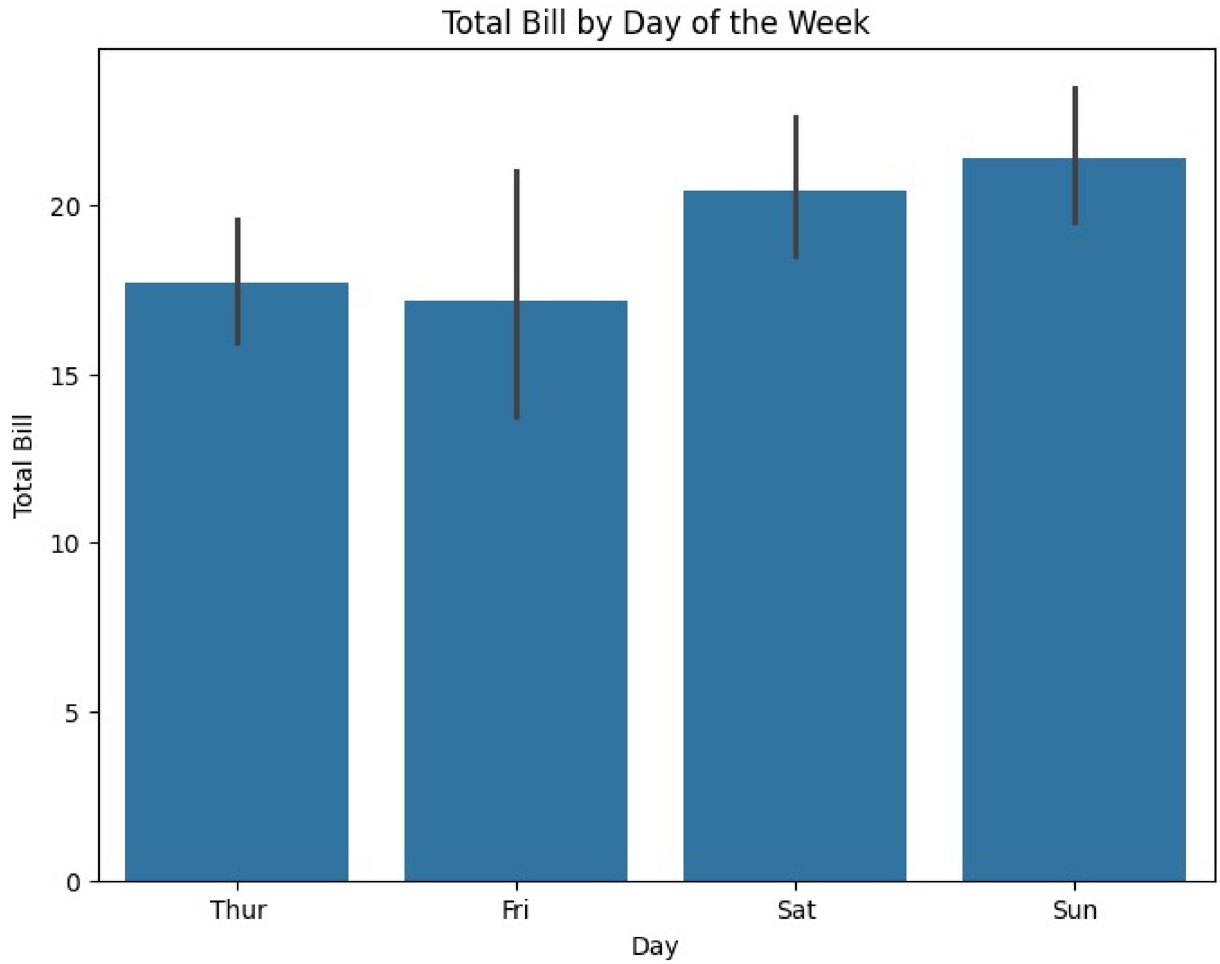
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('/content/district wise rainfall normal.csv')

# Convert the monthly rainfall columns to numeric (if not already)
columns_to_convert = ['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
'AUG', 'SEP', 'OCT', 'NOV', 'DEC']
df[columns_to_convert] = df[columns_to_convert].apply(pd.to_numeric,
errors='coerce')

# Set 'STATE_UT_NAME' and 'DISTRICT' as index for better readability
df.set_index(['STATE_UT_NAME', 'DISTRICT'], inplace=True)

# Select only the columns for the months
monthly_data = df[columns_to_convert]

# Plot the heatmap with fewer columns
plt.figure(figsize=(12, 8))
sns.heatmap(monthly_data, annot=True, cmap='YlGnBu', linewidths=0.5,
fmt='.1f', cbar_kws={'label': 'Rainfall (mm)'})

# Add title and labels
plt.title('Monthly Rainfall Heatmap', fontsize=16)
plt.xlabel('Month', fontsize=12)
plt.ylabel('District', fontsize=12)

