EXPERIMENT 4:

Aim: Demonstration of K-Nearest Neighbors (KNN) for a sample training data set stored as a .CSV file. Calculate the accuracy, precision, and recall for your dataset.

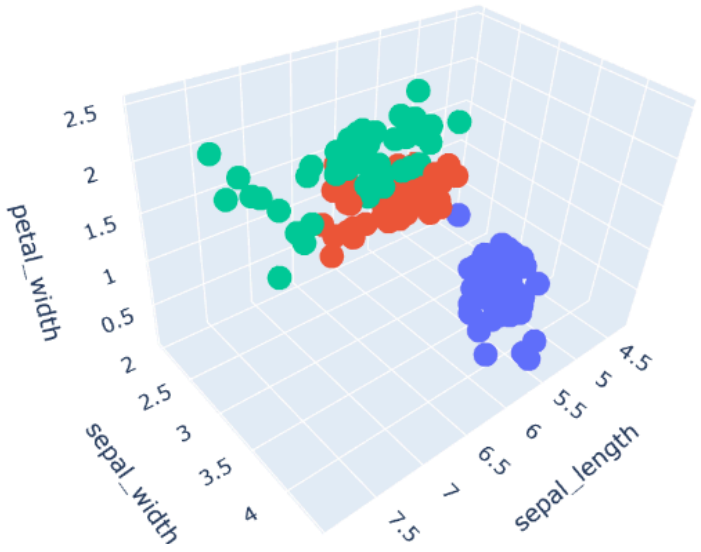
Description:

The Iris dataset is a classic dataset in the field of machine learning, often used for classification tasks. It comprises measurements of iris flowers from three species: Setosa, Versicolor, and Virginica. Each sample includes four features: sepal length, sepal width, petal length, and petal width. In this experiment, we will implement the K-Nearest Neighbors (KNN) algorithm on the Iris dataset to classify iris flowers into their respective species based on these features.

The K-Nearest Neighbors (KNN) algorithm is a simple and effective method for classification. It works on the principle of similarity, where the class of a data point is determined by the classes of its nearest neighbours in the feature space. KNN does not involve explicit training; instead, it stores all available data points and classifies new data points based on a similarity measure, typically using Euclidean distance.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import plotly.express as px
df = px.data.iris()
fig = px.scatter_3d(df, x='sepal_length', y='sepal_width', z='petal_width',  color='species')
fig.show()
```




```
dataset = pd.read_csv('Iris.csv')
X = dataset.iloc[:, 1:-1].values
```

```
y = dataset.iloc[:, -1].values
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
len(y)#Output:150
# from sklearn.model_selection import train_test_split
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5, random_state = 0)
from sklearn.model_selection import RepeatedKFold
kf = RepeatedKFold(n_splits=5, n_repeats=10, random_state=None)
for train_index, test_index in kf.split(X):
    print("Train:", train_index, "Validation:",test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
Train: [  0   1   2   3   4   5   6   7   8  10  11  13  14  15  17  18  20  21
  22  23  25  26  27  28  29  31  32  33  34  36  37  38  39  40  41  43
  45  46  47  48  50  51  52  54  55  56  57  58  60  61  62  64  65  66
  67  69  70  71  72  73  74  75  77  79  80  81  83  84  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 102 103 104 105 106 107 108 109
 110 111 112 113 114 115 116 117 119 121 122 123 126 128 129 130 133 134
 135 136 138 139 140 141 143 144 145 146 147 148] Validation: [  9  12  16  19  24  30  35  42  44  49  53  59  63  68  76  78  82  85
  86 101 118 120 124 125 127 131 132 137 142 149]
Train: [  0   1   2   3   4   5   6   7   8   9  10  11  12  14  15  16  18  19
  20  21  23  24  25  26  27  28  29  30  31  33  35  36  37  38  39  40  41
  42  44  45  46  47  48  49  50  51  52  53  54  56  57  59  62  63  64
  65  66  67  68  70  71  75  76  77  78  79  80  81  82  83  84  85  86
  87  89  90  91  92  93  94  95  96  98  99 100 101 102 103 104 105 106
 107 108 109 110 111 112 114 117 118 119 120 123 124 125 126 127 129 130
 131 132 133 135 136 137 139 142 144 145 146 149] Validation: [ 13  17  22  29  32  34  43  55  58  60  61  69  72  73  74  88  97 113
 115 116 121 122 128 134 138 140 141 143 147 148]
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
len(X_test)#Output:30
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 12, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
print(classifier.predict(sc.transform([[6.2,2.2,4.5,1.5]])))#Output:[1]
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [1 1]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [1 2]
 [2 2]
 [1 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]
 [2 2]]
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[ 6  0  0]
 [ 0 10  2]
 [ 0  0 12]]
0.9333333333333333
```

