Machine Learning Notes

Name: Utkarsh Mishra

INDEX

Practical 1.1
Data Imputation

1. Import Libraries
[Syntax]
```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.cluster import KMeans
```

A. Handling Missing Values
2. Load Dataset
[Syntax]
```
data_03 = [[12, np.nan, 34], [18, 32, np.nan], [np.nan, 11, 20]]
data_05 = pd.read_csv("Data_02.csv")
```

3. Impute missing values
[Syntax]
```
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
data_04 = imputer.fit_transform(data_03)
print("Imputed Example Data:\n", data_04)
```

4. Mode Imputation
[Syntax]
```
print("Sample Data from Data_02:\n", data_05.sample(5))
data_06 = data_05.fillna(data_05.mode().iloc[0])
print("Mode Imputation Result:\n", data_06.sample(5))
data_07 = data_06.isnull().sum()
print("Remaining Missing Values Count:\n", data_07.sample(5))
```

5. Drop Columns With
5.1 Any Missing Values
[Syntax]
```
data_08 = data_05.dropna(axis=1, how='any')
print("Shape after dropping columns with any missing values:",
data_08.shape)
```
5.2 All Missing Values
[Syntax]
```
data_08 = data_05.dropna(axis=1, how=”all”)
```

```
print("Shape after dropping columns with any missing values:",
data_08.shape)
```

[Output]

```

Imputed Example Data:
 [[12.  21.5 34. ]
 [18.  32.  27. ]
 [15.  11.  20. ]]
3]
0s
print("Sample Data from Data_02:\n", data_05.sample(5))
data_06 = data_05.fillna(data_05.mode().iloc[0])
print("Mode Imputation Result:\n", data_06.sample(5))
data_07 = data_06.isnull().sum()
print("Remaining Missing Values Count:\n", data_07.sample(5))
Sample Data from Data_02:
       ID FieldA  FieldB FieldC  FieldD  FieldE   FieldF  FieldG
0    1.0   Good  Better   Best  1024.0     NaN  10241.0       1
9   10.0      A       B      C     2.0     NaN  21111.0      10
5    6.0   Good     NaN   Best    32.0     NaN     32.0       6
18  10.0    NaN     NaN   Best     8.0     NaN    844.0      19
20  10.0      A       B      C     2.0     NaN    111.0      21
Mode Imputation Result:
       ID FieldA  FieldB FieldC  FieldD  FieldE  FieldF  FieldG
7    8.0   Good  Better   Best     8.0     NaN  8111.0       8
20  10.0      A       B      C     2.0     NaN   111.0      21
18  10.0   Good  Better   Best     8.0     NaN   844.0      19
8    9.0   Good  Better   Best     4.0     NaN    41.0       9
3    4.0   Good  Better   Best     2.0     NaN   211.0       4
Remaining Missing Values Count:
 FieldB    0
FieldD    0
FieldA    0
FieldC    0
FieldG    0
dtype: int64

Shape after dropping columns with any missing values: (21, 1)
Shape after dropping columns with any missing values: (21, 7)
```

Practical 1.2.1
Standardization

1. Import Libraries

[Syntax]

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

2. Load Dataset

[Syntax]

```
df = sns.load_dataset('titanic')
```

3. Exploratory Data Analysis

[Syntax]

```
print("First few rows of the original dataset:")
print(df.head())
print("\nDataset Info:")
print(df.info())
print("\nMissing Values:")
print(df.isnull().sum())
```

4. Select Numerical Features for Standardization

[Syntax]

```
numerical_features = df.select_dtypes(include=['float64',
'int64']).columns.tolist()
numerical_features.remove('survived')
print("\nNumerical features selected for standardization:")
print(numerical_features)
```

5. Visualize the Distribution of Numerical Features Before Standardization

[Syntax]

```
plt.figure(figsize=(12, 8))
for i, feature in enumerate(numerical_features, 1):
   plt.subplot(3, 2, i)
   sns.histplot(df[feature], kde=True)
   plt.title(f'Distribution of {feature}')
plt.tight_layout()
```

```
plt.show()
```

6. Standardization of Numerical Features

[Syntax]

```
scaler = StandardScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

7. Visualize

[Syntax]

```
plt.figure(figsize=(12, 8))
for i, feature in enumerate(numerical_features, 1):
   plt.subplot(3, 2, i)
   sns.histplot(df[feature], kde=True)
   plt.title(f'Distribution of Standardized {feature}')
plt.tight_layout()
plt.show()
```

8. Summary

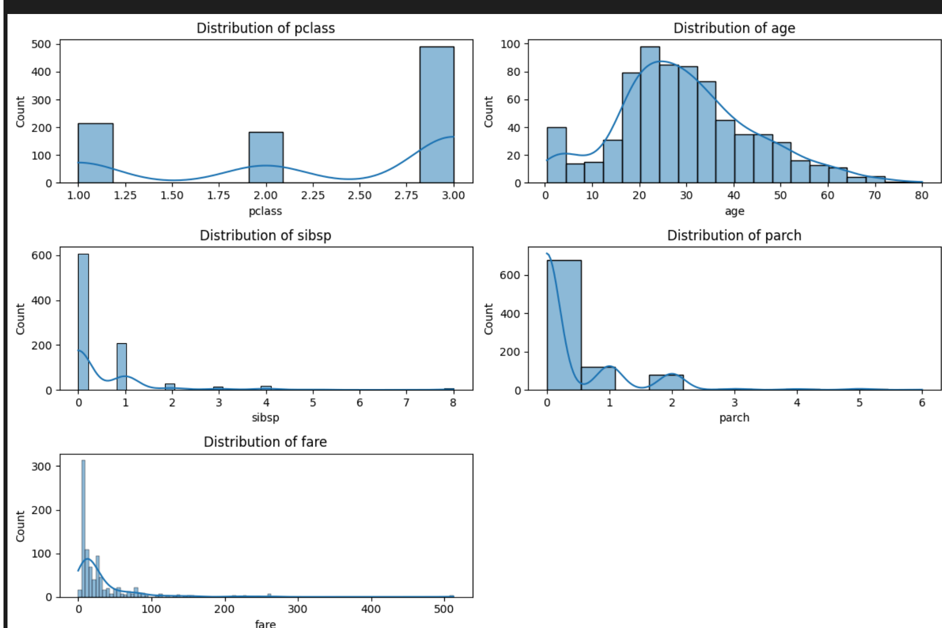
[Syntax]

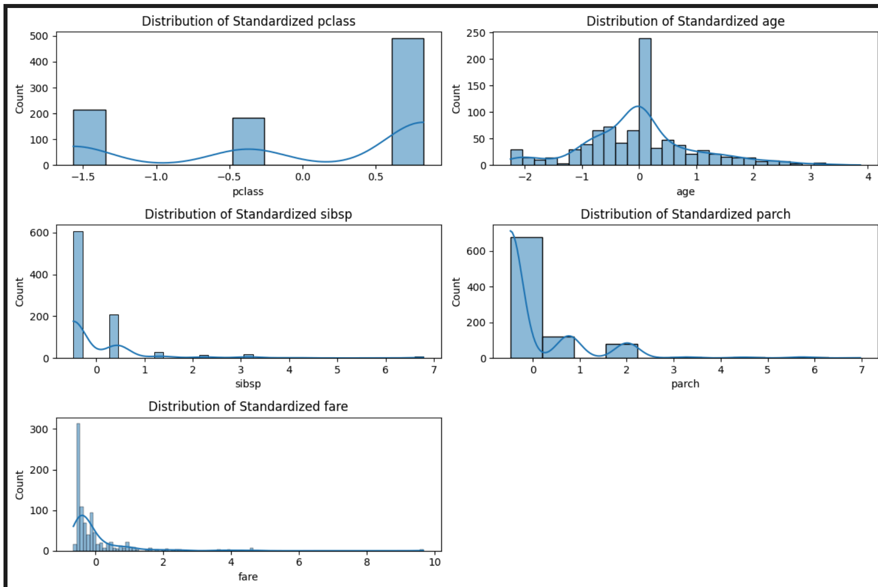
```
print("\nSummary of the standardized DataFrame:")
print(df[numerical_features].describe())
```

[Output]

```
First few rows of the original dataset:
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  
\
0         0       3    male  22.0      1      0   7.2500        S  Third   
1         1       1  female  38.0      1      0  71.2833        C  First   
2         1       3  female  26.0      0      0   7.9250        S  Third   
3         1       1  female  35.0      1      0  53.1000        S  First   
4         0       3    male  35.0      0      0   8.0500        S  Third   

     who  adult_male deck  embark_town alive  alone  
0    man        True  NaN  Southampton    no  False  
1  woman       False    C    Cherbourg   yes  False  
2  woman       False  NaN  Southampton   yes   True  
3  woman       False    C  Southampton   yes  False  
4    man        True  NaN  Southampton    no   True  

Dataset Info:
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
None
```






```
Summary of the standardized DataFrame:
             pclass          age        sibsp        parch
fare
count  8.910000e+02  8.910000e+02  8.910000e+02  8.910000e+02
8.910000e+02
mean  -8.772133e-17  2.232906e-16  4.386066e-17  5.382900e-17
3.987333e-18
std    1.000562e+00  1.000562e+00  1.000562e+00  1.000562e+00
1.000562e+00
min   -1.566107e+00 -2.253155e+00 -4.745452e-01 -4.736736e-01
-6.484217e-01
25%   -3.693648e-01 -5.924806e-01 -4.745452e-01 -4.736736e-01
-4.891482e-01
50%    8.273772e-01  0.000000e+00 -4.745452e-01 -4.736736e-01
-3.573909e-01
75%    8.273772e-01  4.079260e-01  4.327934e-01 -4.736736e-01
-2.424635e-02
max    8.273772e-01  3.870872e+00  6.784163e+00  6.974147e+00
9.667167e+00
```

Practical 1.2.2
Normalization

1. Import Libraries

[Syntax]

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

2. Load Dataset

[Syntax]

```
df = sns.load_dataset('titanic')
```

3. Exploratory Data Analysis

[Syntax]

```
print("First few rows of the original dataset:")
print(df.head())
print("\nDataset Info:")
print(df.info())
print("\nMissing Values:")
print(df.isnull().sum())
```

4. Select Numerical Features for Standardization

[Syntax]

```
numerical_features = df.select_dtypes(include=['float64',
'int64']).columns.tolist()
numerical_features.remove('survived')
print("\nNumerical features selected for standardization:")
print(numerical_features)
```

5. Visualize the Distribution of Numerical Features Before Standardization

[Syntax]

```
plt.figure(figsize=(12, 8))
for i, feature in enumerate(numerical_features, 1):
   plt.subplot(3, 2, i)
   sns.histplot(df[feature], kde=True)
   plt.title(f'Distribution of {feature}')
plt.tight_layout()
plt.show()
```

6. Normalization of Numerical Features

[Syntax]

```
scaler = MinMaxScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

7. Visualize

[Syntax]

```
plt.figure(figsize=(12, 8))
for i, feature in enumerate(numerical_features, 1):
   plt.subplot(3, 2, i)
   sns.histplot(df[feature], kde=True)
   plt.title(f'Distribution of Normalized {feature}')
plt.tight_layout()
plt.show()
```

8. Summary

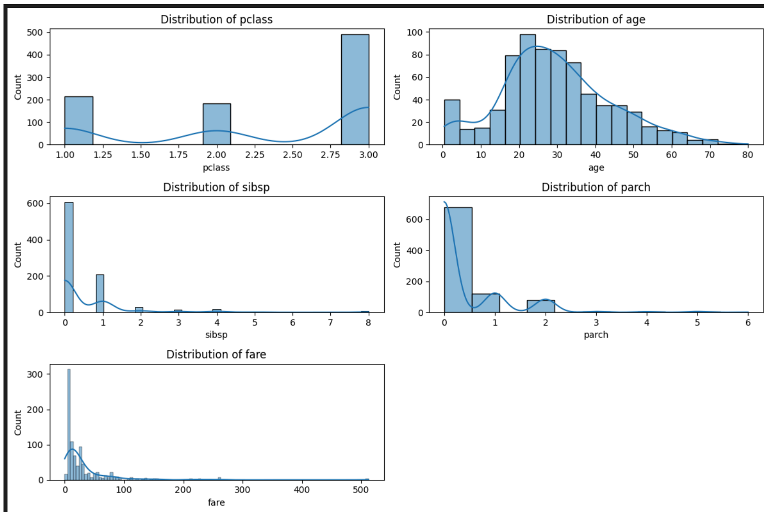
[Syntax]

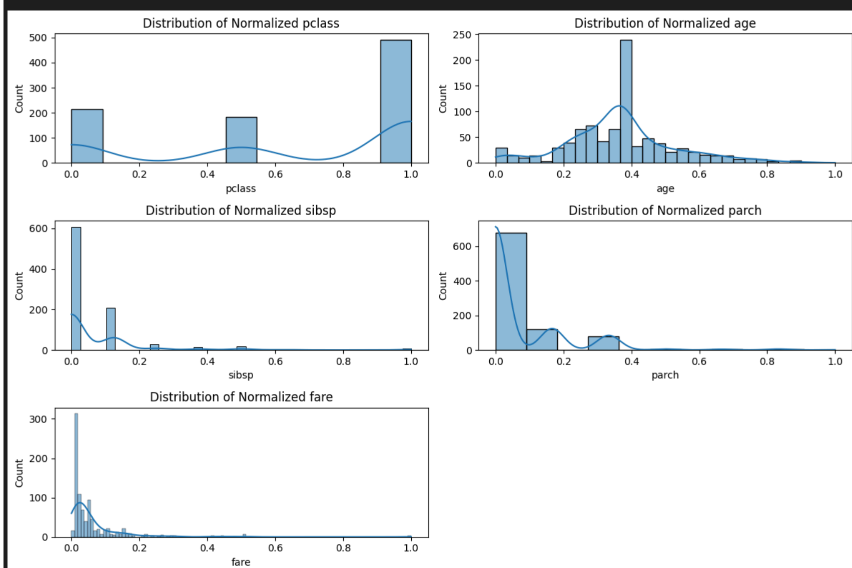
```
print("\nSummary of the normalized DataFrame:")
print(df[numerical_features].describe())
```

[Output]

```
First few rows of the original dataset:
   survived  pclass     sex   age  sibsp  parch     fare embarked  class
\
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third
```

```
     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
None

Numerical features selected for normalization:
['pclass', 'age', 'sibsp', 'parch', 'fare']
```





Summary of the normalized DataFrame:

	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.654321	0.367921	0.065376	0.063599	0.062858
std	0.418036	0.163383	0.137843	0.134343	0.096995
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.500000	0.271174	0.000000	0.000000	0.015440
50%	1.000000	0.367921	0.000000	0.000000	0.028213
75%	1.000000	0.434531	0.125000	0.000000	0.060508
max	1.000000	1.000000	1.000000	1.000000	1.000000

Practical 1.3.1
One Hot Encoding

1. Import Libraries

[Syntax]

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, StandardScaler
```

2. Load Dataset

[Syntax]

```
data_01 = pd.read_csv("/content/customer.csv")
```

3. Exploratory Data Analysis

[Syntax]

```
print(data_01.info())
print(data_01.describe())
print(data_01.isnull().sum())
for col in ["gender", "review", "education", "purchased"]:
   print(f"Unique values in {col}: {data_01[col].unique()}")
```

4. Load Dataset

[Syntax]

```
data_01['gender'].fillna(data_01['gender'].mode()[0], inplace=True)
data_01['education'].fillna(data_01['education'].mode()[0], inplace=True)
data_01['review'].fillna(data_01['review'].mode()[0], inplace=True)
data_01['purchased'].fillna(data_01['purchased'].mode()[0], inplace=True)
```

5. One-Hot Encoding of Categorical Features

[Syntax]

```
data_encoded = pd.get_dummies(data_01, columns=["gender", "education"],
drop_first=True)
encoder = OneHotEncoder(sparse_output=False)
categorical_columns = ["review", "purchased"]
encoded_categories = encoder.fit_transform(data_01[categorical_columns])
```

6. Combine Encoded Features

[Syntax]

```
encoded_df = pd.DataFrame(encoded_categories,
columns=encoder.get_feature_names_out(categorical_columns))
data_final = pd.concat([data_encoded.drop(columns=categorical_columns),
encoded_df], axis=1)
print(data_final.head())
print(data_final.info())
```

[Output]

```
   age  gender   review education purchased
0   30  Female  Average    School        No
1   68  Female     Poor        UG        No
2   70  Female     Good        PG        No
3   72  Female     Good        PG        No
4   16  Female  Average        UG        No
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        50 non-null     int64
 1   gender     50 non-null     object
 2   review     50 non-null     object
 3   education  50 non-null     object
 4   purchased  50 non-null     object
dtypes: int64(1), object(4)
memory usage: 2.1+ KB
None
             age
count  50.000000
```

```
mean    54.160000
std     25.658161
min     15.000000
25%     30.250000
50%     57.000000
75%     74.000000
max     98.000000
age          0
gender       0
review       0
education    0
purchased    0
dtype: int64
Unique values in gender: ['Female' 'Male']
Unique values in review: ['Average' 'Poor' 'Good']
Unique values in education: ['School' 'UG' 'PG']
Unique values in purchased: ['No' 'Yes']
   age  gender_Male  education_School  education_UG  review_Average  \
0   30        False              True         False             1.0
1   68        False             False          True             0.0
2   70        False             False         False             0.0
3   72        False             False         False             0.0
4   16        False             False          True             1.0

   review_Good  review_Poor  purchased_No  purchased_Yes
0          0.0          0.0           1.0            0.0
1          0.0          1.0           1.0            0.0
2          1.0          0.0           1.0            0.0
3          1.0          0.0           1.0            0.0
4          0.0          0.0           1.0            0.0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   age               50 non-null     int64
 1   gender_Male       50 non-null     bool
 2   education_School  50 non-null     bool
 3   education_UG      50 non-null     bool
 4   review_Average    50 non-null     float64
```

```
 5   review_Good        50 non-null      float64
 6   review_Poor        50 non-null      float64
 7   purchased_No       50 non-null      float64
 8   purchased_Yes      50 non-null      float64
dtypes: bool(3), float64(5), int64(1)
memory usage: 2.6 KB
None
```

Practical 1.3.2
Label Encoding

1. Import Libraries

[Syntax]

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
```

2. Load Dataset

[Syntax]

```
data_01 = pd.read_csv("/content/customer.csv")
```

3. Exploratory Data Analysis

[Syntax]

```
print(data_01.info())
print(data_01.describe())
print(data_01.isnull().sum())
for col in ["gender", "review", "education", "purchased"]:
   print(f"Unique values in {col}: {data_01[col].unique()}")
```

4. Load Dataset

[Syntax]

```
data_01['gender'].fillna(data_01['gender'].mode()[0], inplace=True)
data_01['education'].fillna(data_01['education'].mode()[0], inplace=True)
data_01['review'].fillna(data_01['review'].mode()[0], inplace=True)
data_01['purchased'].fillna(data_01['purchased'].mode()[0], inplace=True)
```

5. Initialization Label Encoding

[Syntax]

```
label_encoder = LabelEncoder()
```

6. Apply Label Encoding

[Syntax]

```
label_mappings = {}
for col in ["gender", "education", "review", "purchased"]:
   data_01[col + '_encoded'] = label_encoder.fit_transform(data_01[col])
   label_mappings[col] = dict(enumerate(label_encoder.classes_))
```

7. Display Label Mappings

[Syntax]

```
for col, mapping in label_mappings.items():
   print(f"\nLabel encoding for {col}:")
   for label, encoded in mapping.items():
       print(f"{encoded} -> {label}")
```

8.Combine Encoded Features

[Syntax]

```
print(data_01.head())
print(data_01.info())
```

[Output]

```
    age  gender   review education purchased
0    30  Female  Average    School        No
1    68  Female     Poor        UG        No
2    70  Female     Good        PG        No
3    72  Female     Good        PG        No
4    16  Female  Average        UG        No
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        50 non-null     int64
 1   gender     50 non-null     object
 2   review     50 non-null     object
 3   education  50 non-null     object
 4   purchased  50 non-null     object
dtypes: int64(1), object(4)
```

```
memory usage: 2.1+ KB
None
             age
count  50.000000
mean   54.160000
std    25.658161
min    15.000000
25%    30.250000
50%    57.000000
75%    74.000000
max    98.000000
age          0
gender       0
review       0
education    0
purchased    0
dtype: int64
Unique values in gender: ['Female' 'Male']
Unique values in review: ['Average' 'Poor' 'Good']
Unique values in education: ['School' 'UG' 'PG']
Unique values in purchased: ['No' 'Yes']
   age  gender  review  education  purchased
0   30       0       0          1          0
1   68       0       2          2          0
2   70       0       1          0          0
3   72       0       1          0          0
4   16       0       0          2          0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        50 non-null     int64
 1   gender     50 non-null     int64
 2   review     50 non-null     int64
 3   education  50 non-null     int64
 4   purchased  50 non-null     int64
dtypes: int64(5)
memory usage: 2.1 KB
```

Practical 1.4
Feature Construction

1. Import Libraries
[Syntax]
```
import pandas as pd
import seaborn as sns
```

2. Load Dataset
[Syntax]
```
data_01 = sns.load_dataset('titanic')
```

3. Exploratory Data Analysis
[Syntax]
```
print("First few rows of the original dataset:")
print(data_01.head())
print("\nDataset Info:")
print(data_01.info())
print("\nMissing Values:")
print(data_01.isnull().sum())
```

4. Construct a New Feature 'Family'
[Syntax]
```
data_01['Family'] = data_01['sibsp'] + data_01['parch']
```

5. Create a New Feature 'Alone'
[Syntax]
```
data_01['Alone'] = data_01['Family'] == 0
```

6. Drop Unnecessary Columns
[Syntax]
```
data_02= data_01.drop(columns=['sibsp', 'parch'])
```

7. Select Relevant Features
[Syntax]
```
data_01 = data_01[['sex', 'embarked', 'Alone', 'pclass', 'survived',
'Family']]
```

8. Display the DataFrame
[Syntax]

```
print("\nFinal DataFrame after feature construction:")
print(data_01.head())
```

9. Summary

Syntax]

```
print("\nSummary of the final DataFrame:")
print(data_01.describe())
```

[Output]

```
First few rows of the original dataset:
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  
\
0         0       3    male  22.0      1      0   7.2500        S  Third   
1         1       1  female  38.0      1      0  71.2833        C  First   
2         1       3  female  26.0      0      0   7.9250        S  Third   
3         1       1  female  35.0      1      0  53.1000        S  First   
4         0       3    male  35.0      0      0   8.0500        S  Third   

     who  adult_male deck  embark_town alive  alone  
0    man        True  NaN  Southampton    no  False  
1  woman       False    C    Cherbourg   yes  False  
2  woman       False  NaN  Southampton   yes   True  
3  woman       False    C  Southampton   yes  False  
4    man        True  NaN  Southampton    no   True  

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   survived     891 non-null    int64   
 1   pclass       891 non-null    int64   
 2   sex          891 non-null    object  
 3   age          714 non-null    float64 
 4   sibsp        891 non-null    int64   
 5   parch        891 non-null    int64   
 6   fare         891 non-null    float64 
 7   embarked     889 non-null    object  
 8   class        891 non-null    category
```

```
 9   who          891 non-null    object  
 10  adult_male   891 non-null    bool    
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object  
 13  alive        891 non-null    object  
 14  alone        891 non-null    bool    
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
None

Missing Values:
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64

Final DataFrame after feature construction:
      sex embarked  Alone  pclass  survived  Family
0    male        S  False       3         0       1
1  female        C  False       1         1       1
2  female        S   True       3         1       0
3  female        S  False       1         1       1
4    male        S   True       3         0       0

Summary of the final DataFrame:
           pclass    survived      Family
count  891.000000  891.000000  891.000000
mean     2.308642    0.383838    0.904602
```

```
std      0.836071    0.486592    1.613459
min      1.000000    0.000000    0.000000
25%      2.000000    0.000000    0.000000
50%      3.000000    0.000000    0.000000
75%      3.000000    1.000000    1.000000
max      3.000000    1.000000   10.000000
```