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

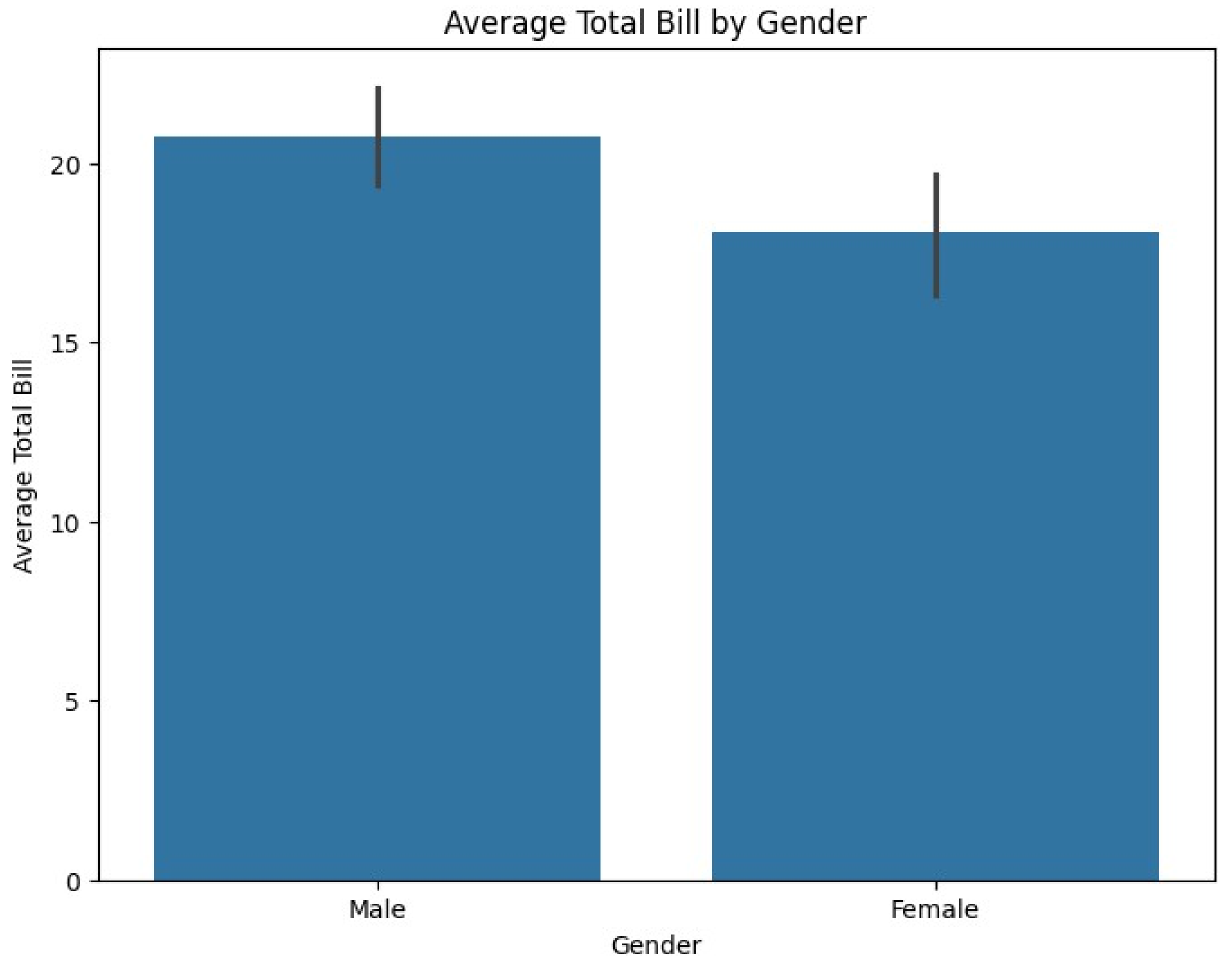


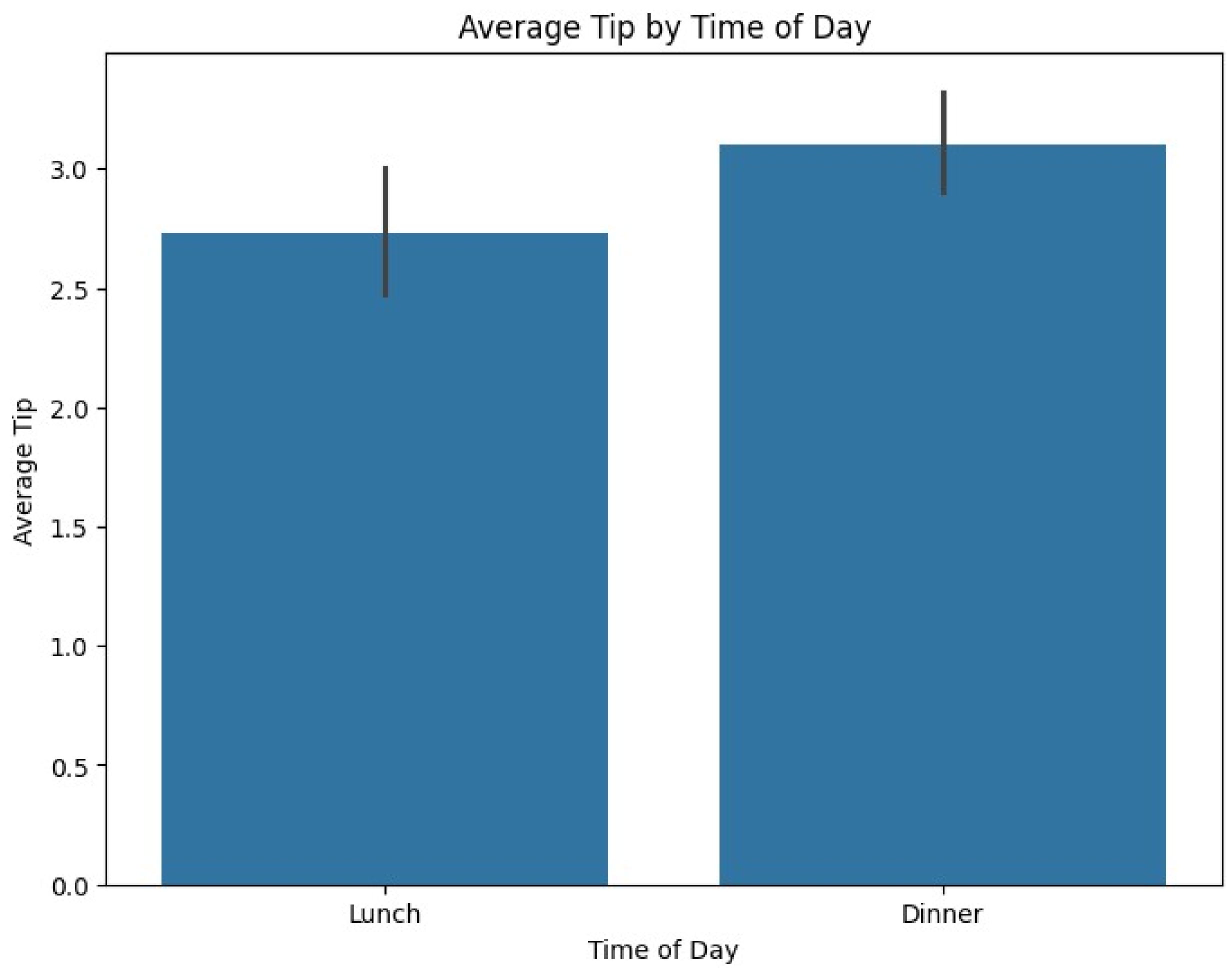
Create different bar plots for variables in any dataset.