```
from sklearn.metrics import precision_score, recall_score, f1_score
precision = precision_score(y_test, y_pred,average='macro')
recall = recall_score(y_test, y_pred ,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
```

```
Precision: 0.9523809523809524
Recall: 0.9444444444444445
F1-Score: 0.9440559440559441
```

EXPERIMENT 3:

Aim: Visualising and reducing a high dimensional data using PCA

Description:
The Wine dataset is a classic dataset commonly used for supervised learning tasks such as classification. It contains the results of a chemical analysis of wines grown in the same region in Italy, but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. In this experiment, we will implement Principal Component Analysis (PCA) on the Wine dataset to reduce its dimensionality while preserving its essential information.
Principal Component Analysis (PCA) is a dimensionality reduction technique that aims to transform high-dimensional data into a lower-dimensional space while preserving most of the variance in the original data. PCA achieves this by identifying the principal components, which are the orthogonal directions in the feature space along which the data varies the most.

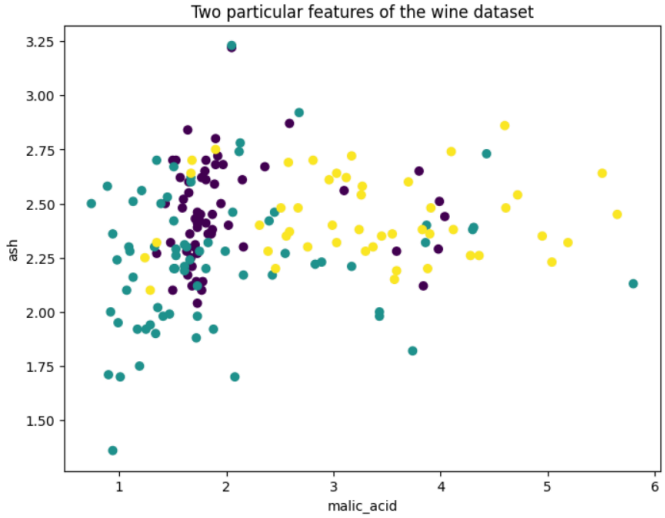
Code:

```
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
winedata = load_wine()
X, y = winedata['data'], winedata['target']
print("X shape:", X.shape)
print("y shape:", y.shape)
```

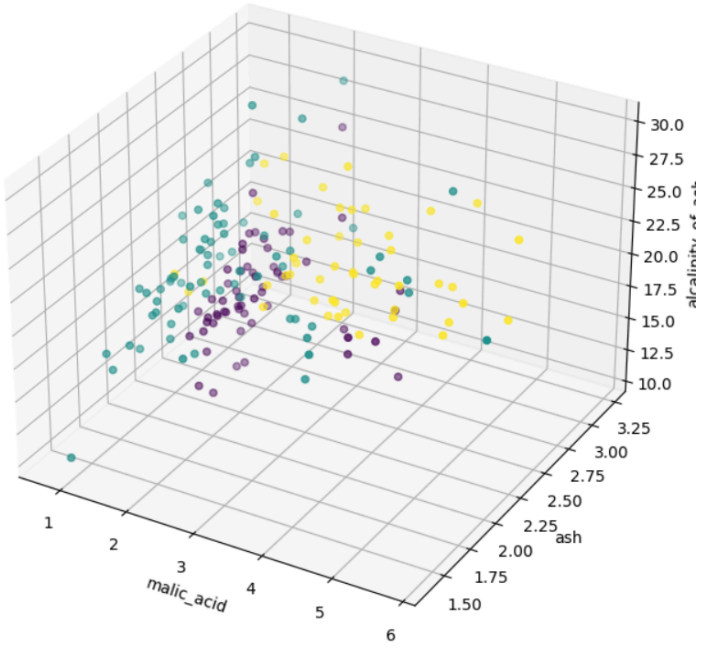
```
X shape: (178, 13)
y shape: (178,)
```

