# GURU JAMBHESHWAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

## (Hisar-Haryana)


## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# Practical file

# Machine Learning

## (PCC-CSEAI301-P)

**Submitted to :**

**Dr. Narender**

**Dept. of CSE**

**Submitted by :**

**Sarita**

**220010150027**

**B.Tech CSE- AI & ML**

# INDEX

# 1. Assignment demonstrating Linear Regression:

## a) Implementing linear regression on placement dataset and predicting the dependent variable.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df=pd.read_csv("placement_data.csv")
```

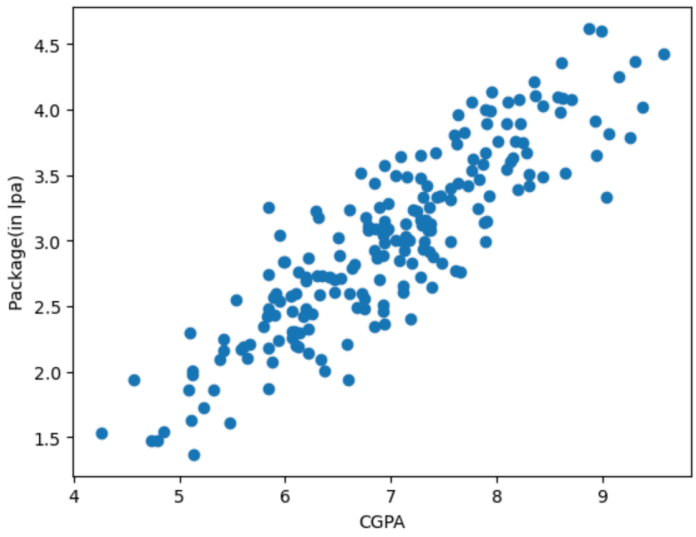
[2]:
```
df.head()
```

[2]:

| | cgpa | package |
|---|---|---|
| 0 | 6.89 | 3.26 |
| 1 | 5.12 | 1.98 |
| 2 | 7.82 | 3.25 |
| 3 | 7.42 | 3.67 |
| 4 | 6.94 | 3.57 |

[3]:
```
plt.scatter(df['cgpa'],df['package'])
plt.xlabel('CGPA')
plt.ylabel('Package(in lpa)')
```

[3]:
```
Text(0, 0.5, 'Package(in lpa)')
```




[4]:
```
X=df.iloc[:,0:1]
Y=df.iloc[:,-1]
```

[5]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2,random_state=123)
```