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

# Load Seaborn's built-in 'tips' dataset
df = sns.load_dataset('tips')

# Plot 1: Bar plot of total bill by day of the week
plt.figure(figsize=(8,6))
sns.barplot(x='day', y='total_bill', data=df)
plt.title('Total Bill by Day of the Week')
plt.xlabel('Day')
plt.ylabel('Total Bill')
plt.show()

# Plot 2: Bar plot of average total bill by gender
plt.figure(figsize=(8,6))
sns.barplot(x='sex', y='total_bill', data=df)
plt.title('Average Total Bill by Gender')
plt.xlabel('Gender')
plt.ylabel('Average Total Bill')
plt.show()
```

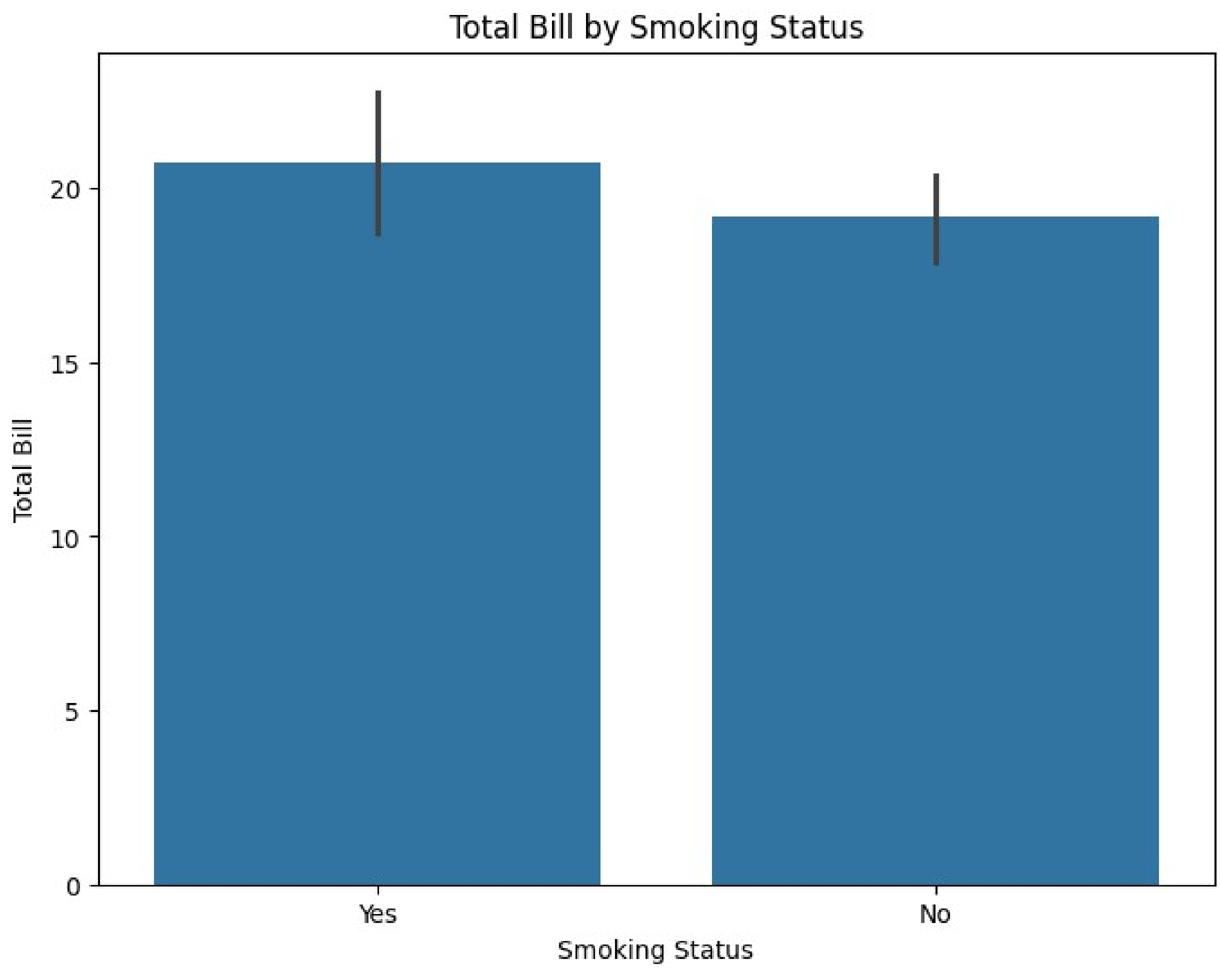
```python
# Plot 3: Bar plot of average tip by time of day (Lunch/Dinner)
plt.figure(figsize=(8,6))
sns.barplot(x='time', y='tip', data=df)
plt.title('Average Tip by Time of Day')
plt.xlabel('Time of Day')
plt.ylabel('Average Tip')
plt.show()

# Plot 4: Bar plot of total bill by smoking status
plt.figure(figsize=(8,6))
sns.barplot(x='smoker', y='total_bill', data=df)
plt.title('Total Bill by Smoking Status')
plt.xlabel('Smoking Status')
plt.ylabel('Total Bill')
plt.show()