```
# Show any two features
plt.figure(figsize=(8,6))
plt.scatter(X[:,1], X[:,2], c=y)
plt.xlabel(winedata["feature_names"][1])
plt.ylabel(winedata["feature_names"][2])
plt.title("Two particular features of the wine dataset")
plt.show()
```

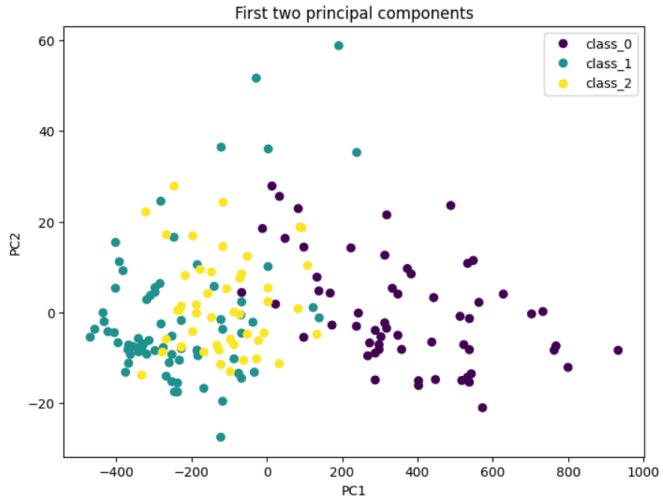
```
X shape: (178, 13)
y shape: (178,)
```




```
# Show any three features
fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(projection='3d')
ax.scatter(X[:,1], X[:,2], X[:,3], c=y)
ax.set_xlabel(winedata["feature_names"][1])
ax.set_ylabel(winedata["feature_names"][2])
ax.set_zlabel(winedata["feature_names"][3])
ax.set_title("Three particular features of the wine dataset")
plt.show()
```

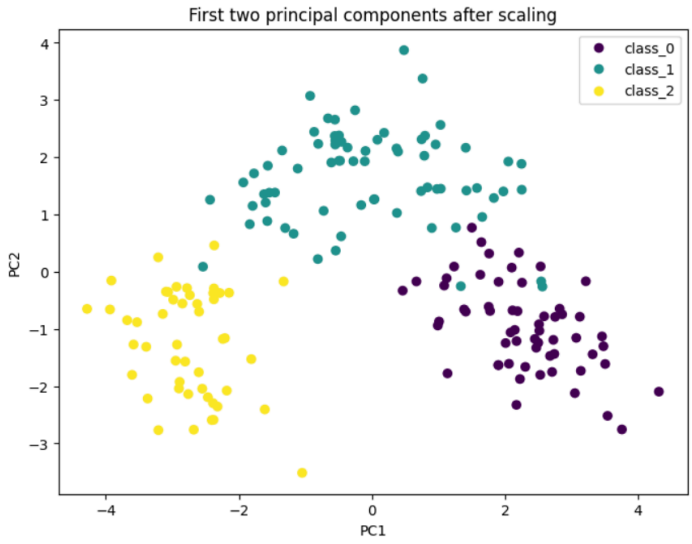


```
# Show first two principal components without scaler
pca = PCA()
plt.figure(figsize=(8,6))
Xt = pca.fit_transform(X)
plot = plt.scatter(Xt[:,0], Xt[:,1], c=y)
plt.legend(handles=plot.legend_elements()[0], labels=list(winedata['target_names']))
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("First two principal components")
plt.show()
```




```
# Show first two principal components with scaler
pca = PCA(2)
pipe = Pipeline([('scaler', StandardScaler()), ('pca', pca)])
plt.figure(figsize=(8,6))
Xt = pipe.fit_transform(X)
print("X shape:", Xt.shape)
print("y shape:", y.shape)
plot = plt.scatter(Xt[:,0], Xt[:,1], c=y)
plt.legend(handles=plot.legend_elements()[0], labels=list(winedata['target_names']))
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("First two principal components after scaling")
plt.show()
```

```
X shape: (178, 2)
y shape: (178,)
```




```
#Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
X = Xt[:,:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
from sklearn.metrics import mean_squared_error, r2_score,accuracy_score
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
# print(Xt)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R2 value:", r2)
```

Linear:

```
Mean Squared Error (MSE): 0.1074160983771034
Root Mean Squared Error (RMSE): 0.32774395246457777
R2 value: 0.8158581170678226
```