C

Practical 1.5.1
Correlation

1. Import Libraries

[Syntax]

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
```

2. Load Dataset

[Syntax]

```
df = load_diabetes()
x = pd.DataFrame(df.data, columns=df.feature_names)
y = df.target
```

3. Sample data and display statistics

[Syntax]

```
print("Feature names:")
print(x.columns.tolist())
print("\nFirst few rows of the dataset:")
print(x.head())
```

4. Correlation Matrix

[Syntax]

```
correlation_matrix = x.corr()
```

5. Visualize the Correlation Matrix

[Syntax]

```
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f",
cmap=plt.cm.CMRmap_r,
           square=True, cbar_kws={"shrink": .8}, linewidths=.5)
plt.title('Correlation Heatmap of Features', fontsize=16)
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.show()
```

6. Threshold for Feature Selection

[Syntax]

```
threshold = 0.5
print(f"\nFeatures with correlation greater than {threshold} with
target:")
correlation_with_target = x.corrwith(pd.Series(y))
high_corr_features = correlation_with_target[abs(correlation_with_target)
> threshold].index
print(high_corr_features)
```

7. Select Features with High Correlation to Target Variable

[Syntax]

```
selected_features = x[high_corr_features]
print("\nSelected features based on correlation threshold:")
print(selected_features.columns.tolist())
```

8. Summary

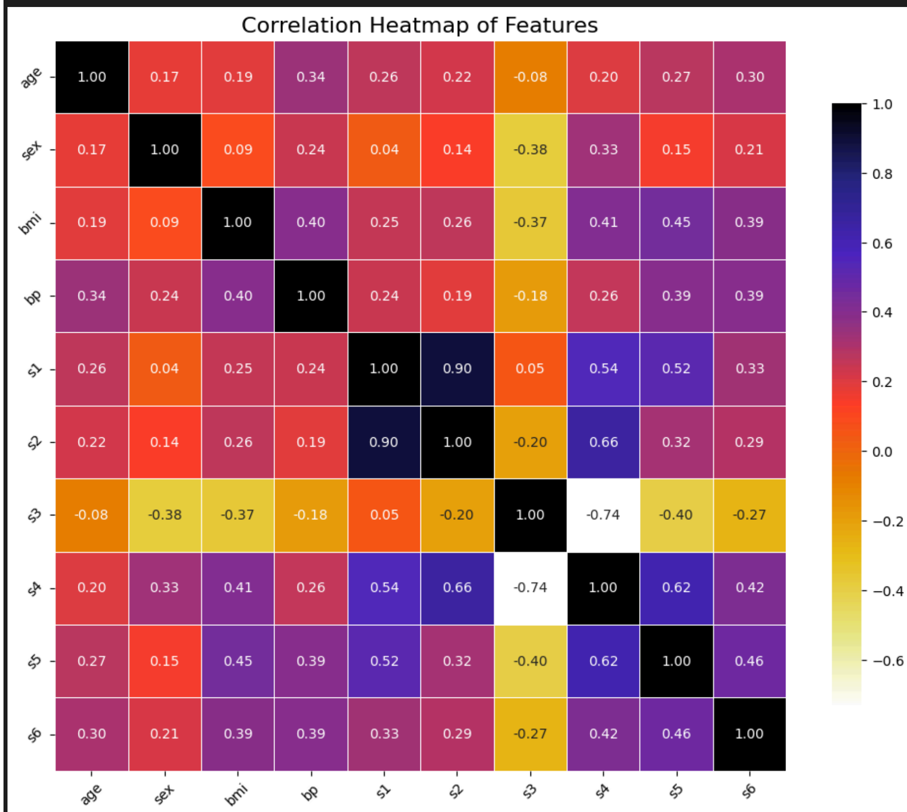
[Syntax]

```
summary = pd.DataFrame({
   'Feature': high_corr_features,
   'Correlation with Target': correlation_with_target[high_corr_features]
})
print("\nSummary of selected features:")
print(summary)
```

[Output]

```
Feature names:
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']

First few rows of the dataset:
        age       sex       bmi        bp        s1        s2        s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
2  0.085299  0.050680  0.044451 -0.005670 -0.045599 -0.034194 -0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142

         s4        s5        s6
0 -0.002592  0.019907 -0.017646
1 -0.039493 -0.068332 -0.092204
```

```
2 -0.002592  0.002861 -0.025930
3  0.034309  0.022688 -0.009362
4 -0.002592 -0.031988 -0.046641
```




```
Features with correlation greater than 0.5 with target:
Index(['bmi', 's5'], dtype='object')

Selected features based on correlation threshold:
['bmi', 's5']

Summary of selected features:
    Feature  Correlation with Target
bmi     bmi                 0.586450
s5       s5                 0.565883
```

Practical 1.5.2
Variance Thresholding

1. Import Libraries
[Syntax]
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_selection import VarianceThreshold
```

2. Load Dataset
[Syntax]
```
data = {
   'F1': [1, 2, 3, 4, 1, 5],
   'F2': [4, 1, 5, 2, 6, 8],
   'F3': [0, 0, 0, 0, 0, 0],
   'F4': [1, 1, 1, 1, 1, 1]
}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
```

3. Calculate Variance
[Syntax]
```
variance = df.var()
print("\nVariance of each feature:")
print(variance)
```

4. Visualize the Variance
[Syntax]
```
plt.bar(df.columns, variance, color='skyblue')
plt.axhline(y=0, color='gray', linestyle='--')
plt.title('Feature Variance')
plt.xlabel('Features')
plt.ylabel('Variance')
plt.show()
```

5. Apply Variance Threshold

[Syntax]

```
threshold = 0.1
var_thre = VarianceThreshold(threshold=threshold)
var_thre.fit(df)
```

6. Get Support Mask and Select Features

[Syntax]

```
df_reduced = df[features_to_keep]
print("\nReduced DataFrame:")
print(df_reduced)
```

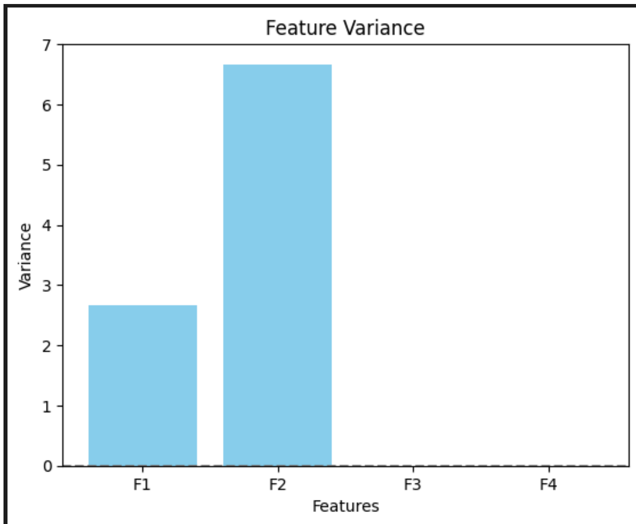
7. Summary of Selected Features

[Syntax]

```
print("\nSummary of selected features:")
summary = pd.DataFrame({
   'Feature': features_to_keep,
   'Variance': variance[features_to_keep]
})
print(summary)
```

[Output]

```
Original DataFrame:
   F1  F2  F3  F4
0   1   4   0   1
1   2   1   0   1
2   3   5   0   1
3   4   2   0   1
4   1   6   0   1
5   5   8   0   1

Variance of each feature:
F1    2.666667
F2    6.666667
F3    0.000000
F4    0.000000
dtype: float64
```




```
Features selected after applying Variance Threshold:
Index(['F1', 'F2'], dtype='object')

Reduced DataFrame:
   F1  F2
0   1   4
1   2   1
2   3   5
3   4   2
4   1   6
5   5   8

Summary of selected features:
   Feature  Variance
F1      F1  2.666667
F2      F2  6.666667
```

Practical 1.6.1
PCA

1. Import Libraries
[Syntax]
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

2. Load Dataset
[Syntax]
```
data_01 = pd.read_csv("Data.csv")
y = data_01.iloc[:, 0]
X = data_01.iloc[:, 1:]
```

3. Standardize Data
[Syntax]
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

4. Principal Component Analysis
[Syntax]
```
pca = PCA(n_components=100)
```

5. Fit PCA and Transform Data
[Syntax]
```
X_pca = pca.fit_transform(X_scaled)
```

6. DataFrame
[Syntax]
```
pca_df = pd.DataFrame(data=X_pca, columns=[f'PC{i+1}' for i in
range(100)])
pca_df['target'] = y
```

6. Visualization
[Syntax]
```
plt.figure(figsize=(8, 6))
```

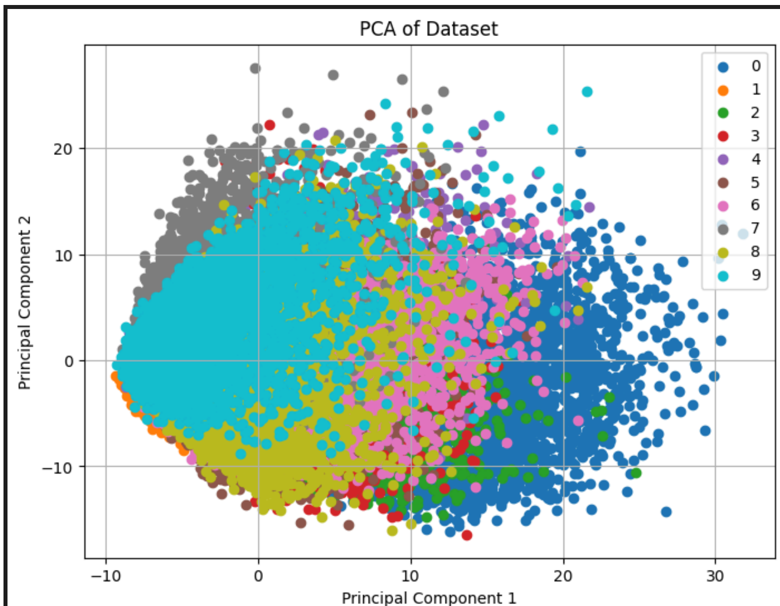
```
unique_targets = np.unique(y)

for target in unique_targets:
   plt.scatter(pca_df[pca_df['target'] == target]['PC1'],
               pca_df[pca_df['target'] == target]['PC2'],
               label=target)

plt.title('PCA of Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid()
plt.show()
```

6. Explained Variance

[Syntax]

```
explained_variance = pca.explained_variance_ratio_
print("Explained variance by each component:", explained_variance)
print("Total explained variance:", np.sum(explained_variance))
```

[Output]




```
Explained variance by each component: [0.05747953 0.04111691 0.03782867
0.02939862 0.02556439 0.02229844
 0.01952552 0.01771605 0.0156266  0.01424956 0.01361009 0.01222541
```

```
 0.01135736 0.0111309  0.01050311 0.01012326 0.00951303 0.00934514
 0.00907259 0.00885327 0.00838907 0.00812027 0.00775161 0.00752312
 0.0072769  0.00698756 0.00690455 0.00664924 0.00630675 0.00616301
 0.00610922 0.00597037 0.00577284 0.00573682 0.00564822 0.00546322
 0.0053943  0.00524361 0.00504843 0.0048853  0.00482244 0.00475901
 0.00460008 0.00457629 0.00449584 0.00446722 0.00443837 0.00436837
 0.00432713 0.00427027 0.00419269 0.0041218  0.00402461 0.00399434
 0.00394891 0.00390805 0.00379899 0.00372454 0.00368413 0.00365723
 0.00353278 0.00351088 0.00345414 0.00341394 0.00337784 0.00336477
 0.0033171  0.00329725 0.00320016 0.00316776 0.00312695 0.00311861
 0.00308212 0.00303276 0.00301509 0.0029714  0.00294849 0.00293629
 0.00287848 0.00286906 0.00283171 0.00282885 0.00281701 0.00279215
 0.00276842 0.00276316 0.00273923 0.0027181  0.00268141 0.00265443
 0.00262072 0.00260164 0.00258196 0.00254877 0.00253504 0.00252401
 0.00250074 0.00245803 0.0024436  0.00241387]
Total explained variance: 0.7145018867909934
```

Practical 2.1
Simple Linear Regression

1. Import Libraries

[Syntax]

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.linear_model import LinearRegression
```

2. Load Dataset

[Syntax]

```
data_01 = pd.read_csv("Data.csv")
```

3. Sample data and display statistics

[Syntax]

```
print(data_01.sample(5))
print(data_01.describe(include="all").T)
```

4. Correlation analysis

[Syntax]

```
correlation_matrix = data_01.corr()
sns.heatmap(correlation_matrix, annot=True)
plt.show()
```

5. Features and target variable

[Syntax]

```
X = data_01["Temperature"].values.reshape(-1, 1)
Y = data_01["Revenue"].values.reshape(-1, 1)
```

6. Train-Test Split

[Syntax]

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=25)
```

7. Model Fitting

[Syntax]

```
regression = LinearRegression()
regression.fit(X_train, Y_train)
```

8. Model coefficients

[Syntax]

```
print("Coefficients:", regression.coef_)
print("Intercept:", regression.intercept_)
```

9. Predictions

[Syntax]

```
Y_estimate = regression.predict(X_test)
print("Predicted Values:", Y_estimate.flatten())
```

10. Compare real and estimated values

[Syntax]

```
data_02 = pd.DataFrame()
data_02["Real_Value"] = Y_test.flatten()
data_02["Estimated_Value"] = Y_estimate.flatten()
data_02['Error'] = data_02["Real_Value"] - data_02["Estimated_Value"]
print(data_02)
```

11. Performance

[Syntax]

```
print("Mean Squared Error:", mean_squared_error(Y_test, Y_estimate))
print("Mean Absolute Error:", mean_absolute_error(Y_test, Y_estimate))
print("Root Mean Squared Error:", np.sqrt(mean_squared_error(Y_test,
Y_estimate)))
print("R-Squared:", r2_score(Y_test, Y_estimate))
```

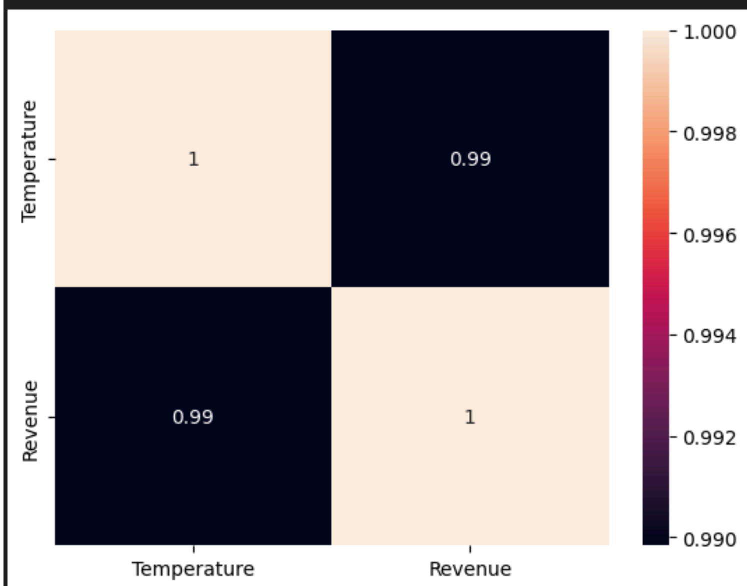
12. Visualize

[Syntax]

```
plt.scatter(X, Y, label="Actual Data")
plt.scatter(X_test, Y_estimate, color="r", label="Predicted Data")
plt.xlabel("Temperature")
plt.ylabel("Revenue")
plt.title("Temperature vs Revenue")
plt.legend()
plt.show()
```

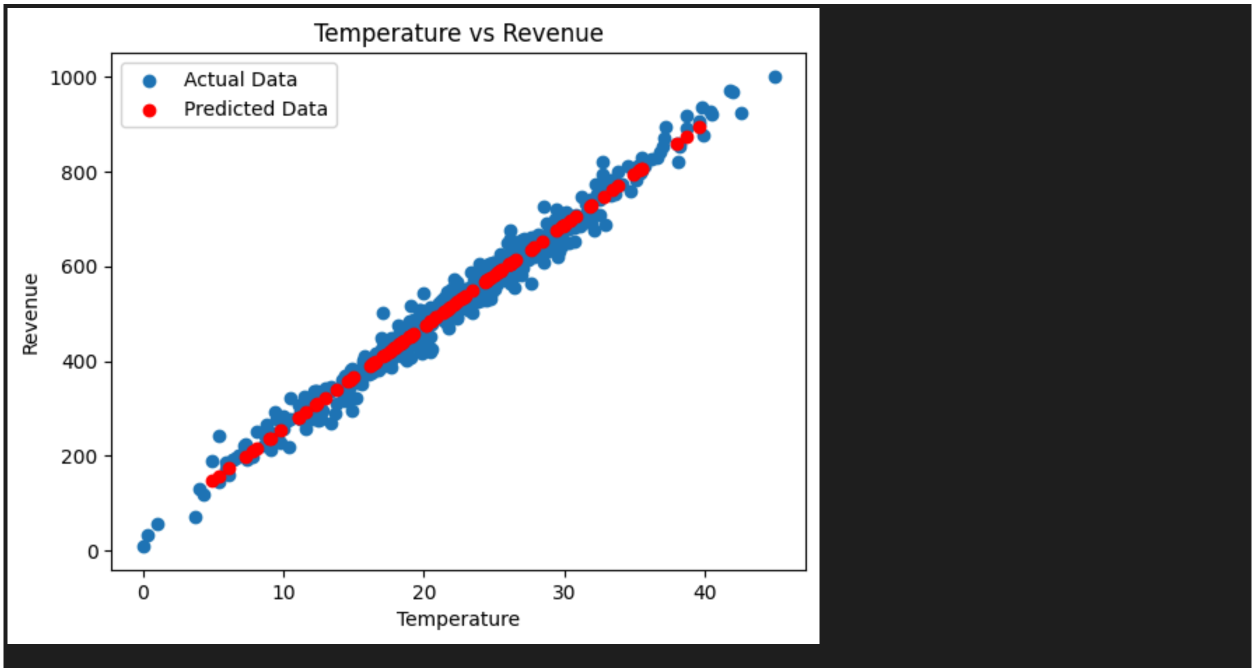
[Output]

```
     Temperature  Revenue
380         20.5      514
285         26.4      575
60          16.4      382
34          35.7      810
109         27.8      652
             count      mean         std   min      25%    50%    75%
max
Temperature  500.0   22.2816    8.097597   0.0   17.175   22.4   27.8
45.0
Revenue      500.0  522.0580  175.410399  10.0  406.000  530.0  643.0
1000.0
```




```
Coefficients: [[21.50681768]]
Intercept: [42.22043519]
Predicted Values: [442.24724397 483.11019756 726.1372373  653.0140572
455.15133458
 199.22020423 409.98701746 683.12360195 392.78156332 476.65815226
 437.94588044 575.58951357 685.27428372 308.90497438 235.78179428
 504.61701523 747.64405498 339.01451912 792.8083721  427.1924716
 399.23360862 216.42565837 450.84997104 728.28791907 571.28815003
 513.21974231 640.1099666  605.69905831 859.4795069  420.7404263
 158.35725064 280.9461114  769.15087266 547.63065059 635.80860306
 612.15110362 491.71292463 209.97361306 414.28838099 893.89041518
 605.69905831 569.13746826 440.0965622  530.42519645 433.6445169
```

```
 360.5213368  508.91837877 805.71246271 696.02769256 147.6038418
 364.82270034 523.97315114 422.89110806 805.71246271 457.30201635
 605.69905831 435.79519867 394.93224508 306.75429261 291.69952024
 390.63088155 321.80906498 704.63041963 397.08292685 590.64428594
 674.52087488 511.06906054 605.69905831 280.9461114  485.26087933
 566.9867865  429.34315337 536.87724175 453.00065281 592.79496771
 252.98724842 586.34292241 590.64428594 874.53427927 450.84997104
 491.71292463 687.42496549 614.30178539 356.21997327 590.64428594
 519.67178761 569.13746826 582.04155887 605.69905831 237.93247604
 760.54814559 173.41202301 569.13746826 440.0965622  801.41109917
 534.72655998 603.54837655 483.11019756 502.46633347 476.65815226]
    Real_Value  Estimated_Value      Error
0          444       442.247244   1.752756
1          514       483.110198  30.889802
2          741       726.137237  14.862763
3          633       653.014057 -20.014057
4          475       455.151335  19.848665
..         ...              ...        ...
95         517       534.726560 -17.726560
96         600       603.548377  -3.548377
97         452       483.110198 -31.110198
98         521       502.466333  18.533667
99         478       476.658152   1.341848

[100 rows x 3 columns]
Mean Squared Error: 704.5829007363823
Mean Absolute Error: 20.149691942088634
Root Mean Squared Error: 26.543980499095877
R-Squared: 0.9746596552577004
```



[Insights]
1. Mean Squared Error (MSE): 704.58
→ Indicates the average squared error; lower values signify better performance.

2. Mean Absolute Error (MAE): 20.15
→ Average absolute deviation of predictions; a lower value suggests more accurate predictions.

3. Root Mean Squared Error (RMSE): 26.54
→ Average error in the same units as the data; indicates predictions are off by about 26.54 units on average.