# Plot 5: Count plot of number of people by day
plt.figure(figsize=(8,6))
sns.countplot(x='day', data=df)
plt.title('Number of People by Day of the Week')
plt.xlabel('Day')
plt.ylabel('Count of People')
plt.show()
```

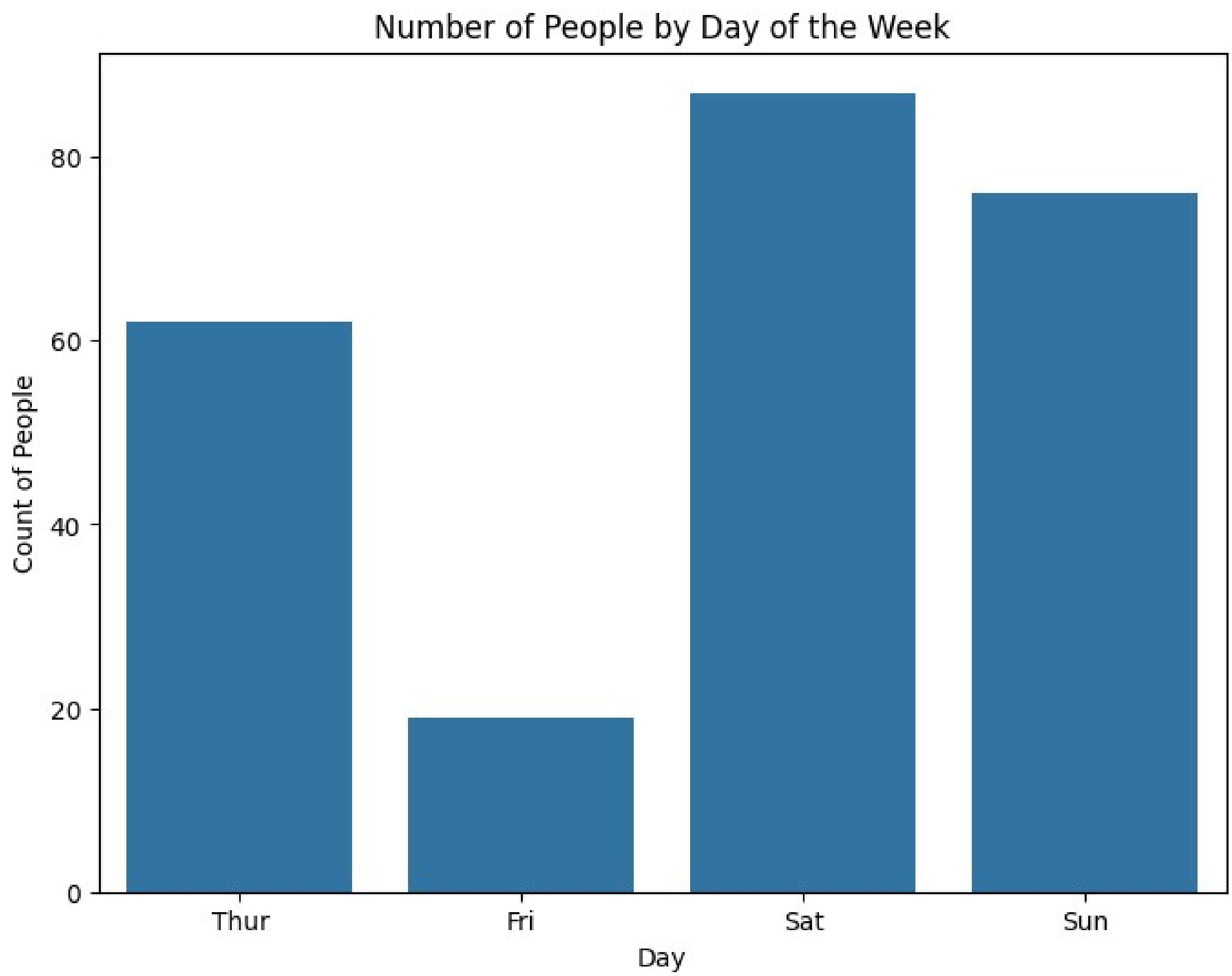

Total Bill by Day of the Week
Total Bill
0
5
10
15
20
Thur
Fri
Sat
Sun
Day


Average Total Bill by Gender
Average Total Bill
0
5
10
15
20
Male
Female
Gender



Average Tip by Time of Day
Average Tip
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Lunch
Dinner
Time of Day

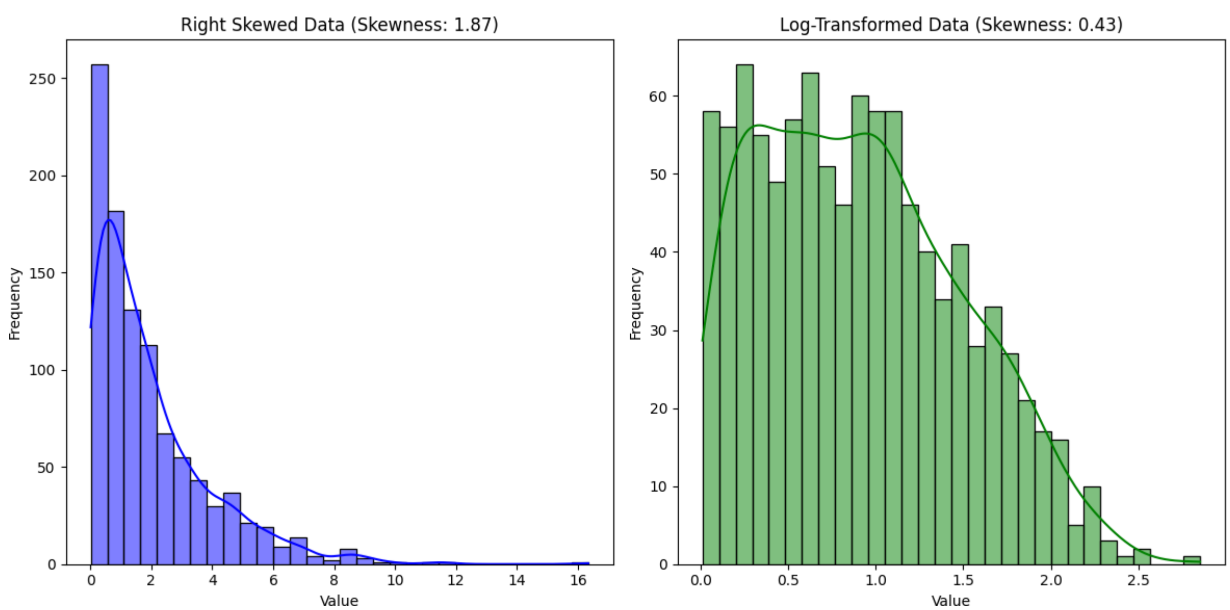

Total Bill by Smoking Status
Total Bill
0
5
10
15
20
Yes
No
Smoking Status



Show an example of Skewed data and removal of skewedness.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew

# Step 1: Generate Skewed Data (Right Skewed)
np.random.seed(42)
right_skewed_data = np.random.exponential(scale=2, size=1000)

# Step 2: Visualize the Right Skewed Data
plt.figure(figsize=(12,6))

# Histogram for Right Skewed Data
plt.subplot(1, 2, 1)
sns.histplot(right_skewed_data, kde=True, color='blue', bins=30)
plt.title(f"Right Skewed Data (Skewness:
{skew(right_skewed_data):.2f})")
plt.xlabel('Value')
```

```
plt.ylabel('Frequency')

# Step 3: Apply Log Transformation to Remove Skewness
log_transformed_data = np.log1p(right_skewed_data)

# Step 4: Visualize the Log-Transformed Data
plt.subplot(1, 2, 2)
sns.histplot(log_transformed_data, kde=True, color='green', bins=30)
plt.title(f"Log-Transformed Data (Skewness:
{skew(log_transformed_data):.2f})")
plt.xlabel('Value')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()




sqrt_transformed_data = np.sqrt(right_skewed_data)


from scipy.stats import boxcox
boxcox_transformed_data, _ = boxcox(right_skewed_data + 1) # Adding 1
to avoid issues with zero values.
```