```
#polynomial
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
```

```
degree = 2  # Quadratic polynomial
# Create polynomial features
poly_features = PolynomialFeatures(degree=degree)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)
model = LinearRegression()
model.fit(X_train_poly, y_train)
y_train_pred = model.predict(X_train_poly)
train_mse = mean_squared_error(y_train, y_train_pred)
print("Training MSE:", train_mse)
y_test_pred = model.predict(X_test_poly)
test_mse = mean_squared_error(y_test, y_test_pred)
print("Testing MSE:", test_mse)
r2_train = r2_score(y_train, y_train_pred)
print("R2 value (Training):", r2_train)
r2_test = r2_score(y_test, y_test_pred)
print("R2 value (Testing):", r2_test)
```

Polynomial(Quadratic):

```
Training MSE: 0.08772780500981965
Testing MSE: 0.05602080382009817
R2 value (Training): 0.8531387745771687
R2 value (Testing): 0.9039643363084031
```

```
#Lasso
alpha = 0.018  # Regularisation parameter
model = Lasso(alpha=alpha)
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)
train_mse = mean_squared_error(y_train, y_train_pred)
print("Training MSE:", train_mse)
y_test_pred = model.predict(X_test)
test_mse = mean_squared_error(y_test, y_test_pred)
print("Testing MSE:", test_mse)
r2_train = r2_score(y_train, y_train_pred)
print("R2 value (Training):", r2_train)
r2_test = r2_score(y_test, y_test_pred)
print("R2 value (Testing):", r2_test)
```

```
Training MSE: 0.12432607449995643
Testing MSE: 0.1081189854071411
R2 value (Training): 0.7918712356814345
R2 value (Testing): 0.8146531678734723
```

```
#Ridge
alpha = 0.01  # Regularisation parameter
model = Ridge(alpha=alpha)
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)
train_mse = mean_squared_error(y_train, y_train_pred)
print("Training MSE:", train_mse)
y_test_pred = model.predict(X_test)
test_mse = mean_squared_error(y_test, y_test_pred)
print("Testing MSE:", test_mse)
r2_train = r2_score(y_train, y_train_pred)
print("R2 value (Training):", r2_train)
r2_test = r2_score(y_test, y_test_pred)
print("R2 value (Testing):", r2_test)
```

```
Training MSE: 0.1241569657835474
Testing MSE: 0.10741700309836141
R2 value (Training): 0.7921543330793317
R2 value (Testing): 0.8158565661170947
```

EXPERIMENT 2:

Aim: Build linear regression model using gradient descent, least squares, polynomial, LASSO and RIDGE approaches also compare all the algorithms and draw a table for all the metrics.

Description:

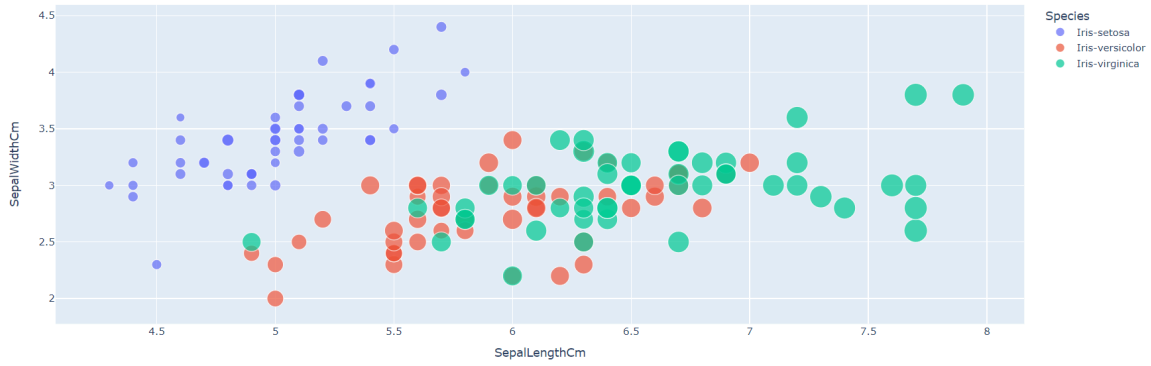
To conduct an experiment comparing various regression algorithms using the Iris dataset, five regression techniques were employed: gradient descent, least squares, polynomial regression, LASSO, and Ridge regression. Beginning with gradient descent, an iterative optimization algorithm was used to minimise the cost function by adjusting model parameters based on the gradient of the cost function. Following this, the least squares method was applied, directly minimising the sum of squared differences between observed and predicted values. Polynomial regression was then implemented, extending the linear regression model to include polynomial features, accommodating non-linear relationships between the features and target variable. Moving on to regularisation techniques, LASSO regression and Ridge regression were utilised. LASSO adds a penalty term to the least squares method, constraining the sum of the absolute values of the coefficients, while Ridge regression adds a penalty term to the least squares method, constraining the sum of the squares of the coefficients. Performance evaluation was conducted using metrics such as mean squared error (MSE), mean absolute error (MAE), and R-squared (R2) score to assess accuracy, precision, and goodness of fit. Results were compared across these metrics and summarised in a table to identify the most effective regression technique for predicting the target variable in the Iris dataset. This experiment aimed to provide a comprehensive comparison of different regression approaches and their suitability for the given dataset.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns #for visualising the data
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

df = pd.read_csv('/content/Iris.csv')
df.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