4. R-Squared: 0.97
→ About 97% of the variance in the data is explained by the model, indicating a strong fit.

Practical 2.2
Multiple Linear Regression

1. Import Libraries

[Syntax]

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.linear_model import LinearRegression
```

2. Load Dataset

[Syntax]

```
data_01 = pd.read_csv("Data.csv")
```

3. Sample data and display statistics

[Syntax]

```
print(data_01.sample(5))
print(data_01.describe(include="all").T)
```

4. Correlation analysis

[Syntax]

```
correlation_matrix = data_01.corr()
sns.heatmap(correlation_matrix, annot=True)
plt.show()
```

5. Features and target variable

[Syntax]

```
X = data_01["Temperature"].values.reshape(-1, 1)
Y = data_01["Revenue"].values.reshape(-1, 1)
```

6. Train-Test Split

[Syntax]

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=25)
```

7. Model Fitting

[Syntax]

```
regression = LinearRegression()
regression.fit(X_train, Y_train)
```

8. Model coefficients

[Syntax]

```
print("Coefficients:", regression.coef_)
print("Intercept:", regression.intercept_)
```

9. Predictions

[Syntax]

```
Y_estimate = regression.predict(X_test)
print("Predicted Values:", Y_estimate.flatten())
```

10. Compare real and estimated values

[Syntax]

```
data_02 = pd.DataFrame()
data_02["Real_Value"] = Y_test.values
data_02["Estimated_Value"] = Y_estimate.flatten()
data_02['Error'] = data_02["Real_Value"] - data_02["Estimated_Value"]
print(data_02)
```

11. Performance

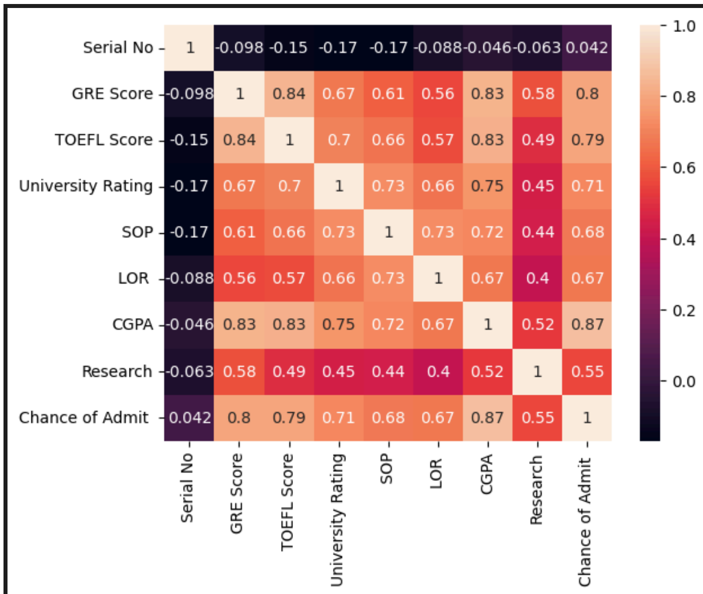
[Syntax]

```
print("Mean Squared Error:", mean_squared_error(Y_test, Y_estimate))
print("Mean Absolute Error:", mean_absolute_error(Y_test, Y_estimate))
print("Root Mean Squared Error:", np.sqrt(mean_squared_error(Y_test,
Y_estimate)))
print("R-Squared:", r2_score(Y_test, Y_estimate))
```

[Output]

```
     Serial No  GRE Score  TOEFL Score  University Rating   SOP  LOR
CGPA  \
373        374        321          109                  3   3.0   3.0
8.54
282        283        312          106                  3   4.0   3.5
8.79
```

```
327         328        295          101                  2   2.5   2.0
7.86
4             5        314          103                  2   2.0   3.0
8.21
91           92        299           97                  3   5.0   3.5
7.66

     Research  Chance of Admit
373         1              0.79
282         1              0.81
327         0              0.69
4           0              0.65
91          0              0.38
                   count        mean         std     min     25%     50%
\
Serial No          400.0  200.500000  115.614301    1.00  100.75  200.50
GRE Score          400.0  316.807500   11.473646  290.00  308.00  317.00
TOEFL Score        400.0  107.410000    6.069514   92.00  103.00  107.00
University Rating  400.0    3.087500    1.143728    1.00    2.00    3.00
 SOP               400.0    3.400000    1.006869    1.00    2.50    3.50
LOR                400.0    3.452500    0.898478    1.00    3.00    3.50
CGPA               400.0    8.598925    0.596317    6.80    8.17    8.61
Research           400.0    0.547500    0.498362    0.00    0.00    1.00
Chance of Admit    400.0    0.724350    0.142609    0.34    0.64    0.73

                        75%     max
Serial No          300.2500  400.00
GRE Score          325.0000  340.00
TOEFL Score        112.0000  120.00
University Rating    4.0000    5.00
 SOP                 4.0000    5.00
LOR                  4.0000    5.00
CGPA                 9.0625    9.92
Research             1.0000    1.00
Chance of Admit      0.8300    0.97
```




```
Coefficients: [ 0.00131371  0.00305671  0.00591875 -0.00379069  0.02444273
0.11631008
  0.02921256]
Intercept: -1.1251812847533555
Predicted Values: [0.93170562 0.69410721 0.78723557 0.53921174 0.69597836
0.88532504
 0.99423997 0.59402473 0.70461353 0.95539849 0.73493312 0.58606356
 0.49904286 0.89701047 0.7734923  0.78127291 0.51015202 0.90787748
 0.60303696 0.58983954 0.88268014 0.90597726 0.82641168 0.82562877
 0.72126662 0.86581365 0.61877191 0.71586698 0.61373614 0.67934741
 0.87422112 0.7168106  0.43523487 0.53770338 0.67549947 0.5869583
 0.78381129 0.80334423 0.81610664 0.72776697 0.70928186 0.61651905
 0.87141532 0.58675855 0.82961041 0.68564994 0.69456296 0.47386422
 0.80739219 0.83011621 0.53043787 0.7747675  0.84817643 0.72813481
 0.79430402 0.7061743  0.77247994 0.92793553 0.87846364 0.58083578
 0.94900488 0.6538332  0.89436181 0.71021358 0.68658971 0.81801625
 0.63150815 0.69968124 0.66920576 0.84171404 0.65110929 0.88199909
 0.7268228  0.93785713 0.63692464 0.51856542 0.85420518 0.65311665
 0.54593246 0.82183928]
    Real_Value  Estimated_Value     Error
0         0.94         0.931706  0.008294
1         0.67         0.694107 -0.024107
2         0.81         0.787236  0.022764
3         0.38         0.539212 -0.159212
4         0.70         0.695978  0.004022
..         ...              ...       ...
```

```
75        0.45        0.518565 -0.068565
76        0.78        0.854205 -0.074205
77        0.62        0.653117 -0.033117
78        0.64        0.545932  0.094068
79        0.87        0.821839  0.048161

[80 rows x 3 columns]
Mean Squared Error: 0.004173399997535151
Mean Absolute Error: 0.049130925846978285
Root Mean Squared Error: 0.06460185753935525
R-Squared: 0.8289825620561656
```

[Insights]
1. Mean Squared Error (MSE): 0.0042
→ This low value indicates that the average squared error is minimal, suggesting good model performance.

2. Mean Absolute Error (MAE): 0.0491
→ The average absolute difference between predictions and actual values is approximately 0.0491, indicating that predictions are quite close to the actual values.

3. Root Mean Squared Error (RMSE): 0.0646
→ This value, in the same units as the data, shows that predictions are, on average, off by about 0.0646 units, reinforcing the model's accuracy.

4. R-Squared: 0.83
→ About 83% of the variance in the dependent variable is explained by the independent variables, indicating a relatively strong fit.

Practical 2.3
Estimation of Linear Regression Parameter using Gradient Descent

1. Import Libraries

[Syntax]

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

2. Generate Dataset

[Syntax]

```
np.random.seed(0)
x = np.random.randn(100, 1)
y = 14 * x + np.random.randn(100, 1)
```

3. Scatter Plot

[Syntax]

```
plt.scatter(x, y, alpha=0.6)
plt.title("Scatter Plot of Generated Data")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

4. Gradient Function

[Syntax]

```
def Gradient_Descent(x, y, m, b, lr):
    """
    Perform gradient descent for a single iteration to learn parameters m
(slope) and b (intercept).

    Parameters:
    x (numpy.ndarray): Independent variable values.
    y (numpy.ndarray): Dependent variable values.
    m (float): Current value of the slope.
    b (float): Current value of the intercept.
    lr (float): Learning rate.

    Returns:
    tuple: Updated values of m and b.
    """
```

```
    N = x.shape[0]
    dm = 0.0
    db = 0.0
    for xi, yi in zip(x, y):
        prediction = m * xi + b
        dm += -2 * xi * (yi - prediction)
        db += -2 * (yi - prediction)
    dm /= N
    db /= N
    m -= lr * dm
    b -= lr * db
    return m, b
```

5. Initialize Parameters

[Syntax]

```
m = np.array([0.0])
b = np.array([0.0])
learning_rate = 0.1
epochs = 20
```

6. Store losses

[Syntax]

```
losses = []
m_values = []
b_values = []
```

7. Gradient Descent Loop

[Syntax]

```
for epoch in range(epochs):
    m, b = Gradient_Descent(x, y, m, b, learning_rate)
    y_hat = m * x + b
    loss = np.mean((y - y_hat) ** 2)
    losses.append(loss)
    m_values.append(m[0])
    b_values.append(b[0])

    print(f"At epoch {epoch + 1}, loss is {loss:.4f} with parameter m =
{m[0]:.4f} and b = {b[0]:.4f}")
```

8. Plot loss over epochs

[Syntax]

```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(1, epochs + 1), losses, marker='o')
plt.title("Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss (Mean Squared Error)")
plt.grid()
```

9. Plot of the regression line

[Syntax]

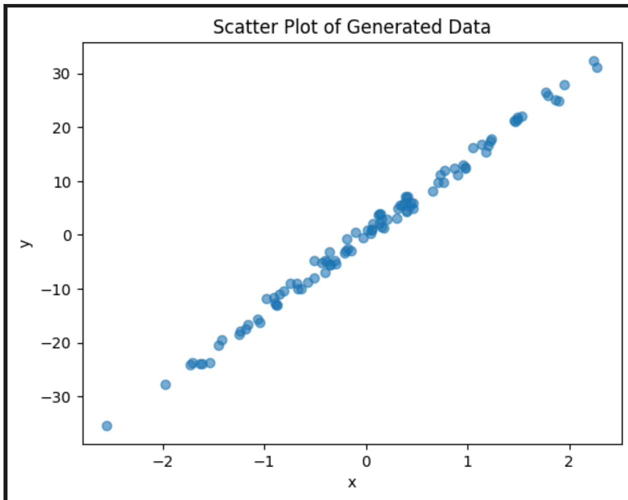
```
plt.subplot(1, 2, 2)
plt.scatter(x, y, label='Data points', alpha=0.6)
plt.plot(x, y_hat, color='red', label='Fitted line')
plt.title("Linear Regression Fit")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()

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

10. Plot parameter values over epochs

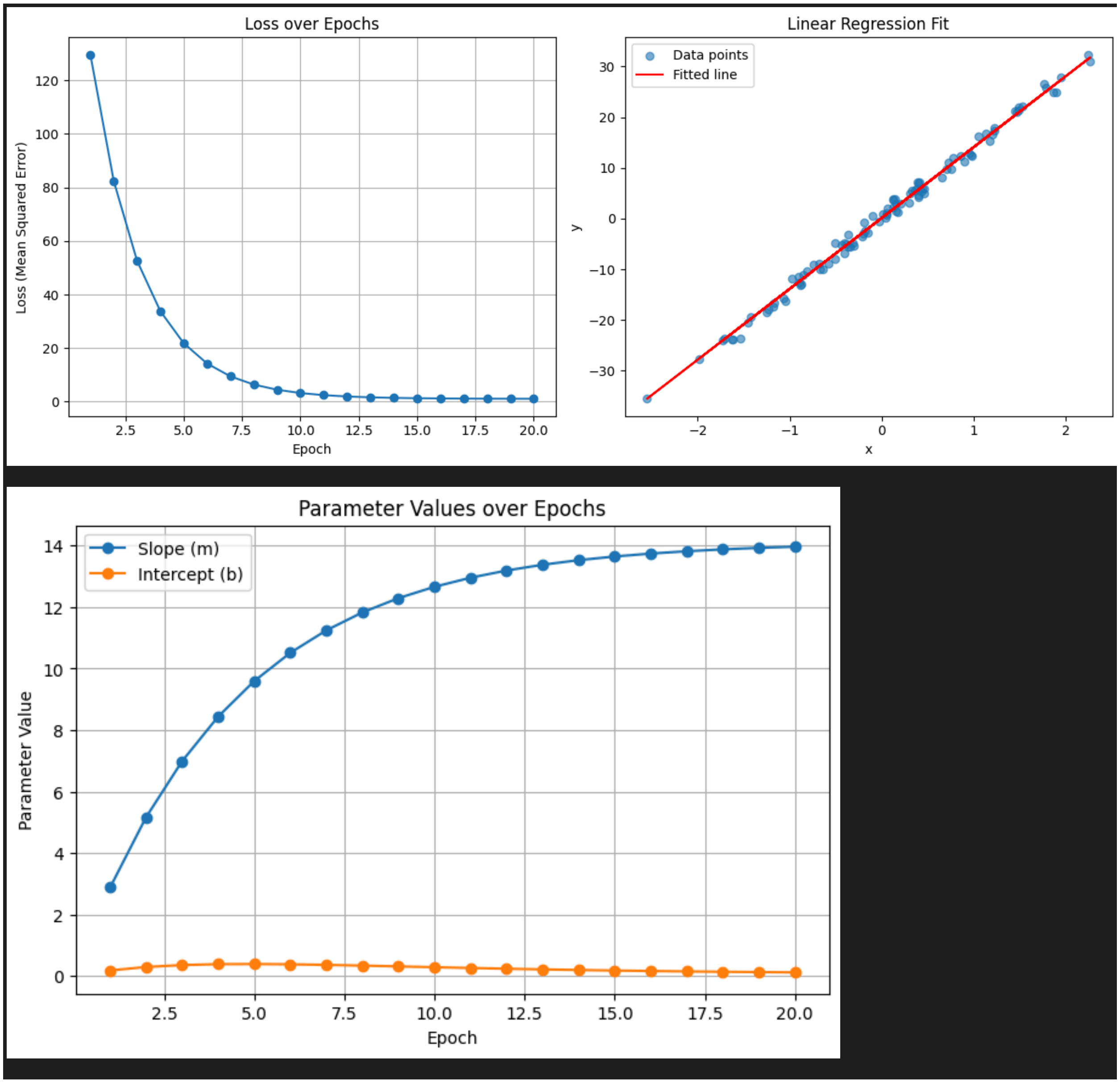
[Syntax]

```
plt.figure(figsize=(8, 5))
plt.plot(range(1, epochs + 1), m_values, label='Slope (m)', marker='o')
plt.plot(range(1, epochs + 1), b_values, label='Intercept (b)',
marker='o')
plt.title("Parameter Values over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Parameter Value")
plt.legend()
plt.grid()
plt.show()
```

[Output]




```
At epoch 1, loss is 129.6222 with parameter m = 2.8786 and b = 0.1839
At epoch 2, loss is 82.4634 with parameter m = 5.1681 and b = 0.2965
At epoch 3, loss is 52.6490 with parameter m = 6.9895 and b = 0.3593
At epoch 4, loss is 33.7835 with parameter m = 8.4388 and b = 0.3877
At epoch 5, loss is 21.8354 with parameter m = 9.5923 and b = 0.3931
At epoch 6, loss is 14.2616 with parameter m = 10.5105 and b = 0.3836
At epoch 7, loss is 9.4562 with parameter m = 11.2416 and b = 0.3650
At epoch 8, loss is 6.4045 with parameter m = 11.8239 and b = 0.3414
At epoch 9, loss is 4.4648 with parameter m = 12.2878 and b = 0.3156
At epoch 10, loss is 3.2307 with parameter m = 12.6574 and b = 0.2893
At epoch 11, loss is 2.4448 with parameter m = 12.9519 and b = 0.2639
At epoch 12, loss is 1.9439 with parameter m = 13.1867 and b = 0.2401
At epoch 13, loss is 1.6243 with parameter m = 13.3740 and b = 0.2182
At epoch 14, loss is 1.4202 with parameter m = 13.5233 and b = 0.1984
At epoch 15, loss is 1.2898 with parameter m = 13.6424 and b = 0.1809
At epoch 16, loss is 1.2063 with parameter m = 13.7374 and b = 0.1654
At epoch 17, loss is 1.1529 with parameter m = 13.8133 and b = 0.1518
At epoch 18, loss is 1.1187 with parameter m = 13.8738 and b = 0.1401
At epoch 19, loss is 1.0967 with parameter m = 13.9221 and b = 0.1300
At epoch 20, loss is 1.0826 with parameter m = 13.9607 and b = 0.1213
```


Loss over Epochs
Loss (Mean Squared Error)
Epoch
Linear Regression Fit
Data points
Fitted line
y
x
Parameter Values over Epochs
Slope (m)
Intercept (b)
Parameter Value
Epoch

Practical 2.4
Assumption of Linear Regression

1. Import Libraries
[Syntax]
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import scipy.stats as stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

2. Load Dataset
[Syntax]
```
housing = fetch_california_housing()
data_01 = pd.DataFrame(housing.data, columns=housing.feature_names)
data_01['MEDV'] = housing.target
```

3. Sample data and display statistics
[Syntax]
```
print(data_01.head())
print(data_01.describe(include="all").T)=
```

4. Features and target variable
[Syntax]
```
X = data_01.drop('MEDV', axis=1).values
y = data_01['MEDV'].values
```

5. Train-Test Split
[Syntax]
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1)
```

6. Model Fitting
[Syntax]
```
model = LinearRegression()
```

```
model.fit(X_train, y_train)
```

7. Predictions & Residuals

[Syntax]

```
y_pred = model.predict(X_test)
residual = y_test - y_pred
```

A. Linearity

[Syntax]

```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(y_pred, y_test)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.title("Predicted vs Actual")
plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")

data_01.corr()
```

B. Multicollinearity

[Syntax]

```
vif = []

for i in range(X_train.shape[1]):
   vif.append(variance_inflation_factor(X_train, i))

pd.DataFrame({'vif': vif}, index=data_01.columns[0:8]).T

correlation_matrix = data_01.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

C. Normality of Residuals

[Syntax]

```
plt.figure(figsize=(6, 4))
sns.histplot(residual, kde=True)
plt.title("Residuals Distribution")
```

```
plt.xlabel("Residuals")
plt.show()

plt.figure(figsize=(6, 4))
stats.probplot(residual, dist="norm", plot=plt)
plt.title("QQ Plot of Residuals")
plt.show()
```

D. Homoscedasticity

[Syntax]

```
plt.figure(figsize=(6, 4))
plt.scatter(y_pred, residual)
plt.axhline(0, color='red', linestyle='--')
plt.title("Residuals vs Predicted")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.tight_layout()
plt.show()
```

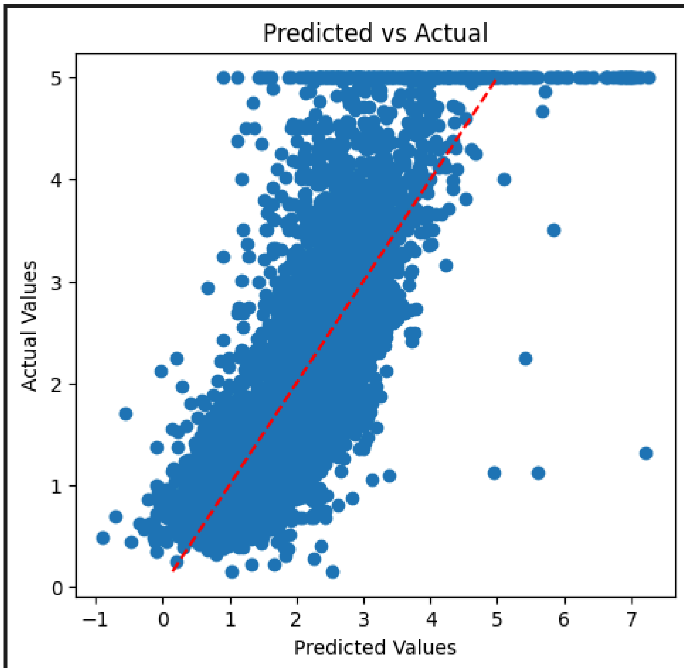
E. Auto Correlation of Residuals

[Syntax]

```
dw_statistic = durbin_watson(residual)
print("Durbin-Watson Statistic:", dw_statistic)

if dw_statistic < 1:
   print("Positive autocorrelation detected.")
elif dw_statistic > 3:
   print("Negative autocorrelation detected.")
else:
   print("No autocorrelation detected.")

plt.figure(figsize=(10, 5))
plt.plot(residual, marker='o', linestyle='none', color='blue')
plt.axhline(0, color='red', linestyle='--', linewidth=2)
plt.title("Residuals Plot")
plt.xlabel("Observation Index")
plt.ylabel("Residuals")
plt.grid(True)
plt.show()
```

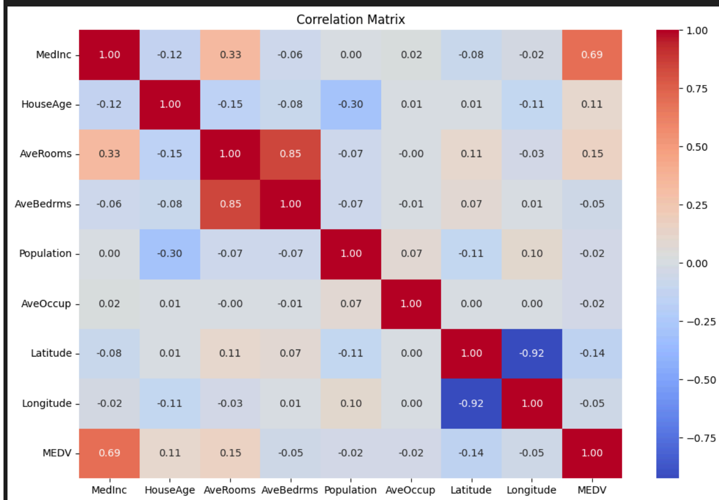
[Output]

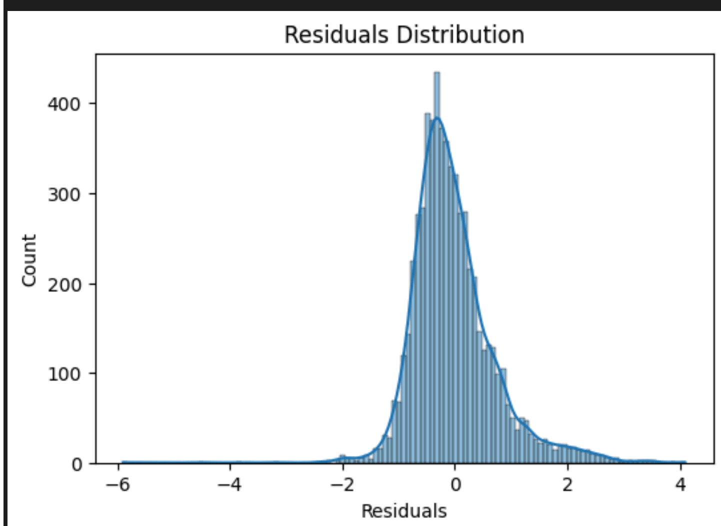
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude   MEDV
0    -122.23  4.526
1    -122.22  3.585
2    -122.24  3.521
3    -122.25  3.413
4    -122.25  3.422
              count         mean          std         min          25%  \
MedInc      20640.0     3.870671     1.899822    0.499900     2.563400
HouseAge    20640.0    28.639486    12.585558    1.000000    18.000000
AveRooms    20640.0     5.429000     2.474173    0.846154     4.440716
AveBedrms   20640.0     1.096675     0.473911    0.333333     1.006079
Population  20640.0  1425.476744  1132.462122    3.000000   787.000000
AveOccup    20640.0     3.070655    10.386050    0.692308     2.429741
Latitude    20640.0    35.631861     2.135952   32.540000    33.930000
Longitude   20640.0  -119.569704     2.003532 -124.350000 -121.800000
MEDV        20640.0     2.068558     1.153956    0.149990     1.196000

                    50%          75%           max
MedInc         3.534800     4.743250     15.000100
HouseAge      29.000000    37.000000     52.000000
AveRooms       5.229129     6.052381    141.909091
AveBedrms      1.048780     1.099526     34.066667
Population  1166.000000  1725.000000  35682.000000
AveOccup       2.818116     3.282261   1243.333333
Latitude      34.260000    37.710000     41.950000
Longitude   -118.490000  -118.010000   -114.310000
MEDV           1.797000     2.647250      5.000010
```

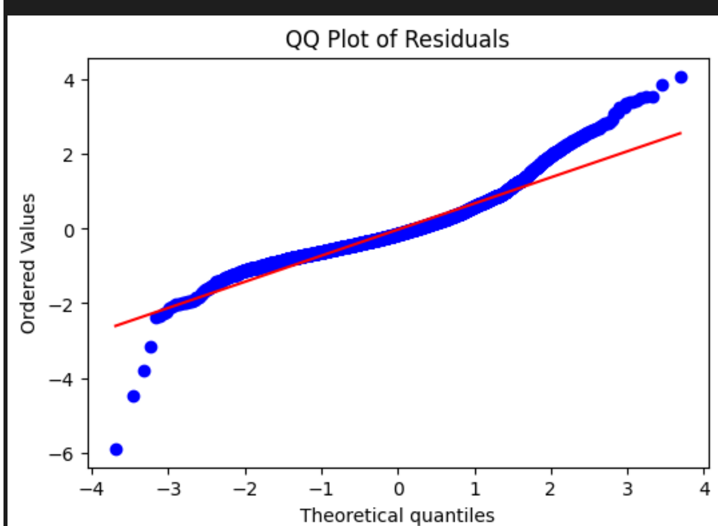


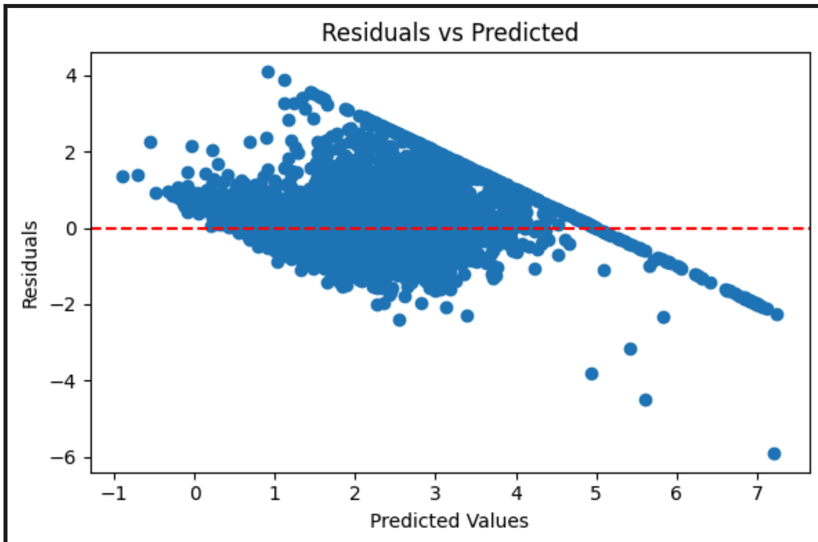