[6]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train,Y_train)
```

[6]:
```
LinearRegression
LinearRegression()
```

[7]:
```
X_test[:5]
```

[7]:

| | cgpa |
|---|---|
| **50** | 9.58 |
| **127** | 6.78 |
| **37** | 5.90 |
| **149** | 8.28 |
| **19** | 7.48 |

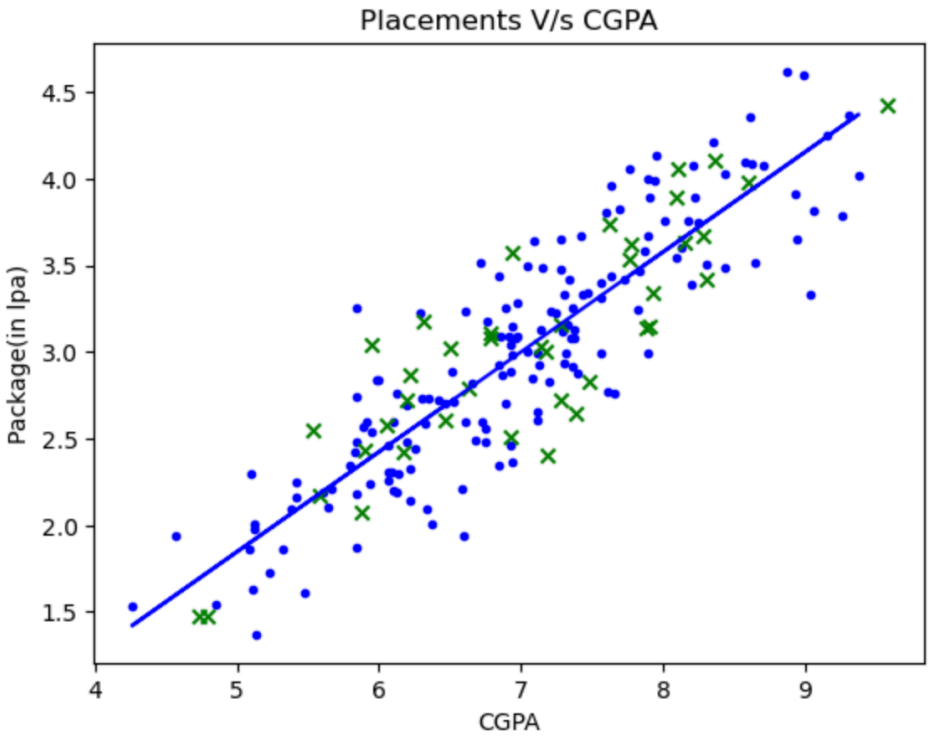
[8]:
```
Y_test[:5]
```

[8]:
```
50     4.43
127    3.11
37     2.43
149    3.67
19     2.83
Name: package, dtype: float64
```

[9]:
```
Y_predicted = lr.predict(X_train)
```

[10]:
```
# Plot the linear fit
plt.scatter(X_train, Y_train, marker='.', c='b')
plt.plot(X_train, Y_predicted, c = "b")
plt.scatter(X_test, Y_test, marker='x', c='g')
plt.title("Placements V/s CGPA")
plt.ylabel('Package(in lpa)')
plt.xlabel('CGPA')
```

[10]:
```
Text(0.5, 0, 'CGPA')
```



## b) Implementing linear regression on randomly generated dataset and evaluation of the regression model using R2 score.

[1]:
```
# importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
```

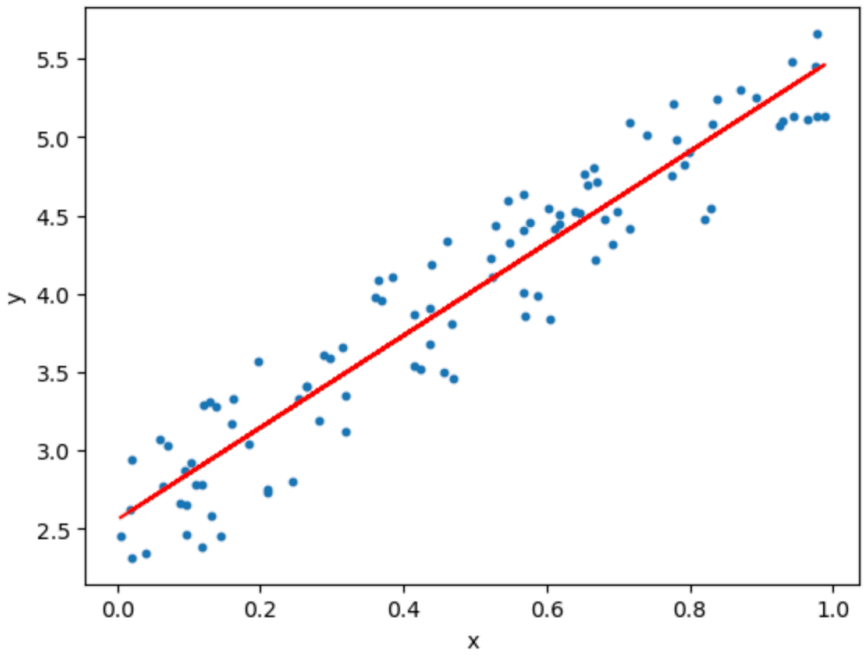
[2]:
```
#generating  a random dataset
np.random.seed(0)
x=np.random.rand(100,1)
y=2+3*x+np.random.rand(100,1)

#sckit_learn implementation

#model_initialisation
r_model=LinearRegression()
#fit the data(Train the model)
r_model.fit(x,y)
#predict
y_predicted=r_model.predict(x)
```

[3]:
```
#model evaluation
rmse=mean_squared_error(y,y_predicted)
r2=r2_score(y,y_predicted)
```

[4]:
```
#printing values
print('slope:',r_model.coef_)
print('intercept:',r_model.intercept_)
print("root mean squared error:",rmse)
print("R2 score:",r2)
```

```
slope: [[2.93655106]]
intercept: [2.55808002]
root mean squared error: 0.07623324582875009
R2 score: 0.9038655568672764
```

[5]:
```
#plotting values
#data points
plt.scatter(x,y,s=10)
plt.xlabel('x')
plt.ylabel('y')