```
import plotly.express as px
fig = px.scatter(df, x="SepalLengthCm", y="SepalWidthCm", color="Species",
                 size='PetalLengthCm', hover_data=['PetalWidthCm'])
fig.show()
```




```
df.head(10)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

df.isnull().sum()

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm','PetalWidthCm']] =
scaler.fit_transform(df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm','PetalWidthCm']])
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	0.222222	0.625000	0.067797	0.041667	0
1	2	0.166667	0.416667	0.067797	0.041667	0
2	3	0.111111	0.500000	0.050847	0.041667	0
3	4	0.083333	0.458333	0.084746	0.041667	0
4	5	0.194444	0.666667	0.067797	0.041667	0

```
from scipy import stats
z_scores = stats.zscore(df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm','PetalWidthCm']])
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)  # Keep only the rows where all feature values have a
Z-score less than 3
X_without_outliers = df[filtered_entries]
df = X_without_outliers
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	0.222222	0.625000	0.067797	0.041667	0
1	2	0.166667	0.416667	0.067797	0.041667	0
2	3	0.111111	0.500000	0.050847	0.041667	0
3	4	0.083333	0.458333	0.084746	0.041667	0
4	5	0.194444	0.666667	0.067797	0.041667	0

```
X = df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm','PetalWidthCm']]
# Dependent variable (target)
y = df['Species']
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

y_pred = model.predict(X_test)

from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error

mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("R2 value:", r2)
```

```
Mean Squared Error (MSE): 0.054023465548463184
Root Mean Squared Error (RMSE): 0.23242948510992142
Mean Absolute Error (MAE): 0.18570205937097298
R2 value: 0.9217051223935316
```

```
y_pred_train = model.predict(X_train)

from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

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

# Define the degree of the polynomial
degree = 2  # Quadratic polynomial
poly_features = PolynomialFeatures(degree=degree)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)

model = LinearRegression()
model.fit(X_train_poly, y_train)

y_train_poly_pred = model.predict(X_train_poly)
y_test_poly_pred = model.predict(X_test_poly)
test_mse_polynomial = mean_squared_error(y_test, y_test_poly_pred)
```

```
r2_test_poly = r2_score(y_test, y_test_poly_pred)
rmse_test_polynomial = mean_squared_error(y_test, y_test_poly_pred, squared=False)
mae_test_polynomial = mean_absolute_error(y_test, y_test_poly_pred)

print("Testing MSE for polynomial regression:", test_mse_polynomial)
print("R2 value (Testing) for polynomial regression:", r2_test_poly)
print("Testing RMSE for polynomial regression:", rmse_test_polynomial)
print("Testing MAE for polynomial regression:", mae_test_polynomial)
```

```
Testing MSE for polynomial regression: 0.0660378187647385
R2 value (Testing) for polynomial regression: 0.9055102752173853
Testing RMSE for polynomial regression: 0.25697824570328615
Testing MAE for polynomial regression: 0.17936397677597282
```