```
     MedInc      HouseAge    AveRooms    AveBedrms   Population  AveOccup
Latitude    Longitude   MEDV
MedInc      1.000000    -0.119034   0.326895    -0.062040   0.004834
0.018766    -0.079809   -0.015176   0.688075
HouseAge    -0.119034   1.000000    -0.153277   -0.077747   -0.296244
0.013191    0.011173    -0.108197   0.105623
AveRooms    0.326895    -0.153277   1.000000    0.847621    -0.072213
-0.004852   0.106389    -0.027540   0.151948
AveBedrms   -0.062040   -0.077747   0.847621    1.000000    -0.066197
-0.006181   0.069721    0.013344    -0.046701
Population  0.004834    -0.296244   -0.072213   -0.066197   1.000000
0.069863    -0.108785   0.099773    -0.024650
AveOccup    0.018766    0.013191    -0.004852   -0.006181   0.069863
1.000000    0.002366    0.002476    -0.023737
Latitude    -0.079809   0.011173    0.106389    0.069721    -0.108785
0.002366    1.000000    -0.924664   -0.144160
Longitude   -0.015176   -0.108197   -0.027540   0.013344    0.099773
0.002476    -0.924664   1.000000    -0.045967
MEDV   0.688075    0.105623    0.151948    -0.046701   -0.024650   -0.023737
-0.144160   -0.045967   1.000000


MedInc      HouseAge    AveRooms    AveBedrms   Population  AveOccup
Latitude    Longitude
```

```
vif   11.486592   7.165424   45.047286   41.542747   2.951747   1.073956
559.704788  630.756527
```

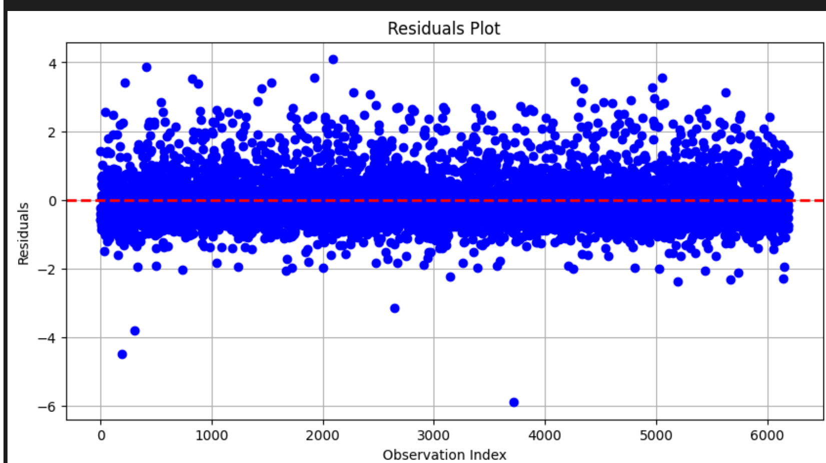









```
Durbin-Watson Statistic: 1.957776782949295
No autocorrelation detected.
```



Practical 2.5
Polynomial Regression

1. Import Libraries

[Syntax]

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.linear_model import LinearRegression
```

2. Load Dataset

[Syntax]

```
data_01 = pd.read_csv("Data.csv")
```

3. Sample data and display statistics

[Syntax]

```
print(data_01.sample(5))
print(data_01.describe(include="all").T)
```

4. Features and target variable

[Syntax]

```
X = data_01["Temperature"].values.reshape(-1, 1)
Y = data_01["Revenue"].values.reshape(-1, 1)
```

5. Transform features for polynomial regression

[Syntax]

```
 regression_polynomial = PolynomialFeatures(degree=2)
X_poly = regression_polynomial.fit_transform(X)
```

6. Train-Test Split

[Syntax]

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=25)
```

7. Model Fitting

[Syntax]

```
regression = LinearRegression()
regression.fit(X_train, Y_train)
```

8. Model coefficients

[Syntax]

```
print("Coefficients:", regression.coef_)
print("Intercept:", regression.intercept_)
```

9. Predictions

[Syntax]

```
Y_estimate = regression.predict(X_test)
print("Predicted Values:", Y_estimate.flatten())
```

10. Compare real and estimated values

[Syntax]

```
data_02 = pd.DataFrame()
data_02["Real_Value"] = Y_test.values
data_02["Estimated_Value"] = Y_estimate.flatten()
data_02['Error'] = data_02["Real_Value"] - data_02["Estimated_Value"]
print(data_02)
```

11. Performance

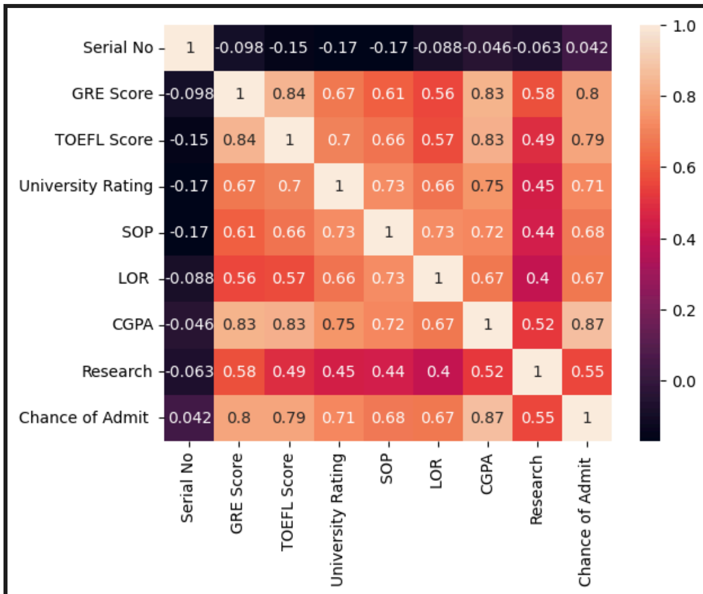
[Syntax]

```
print("Mean Squared Error:", mean_squared_error(Y_test, Y_estimate))
print("Mean Absolute Error:", mean_absolute_error(Y_test, Y_estimate))
print("Root Mean Squared Error:", np.sqrt(mean_squared_error(Y_test,
Y_estimate)))
print("R-Squared:", r2_score(Y_test, Y_estimate))
```

[Output]

```
     Serial No  GRE Score  TOEFL Score  University Rating  SOP  LOR
CGPA  \
373        374        321          109                  3  3.0  3.0
8.54
282        283        312          106                  3  4.0  3.5
8.79
```

```
327         328        295          101                  2   2.5   2.0
7.86
4             5        314          103                  2   2.0   3.0
8.21
91           92        299           97                  3   5.0   3.5
7.66

     Research  Chance of Admit
373         1             0.79
282         1             0.81
327         0             0.69
4           0             0.65
91          0             0.38
                   count        mean         std     min      25%      50%
\
Serial No          400.0  200.500000  115.614301    1.00   100.75   200.50
GRE Score          400.0  316.807500   11.473646  290.00   308.00   317.00
TOEFL Score        400.0  107.410000    6.069514   92.00   103.00   107.00
University Rating  400.0    3.087500    1.143728    1.00     2.00     3.00
 SOP               400.0    3.400000    1.006869    1.00     2.50     3.50
LOR                400.0    3.452500    0.898478    1.00     3.00     3.50
CGPA               400.0    8.598925    0.596317    6.80     8.17     8.61
Research           400.0    0.547500    0.498362    0.00     0.00     1.00
Chance of Admit    400.0    0.724350    0.142609    0.34     0.64     0.73

                        75%     max
Serial No          300.2500  400.00
GRE Score          325.0000  340.00
TOEFL Score        112.0000  120.00
University Rating    4.0000    5.00
 SOP                 4.0000    5.00
LOR                  4.0000    5.00
CGPA                 9.0625    9.92
Research             1.0000    1.00
Chance of Admit      0.8300    0.97
```




```
Coefficients: [-4.61204926e+08  9.63841115e-03  8.48802370e-02
3.49333180e-02
 -3.57735087e-01  2.41868538e-01  1.14172908e-01 -2.41421089e-01
  1.48331642e-05 -1.29377743e-04 -6.36452530e-04  5.43715810e-04
  1.50061060e-05 -5.98257741e-04  9.53952741e-04 -7.85099274e-05
  2.20904430e-03  4.97761510e-03 -1.71752976e-03 -4.52522183e-03
 -3.97233863e-03 -1.67979412e-02  3.24909602e-02 -4.48949870e-03
 -6.89314586e-03  6.90820769e-03 -1.78899911e-02  2.37966525e-03
 -3.84182374e-02 -5.19501774e-03  1.25778015e-02 -1.53001399e-02
  1.70435373e-02  4.91040830e-02  6.76016906e-02 -2.41421089e-01]
Intercept: 461204919.68414897
Predicted Values: [0.92182332 0.69911849 0.76735204 0.55700815 0.71707332
0.88450819
 0.94533765 0.61021364 0.69978797 0.96659875 0.74417287 0.63984334
 0.53913128 0.92391634 0.76869786 0.7174592  0.52616566 0.90170252
 0.63663942 0.59746063 0.85731584 0.91891223 0.8373552  0.85190469
 0.71863109 0.88326538 0.6361379  0.73554766 0.63859326 0.62970573
 0.89019692 0.71184933 0.46898484 0.52312809 0.68243605 0.60161269
 0.77754545 0.82546753 0.84736818 0.6968922  0.67864412 0.59458458
 0.86340594 0.58457166 0.86349487 0.68412906 0.73205972 0.47605819
 0.82050544 0.84421331 0.54516119 0.78876597 0.81420588 0.73109299
 0.79874343 0.67475927 0.78017986 0.89402753 0.85176897 0.56221783
 0.92983812 0.67036545 0.8965041  0.71689111 0.69456911 0.84982622
 0.63354081 0.73174614 0.64363354 0.81190324 0.64653605 0.89398277
 0.75714785 0.94957292 0.64813936 0.51450735 0.86549544 0.63142854
 0.54087317 0.81810719]
```

```
    Real_Value  Estimated_Value     Error
0         0.94         0.921823  0.018177
1         0.67         0.699118 -0.029118
2         0.81         0.767352  0.042648
3         0.38         0.557008 -0.177008
4         0.70         0.717073 -0.017073
..         ...              ...       ...
75        0.45         0.514507 -0.064507
76        0.78         0.865495 -0.085495
77        0.62         0.631429 -0.011429
78        0.64         0.540873  0.099127
79        0.87         0.818107  0.051893

[80 rows x 3 columns]
Mean Squared Error: 0.0049331992127593215
Mean Absolute Error: 0.05381584253907203
Root Mean Squared Error: 0.07023673691708152
R-Squared: 0.797847536605426
```

Practical 3.1
Ridge Regression (L2 Regularization)

1. Import Libraries

[Syntax]

```
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error,
mean_squared_error
import matplotlib.pyplot as plt
```

A. Single Feature

2. Generate Synthetic Data

[Syntax]

```
X, y = make_regression(n_samples=100, n_features=1, noise=0.1,
random_state=42)
```

3. Scatter Plot

[Syntax]

```
plt.scatter(X, y)
plt.xlabel("Feature 1")
```

```
plt.ylabel("Target")
plt.title("Scatter Plot of Feature 1 vs Target")
plt.show()
```

4. Train-Test Split

[Syntax]

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

5. Model Fitting

[Syntax]

```
ridge = Ridge(alpha=0)
ridge.fit(X_train, y_train)
```

6. Predictions

[Syntax]

```
y_pred = ridge.predict(X_test)
```

7. Performance

[Syntax]

```
print(f"R2 Score: {r2_score(y_test, y_pred)}")
print(f"Mean Absolute Error: {mean_absolute_error(y_test, y_pred)}")
print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred)}")
print(f"Coefficients: {ridge.coef_}")
print(f"Intercept: {ridge.intercept_}")
```

8. Visualize

[Syntax]

```
plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, linewidth=2, label='Model')
plt.title('Ridge Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```

B. Multiple Feature

2. Generate Synthetic Data

[Syntax]

```
X, y = make_regression(n_samples=100, n_features=10, noise=0.1,
random_state=42)
```

3. Scatter Plot

[Syntax]

```
num_features = X.shape[1]
fig, axs = plt.subplots(4, 3, figsize=(15, 12))
axs = axs.flatten()
for i in range(num_features):
   axs[i].scatter(X[:, i], y)
   axs[i].set_xlabel(f"Feature {i}")
   axs[i].set_ylabel("Target")
   axs[i].set_title(f"Scatter Plot of Feature {i} vs Target")

for j in range(num_features, len(axs)):
   axs[j].axis('off')

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

4. Train-Test Split

[Syntax]

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

5. Model Fitting

[Syntax]

```
ridge_models = {
   'Model 01': Ridge(alpha=0),
   'Model 02': Ridge(alpha=10),
   'Model 03': Ridge(alpha=100),
   'Model 04': Ridge(alpha=1000)
}
```

6. Predictions

[Syntax]

```
predictions = {}
for name, model in ridge_models.items():
   model.fit(X_train, y_train)
   predictions[name] = model.predict(X_test)
   print(f"{name} - R2 Score: {r2_score(y_test, predictions[name])}")
   print(f"{name} - Mean Absolute Error: {mean_absolute_error(y_test,
predictions[name])}")
   print(f"{name} - Mean Squared Error: {mean_squared_error(y_test,
predictions[name])}")
   print(f"{name} - Coefficients: {model.coef_}")
   print(f"{name} - Intercept: {model.intercept_}\n")
```

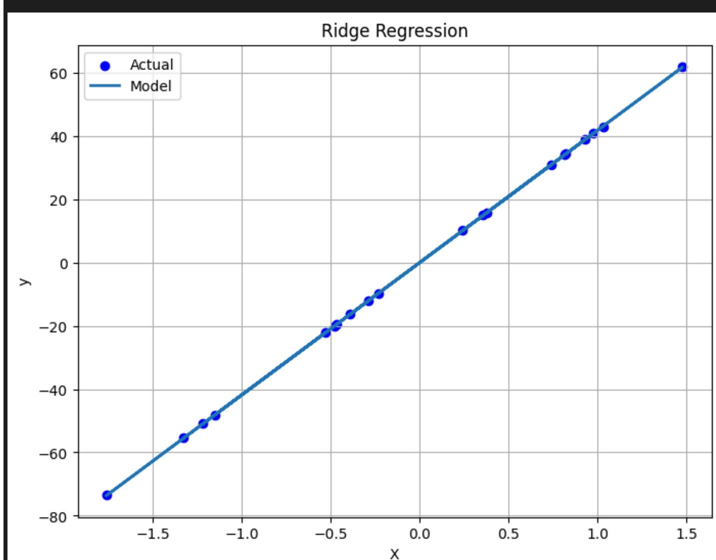
8. Visualize

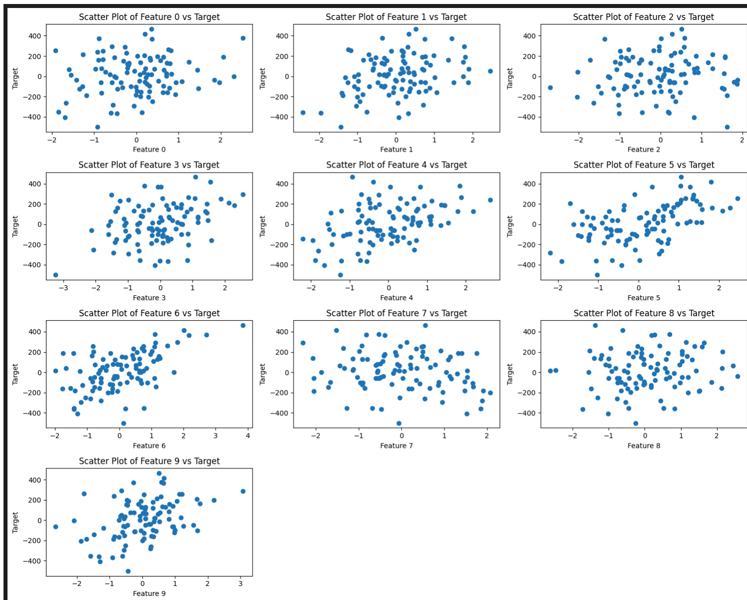
[Syntax]

```
num_features = X_test.shape[1]
fig, axs = plt.subplots(4, 3, figsize=(15, 12))
axs = axs.flatten()

for i in range(num_features):
   axs[i].scatter(X_test[:, i], y_test, color='blue', label='Actual',
alpha=0.5)
   for name, pred in predictions.items():
       axs[i].scatter(X_test[:, i], pred, linewidth=2, label=name,
alpha=0.7)
   axs[i].set_title(f'Ridge Regression - Feature {i}')
   axs[i].set_xlabel(f'Feature {i}')
   axs[i].set_ylabel('y')
   axs[i].legend()
   axs[i].grid(True)

for j in range(num_features, len(axs)):
   axs[j].axis('off')
plt.tight_layout()
plt.show()
```

[Output]

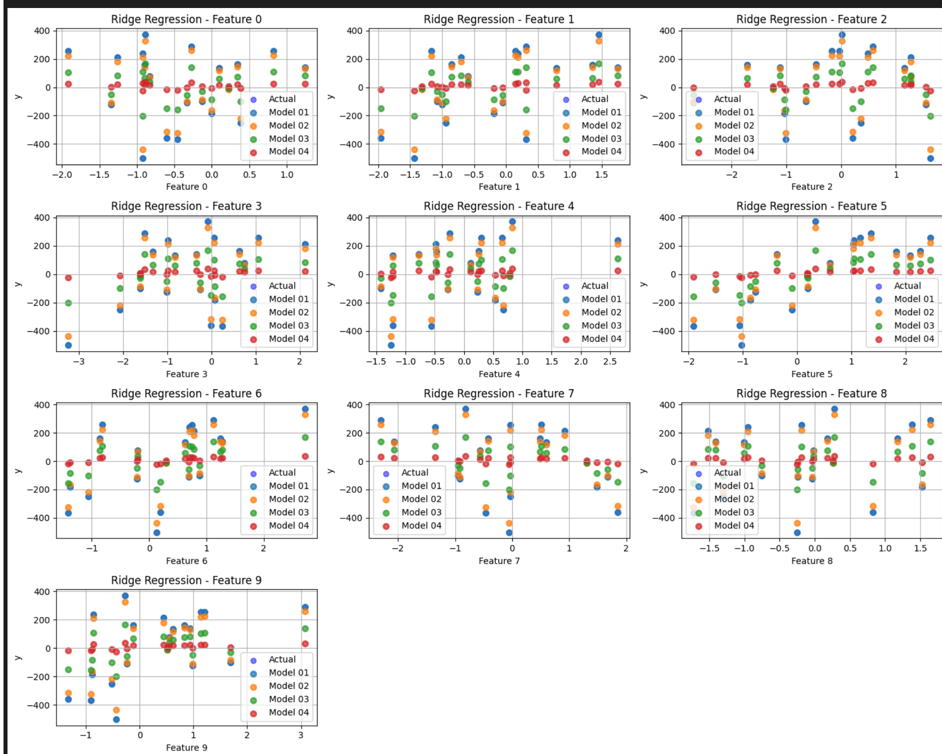

```
R2 Score: 0.9999925261586983
Mean Absolute Error: 0.08416659922208973
Mean Squared Error: 0.010420222653186813
Coefficients: [41.76613113]
Intercept: 0.00099222142259358
```






```
Model 01 - R2 Score: 0.9999998282331861
Model 01 - Mean Absolute Error: 0.07556674834423119
Model 01 - Mean Squared Error: 0.010265673458289933
Model 01 - Coefficients: [16.7712358  54.13782324  5.18097686 63.64362199
93.61309994 70.63686589
 87.0713662  10.43882574  3.15690876 70.90887261]
Model 01 - Intercept: 0.017550255465828002

Model 02 - R2 Score: 0.9847661835848986
Model 02 - Mean Absolute Error: 25.997196039460448
Model 02 - Mean Squared Error: 910.4516831231938
Model 02 - Coefficients: [17.88856321 46.40514956  4.89279765 55.50587225
80.80473461 62.39518869
 77.68820784  6.06125249  3.8243928  60.04934268]
Model 02 - Intercept: 0.862136232402575

Model 03 - R2 Score: 0.6691558494864291
Model 03 - Mean Absolute Error: 122.1797251420646
Model 03 - Mean Squared Error: 19772.95810050228
Model 03 - Coefficients: [13.45753426 19.76892349  3.55530702 26.6822633
36.85627286 30.42172633
 40.49841771 -3.25683516  3.4735336  25.7645111 ]
Model 03 - Intercept: 5.204682254676274

Model 04 - R2 Score: 0.13676508085065853
Model 04 - Mean Absolute Error: 198.827578913156
```

```
Model 04 - Mean Squared Error: 51591.38482797587
Model 04 - Coefficients: [ 2.71907595  2.84893518  0.81003631  4.47516966
5.82111316  5.02273078
  7.17899711 -1.48023703  0.73880394  3.90280059]
Model 04 - Intercept: 9.307377723619343
```



Practical 3.2
Lasso Regression (L1 Regularization)

1. Import Libraries

[Syntax]

```
import numpy as np
from sklearn.linear_model import Lasso
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error,
mean_squared_error
import matplotlib.pyplot as plt
```

<u>A. Single Feature</u>

2. Generate Synthetic Data

[Syntax]

```
X, y = make_regression(n_samples=100, n_features=1, noise=0.1,
random_state=42)
```

3. Scatter Plot

[Syntax]

```
plt.scatter(X, y)
plt.xlabel("Feature 1")
plt.ylabel("Target")
plt.title("Scatter Plot of Feature 1 vs Target")
plt.show()
```

4. Train-Test Split

[Syntax]

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

5. Model Fitting

[Syntax]

```
lasso = Lasso(alpha=0)
lasso.fit(X_train, y_train)
```

6. Predictions

[Syntax]

```
y_pred = lasso.predict(X_test)
```

7. Performance

[Syntax]

```
print(f"R2 Score: {r2_score(y_test, y_pred)}")
print(f"Mean Absolute Error: {mean_absolute_error(y_test, y_pred)}")
print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred)}")
print(f"Coefficients: {ridge.coef_}")
print(f"Intercept: {ridge.intercept_}")
```

8. Visualize

[Syntax]

```
plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, linewidth=2, label='Model')
```

```
plt.title('Lasso Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```

B. Multiple Feature

2. Generate Synthetic Data

[Syntax]

```
X, y = make_regression(n_samples=100, n_features=10, noise=0.1,
random_state=42)
```

3. Scatter Plot

[Syntax]

```
num_features = X.shape[1]
fig, axs = plt.subplots(4, 3, figsize=(15, 12))
axs = axs.flatten()
for i in range(num_features):
   axs[i].scatter(X[:, i], y)
   axs[i].set_xlabel(f"Feature {i}")
   axs[i].set_ylabel("Target")
   axs[i].set_title(f"Scatter Plot of Feature {i} vs Target")

for j in range(num_features, len(axs)):
   axs[j].axis('off')

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

4. Train-Test Split

[Syntax]

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

5. Model Fitting

[Syntax]

```
lasso_models = {
   'Model 01': Lasso(alpha=0),
```

```
    'Model 02': Lasso(alpha=10),
    'Model 03': Lasso(alpha=100),
    'Model 04': Lasso(alpha=1000)
}
```

6. Predictions

[Syntax]