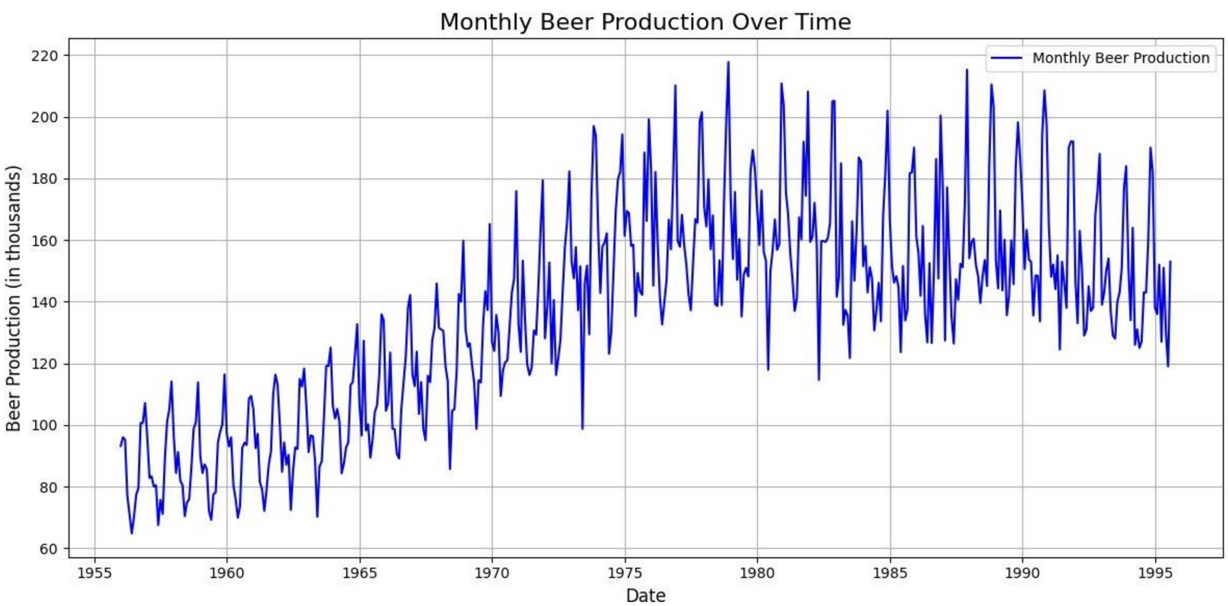


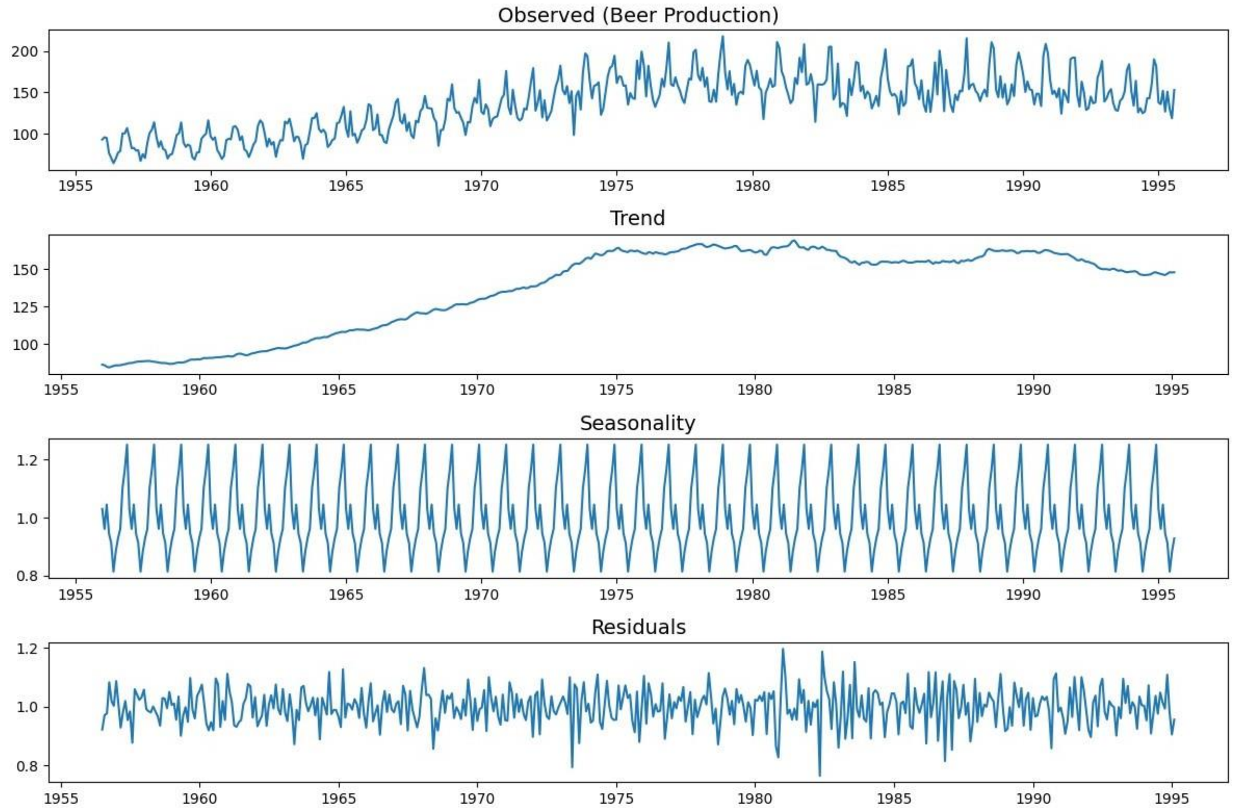
For a sales dataset do Time Series visualization.