```
#Lasso
alpha = 0.01  # Regularisation parameter
model = Lasso(alpha=alpha)
model.fit(X_train, y_train)

y_train_lasso_pred = model.predict(X_train)
train_lasso_mse = mean_squared_error(y_train, y_train_lasso_pred)
y_test_lasso_pred = model.predict(X_test)
test_lasso_mse = mean_squared_error(y_test, y_test_lasso_pred)

rmse_test_lasso = mean_squared_error(y_test, y_test_lasso_pred, squared=False)
mae_test_lasso = mean_absolute_error(y_test, y_test_lasso_pred)
r2_test_lasso = r2_score(y_test, y_test_lasso_pred)

print("Testing MSE for lasso:", test_lasso_mse)
print("Testing RMSE for lasso:", rmse_test_lasso)
print("Testing MAE for lasso:", mae_test_lasso)
print("R2 value (Testing) for lasso:", r2_test_lasso)
```

```
Testing MSE for lasso: 0.056584478335368955
Testing RMSE for lasso: 0.23787492161925972
Testing MAE for lasso: 0.17763058251751598
R2 value (Testing) for lasso: 0.9190365174851637
```

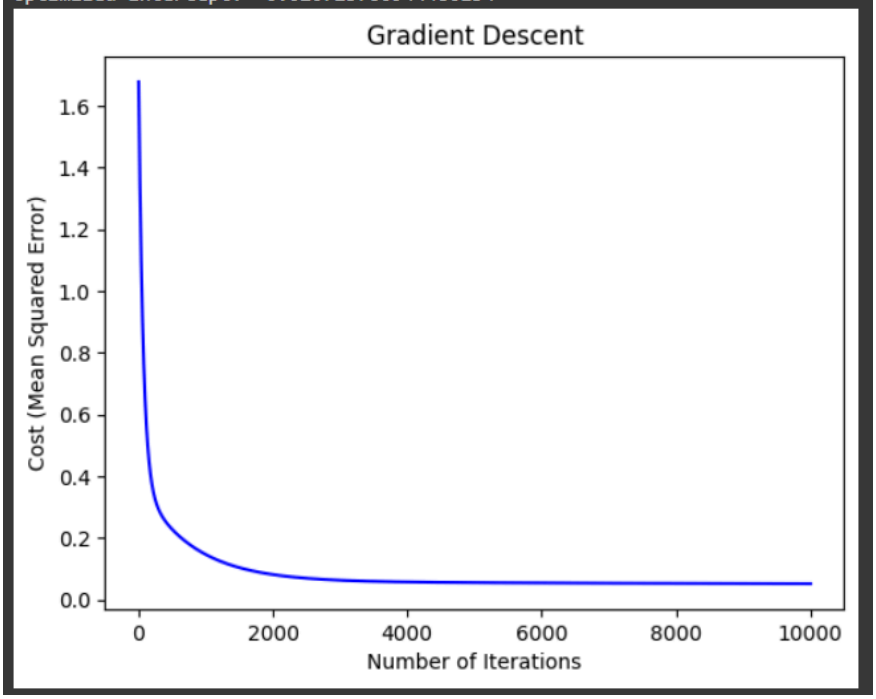
```
#Ridge
alpha = 0.06  # Regularization parameter
model = Ridge(alpha=alpha)
```

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

test_ridge_mse = mean_squared_error(y_test, y_test_ridge_pred)
rmse_test_ridge = mean_squared_error(y_test, y_test_ridge_pred, squared=False)
mae_test_ridge = mean_absolute_error(y_test, y_test_ridge_pred)
r2_test_ridge = r2_score(y_test, y_test_ridge_pred)

print("Testing MSE for ridge:", test_ridge_mse)
print("Testing RMSE for ridge:", rmse_test_ridge)
print("Testing MAE for ridge:", mae_test_ridge)
print("R2 value (Testing) for ridge:", r2_test_ridge)
```

```
Testing MSE for ridge: 0.037150813373876 14
Testing RMSE for ridge: 0.19274546265444523
Testing MAE for ridge: 0.14622221075330616
R2 value (Testing) for ridge: 0.9468430333283171
```

```
#Gradient Descient
coefficients = np.zeros(X.shape[1])
intercept = 0
learning_rate = 0.002
epochs = 10000 #if we are getting nan or -inf as the output adjust the values of l and epochs
n = len(X)
cost_history = []
for i in range(epochs):
    y_pred = np.dot(X, coefficients) + intercept
    cost = np.mean((y_pred - y) ** 2)
    cost_history.append(cost)
    D_coefficients = (-2/n) * np.dot(X.T, (y - y_pred))
    D_intercept = (-2/n) * np.sum(y - y_pred)
    coefficients = coefficients - learning_rate * D_coefficients
    intercept = intercept - learning_rate * D_intercept
print("Optimized coefficients:", coefficients)
print("Optimized intercept:", intercept)
plt.plot(range(1, epochs + 1), cost_history, color='blue')
plt.xlabel('Number of Iterations')
plt.ylabel('Cost (Mean Squared Error)')
plt.title('Gradient Descent')
plt.show()
```

```
Optimized coefficients: [ 0.32277355 -0.36910654  1.01772959  1.24049485]
Optimized intercept: -0.020723760944480134
```