```
predictions = {}
for name, model in ridge_models.items():
    model.fit(X_train, y_train)
    predictions[name] = model.predict(X_test)
    print(f"{name} - R2 Score: {r2_score(y_test, predictions[name])}")
    print(f"{name} - Mean Absolute Error: {mean_absolute_error(y_test,
predictions[name])}")
    print(f"{name} - Mean Squared Error: {mean_squared_error(y_test,
predictions[name])}")
    print(f"{name} - Coefficients: {model.coef_}")
    print(f"{name} - Intercept: {model.intercept_}\n")
```

8. Visualize

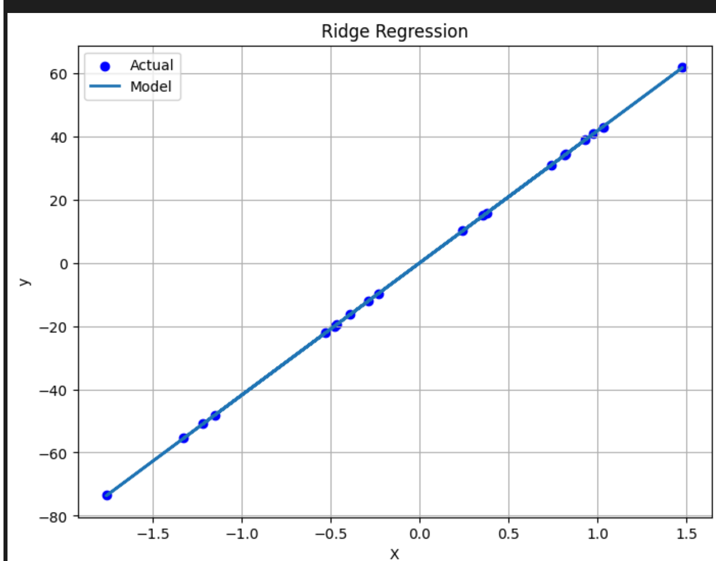
[Syntax]

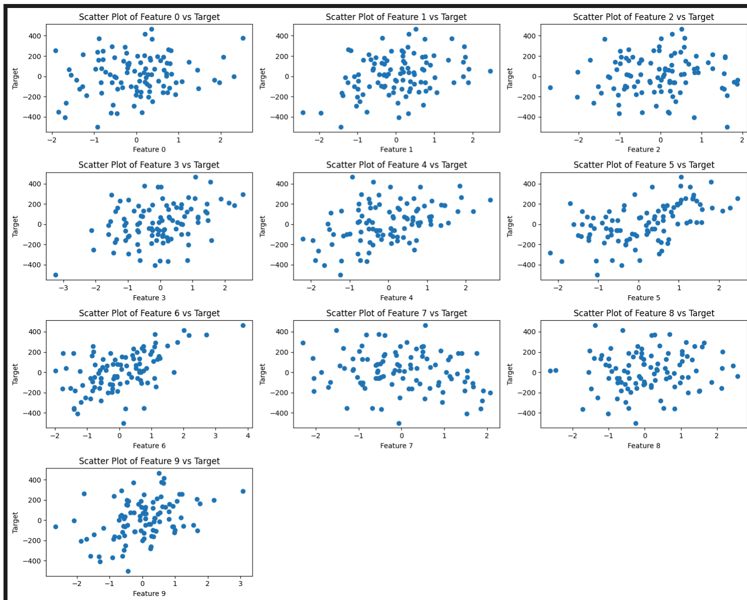
```
num_features = X_test.shape[1]
fig, axs = plt.subplots(4, 3, figsize=(15, 12))
axs = axs.flatten()

for i in range(num_features):
    axs[i].scatter(X_test[:, i], y_test, color='blue', label='Actual',
alpha=0.5)
    for name, pred in predictions.items():
        axs[i].scatter(X_test[:, i], pred, linewidth=2, label=name,
alpha=0.7)
    axs[i].set_title(f'Lasso Regression - Feature {i}')
    axs[i].set_xlabel(f'Feature {i}')
    axs[i].set_ylabel('y')
    axs[i].legend()
    axs[i].grid(True)

for j in range(num_features, len(axs)):
    axs[j].axis('off')
```

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

[Output]

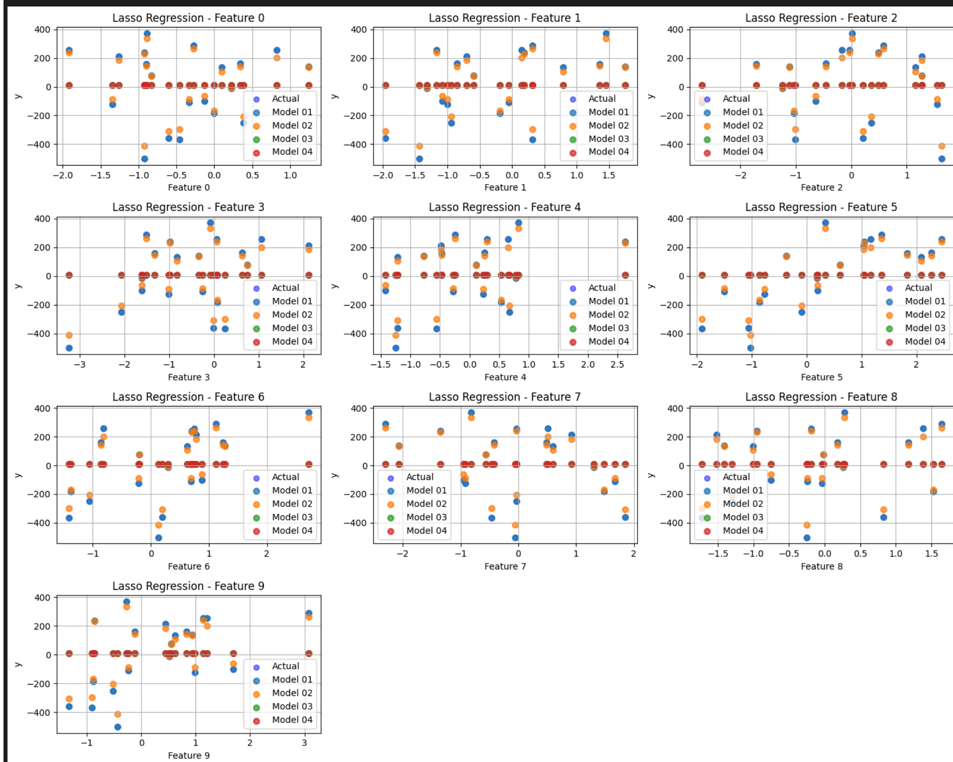

```
R2 Score: 0.9999925261586983
Mean Absolute Error: 0.08416659922208973
Mean Squared Error: 0.010420222653186813
Coefficients: [41.76613113]
Intercept: 0.000992222142259358
```






```
Model 01 - R2 Score: 0.9999998248721161
Model 01 - Mean Absolute Error: 0.07658870727765103
Model 01 - Mean Squared Error: 0.010466548388298226
Model 01 - Coefficients: [16.77220561 54.14087025  5.17899466 63.64494836
93.61215072 70.63803909
 87.06972731 10.43763963  3.15682746 70.90827124]
Model 01 - Intercept: 0.016884012285636663

Model 02 - R2 Score: 0.9759087693486258
Model 02 - Mean Absolute Error: 30.870482861284188
Model 02 - Mean Squared Error: 1439.8165828826345
Model 02 - Coefficients: [ 7.88357416 41.43085042  0.          52.6042667
80.53726745 59.91064109
 77.47428958  0.          0.          58.55025218]
Model 02 - Intercept: 3.0425193637953054

Model 03 - R2 Score: -0.00030940031275816793
Model 03 - Mean Absolute Error: 214.31953802951338
Model 03 - Mean Squared Error: 59783.66499519361
Model 03 - Coefficients: [ 0.   0.   0.   0.   0.   0.   0. -0.   0.   0.]
Model 03 - Intercept: 10.153184707338482

Model 04 - R2 Score: -0.00030940031275816793
Model 04 - Mean Absolute Error: 214.31953802951338
Model 04 - Mean Squared Error: 59783.66499519361
Model 04 - Coefficients: [ 0.   0.   0.   0.   0.   0.   0. -0.   0.   0.]
```

```
Model 04 - Intercept: 10.153184707338482
```



Practical 4.1
Logistic Regression

1. Import Libraries

[Syntax]

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

2. Load Dataset

[Syntax]

```
data = pd.read_csv("insurance_data.csv")
```

3. Sample data and display statistics

[Syntax]

```
data_01.sample(5)
data.info()
data.describe().T
```

4. Data Visualization

[Syntax]

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='age', y='bought_insurance', hue='bought_insurance',
data=data, palette='viridis')
plt.xlabel('Age')
plt.ylabel('Bought Insurance')
plt.title('Scatter Plot of Age vs Bought Insurance')
plt.legend(title='Category')
plt.show()
```

5. Features and target variable

[Syntax]

```
X = data['age'].values.reshape(-1, 1)
y = data['bought_insurance'].values.reshape(-1, 1)
```

6. Linear Regression

[Syntax]

```
linear_model = LinearRegression()
linear_model.fit(X, y)
```

7. Linear Regression Plot
[Syntax]

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='age', y='bought_insurance', hue='bought_insurance',
data=data, palette='viridis')
plt.plot(X, linear_model.predict(X), color='blue', linewidth=2,
label='Linear Regression')
plt.xlabel('Age')
plt.ylabel('Bought Insurance')
plt.title('Linear Regression of Age vs Bought Insurance')
plt.legend(title='Category')
plt.show()
```

8. Logistic Regression
[Syntax]

```
logistic_model = LogisticRegression()
logistic_model.fit(X, y.ravel())
```

9. Probability Predictions for Logistic Model
[Syntax]

```
x_values = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
predictions = logistic_model.predict_proba(x_values)[:, 1]
```

10. Logistic Regression Plot
[Syntax]

```
x_values = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
predictions = logistic_model.predict_proba(x_values)[:, 1]
```

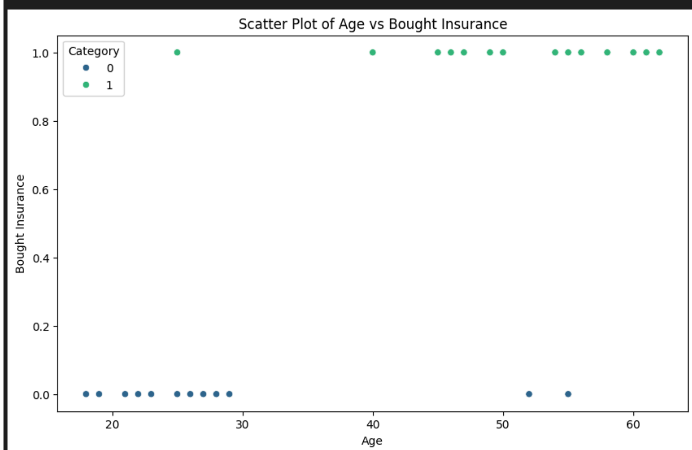
11. Prediction for a 25-Year-Old
[Syntax]

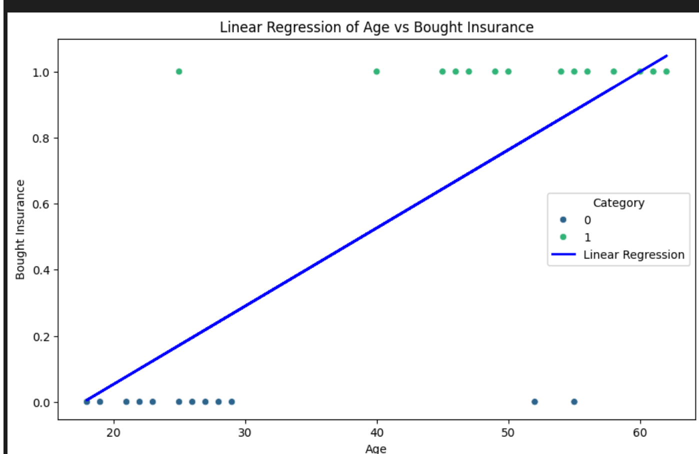
```
prob_25 = logistic_model.predict_proba([[25]])[0, 1]
print(f"Probability of a 25-year-old buying insurance: {prob_25:.2f}")
```

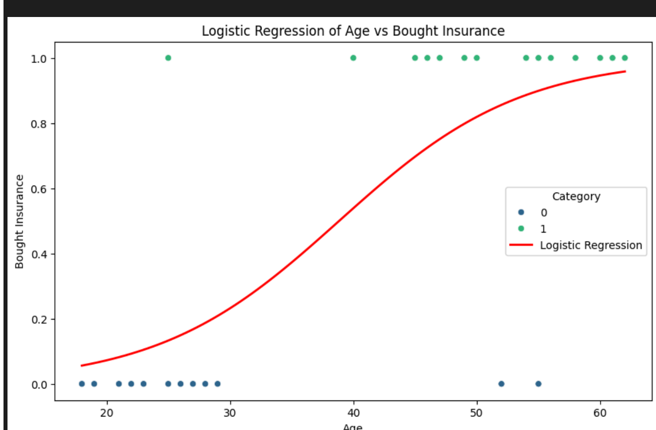
[Output]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27 entries, 0 to 26
```

```
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   age               27 non-null     int64
 1   bought_insurance  27 non-null     int64
dtypes: int64(2)
memory usage: 560.0 bytes
```







```
Probability of a 25-year-old buying insurance: 0.13
```

Practical 4.2
Decision Tree

1. Import Libraries

[Syntax]

```
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold,
cross_val_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report
```

2. Function for Entropy

[Syntax]

```
def entropy(proportion):
   if proportion in [0, 1]:
       return 0
   negative = 1 - proportion
   return -((proportion) * math.log2(proportion)) - ((negative) *
math.log2(negative))
```

3. Calculate Information Gain

[Syntax]

```
parent = entropy(9 / 14)
outlook = (5 / 14) * entropy(2 / 5) + (5 / 14) * entropy(3 / 5)
temperature = (4 / 14) * entropy(2 / 4) + (6 / 14) * entropy(4 / 6) + (4 /
14) * entropy(3 / 4)
humidity = (7 / 14) * entropy(3 / 7) + (7 / 14) * entropy(6 / 7)
wind = (8 / 14) * entropy(6 / 8) + (6 / 14) * entropy(3 / 6)
print(f"IG(S, Outlook) = {parent - outlook}")
print(f"IG(S, Temperature) = {parent - temperature}")
print(f"IG(S, Humidity) = {parent - humidity}")
print(f"IG(S, Wind) = {parent - wind}")
```

4. Function to Remove Outliers Using IQR

[Syntax]

```
def remove_outliers_iqr(df):
   df_no_outliers = df.copy()
   for col in df.select_dtypes(include=[np.number]).columns:
       Q1 = df[col].quantile(0.25)
       Q3 = df[col].quantile(0.75)
       IQR = Q3 - Q1
       lower_bound = Q1 - 1.5 * IQR
       upper_bound = Q3 + 1.5 * IQR
       df_no_outliers = df_no_outliers[(df_no_outliers[col] >=
lower_bound) & (df_no_outliers[col] <= upper_bound)]
   return df_no_outliers
```

5. Load Dataset

[Syntax]

```
data =
pd.read_csv("https://raw.githubusercontent.com/kishan0725/Breast-Cancer-Wi
sconsin-Diagnostic/master/data.csv")
df = remove_outliers_iqr(data)
```

6. Data Exploration

[Syntax]

```
print(df.shape)
print(df.sample(10))
print(df.info())
print(df.isna().sum())
print(df.describe(include="all"))
```

7. Standardization

[Syntax]

```
numeric_cols = df.select_dtypes(include=['number']).columns
scaler = StandardScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

8. Features and target variable

[Syntax]

```
X = df.drop(['id','diagnosis','Unnamed: 32'], axis=1)
y = df['diagnosis'].astype('category')
```

9. Train-Test Split

[Syntax]

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=14)
```

10. Cross-Validation Setup

[Syntax]

```
kf = KFold(n_splits=10, shuffle=True, random_state=42)
```

11. Decision Tree Classifier

[Syntax]

```
model = DecisionTreeClassifier(criterion="entropy", random_state=42)
model.fit(X_train, y_train)
```

12. Predictions and Evaluation

[Syntax]

```
y_pred = model.predict(X_test)
print(f"Accuracy of Decision Tree: {accuracy_score(y_test, y_pred)}")
print(f"Classification Report of Decision
Tree:\n{classification_report(y_test, y_pred)}")
```

13. Cross-Validation Scores

[Syntax]

```
scores = cross_val_score(model, X, y, cv=kf, scoring='accuracy')
print(f"Accuracy scores for each fold of Decision Tree: {scores}")
print(f"Mean accuracy for folds of Decision Tree: {np.mean(scores)}")
print(f"Standard deviation for folds of Decision Tree: {np.std(scores)}")
```

14. Decision Tree Visualization

[Syntax]

```
plt.figure(figsize=(14, 10))
plot_tree(model, filled=True, feature_names=X.columns,
class_names=["Malignant", "Benign"])
plt.title("Decision Tree Visualization", fontsize=18)
plt.show()
```

[Output]

```
IG(S, Outlook) = 0.24674981977443933
IG(S, Temperature) = 0.02922256565895487
IG(S, Humidity) = 0.15183550136234159
```

```
IG(S, Wind) = 0.048127030408269490
(569, 33)
         id diagnosis  radius_mean  texture_mean  perimeter_mean
area_mean  \
406  905189         B        16.14         14.86          104.30
800.0
120  865137         B        11.41         10.82           73.34
403.3
158  871122         B        12.06         12.74           76.84
448.6
385   90291         M        14.60         23.29           93.97
664.7
492  914062         M        18.01         20.56          118.40
1007.0

     smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
406          0.09495           0.08501         0.05500
0.04528
120          0.09373           0.06685         0.03512
0.02623
158          0.09311           0.05241         0.01972
0.01963
385          0.08682           0.06636         0.08390
0.05271
492          0.10010           0.12890         0.11700
0.07762

     ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
406  ...          19.58           115.90       947.9            0.1206
120  ...          15.97            83.74       510.5            0.1548
158  ...          18.41            84.08       532.8            0.1275
385  ...          31.71           102.20       758.2            0.1312
492  ...          26.06           143.40      1426.0            0.1309

     compactness_worst  concavity_worst  concave points_worst
symmetry_worst  \
406             0.1722          0.23100               0.11290
0.2778
```

```
120            0.2390         0.21020              0.08958
0.3016
158            0.1232         0.08636              0.07025
0.2514
385            0.1581         0.26750              0.13590
0.2477
492            0.2327         0.25440              0.14890
0.3251

     fractal_dimension_worst  Unnamed: 32
406                  0.07012          NaN
120                  0.08523          NaN
158                  0.07898          NaN
385                  0.06836          NaN
492                  0.07625          NaN

[5 rows x 33 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
```

```
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
None
id                             0
diagnosis                      0
radius_mean                    0
texture_mean                   0
perimeter_mean                 0
area_mean                      0
smoothness_mean                0
compactness_mean               0
concavity_mean                 0
concave points_mean            0
symmetry_mean                  0
fractal_dimension_mean         0
radius_se                      0
texture_se                     0
perimeter_se                   0
area_se                        0
smoothness_se                  0
compactness_se                 0
concavity_se                   0
concave points_se              0
symmetry_se                    0
fractal_dimension_se           0
```

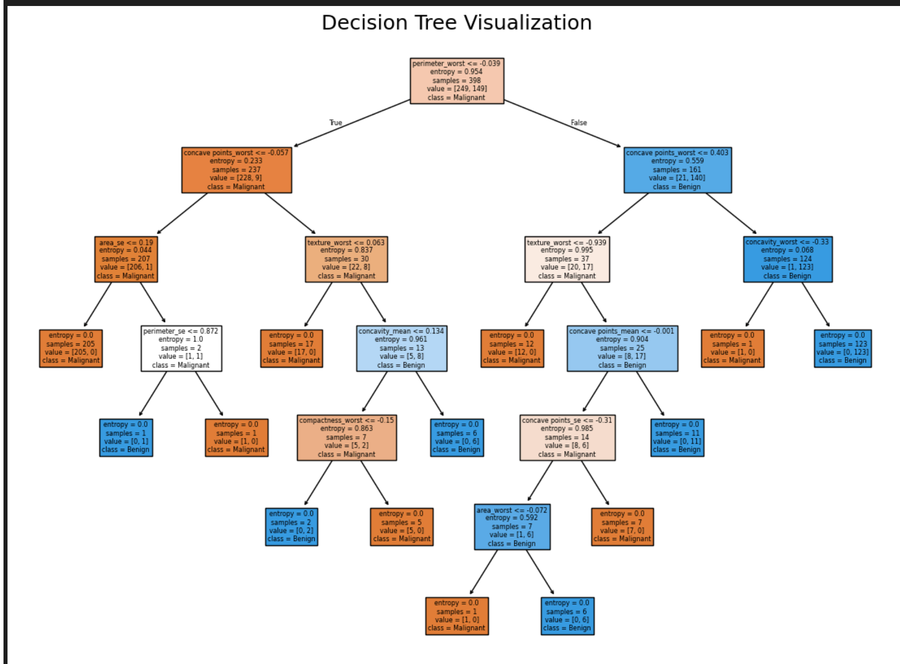
```
radius_worst                0
texture_worst               0
perimeter_worst             0
area_worst                  0
smoothness_worst            0
compactness_worst           0
concavity_worst             0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
Unnamed: 32               569
dtype: int64
                  id diagnosis  radius_mean  texture_mean  perimeter_mean  \
count   5.690000e+02       569   569.000000    569.000000      569.000000
unique           NaN         2          NaN           NaN             NaN
top              NaN         B          NaN           NaN             NaN
freq             NaN       357          NaN           NaN             NaN
mean    3.037183e+07       NaN    14.127292     19.289649       91.969033
std     1.250206e+08       NaN     3.524049      4.301036       24.298981
min     8.670000e+03       NaN     6.981000      9.710000       43.790000
25%     8.692180e+05       NaN    11.700000     16.170000       75.170000
50%     9.060240e+05       NaN    13.370000     18.840000       86.240000
75%     8.813129e+06       NaN    15.780000     21.800000      104.100000
max     9.113205e+08       NaN    28.110000     39.280000      188.500000

          area_mean  smoothness_mean  compactness_mean  concavity_mean  \
count    569.000000       569.000000        569.000000      569.000000
unique          NaN              NaN               NaN             NaN
top             NaN              NaN               NaN             NaN
freq            NaN              NaN               NaN             NaN
mean     654.889104         0.096360          0.104341        0.088799
std      351.914129         0.014064          0.052813        0.079720
min      143.500000         0.052630          0.019380        0.000000
25%      420.300000         0.086370          0.064920        0.029560
50%      551.100000         0.095870          0.092630        0.061540
75%      782.700000         0.105300          0.130400        0.130700
max     2501.000000         0.163400          0.345400        0.426800
```

```
       concave points_mean  ...  texture_worst  perimeter_worst
area_worst  \
count            569.000000  ...     569.000000       569.000000
569.000000
unique                  NaN  ...            NaN              NaN
NaN
top                     NaN  ...            NaN              NaN
NaN
freq                    NaN  ...            NaN              NaN
NaN
mean               0.048919  ...      25.677223       107.261213
880.583128
std                0.038803  ...       6.146258        33.602542
569.356993
min                0.000000  ...      12.020000        50.410000
185.200000
25%                0.020310  ...      21.080000        84.110000
515.300000
50%                0.033500  ...      25.410000        97.660000
686.500000
75%                0.074000  ...      29.720000       125.400000
1084.000000
max                0.201200  ...      49.540000       251.200000
4254.000000

       smoothness_worst  compactness_worst  concavity_worst  \
count        569.000000         569.000000       569.000000
unique              NaN                NaN              NaN
top                 NaN                NaN              NaN
freq                NaN                NaN              NaN
mean           0.132369           0.254265         0.272188
std            0.022832           0.157336         0.208624
min            0.071170           0.027290         0.000000
25%            0.116600           0.147200         0.114500
50%            0.131300           0.211900         0.226700
75%            0.146000           0.339100         0.382900
max            0.222600           1.058000         1.252000

       concave points_worst  symmetry_worst  fractal_dimension_worst  \
count            569.000000      569.000000               569.000000
```

```
unique                  NaN             NaN                        NaN
top                     NaN             NaN                        NaN
freq                    NaN             NaN                        NaN
mean               0.114606        0.290076                   0.083946
std                0.065732        0.061867                   0.018061
min                0.000000        0.156500                   0.055040
25%                0.064930        0.250400                   0.071460
50%                0.099930        0.282200                   0.080040
75%                0.161400        0.317900                   0.092080
max                0.291000        0.663800                   0.207500

        Unnamed: 32
count           0.0
unique          NaN
top             NaN
freq            NaN
mean            NaN
std             NaN
min             NaN
25%             NaN
50%             NaN
75%             NaN
max             NaN

[11 rows x 33 columns]
Accuracy of Random Forest: 0.9122807017543859
Classification Report of Random Forest:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93       108
           1       0.86      0.90      0.88        63

    accuracy                           0.91       171
   macro avg       0.90      0.91      0.91       171
weighted avg       0.91      0.91      0.91       171

/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1137:
RuntimeWarning: invalid value encountered in divide
  updated_mean = (last_sum + new_sum) / updated_sample_count
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1142:
RuntimeWarning: invalid value encountered in divide
  T = new_sum / new_sample_count
/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1162:
RuntimeWarning: invalid value encountered in divide
  new_unnormalized_variance -= correction**2 / new_sample_count
Accuracy scores for each fold of Random Forest: [0.94736842 0.9122807
0.92982456 0.96491228 0.92982456 0.92982456
 0.96491228 0.96491228 0.94736842 0.92857143]
Mean accuracy for folds of Random Forest: 0.9419799498746867
Standard deviation for folds of Random Forest: 0.01772241750965193
Feature ranking:
1. feature perimeter_worst (0.6177222195454725)
2. feature concave points_worst (0.17302969054802153)
3. feature texture_worst (0.07063304280903168)
4. feature concave points_se (0.02542004910012378)
5. feature concave points_mean (0.023220454975018)
6. feature concavity_worst (0.022100342780427273)
7. feature area_se (0.0187860604874988)
8. feature concavity_mean (0.016999135428820842)
9. feature compactness_worst (0.015912989483805783)
10. feature area_worst (0.010908421194353549)
11. feature perimeter_se (0.005267593647426239)
12. feature fractal_dimension_worst (0.0)
13. feature compactness_mean (0.0)
14. feature perimeter_mean (0.0)
15. feature smoothness_mean (0.0)
16. feature area_mean (0.0)
17. feature fractal_dimension_mean (0.0)
18. feature texture_mean (0.0)
19. feature symmetry_mean (0.0)
20. feature smoothness_se (0.0)
21. feature radius_se (0.0)
22. feature texture_se (0.0)
23. feature symmetry_worst (0.0)
24. feature compactness_se (0.0)
25. feature concavity_se (0.0)
26. feature symmetry_se (0.0)
27. feature fractal_dimension_se (0.0)
28. feature radius_worst (0.0)
```

```
29. feature smoothness_worst (0.0)
30. feature radius_mean (0.0)
```