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

```python
from statsmodels.tsa.seasonal import seasonal_decompose

# Load the dataset (replace 'beer_production.csv' with the actual path
to your dataset)
df = pd.read_csv('/content/monthly-beer-production-in-austr.csv')

# Convert the 'Month' column to datetime
df['Month'] = pd.to_datetime(df['Month'], format='%Y-%m')

# Set 'Month' as the index for time series analysis
df.set_index('Month', inplace=True)

# Display the first few rows to verify the data
print(df.head())

# 1. Visualizing the Time Series (Beer Production over Time)
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Monthly beer production'], label='Monthly Beer
Production', color='blue')
plt.title('Monthly Beer Production Over Time', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Beer Production (in thousands)', fontsize=12)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# 2. Decompose the Time Series to observe trend, seasonality, and
residuals
decomposition = seasonal_decompose(df['Monthly beer production'],
model='multiplicative', period=12)

# Plot the decomposition results
plt.figure(figsize=(12, 8))

plt.subplot(411)
plt.plot(decomposition.observed)
plt.title('Observed (Beer Production)', fontsize=14)

plt.subplot(412)
plt.plot(decomposition.trend)
plt.title('Trend', fontsize=14)

plt.subplot(413)
plt.plot(decomposition.seasonal)
plt.title('Seasonality', fontsize=14)

plt.subplot(414)
plt.plot(decomposition.resid)
plt.title('Residuals', fontsize=14)
```

```
plt.tight_layout()
plt.show()

            Monthly beer production
Month
1956-01-01                     93.2
1956-02-01                     96.0
1956-03-01                     95.2
1956-04-01                     77.1
1956-05-01                     70.9
```