EXPERIMENT 1:

Aim: Demonstrating Data Preprocessing and Linear Regression

Description:
Linear Regression is a foundational statistical technique used for modelling the relationship between a dependent variable (often denoted as 'y') and one or more independent variables (often denoted as 'x'). The goal of linear regression is to fit a linear equation to the observed data that best predicts the dependent variable based on the independent variables. The linear equation has the form:
$y=b0+b1x1+b2x2+b3x3….+bnxn + \epsilon$
Linear regression assumes a linear relationship between the independent and dependent variables. It is often used for prediction and inference tasks in various fields such as economics, finance, social sciences, and engineering.
Data Preprocessing is a crucial step in the machine learning pipeline aimed at preparing the dataset for analysis and modelling. It involves cleaning, transforming, and organising the raw data into a format suitable for machine learning algorithms. Some common techniques involved in data preprocessing include handling missing values, feature scaling, feature encoding, feature engineering, train-test split, and outlier detection and removal. Data preprocessing ensures that the data is clean, consistent, and suitable for analysis, ultimately leading to the development of robust and accurate machine learning models.

Code:
```
#data preprocessing
#step 1 - importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns #for visualising the data
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler #For feature engineering

#step 2 - loading the dataset
df = pd.read_csv('/content/bmi.csv')
df.describe()
```

	Height	Weight	Index
count	500.000000	500.000000	500.000000
mean	169.944000	106.000000	3.748000
std	16.375261	32.382607	1.355053
min	140.000000	50.000000	0.000000
25%	156.000000	80.000000	3.000000
50%	170.500000	106.000000	4.000000
75%	184.000000	136.000000	5.000000
max	199.000000	160.000000	5.000000

df.head()

	Gender	Height	Weight	Index
0	Male	174	96	4
1	Male	189	87	2
2	Female	185	110	4
3	Female	195	104	3
4	Male	149	61	3

#check for the null values

df.isnull().sum()

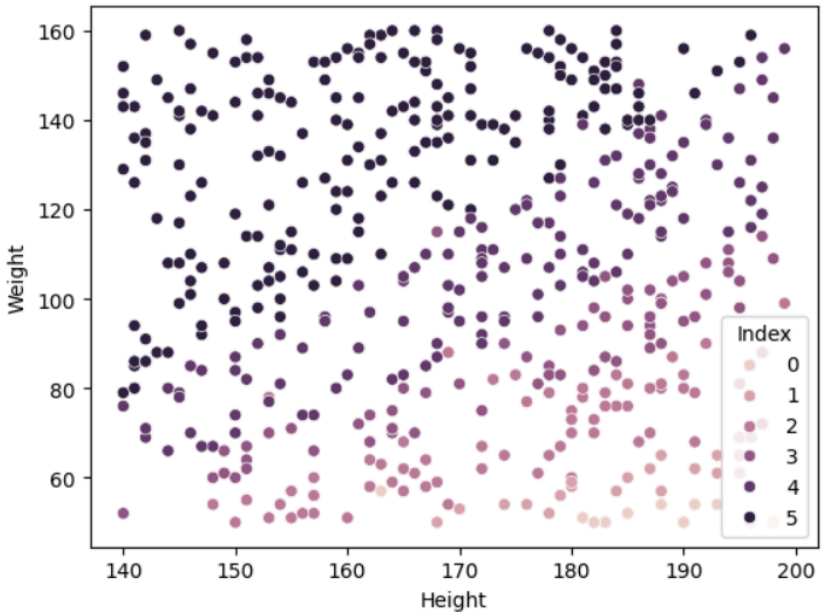
```
Gender    0
Height    0
Weight    0
Index     0
dtype: int64
```

#step -3 statistical analysis

#use seaborn for data visualisation

sns.scatterplot(x=df['Height'], y=df['Weight'], hue=df["Index"])

```
<Axes: xlabel='Height', ylabel='Weight'>
```