Practical 4.3
Random Forest

1. Import Libraries
[Syntax]
```
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold,
cross_val_score
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

2. Function to Remove Outliers Using IQR
[Syntax]
```
def remove_outliers_iqr(df):
   df_no_outliers = df.copy()
   for col in df.select_dtypes(include=[np.number]).columns:
       Q1 = df[col].quantile(0.25)
       Q3 = df[col].quantile(0.75)
       IQR = Q3 - Q1
       lower_bound = Q1 - 1.5 * IQR
       upper_bound = Q3 + 1.5 * IQR
       df_no_outliers = df_no_outliers[(df_no_outliers[col] >=
lower_bound) & (df_no_outliers[col] <= upper_bound)]
   return df_no_outliers
```

3. Load Dataset
[Syntax]
```
data =
pd.read_csv("https://raw.githubusercontent.com/kishan0725/Breast-Cancer-Wi
sconsin-Diagnostic/master/data.csv")
df = remove_outliers_iqr(data)
```

4. Data Exploration
[Syntax]

```
print(df.shape)
print(df.sample(10))
print(df.info())
print(df.isna().sum())
print(df.describe(include="all"))
```

5. Standardization

[Syntax]

```
numeric_cols = df.select_dtypes(include=['number']).columns
scaler = StandardScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

6. Features and target variable

[Syntax]

```
X = df.drop(['id','diagnosis','Unnamed: 32'], axis=1)
y = df['diagnosis'].astype('category')
```

7. Train-Test Split

[Syntax]

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=14)
```

8. Cross-Validation Setup

[Syntax]

```
kf = KFold(n_splits=10, shuffle=True, random_state=42)
```

9. Random Forest Classifier

[Syntax]

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

10. Predictions and Evaluation

[Syntax]

```
y_pred = model.predict(X_test)
print(f"Accuracy of Decision Tree: {accuracy_score(y_test, y_pred)}")
print(f"Classification Report of Decision
Tree:\n{classification_report(y_test, y_pred)}")
```

11. Cross-Validation Scores

[Syntax]

```
scores = cross_val_score(model, X, y, cv=kf, scoring='accuracy')
print(f"Accuracy scores for each fold of Decision Tree: {scores}")
print(f"Mean accuracy for folds of Decision Tree: {np.mean(scores)}")
print(f"Standard deviation for folds of Decision Tree: {np.std(scores)}")
```

12. Decision Tree Visualization

[Syntax]

```
plt.figure(figsize=(14, 10))
plot_tree(model, filled=True, feature_names=X.columns,
class_names=["Malignant", "Benign"])
plt.title("Random Forest Visualization", fontsize=18)
plt.show()
```

[Output]

```
           id diagnosis  radius_mean  texture_mean  perimeter_mean
area_mean  \
280   8912049         M        19.16         26.60          126.20
1138.0
488    913512         B        11.68         16.17           75.49
420.5
30     853401         M        18.63         25.11          124.80
1088.0
247    884626         B        12.89         14.11           84.95
512.2
506  91544001         B        12.22         20.04           79.47
453.1

     smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
280           0.1020           0.14530         0.19210
0.09664
488           0.1128           0.09263         0.04279
0.03132
30            0.1064           0.18870         0.23190
0.12440
247           0.0876           0.13460         0.13740
0.03980
506           0.1096           0.11520         0.08175
0.02166
```

```
     ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
280  ...          35.90           159.80      1724.0            0.1782
488  ...          21.59            86.57       549.8            0.1526
30   ...          34.01           160.50      1670.0            0.1491
247  ...          17.70           105.00       639.1            0.1254
506  ...          24.17            85.13       515.3            0.1402

     compactness_worst  concavity_worst  concave points_worst
symmetry_worst  \
280             0.3841           0.5754               0.18720
0.3258
488             0.1477           0.1490               0.09815
0.2804
30              0.4257           0.6133               0.18480
0.3444
247             0.5849           0.7727               0.15610
0.2639
506             0.2315           0.3535               0.08088
0.2709

     fractal_dimension_worst  Unnamed: 32
280                  0.09720          NaN
488                  0.08024          NaN
30                   0.09782          NaN
247                  0.11780          NaN
506                  0.08839          NaN

[5 rows x 33 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   id                        569 non-null    int64
 1   diagnosis                 569 non-null    object
 2   radius_mean               569 non-null    float64
 3   texture_mean              569 non-null    float64
 4   perimeter_mean            569 non-null    float64
 5   area_mean                 569 non-null    float64
```

```
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
None
id                             0
diagnosis                      0
radius_mean                    0
texture_mean                   0
perimeter_mean                 0
area_mean                      0
smoothness_mean                0
compactness_mean               0
concavity_mean                 0
concave points_mean            0
```

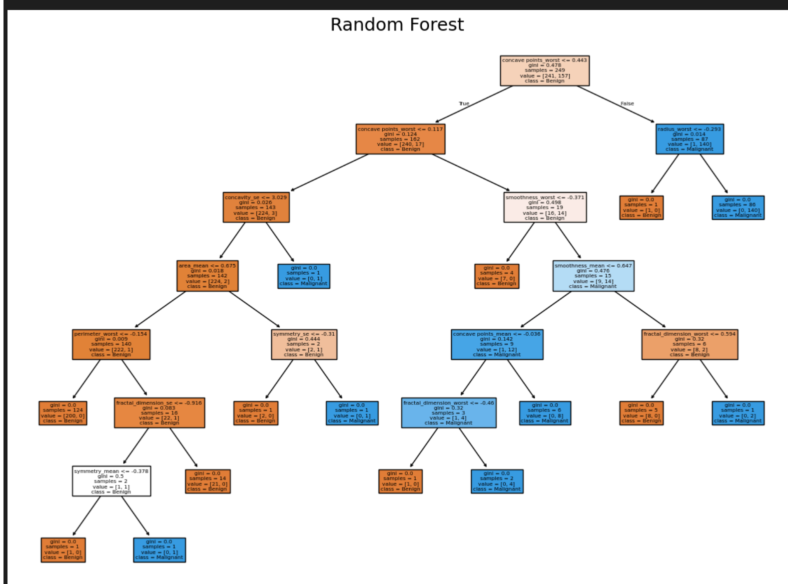
```
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
                  id diagnosis  radius_mean  texture_mean  perimeter_mean
\
count   5.690000e+02       569   569.000000    569.000000      569.000000
unique           NaN         2          NaN           NaN             NaN
top              NaN         B          NaN           NaN             NaN
freq             NaN       357          NaN           NaN             NaN
mean    3.037183e+07       NaN    14.127292     19.289649       91.969033
std     1.250206e+08       NaN     3.524049      4.301036       24.298981
min     8.670000e+03       NaN     6.981000      9.710000       43.790000
25%     8.692180e+05       NaN    11.700000     16.170000       75.170000
50%     9.060240e+05       NaN    13.370000     18.840000       86.240000
75%     8.813129e+06       NaN    15.780000     21.800000      104.100000
max     9.113205e+08       NaN    28.110000     39.280000      188.500000

         area_mean  smoothness_mean  compactness_mean  concavity_mean  \
count   569.000000       569.000000        569.000000      569.000000
```

```
unique          NaN              NaN                NaN              NaN
top             NaN              NaN                NaN              NaN
freq            NaN              NaN                NaN              NaN
mean     654.889104         0.096360           0.104341         0.088799
std      351.914129         0.014064           0.052813         0.079720
min      143.500000         0.052630           0.019380         0.000000
25%      420.300000         0.086370           0.064920         0.029560
50%      551.100000         0.095870           0.092630         0.061540
75%      782.700000         0.105300           0.130400         0.130700
max     2501.000000         0.163400           0.345400         0.426800

        concave points_mean  ...  texture_worst  perimeter_worst
area_worst  \
count            569.000000  ...     569.000000       569.000000
569.000000
unique                  NaN  ...            NaN              NaN
NaN
top                     NaN  ...            NaN              NaN
NaN
freq                    NaN  ...            NaN              NaN
NaN
mean               0.048919  ...      25.677223       107.261213
880.583128
std                0.038803  ...       6.146258        33.602542
569.356993
min                0.000000  ...      12.020000        50.410000
185.200000
25%                0.020310  ...      21.080000        84.110000
515.300000
50%                0.033500  ...      25.410000        97.660000
686.500000
75%                0.074000  ...      29.720000       125.400000
1084.000000
max                0.201200  ...      49.540000       251.200000
4254.000000

        smoothness_worst  compactness_worst  concavity_worst  \
count         569.000000         569.000000       569.000000
unique               NaN                NaN              NaN
top                  NaN                NaN              NaN
```

```
freq              NaN               NaN              NaN
mean         0.132369          0.254265         0.272188
std          0.022832          0.157336         0.208624
min          0.071170          0.027290         0.000000
25%          0.116600          0.147200         0.114500
50%          0.131300          0.211900         0.226700
75%          0.146000          0.339100         0.382900
max          0.222600          1.058000         1.252000

        concave points_worst  symmetry_worst  fractal_dimension_worst  \
count             569.000000      569.000000               569.000000
unique                   NaN             NaN                      NaN
top                      NaN             NaN                      NaN
freq                     NaN             NaN                      NaN
mean                0.114606        0.290076                 0.083946
std                 0.065732        0.061867                 0.018061
min                 0.000000        0.156500                 0.055040
25%                 0.064930        0.250400                 0.071460
50%                 0.099930        0.282200                 0.080040
75%                 0.161400        0.317900                 0.092080
max                 0.291000        0.663800                 0.207500

        Unnamed: 32
count           0.0
unique          NaN
top             NaN
freq            NaN
mean            NaN
std             NaN
min             NaN
25%             NaN
50%             NaN
75%             NaN
max             NaN

[11 rows x 33 columns]
/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1137:
RuntimeWarning: invalid value encountered in divide
  updated_mean = (last_sum + new_sum) / updated_sample_count
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1142:
RuntimeWarning: invalid value encountered in divide
  T = new_sum / new_sample_count
/usr/local/lib/python3.10/dist-packages/sklearn/utils/extmath.py:1162:
RuntimeWarning: invalid value encountered in divide
  new_unnormalized_variance -= correction**2 / new_sample_count
Accuracy of Random Forest: 0.9415204678362573
Classification Report of Random Forest:
              precision    recall  f1-score   support

           0       0.92      0.99      0.96       108
           1       0.98      0.86      0.92        63

    accuracy                           0.94       171
   macro avg       0.95      0.92      0.94       171
weighted avg       0.94      0.94      0.94       171

Accuracy scores for each fold of Random Forest: [0.96491228 0.96491228
0.98245614 0.96491228 0.96491228 0.94736842
 0.96491228 0.94736842 0.96491228 0.96428571]
Mean accuracy for folds of Random Forest: 0.9630952380952381
Standard deviation for folds of Random Forest: 0.00943788737641594
Feature ranking:
1. feature concave points_worst (0.15202612945430058)
2. feature area_worst (0.12874743014382417)
3. feature concave points_mean (0.11558042945096265)
4. feature perimeter_worst (0.095172929513616)
5. feature concavity_mean (0.07926403687338594)
6. feature radius_worst (0.07230949427019451)
7. feature perimeter_mean (0.0567948037815623)
8. feature radius_mean (0.04517994200466907)
9. feature concavity_worst (0.03718781740954586)
10. feature area_se (0.03335379510191234)
11. feature area_mean (0.03199930818344027)
12. feature texture_worst (0.020205821758160873)
13. feature compactness_worst (0.016559985905535996)
14. feature compactness_mean (0.01352094018268543)
15. feature smoothness_worst (0.013254669523309223)
16. feature texture_mean (0.012202057674059808)
17. feature radius_se (0.011990737544406547)
```

```
18. feature perimeter_se (0.011739568230596228)
19. feature compactness_se (0.008272379975674664)
20. feature symmetry_worst (0.006434536557988434)
21. feature smoothness_mean (0.006333239097768517)
22. feature fractal_dimension_worst (0.005492701210970268)
23. feature concave points_se (0.004031995407528327)
24. feature concavity_se (0.003747836797594058)
25. feature fractal_dimension_se (0.003546927506014493)
26. feature smoothness_se (0.003356895028184317 2)
27. feature texture_se (0.003207292053706029)
28. feature fractal_dimension_mean (0.003185085516525311 5)
29. feature symmetry_mean (0.003087541569290326 8)
30. feature symmetry_se (0.002213672272587238 7)
```



Practical 4.4
Naive Bayes

1. Import Libraries

[Syntax]

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB, GaussianNB
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

2. Load Dataset

[Syntax]

```
data = load_breast_cancer()
X = data.data
y = data.target
feature_names = data.feature_names
data_01 = pd.DataFrame(X, columns=feature_names)
data_01['Target'] = y
```

3. Data Exploration

[Syntax]

```
print(f"Sample 5 rows of the dataset: {data_01.sample(5)}")
print(f"\nDataset Summary: {data_01.describe()}")
```

4. Visualizations

[Syntax]

```
sns.set(style='whitegrid')
plt.figure(figsize=(8, 5))
sns.countplot(x='target', data=df, palette='pastel')
plt.title('Distribution of Target Variable', fontsize=16)
plt.xlabel('Target', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks([0, 1], ['Malignant', 'Benign'], fontsize=12)
counts = df['target'].value_counts()
for index, value in enumerate(counts):
```

```
    plt.text(index, value, str(value), ha='center', va='bottom',
fontsize=12)
plt.tight_layout()
plt.show()

sns.pairplot(data_01, hue='Target', vars=feature_names[:5])
plt.title('Pairplot of First 5 Features')
plt.show()
```

5. Train-Test Split

[Syntax]

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

7. Function

[Syntax]

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

    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)

    print(f"Accuracy: {accuracy:.2f}")
    print("Confusion Matrix:")
    print(conf_matrix)
    print("Classification Report:")
    print(class_report)
print()
```

A. Gaussian Naive Bayes

[Syntax]

```
print("Gaussian Naive Bayes Results:")
evaluate_model(GaussianNB())
```

B. Bernoulli Naive Bayes

[Syntax]

```
print("Bernoulli Naive Bayes Results:")
evaluate_model(BernoulliNB())
```

C. Multinomial Naive Bayes

[Syntax]

```
print("Multinomial Naive Bayes Results:")
evaluate_model(MultinomialNB())
```

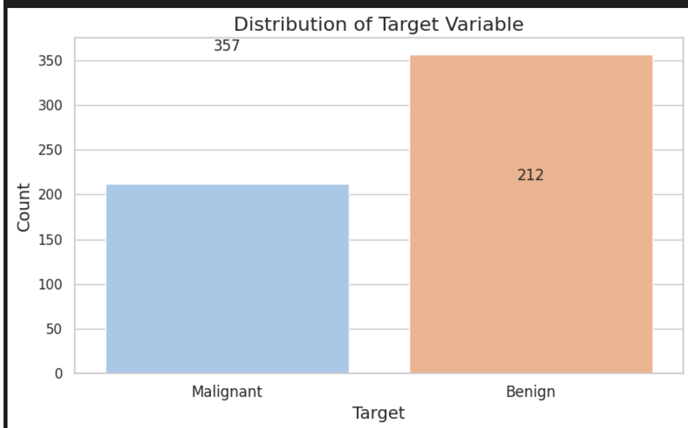
[Output]

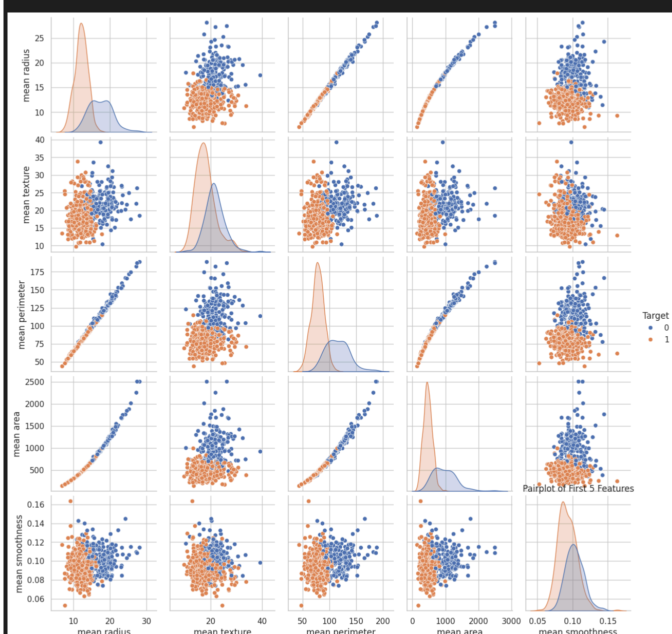
```
Sample 5 rows of the dataset:      mean radius  mean texture  mean
perimeter  mean area  mean smoothness  \
535        20.55         20.86          137.80     1308.0          0.10460
472        14.92         14.93           96.45      686.9          0.08098
502        12.54         16.32           81.25      476.3          0.11580
508        16.30         15.70          104.70      819.8          0.09427
401        11.93         10.91           76.14      442.7          0.08872

     mean compactness  mean concavity  mean concave points  mean symmetry
\
535           0.17390         0.20850              0.13220         0.2127
472           0.08549         0.05539              0.03221         0.1687
502           0.10850         0.05928              0.03279         0.1943
508           0.06712         0.05526              0.04563         0.1711
401           0.05242         0.02606              0.01796         0.1601

     mean fractal dimension  ...  worst texture  worst perimeter  worst
area  \
535                 0.06251  ...          25.48           160.20
1809.0
472                 0.05669  ...          18.22           112.00
906.6
502                 0.06612  ...          21.40            86.67
552.0
508                 0.05657  ...          17.76           109.80
928.2
401                 0.05541  ...          20.14            87.64
589.5

     worst smoothness  worst compactness  worst concavity  \
535            0.1268             0.3135           0.4433
472            0.1065             0.2791           0.3151
502            0.1580             0.1751           0.1889
```

```
508           0.1354             0.1361          0.1947
401           0.1374             0.1575          0.1514

     worst concave points  worst symmetry  worst fractal dimension  Target
535               0.21480          0.3077                  0.07569       0
472               0.11470          0.2688                  0.08273       1
502               0.08411          0.3155                  0.07538       1
508               0.13570          0.2300                  0.07230       1
401               0.06876          0.2460                  0.07262       1

[5 rows x 31 columns]

Dataset Summary:        mean radius  mean texture  mean perimeter    mean
area  \
count   569.000000    569.000000      569.000000   569.000000
mean     14.127292     19.289649       91.969033   654.889104
std       3.524049      4.301036       24.298981   351.914129
min       6.981000      9.710000       43.790000   143.500000
25%      11.700000     16.170000       75.170000   420.300000
50%      13.370000     18.840000       86.240000   551.100000
75%      15.780000     21.800000      104.100000   782.700000
max      28.110000     39.280000      188.500000  2501.000000

       mean smoothness  mean compactness  mean concavity  mean concave
points  \
count       569.000000        569.000000      569.000000
569.000000
mean          0.096360          0.104341        0.088799
0.048919
std           0.014064          0.052813        0.079720
0.038803
min           0.052630          0.019380        0.000000
0.000000
25%           0.086370          0.064920        0.029560
0.020310
50%           0.095870          0.092630        0.061540
0.033500
75%           0.105300          0.130400        0.130700
0.074000
```

```
max          0.163400          0.345400        0.426800
0.201200

       mean symmetry  mean fractal dimension  ...  worst texture  \
count     569.000000              569.000000  ...     569.000000
mean        0.181162                0.062798  ...      25.677223
std         0.027414                0.007060  ...       6.146258
min         0.106000                0.049960  ...      12.020000
25%         0.161900                0.057700  ...      21.080000
50%         0.179200                0.061540  ...      25.410000
75%         0.195700                0.066120  ...      29.720000
max         0.304000                0.097440  ...      49.540000

       worst perimeter   worst area  worst smoothness  worst compactness
\
count       569.000000   569.000000        569.000000         569.000000
mean        107.261213   880.583128          0.132369           0.254265
std          33.602542   569.356993          0.022832           0.157336
min          50.410000   185.200000          0.071170           0.027290
25%          84.110000   515.300000          0.116600           0.147200
50%          97.660000   686.500000          0.131300           0.211900
75%         125.400000  1084.000000          0.146000           0.339100
max         251.200000  4254.000000          0.222600           1.058000

       worst concavity  worst concave points  worst symmetry  \
count       569.000000            569.000000      569.000000
mean          0.272188              0.114606        0.290076
std           0.208624              0.065732        0.061867
min           0.000000              0.000000        0.156500
25%           0.114500              0.064930        0.250400
50%           0.226700              0.099930        0.282200
75%           0.382900              0.161400        0.317900
max           1.252000              0.291000        0.663800

       worst fractal dimension      Target
count               569.000000  569.000000
mean                  0.083946    0.627417
std                   0.018061    0.483918
min                   0.055040    0.000000
25%                   0.071460    0.000000
```

```
50%                       0.080040    1.000000
75%                       0.092080    1.000000
max                       0.207500    1.000000

[8 rows x 31 columns]
```






```
Gaussian Naive Bayes Results:
Accuracy: 0.94
Confusion Matrix:
[[ 57   6]
 [  4 104]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.90      0.92        63
```

```
           1       0.95      0.96      0.95       108

    accuracy                           0.94       171
   macro avg       0.94      0.93      0.94       171
weighted avg       0.94      0.94      0.94       171

Bernoulli Naive Bayes Results:
Accuracy: 0.63
Confusion Matrix:
[[  0  63]
 [  0 108]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        63
           1       0.63      1.00      0.77       108

    accuracy                           0.63       171
   macro avg       0.32      0.50      0.39       171
weighted avg       0.40      0.63      0.49       171

Multinomial Naive Bayes Results:
Accuracy: 0.91
Confusion Matrix:
[[ 50  13]
 [  2 106]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.79      0.87        63
           1       0.89      0.98      0.93       108

    accuracy                           0.91       171
   macro avg       0.93      0.89      0.90       171
weighted avg       0.92      0.91      0.91       171
```