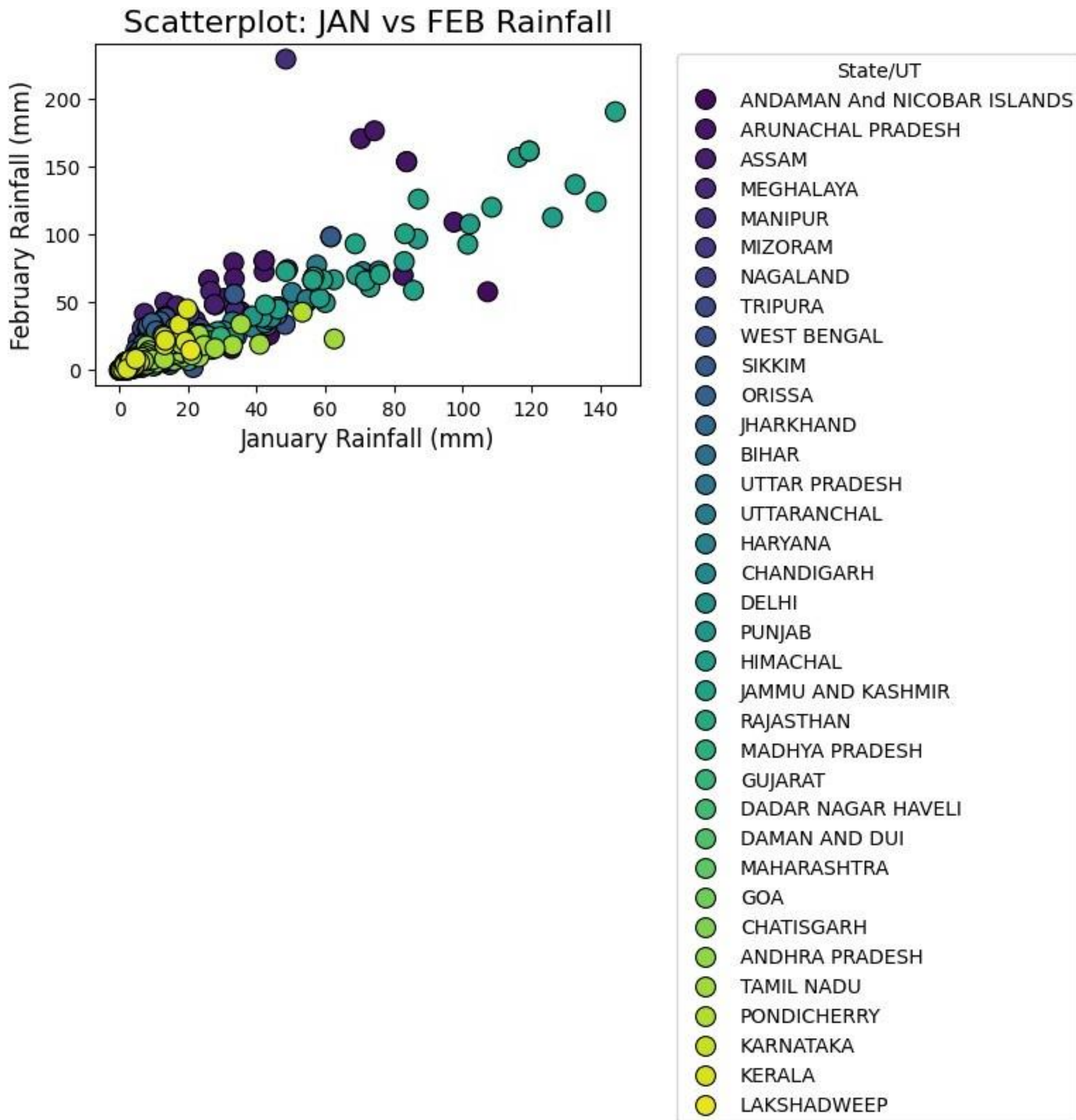


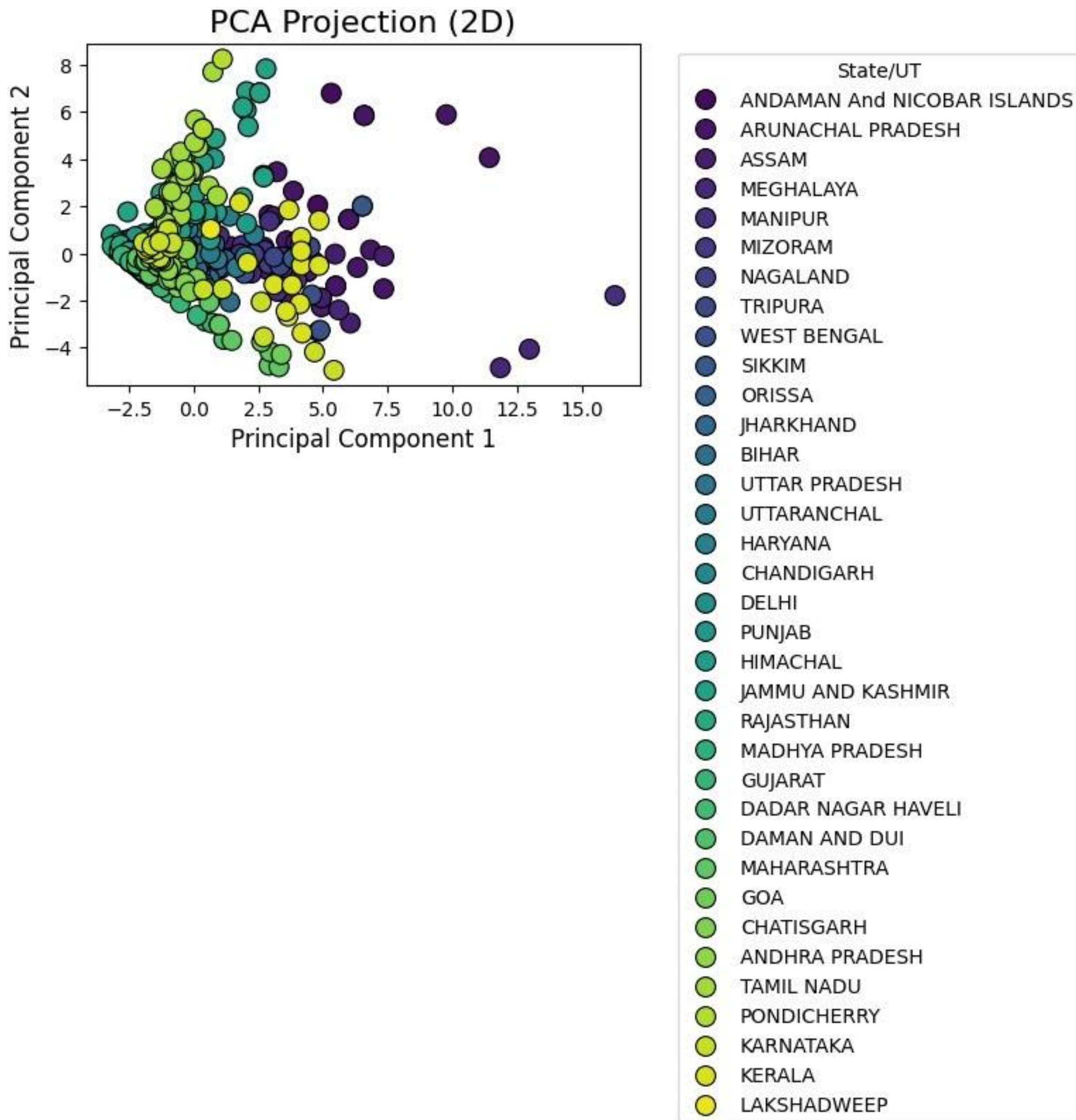


Build a Scatterplot and suggest dimension reduction.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv('/content/district wise rainfall normal.csv')

# Convert the rainfall columns to numeric (if not already)
columns_to_convert = ['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
'AUG', 'SEP', 'OCT', 'NOV', 'DEC']
df[columns_to_convert] = df[columns_to_convert].apply(pd.to_numeric,
errors='coerce')

# Scatterplot: Plot JAN vs FEB to see correlation
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['JAN'], y=df['FEB'], hue=df['STATE_UT_NAME'],
palette='viridis', s=100, edgecolor='k')
plt.title('Scatterplot: JAN vs FEB Rainfall', fontsize=16)
plt.xlabel('January Rainfall (mm)', fontsize=12)
plt.ylabel('February Rainfall (mm)', fontsize=12)
plt.legend(title='State/UT', bbox_to_anchor=(1.05, 1), loc='upper
```

```
left')
plt.tight_layout()
plt.show()