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

	Height	Weight	Index	Gender_Male
0	174	96	4	1
1	189	87	2	1
2	185	110	4	0
3	195	104	3	0
4	149	61	3	1

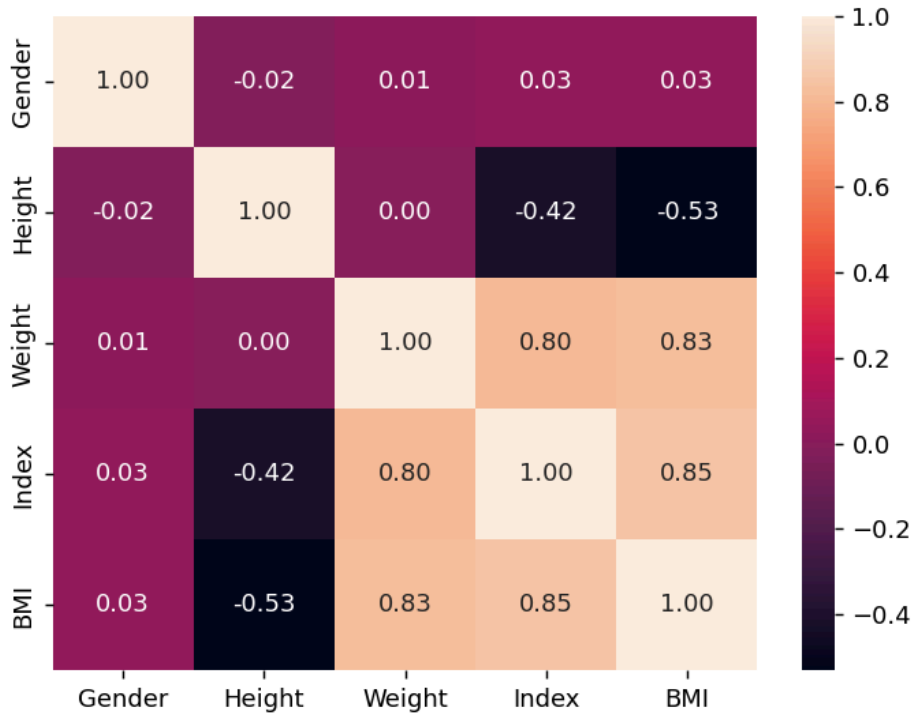
```
#feature engineering
df['BMI'] = df['Weight'] / ((df['Height'] / 100) ** 2)
df.head()
```

	Height	Weight	Index	Gender_Male	BMI
0	174	96	4	1	31.708284
1	189	87	2	1	24.355421
2	185	110	4	0	32.140248
3	195	104	3	0	27.350427
4	149	61	3	1	27.476240

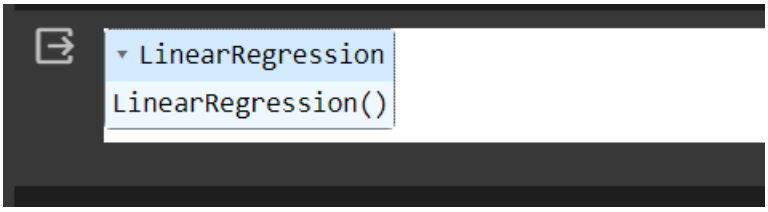
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[['Height', 'Weight','BMI']] = scaler.fit_transform(df[['Height', 'Weight','BMI']])
```

	Gender	Height	Weight	Index	BMI
0	1	0.576271	0.418182	4	0.286756
1	1	0.830508	0.336364	2	0.175517
2	0	0.762712	0.545455	4	0.293291
3	0	0.932203	0.490909	3	0.220828
4	1	0.152542	0.100000	3	0.222731

```
corr = df.corr()
plt.figure(dpi=130)
sns.heatmap(df.corr(), annot=True, fmt= '.2f')
plt.show()
```



```
X = df[['Gender','Height','Weight']].values
y = df['Index'].values
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
```




```
y_pred = model.predict(X_test)

from sklearn.metrics import mean_squared_error, r2_score,accuracy_score
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R2 value:", r2)
```

```
Mean Squared Error (MSE): 0.3393556144069988
Root Mean Squared Error (RMSE): 0.5825423713404878
R2 value: 0.7962929261018075
```