D. Text Classification

1. Import Libraries

[Syntax]

```
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import BernoulliNB, MultinomialNB,GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

2. Load Dataset

[Syntax]

```
data_01 = fetch_20newsgroups(subset='all')
X = data_01.data
y = data_01.target
```

3. Function

[Syntax]

```
def evaluate_model(model):
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)
   print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
   print("Confusion Matrix:")
   print(confusion_matrix(y_test, y_pred))
   print("Classification Report:")
   print(classification_report(y_test, y_pred))
```

4. Vectorization

[Syntax]

```
vectorizers = {
   "binary": CountVectorizer(binary=True),
   "count": CountVectorizer(binary=False),
   "binary_stop_words": CountVectorizer(binary=True,
stop_words='english'),
   "count_stop_words": CountVectorizer(binary=False, stop_words='english')
}
```

5. Models Building

[Syntax]

```
for key, vectorizer in vectorizers.items():
   X_transformed = vectorizer.fit_transform(X)
```

```
    X_train, X_test, y_train, y_test = train_test_split(X_transformed, y,
test_size=0.33, random_state=1)

    print(f"Evaluating BernoulliNB with {key} vectorizer:")
    evaluate_model(BernoulliNB())

    print(f"Evaluating MultinomialNB with {key} vectorizer:")
    evaluate_model(MultinomialNB())

    print(f"Evaluating Gaussian with {key} vectorizer:")
    evaluate_model(GaussianNB())
```

[Output]

```
Evaluating BernoulliNB with binary vectorizer:
Accuracy: 0.6586816720257235
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.23      0.36       274
           1       0.83      0.55      0.66       321
           2       0.67      0.02      0.03       351
           3       0.51      0.75      0.61       324
           4       0.36      0.89      0.51       304
           5       0.92      0.56      0.70       319
           6       0.39      0.89      0.54       329
           7       0.89      0.77      0.83       333
           8       0.80      0.92      0.85       325
           9       0.70      0.89      0.78       314
          10       0.99      0.88      0.93       345
          11       0.92      0.76      0.83       341
          12       0.47      0.85      0.60       313
          13       0.90      0.69      0.78       329
          14       0.95      0.68      0.79       342
          15       0.54      0.83      0.66       317
          16       0.81      0.69      0.74       301
          17       0.93      0.71      0.81       289
          18       0.89      0.24      0.38       263
          19       0.00      0.00      0.00       186

    accuracy                           0.66      6220
```

```
   macro avg       0.72      0.64      0.62      6220
weighted avg       0.74      0.66      0.64      6220

Evaluating MultinomialNB with binary vectorizer:
Accuracy: 0.8409967845659164
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.84      0.86       274
           1       0.75      0.84      0.79       321
           2       0.95      0.23      0.37       351
           3       0.58      0.87      0.70       324
           4       0.90      0.85      0.88       304
           5       0.76      0.86      0.81       319
           6       0.95      0.68      0.79       329
           7       0.87      0.88      0.87       333
           8       0.94      0.94      0.94       325
           9       0.96      0.97      0.97       314
          10       0.98      0.97      0.97       345
          11       0.78      0.97      0.86       341
          12       0.83      0.81      0.82       313
          13       0.92      0.94      0.93       329
          14       0.91      0.96      0.94       342
          15       0.73      0.96      0.83       317
          16       0.78      0.96      0.86       301
          17       0.88      0.99      0.93       289
          18       0.96      0.77      0.85       263
          19       0.96      0.38      0.54       186

    accuracy                           0.84      6220
   macro avg       0.86      0.83      0.83      6220
weighted avg       0.86      0.84      0.83      6220

Evaluating BernoulliNB with count vectorizer:
Accuracy: 0.6586816720257235
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.23      0.36       274
           1       0.83      0.55      0.66       321
```

```
           2       0.67      0.02      0.03       351
           3       0.51      0.75      0.61       324
           4       0.36      0.89      0.51       304
           5       0.92      0.56      0.70       319
           6       0.39      0.89      0.54       329
           7       0.89      0.77      0.83       333
           8       0.80      0.92      0.85       325
           9       0.70      0.89      0.78       314
          10       0.99      0.88      0.93       345
          11       0.92      0.76      0.83       341
          12       0.47      0.85      0.60       313
          13       0.90      0.69      0.78       329
          14       0.95      0.68      0.79       342
          15       0.54      0.83      0.66       317
          16       0.81      0.69      0.74       301
          17       0.93      0.71      0.81       289
          18       0.89      0.24      0.38       263
          19       0.00      0.00      0.00       186

    accuracy                           0.66      6220
   macro avg       0.72      0.64      0.62      6220
weighted avg       0.74      0.66      0.64      6220

Evaluating MultinomialNB with count vectorizer:
Accuracy: 0.8331189710610932
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.86      0.87       274
           1       0.63      0.90      0.74       321
           2       0.91      0.09      0.16       351
           3       0.61      0.84      0.71       324
           4       0.89      0.84      0.86       304
           5       0.70      0.87      0.78       319
           6       0.93      0.62      0.75       329
           7       0.87      0.86      0.87       333
           8       0.92      0.93      0.92       325
           9       0.95      0.97      0.96       314
          10       0.98      0.97      0.97       345
          11       0.79      0.96      0.87       341
```

```
          12       0.86      0.80      0.83       313
          13       0.96      0.93      0.94       329
          14       0.93      0.95      0.94       342
          15       0.77      0.96      0.85       317
          16       0.82      0.94      0.88       301
          17       0.87      0.98      0.93       289
          18       0.82      0.84      0.83       263
          19       0.96      0.47      0.63       186

    accuracy                           0.83      6220
   macro avg       0.85      0.83      0.82      6220
weighted avg       0.85      0.83      0.82      6220

Evaluating BernoulliNB with binary_stop_words vectorizer:
Accuracy: 0.677652733118971
Classification Report:
             precision    recall  f1-score   support

           0       0.94      0.22      0.36       274
           1       0.82      0.54      0.65       321
           2       0.67      0.02      0.03       351
           3       0.50      0.77      0.60       324
           4       0.33      0.95      0.49       304
           5       0.93      0.55      0.69       319
           6       0.65      0.89      0.75       329
           7       0.87      0.80      0.83       333
           8       0.77      0.94      0.85       325
           9       0.69      0.96      0.80       314
          10       0.99      0.88      0.93       345
          11       0.94      0.80      0.87       341
          12       0.45      0.90      0.60       313
          13       0.92      0.71      0.80       329
          14       0.96      0.70      0.81       342
          15       0.54      0.88      0.67       317
          16       0.82      0.71      0.76       301
          17       0.95      0.73      0.83       289
          18       0.91      0.24      0.38       263
          19       0.00      0.00      0.00       186

    accuracy                           0.68      6220
```

```
   macro avg       0.73      0.66      0.64      6220
weighted avg       0.75      0.68      0.65      6220

Evaluating MultinomialNB with binary_stop_words vectorizer:
Accuracy: 0.8617363344051447
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.88      0.89       274
           1       0.70      0.87      0.77       321
           2       0.96      0.38      0.54       351
           3       0.65      0.87      0.74       324
           4       0.91      0.86      0.89       304
           5       0.76      0.88      0.82       319
           6       0.94      0.76      0.84       329
           7       0.90      0.88      0.89       333
           8       0.95      0.94      0.95       325
           9       0.96      0.98      0.97       314
          10       0.97      0.97      0.97       345
          11       0.85      0.96      0.90       341
          12       0.86      0.81      0.83       313
          13       0.95      0.94      0.95       329
          14       0.90      0.96      0.93       342
          15       0.79      0.94      0.86       317
          16       0.81      0.95      0.87       301
          17       0.90      0.99      0.94       289
          18       0.93      0.81      0.86       263
          19       0.95      0.46      0.62       186

    accuracy                           0.86      6220
   macro avg       0.88      0.86      0.85      6220
weighted avg       0.88      0.86      0.86      6220

Evaluating BernoulliNB with count_stop_words vectorizer:
Accuracy: 0.677652733118971
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.22      0.36       274
           1       0.82      0.54      0.65       321
```

```
           2       0.67      0.02      0.03       351
           3       0.50      0.77      0.60       324
           4       0.33      0.95      0.49       304
           5       0.93      0.55      0.69       319
           6       0.65      0.89      0.75       329
           7       0.87      0.80      0.83       333
           8       0.77      0.94      0.85       325
           9       0.69      0.96      0.80       314
          10       0.99      0.88      0.93       345
          11       0.94      0.80      0.87       341
          12       0.45      0.90      0.60       313
          13       0.92      0.71      0.80       329
          14       0.96      0.70      0.81       342
          15       0.54      0.88      0.67       317
          16       0.82      0.71      0.76       301
          17       0.95      0.73      0.83       289
          18       0.91      0.24      0.38       263
          19       0.00      0.00      0.00       186

    accuracy                           0.68      6220
   macro avg       0.73      0.66      0.64      6220
weighted avg       0.75      0.68      0.65      6220

Evaluating MultinomialNB with count_stop_words vectorizer:
Accuracy: 0.8585209003215434
Classification Report:
             precision    recall  f1-score   support

           0       0.89      0.89      0.89       274
           1       0.65      0.91      0.75       321
           2       0.96      0.23      0.36       351
           3       0.64      0.85      0.73       324
           4       0.87      0.88      0.87       304
           5       0.71      0.87      0.78       319
           6       0.92      0.74      0.82       329
           7       0.90      0.90      0.90       333
           8       0.92      0.96      0.94       325
           9       0.96      0.97      0.97       314
          10       0.97      0.97      0.97       345
          11       0.90      0.96      0.93       341
```

```
          12       0.88      0.81      0.84       313
          13       0.97      0.93      0.95       329
          14       0.94      0.95      0.95       342
          15       0.83      0.96      0.89       317
          16       0.82      0.96      0.88       301
          17       0.93      0.99      0.96       289
          18       0.89      0.84      0.86       263
          19       0.93      0.54      0.68       186

    accuracy                           0.86      6220
   macro avg       0.87      0.85      0.85      6220
weighted avg       0.87      0.86      0.85      6220
```

Practical 4.5
K-nearest neighbors (KNN)

1. Import Libraries
[Syntax]
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, RepeatedKFold,
GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

A. Pima Indians Diabetes Dataset

2. Load Dataset
[Syntax]
```
data_01 = pd.read_csv("Data.csv",names=["Pregnancies", "Glucose",
"BloodPressure",  "SkinThickness",Insulin", "BMI",
"DiabetesPedigreeFunction", "Age", "Outcome"],header=None)
```

3. Exploratory Data Analysis (EDA)
[Syntax]
```
print(data_01.sample(5))
print(f"Missing values in each column: {data_01.isnull().sum()}")
print(f"Basic statistics of the dataset: {data_01.describe()}")
print(f"Data types of each column: {data_01.dtypes}")
data_01["Outcome"].value_counts()
```

4. Visualization
[Syntax]
```
sns.set(style="whitegrid")
plt.figure(figsize=(8, 5))
sns.countplot(x=data_01['Outcome'], palette='pastel')
plt.title("Distribution of Diabetes Cases", fontsize=16,
fontweight='bold')
plt.xlabel("Diabetes Status", fontsize=14)
```

```
plt.ylabel("Count", fontsize=14)
for p in plt.gca().patches:
   plt.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2.,
p.get_height()),ha='center', va='bottom', fontsize=12)
plt.show()

sns.set(style="white")
plt.figure(figsize=(8, 5))
sns.heatmap(data_01.corr(), annot=True, fmt=".2f", cmap='coolwarm',
square=True,cbar_kws={"shrink": .8}, linewidths=0.5, linecolor='gray')
plt.title("Correlation Heatmap", fontsize=18, fontweight='bold')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```

5. Split Data into Features and Target
[Syntax]

```
X = data_01.iloc[:, :-1]
Y = data_01.iloc[:, -1]
```

6. Train-Test Split
[Syntax]

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=1,
test_size=0.25)
print("Shape of training and test data:", X.shape, Y.shape)
```

7. Hyperparameter Tuning for K-Neighbors Classifier
[Syntax]

```
parameters = {"n_neighbors": list(range(15, 26))}
K_Classifier = KNeighborsClassifier()
K_fold = RepeatedKFold(n_splits=10, n_repeats=5, random_state=15)
Grid_Search = GridSearchCV(K_Classifier, parameters, cv=K_fold)
```

8. Fit Grid Search
[Syntax]

```
Grid_Search.fit(X, Y)
best_params = Grid_Search.best_params_
```

```
print("Best parameters for KNN:", best_params)
```

9. Train Classifier with Best Parameters

[Syntax]

```
K_Classifer_02 =
KNeighborsClassifier(n_neighbors=best_params['n_neighbors'])
K_Classifer_02.fit(X_train,Y_train)
```

10. Predictions and Evaluation

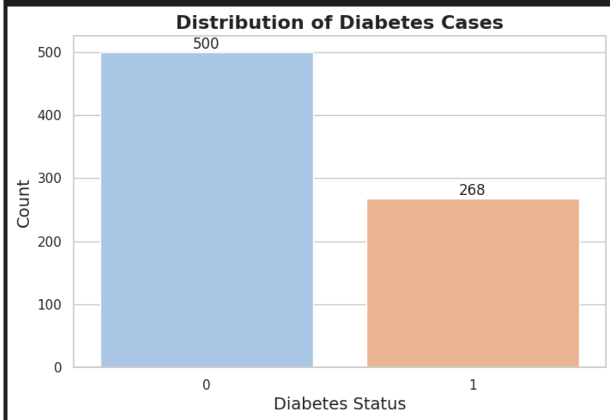
[Syntax]

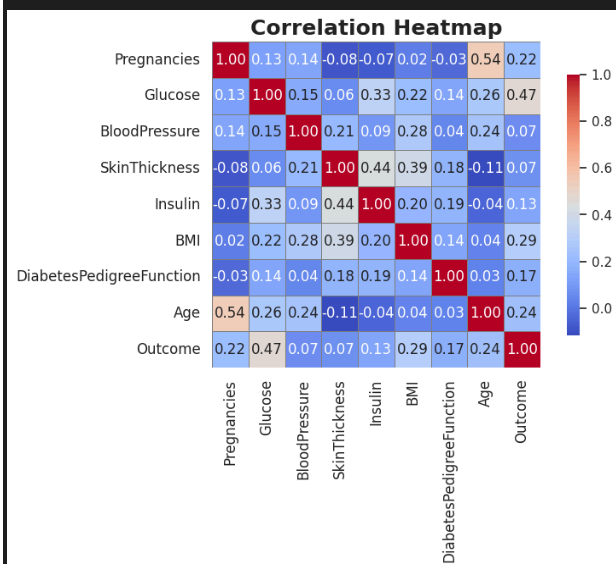
```
Y_prediction = K_Classifer_02.predict(X_test)
print(f"Accuracy : {accuracy_score(Y_test,Y_prediction)}")
print(f"Confusion Matrix : \n {confusion_matrix(Y_test,Y_prediction)}")
print("Classisficatin report: \n")
print(classification_report(Y_test,Y_prediction))
```

[Output]

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
348            3       99             62             19       74  21.8
658           11      127            106              0        0  39.0
672           10       68            106             23       49  35.5
702            1      168             88             29        0  35.0
186            8      181             68             36      495  30.1

     DiabetesPedigreeFunction  Age  Outcome
348                     0.279   26        0
658                     0.190   51        0
672                     0.285   47        0
702                     0.905   52        1
186                     0.615   60        1

Missing values in each column: Pregnancies                0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
```

```
Outcome                     0
dtype: int64

Basic statistics of the dataset:        Pregnancies     Glucose
BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000

Data types of each column: Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object

      count
Outcome
0     500
1     268
dtype: int64
```






```
Shape of training and test data: (768, 8) (768,)
Best parameters for KNN: {'n_neighbors': 16}
Accuracy : 0.7864583333333334
Confusion Matrix :
 [[114   9]
 [ 32  37]]

Classisficatin report:

              precision    recall  f1-score   support

           0       0.78      0.93      0.85       123
           1       0.80      0.54      0.64        69
```

```
    accuracy                           0.79       192
   macro avg       0.79      0.73      0.75       192
weighted avg       0.79      0.79      0.77       192
```

[Insights]

1. The outcome shows a dataset with two classes, where class 0 has 500 instances and class 1 has 268 instances. This indicates a class imbalance, with class 0 being the majority class.
2. The model demonstrates a reasonably good performance overall, with an accuracy of approximately 78.65%.
3. Confusion Matrix Analysis:

- True Negatives (TN): 114
- False Positives (FP): 9
- False Negatives (FN): 32
- True Positives (TP): 37

This indicates that the model is more effective at correctly predicting class 0 (with a high true negative rate) than class 1, as shown by the lower true positive count.

4. Precision and Recall:

- For class 0:
  - Precision: 0.78 (indicates the proportion of predicted positives that are actually positive)
  - Recall: 0.93 (indicates the proportion of actual positives that were correctly identified)
- For class 1:
  - Precision: 0.80
  - Recall: 0.54 (the model struggles to identify all actual positive cases)

5. F1-Score:

- The F1-score for class 0 (0.85) is higher than that for class 1 (0.64), reflecting better balance between precision and recall for the majority class.

<u>B. Iris Dataset</u>

2. Load Dataset

[Syntax]

```
iris = load_iris()
```

```
data_01 = pd.DataFrame(data=iris.data, columns=iris.feature_names)
data_01['target'] = iris.target
```

3. Exploratory Data Analysis (EDA)

[Syntax]

```
print(data_01.sample(5))
print("Basic statistics of the iris dataset:")
print(data_01.describe())
print(f"Data types of each column: {data_01.dtypes}")
data_01["target"].value_counts()
```

4. Visualization

[Syntax]

```
sns.pairplot(iris_df, hue='target', palette='viridis')
plt.title("Pairplot of Iris Features")
plt.show()
```

5. Split Data into Features and Target

[Syntax]

```
X = data_01.iloc[:, :-1]
Y = data_01.iloc[:, -1]
```

6. Train-Test Split

[Syntax]

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=1,
test_size=0.25)
print("Shape of training and test data:", X.shape, Y.shape)
```

7. Accuracy Evaluation for K-Neighbors Classifier on Iris Dataset

[Syntax]

```
data_03 = {}

for i in range(1, round((len(X) + 1)**(1/2))):
   K_Classifier = KNeighborsClassifier(n_neighbors=i)
   K_Classifier.fit(X_train, Y_train)
   Y_prediction = K_Classifier.predict(X_test)
   accuracy = accuracy_score(Y_test, Y_prediction)
   data_03[i] = accuracy
```

8. Results for Different Values of k

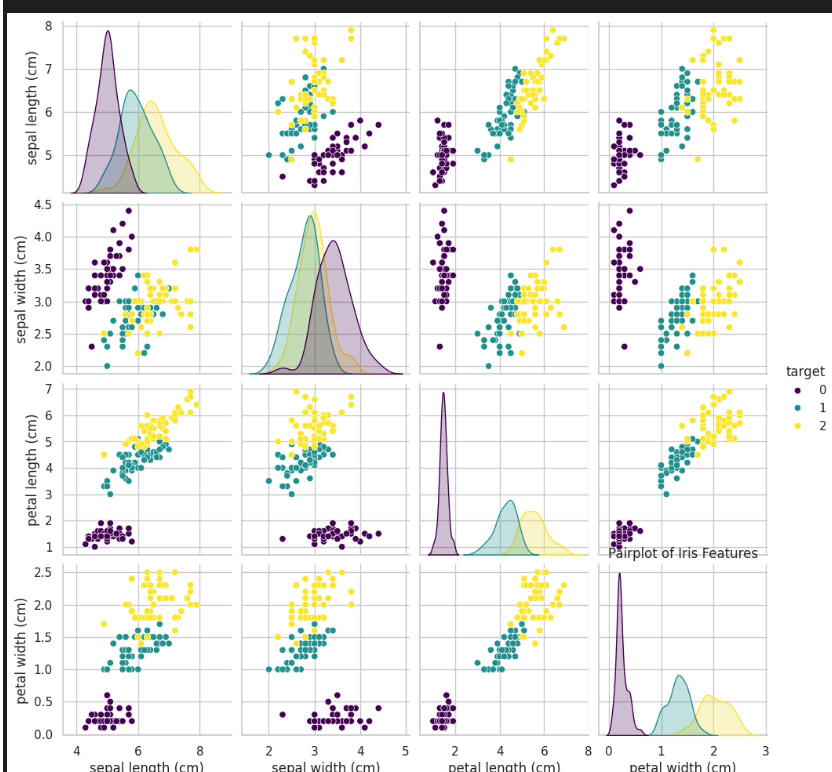
[Syntax]

```
print("Accuracy for different values of k:")
data_03
```

[Output]

```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)   \
138                6.0               3.0                4.8
1.8
77                 6.7               3.0                5.0
1.7
148                6.2               3.4                5.4
2.3
47                 4.6               3.2                1.4
0.2
82                 5.8               2.7                3.9
1.2

     target
138       2
77        1
148       2
47        0
82        1


Basic statistics of the iris dataset:
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)      target
count        150.000000  150.000000
```

```
mean          1.199333    1.000000
std           0.762238    0.819232
min           0.100000    0.000000
25%           0.300000    0.000000
50%           1.300000    1.000000
75%           1.800000    2.000000
max           2.500000    2.000000

Data types of each column: sepal length (cm)    float64
sepal width (cm)     float64
petal length (cm)    float64
petal width (cm)     float64
target                 int64
dtype: object

count
target
0     50
1     50
2     50
dtype: int64
```



```
Shape of training and test data: (150, 4) (150,)
Accuracy for different values of k:
{1: 1.0,
 2: 1.0,
 3: 1.0,
 4: 1.0,
 5: 1.0,
 6: 1.0,
 7: 0.9736842105263158,
 8: 1.0,
 9: 0.9736842105263158,
 10: 0.9736842105263158,
 11: 0.9736842105263158}
```

[Insights]

1. The outcome shows a dataset with three classes, where class 0 has 50 instances , class 1 has 50 instances and class 2 has 50 instances. This indicates a class completely balance.

2. Accuracy by k
Perfect Accuracy:

- For k=1 to k=6 and k=8, the accuracy is 1.0. This indicates that the model perfectly classifies all instances for these values of kkk.

Slight Decrease in Accuracy:

- For k=7, k=9, k=10, and k=11, the accuracy drops slightly to approximately 97.37%. While this is still a strong performance, it suggests that increasing kkk beyond 6 starts to impact the model's ability to generalize, potentially introducing some noise from neighboring points.

Practical 4.6
Support Vector Machine

1. Import Libraries

[Syntax]

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,
f1_score, precision_score
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Load Dataset

[Syntax]

```
cancer = load_breast_cancer()
data_01 = pd.DataFrame(cancer.data,columns = cancer.feature_names)
data_01["Target"] = cancer.target
```

3. Data Exploration

[Syntax]

```
data_01.columns
data_01["Target"].value_counts()
```

4. Split Data into Features and Target

[Syntax]