# Dimensionality Reduction: Applying PCA

# Drop non-numeric columns like 'STATE_UT_NAME' and 'DISTRICT'
df_numeric = df[columns_to_convert]

# Standardize the data before applying PCA
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_numeric)

# Apply PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
pca_components = pca.fit_transform(df_scaled)

# Create a DataFrame with the PCA results
pca_df = pd.DataFrame(data=pca_components, columns=['PC1', 'PC2'])

# Plot the 2D PCA projection
plt.figure(figsize=(8, 6))
sns.scatterplot(x=pca_df['PC1'], y=pca_df['PC2'],
hue=df['STATE_UT_NAME'], palette='viridis', s=100, edgecolor='k')
plt.title('PCA Projection (2D)', fontsize=16)
plt.xlabel('Principal Component 1', fontsize=12)
plt.ylabel('Principal Component 2', fontsize=12)
plt.legend(title='State/UT', bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.tight_layout()
plt.show()
```


Scatterplot: JAN vs FEB Rainfall
February Rainfall (mm)
January Rainfall (mm)
0
50
100
150
200
0
20
40
60
80
100
120
140
State/UT
ANDAMAN And NICOBAR ISLANDS
ARUNACHAL PRADESH
ASSAM
MEGHALAYA
MANIPUR
MIZORAM
NAGALAND
TRIPURA
WEST BENGAL
SIKKIM
ORISSA
JHARKHAND
BIHAR
UTTAR PRADESH
UTTARANCHAL
HARYANA
CHANDIGARH
DELHI
PUNJAB
HIMACHAL
JAMMU AND KASHMIR
RAJASTHAN
MADHYA PRADESH
GUJARAT
DADAR NAGAR HAVELI
DAMAN AND DUI
MAHARASHTRA
GOA
CHATISGARH
ANDHRA PRADESH
TAMIL NADU
PONDICHERRY
KARNATAKA
KERALA
LAKSHADWEEP



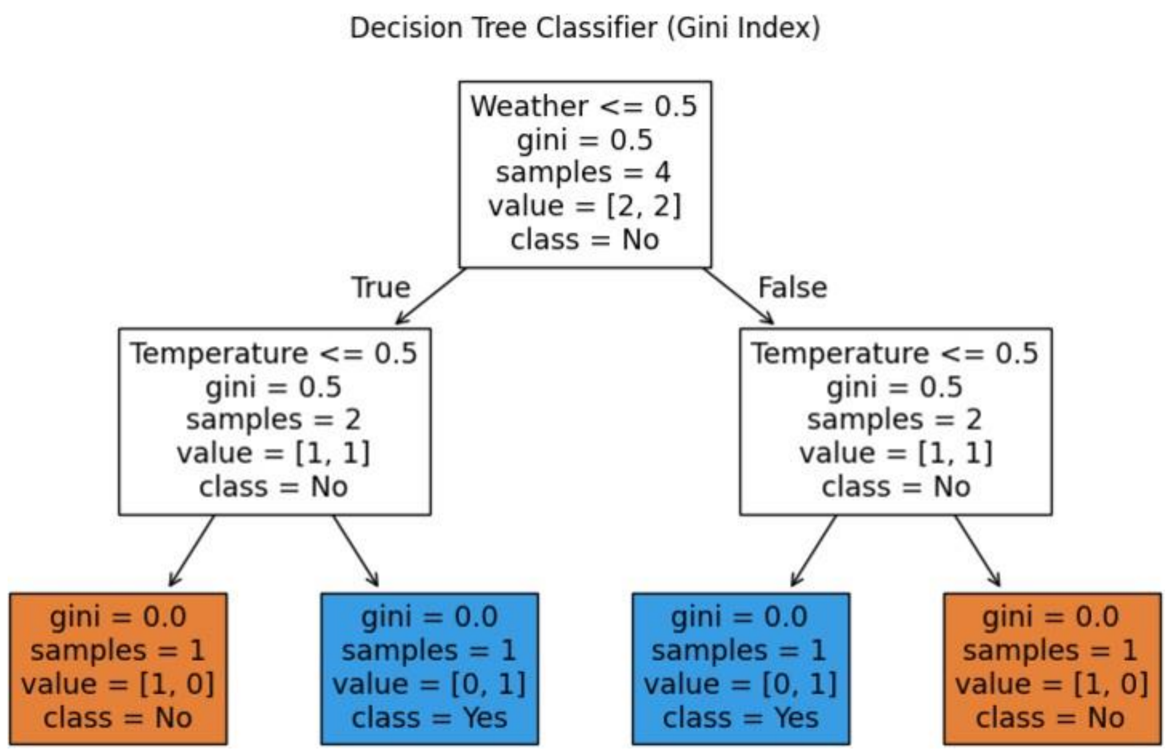
PCA Projection (2D)
Principal Component 1
Principal Component 2
State/UT
ANDAMAN And NICOBAR ISLANDS
ARUNACHAL PRADESH
ASSAM
MEGHALAYA
MANIPUR
MIZORAM
NAGALAND
TRIPURA
WEST BENGAL
SIKKIM
ORISSA
JHARKHAND
BIHAR
UTTAR PRADESH
UTTARANCHAL
HARYANA
CHANDIGARH
DELHI
PUNJAB
HIMACHAL
JAMMU AND KASHMIR
RAJASTHAN
MADHYA PRADESH
GUJARAT
DADAR NAGAR HAVELI
DAMAN AND DUI
MAHARASHTRA
GOA
CHATISGARH
ANDHRA PRADESH
TAMIL NADU
PONDICHERRY
KARNATAKA
KERALA
LAKSHADWEEP

K-Nearest Neighbors (KNN) Classifier on Iris Dataset

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
#from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)
# Train the model
knn.fit(X_train, y_train)
# Make predictions
y_pred = knn.predict(X_test)
# Evaluate the model

print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))

OUTPUT:
Accuracy Score:
1.0
```

Decision Tree Classifier with Gini Index on Categorical Dataset

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import matplotlib.pyplot as plt

# Sample Dataset
data = {
    'ID': [1, 2, 3, 4],
    'Weather': ['Sunny', 'Sunny', 'Rainy', 'Rainy'],
    'Temperature': ['Hot', 'Cold', 'Hot', 'Cold'],
    'Play': ['No', 'Yes', 'Yes', 'No']
}

# Create a DataFrame
df = pd.DataFrame(data)

# Features and Target
X = df[['Weather', 'Temperature']]
y = df['Play']

# Encode categorical features and target using LabelEncoder
encoder = LabelEncoder

X_encoded = X.apply(encoder.fit_transform) # Apply encoding to each
                                             column
y_encoded = encoder.fit_transform(y)       # Encode target variable

# Initialize Decision Tree Classifier with Gini Index
clf = DecisionTreeClassifier(criterion='gini', random_state=42)

# Train the model
clf.fit(X_encoded, y_encoded)

# Visualize the Decision Tree
plt.figure(figsize=(10, 6))
plot_tree(
    clf,
    feature_names=['Weather', 'Temperature'],
    class_names=encoder.classes_,
    filled=True
)
plt.title("Decision Tree Classifier (Gini Index)")
plt.show()
```


Decision Tree Classifier (Gini Index)
Weather <= 0.5
gini = 0.5
samples = 4
value = [2, 2]
class = No
True
False
Temperature <= 0.5
gini = 0.5
samples = 2
value = [1, 1]
class = No
Temperature <= 0.5
gini = 0.5
samples = 2
value = [1, 1]
class = No
gini = 0.0
samples = 1
value = [1, 0]
class = No
gini = 0.0
samples = 1
value = [0, 1]
class = Yes
gini = 0.0
samples = 1
value = [0, 1]
class = Yes
gini = 0.0
samples = 1
value = [1, 0]
class = No

Support Vector Classifier (SVC) with Linear Kernel on Iris Dataset

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
#from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X  =  iris.data
y = iris.target
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Create KNN classifier
support_vec= SVC(kernel='linear')
# Train the model
support_vec.fit(X_train, y_train)
# Make predictions
y_pred = support_vec.predict(X_test)
# Evaluate the model
print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))

OUTPUT:
Accuracy Score:
1.0
```

K-Nearest Neighbors (KNN) Classifier with 5-Fold Cross-Validation on Iris Dataset

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score, KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Create a KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)

# Define a 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(knn, X, y, cv=kf, scoring='accuracy')

# Print the cross-validation scores
print(f"Cross-validation scores: {cv_scores}")
print(f"Mean accuracy: {cv_scores.mean():.2f}")
print(f"Standard deviation of accuracy: {cv_scores.std():.2f}")
```

```
OUTPUT:
Cross-validation scores: [1.         0.96666667 0.96666667 0.93333333
0.96666667]
Mean accuracy: 0.97

Standard deviation of accuracy: 0.02
```

K-Nearest Neighbors (KNN) Classifier with 5-Fold Cross-Validation on Iris Dataset (k=5)

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score, KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X  =  data.data
y = data.target

# Create a KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)

# Define a 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(knn, X, y, cv=kf, scoring='accuracy')

# Print the cross-validation scores
print(f"Cross-validation scores: {cv_scores}")
print(f"Mean accuracy: {cv_scores.mean():.2f}")
print(f"Standard deviation of accuracy: {cv_scores.std():.2f}")

OUTPUT:
Cross-validation scores: [1.         1.         0.96666667 0.93333333
0.96666667]
Mean accuracy: 0.97
Standard deviation of accuracy: 0.02
```