```
X = data_01.drop(columns=['Target'])
Y = data_01['Target']
```

5. Feature Scaling

[Syntax]

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

6. Train-Test Split

[Syntax]

```
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y,
test_size=0.3, random_state=42)
```

7. Model Training and Evaluation

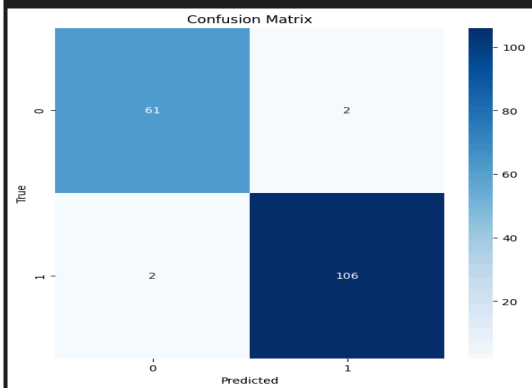
[Syntax]

```
list = ["linear","poly","rbf"]
for i in list:
   print("For Kernal : ",i)
   svm = SVC(kernel=i)
   svm.fit(X_train, Y_train)
   Y_prediction_svm = svm.predict(X_test)
   print(f"F1 Score : {f1_score(Y_test, Y_prediction_svm,
average='weighted')}")
   print(f"Precision Score : {precision_score(Y_test, Y_prediction_svm,
average='weighted')}")
   print("Confusion Matrix")
   plt.figure(figsize=(7, 7))
   sns.heatmap(confusion_matrix(Y_test, Y_prediction_svm), annot=True,
fmt='d', cmap='Blues')
   plt.xlabel('Predicted')
   plt.ylabel('True')
   plt.title('Confusion Matrix')
   plt.show()
   print()
```

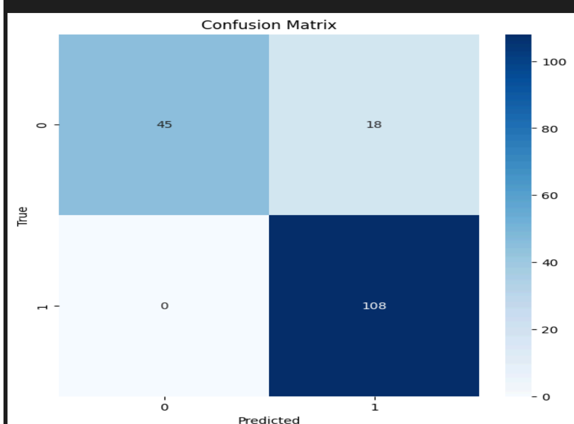
[Output]

```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension
error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal
dimension',
       'Target'],
      dtype='object')


count
Target
```

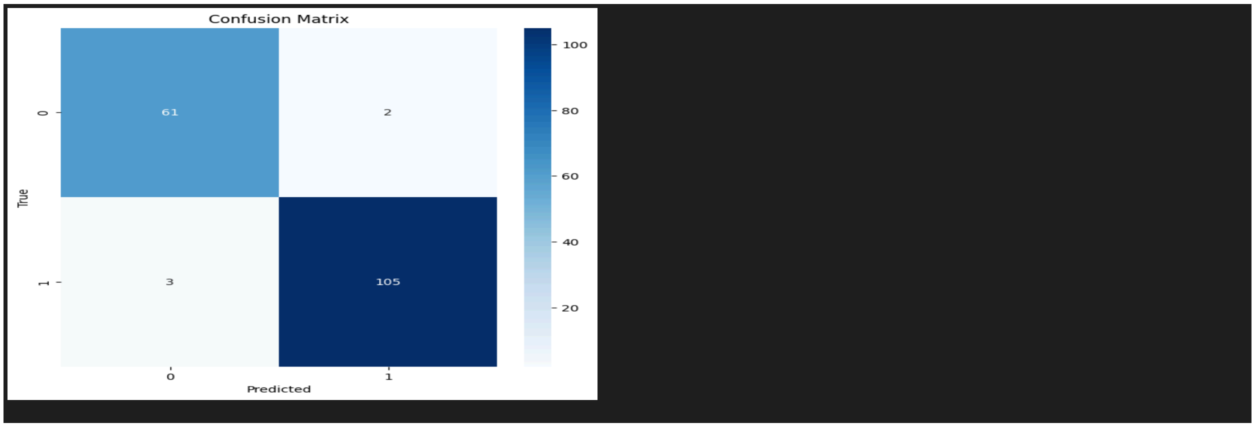
```
1     357
0     212
dtype: int64

For Kernal :  linear
F1 Score : 0.9766081871345029
Precision Score : 0.9766081871345029
Confusion Matrix
```




```
For Kernal :  poly
F1 Score : 0.8900134952766531
Precision Score : 0.9097744360902255
Confusion Matrix
```




```
For Kernal :  rbf
F1 Score : 0.970807351651423
Precision Score : 0.9709250491883916
Confusion Matrix
```



[Insights]

1. The outcome shows a dataset with two classes, where class 0 has 212 instances and class 1 has 357 instances. This indicates a class imbalance, with class 1 being the majority class.

2. Performance Analysis for Different Kernels:

A. Linear Kernel

- F1 Score: 0.9766
- Precision: 0.9766
- Confusion Matrix:

  a} True Positives (TP): 106 (correctly predicted Target 1)

  b} True Negatives (TN): 61 (correctly predicted Target 0)

  c} False Positives (FP): 2 (incorrectly predicted Target 1)

  d} False Negatives (FN): 2 (incorrectly predicted Target 0)

The **linear kernel** is the most effective, achieving the highest F1 score and precision while maintaining a low number of misclassifications.

B. Polynomial Kernel

- F1 Score: 0.8900
- Precision: 0.9098
- Confusion Matrix:

  a} True Positives (TP): 108 (correctly predicted Target 1)

  b} True Negatives (TN): 45 (correctly predicted Target 0) c} False Positives (FP): 18 (incorrectly predicted Target 1)

d} False Negatives (FN): 0 (incorrectly predicted Target 0)

The **polynomial kernel** struggles with the false positive rate and does not generalize well for Target 0, even though it identifies all instances of Target 1.

<u>C. RBF Kernel</u>

- F1 Score: 0.9708
- Precision: 0.9709
- Confusion Matrix:

a} True Positives (TP): 105 (correctly predicted Target 1)

b} True Negatives (TN): 61 (correctly predicted Target 0)

c} False Positives (FP): 2 (incorrectly predicted Target 1)

d} False Negatives (FN): 3 (incorrectly predicted Target 0)

The **RBF kernel** is also strong, performing similarly to the linear kernel, making it a viable alternative if non-linearity is expected in the data.

Practical 5.1
K Means

1. Import Libraries

[Syntax]

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

2. Load Dataset

[Syntax]

```
data_01 = pd.read_excel("student_clustering.xlsx")
X = data_01[['cgpa', 'ML']]
```

3. Standardization

[Syntax]

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

4. K Means Clustering

[Syntax]

```
kmeans = KMeans(n_clusters=4, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
```

5. Silhouette Score

[Syntax]

```
print(f'Silhouette Score: {silhouette_score(X_scaled, clusters):.2f}')
```

6. Create a DataFrame

[Syntax]

```
df = pd.DataFrame(X, columns=['cgpa', 'ML'])
df['Cluster'] = clusters
cluster_names = {0: 'Low Performer', 1: 'Average Performer', 2: 'High
Performer',3:'Excellent Performer'}
df['Cluster Name'] = df['Cluster'].map(cluster_names)
df_plot = df.copy()
df_plot['Cluster Name'] = df_plot['Cluster Name'].astype(str)
```

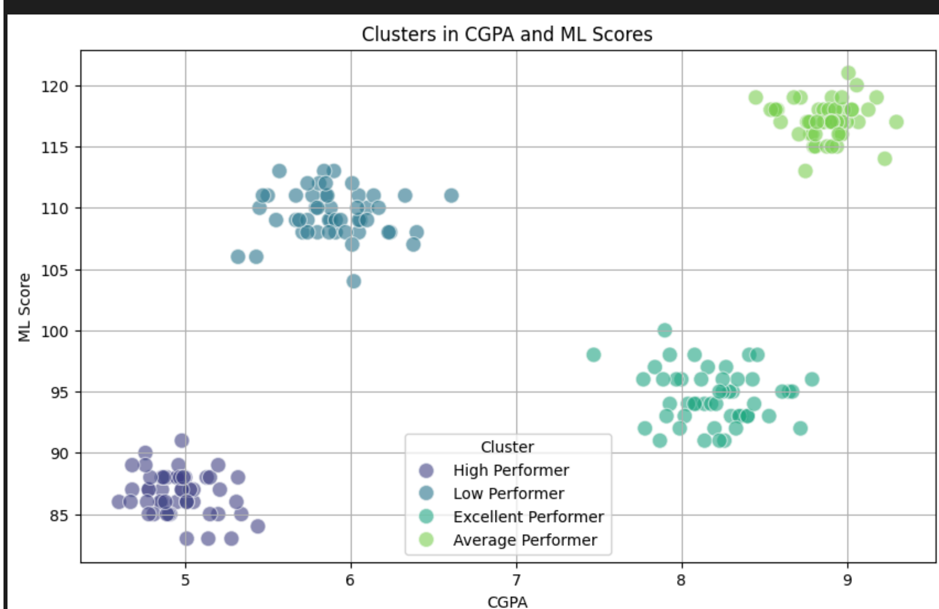
7. Visualize

[Syntax]

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_plot, x='cgpa', y='ML',
                hue='Cluster Name', palette='viridis', s=100, alpha=0.6)
plt.title('Clusters in CGPA and ML Scores')
plt.xlabel('CGPA')
plt.ylabel('ML Score')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```

[Output]

```
Silhouette Score: 0.86
   cgpa   ML  Cluster           Cluster Name
0  5.13   88        2       High Performer
1  5.90  113        0        Low Performer
2  8.36   93        3  Excellent Performer
3  8.27   97        3  Excellent Performer
4  5.45  110        0        Low Performer
```



Practical 5.2

Hierarchical Clustering

1. Import Libraries

[Syntax]

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster import hierarchy as SCH
from sklearn.cluster import AgglomerativeClustering
```

2. Load Dataset

[Syntax]

```
df = pd.read_csv("Mall_Customers.csv")
X = df[['Annual Income (k$)', 'Spending Score (1-100)']]
```

3. Dendrogram Visualization

[Syntax]

```
plt.figure(figsize=(8, 8))
plt.title('Dendrogram for Hierarchical Clustering')
dendrogram = SCH.dendrogram(SCH.linkage(X, method='ward'))
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
```

4.Agglomerative Clustering

[Syntax]

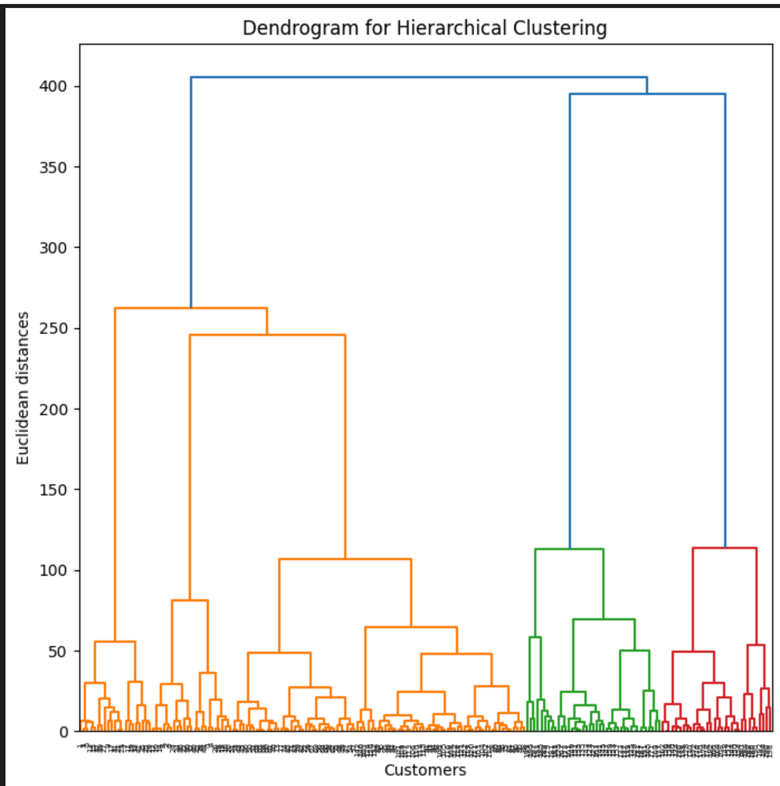
```
ac = AgglomerativeClustering(n_clusters=5)
labels = ac.fit_predict(X)
```

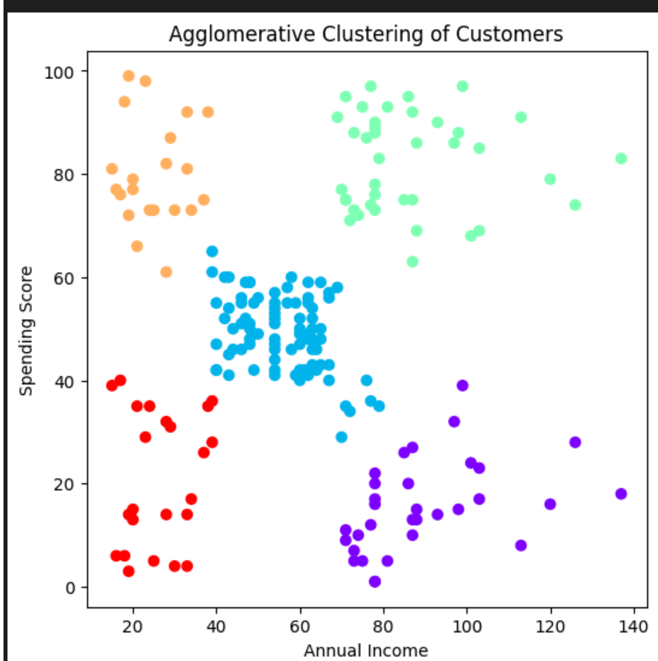
5. Visualizing the Clusters

[Syntax]

```
plt.figure(figsize=(6, 6))
plt.scatter(X['Annual Income (k$)'], X['Spending Score (1-100)'],
c=labels, cmap='rainbow')
plt.title('Agglomerative Clustering of Customers')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.show()
```

[Output]


Dendrogram for Hierarchical Clustering
Euclidean distances
400
350
300
250
200
150
100
50
0
Customers


Agglomerative Clustering of Customers
Spending Score
100
80
60
40
20
0
20
40
60
80
100
120
140
Annual Income

Practical 5.3
Density Based Clustering

1. Import Libraries
[Syntax]
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
```

2. Load Dataset
[Syntax]
```
df = pd.read_csv('DBSCAN.csv')
print(df.head())
x = df.iloc[:, 1:]
```

3. Visualize
[Syntax]
```
plt.figure(figsize=(10, 8))
plt.scatter(df['A'], df['B'])
plt.title('Data Scatter Plot')
plt.xlabel('Feature A')
plt.ylabel('Feature B')
plt.show()
```

4.Elbow Method for Optimal Clusters
[Syntax]
```
wcss_list = []

for i in range(1, 11):
   kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
   kmeans.fit(x)
   wcss_list.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss_list)
plt.title('The Elbow Method Graph')
plt.xlabel('Number of clusters (k)')
plt.ylabel('WCSS')
plt.show()
```

5. KMeans Clustering

[Syntax]

```
X = df.iloc[:, 1:].values
km = KMeans(n_clusters=4)
y_means = km.fit_predict(X)

plt.figure(figsize=(10, 8))
plt.scatter(X[y_means == 0, 0], X[y_means == 0, 1], color='blue',
label='Cluster 1')
plt.scatter(X[y_means == 1, 0], X[y_means == 1, 1], color='red',
label='Cluster 2')
plt.scatter(X[y_means == 2, 0], X[y_means == 2, 1], color='green',
label='Cluster 3')
plt.scatter(X[y_means == 3, 0], X[y_means == 3, 1], color='yellow',
label='Cluster 4')
plt.title('KMeans Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

6. Agglomerative Clustering

[Syntax]

```
cluster = AgglomerativeClustering(n_clusters=5, linkage='ward')
labels_ = cluster.fit_predict(X)

plt.figure(figsize=(10, 8))
plt.scatter(X[:, 0], X[:, 1], c=cluster.labels_, cmap='rainbow')
plt.title('Agglomerative Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

7. Agglomerative Clustering

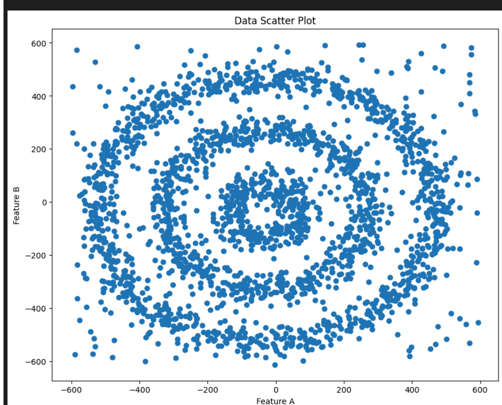
[Syntax]

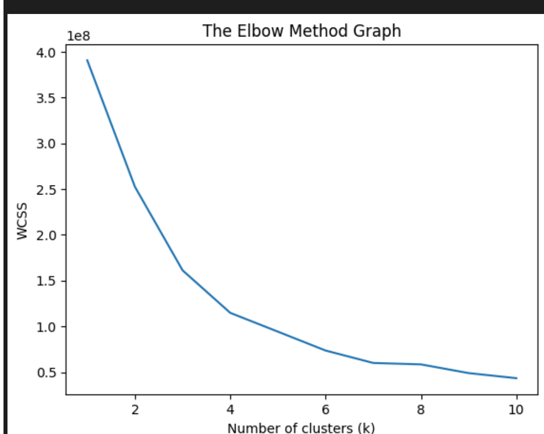
```
DB = DBSCAN(eps=30, min_samples=5)
DB.fit(df[['A', 'B']])
df['DBSCAN_labels'] = DB.labels_

plt.figure(figsize=(10, 8))
```

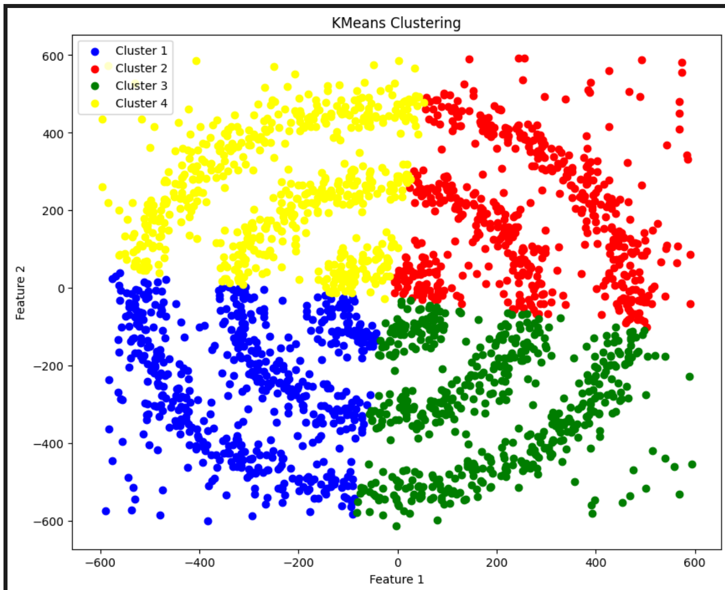
```
plt.scatter(df['A'], df['B'], c=df['DBSCAN_labels'], cmap='rainbow', s=15)
plt.title('DBSCAN Clustering', fontsize=20)
plt.xlabel('Feature 1', fontsize=14)
plt.ylabel('Feature 2', fontsize=14)
plt.show()
```

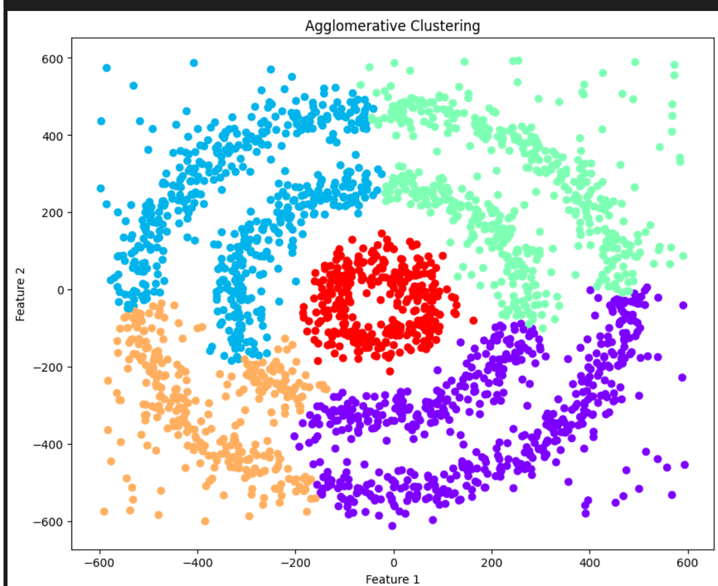
[Output]

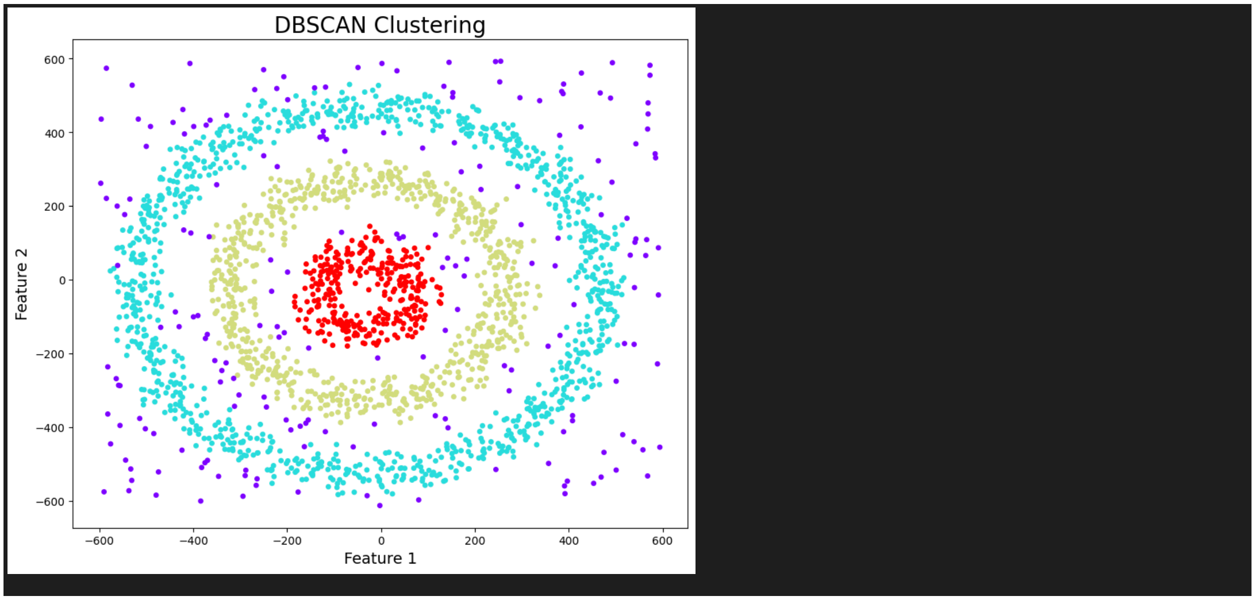
```
  sr.no           A          B
0      0  484.891555 -31.006357
1      1  489.391178  21.973916
2      2  462.886575 -27.599889
3      3  517.218479   5.588090
4      4  455.669049   1.982181
```






KMeans Clustering
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Feature 1
Feature 2
−600
−400
−200
0
200
400
600


Agglomerative Clustering
Feature 1
Feature 2
−600
−400
−200
0
200
400
600



DBSCAN Clustering
Feature 2
Feature 1
600
400
200
0
−200
−400
−600
−600
−400
−200
0
200
400
600