#predicted values
plt.plot(x,y_predicted, c='r')
plt.show()
```



# 2. Implementing and demonstrating the Find-S algorithm for finding most specific hypothesis.

[1]:
```python
#Initialize the hypothesis with the most specific hypothesis
def initialize_hypothesis(attributes):
    hypothesis = {}
    for attribute in attributes:
        hypothesis[attribute] = "null"
    return hypothesis
```

[2]:
```python
# Update the hypothesis based on a positive example
def update_hypothesis(hypothesis, example):
    for attribute, value in example.items():
        if hypothesis[attribute] == "null":
            hypothesis[attribute] = value
        elif hypothesis[attribute] != value:
            hypothesis [attribute]="?"
    return hypothesis
```

[3]:
```python
#Find-S algorithm
def find_s(training_data):
    attributes= list(training_data[0].keys())
    hypothesis= initialize_hypothesis(attributes)
    for example in training_data:
        if example['target'] == 'cat':
            hypothesis = update_hypothesis(hypothesis, example)
    return hypothesis
```

[4]:
```python
#Example training data
training_data = [
     {'color': 'brown', 'size': 'small', 'tail': 'long', 'target': 'cat'},
     {'color': 'gray', 'size': 'medium', 'tail': 'short', 'target': 'cat'},
     {'color': 'black', 'size': 'large', 'tail': 'long', 'target': 'not_cat'},
     {'color': 'white', 'size': 'small', 'tail': 'short', 'target': 'not_cat'}
]
```

[5]:
```python
#Apply Find-S algorithm
learned_hypothesis = find_s(training_data)
print("Learned Hypothesis:", learned_hypothesis)
```

```
Learned Hypothesis: {'color': '?', 'size': '?', 'tail': '?', 'target': 'cat'}
```

# 3. Implementing Candidate Elimination algorithm and finding specific and general boundary sets of hypotheses consistent with EnjoySport dataset.

[1]:
```
import numpy as np
import pandas as pd
data = pd.read_csv('enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\n Instances are:\n", concepts)
target = np.array(data.iloc[:,-1])
print("\n Target Values are: ",target)
```

```
 Instances are:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

 Target Values are:  ['yes' 'yes' 'no' 'yes']
```

[2]:
```
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\n Initlalization of specific_h and genearal_h")
    print("\n Specific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ", general_h)

    for i,h in enumerate(concepts):
        print("\nInstance ", i+1, " is ", h)
        if target[i] == "yes":
            print("Instance is Positive")
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = "?"
                    general_h[x][x] = '?'

        if target[i] == "no":
            print("Instance is Negative")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])

    return specific_h, general_h
```

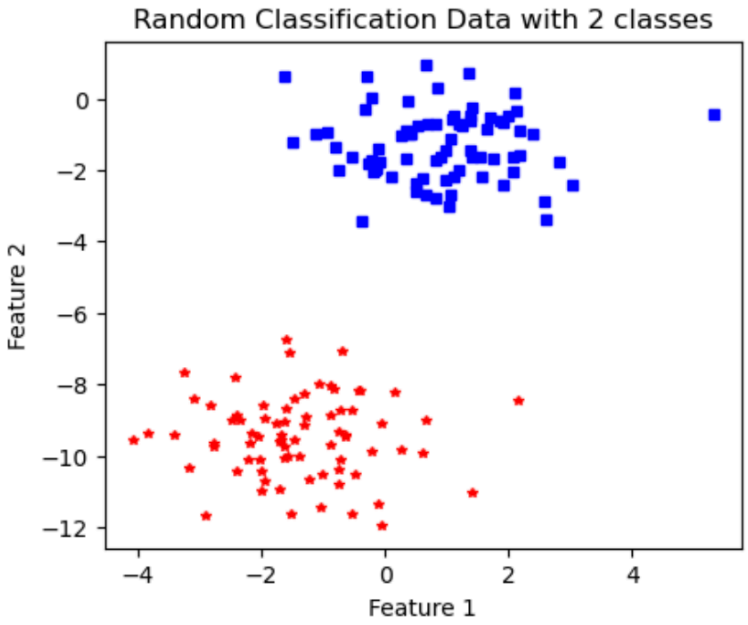
[3]:
```
s_final, g_final = learn(concepts, target)
print("Final Specific_h: ", s_final, sep="\n")
print("Final General h: ", g_final, sep="\n")
```

```
 Initlalization of specific_h and genearal_h

 Specific Boundary:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance  1  is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  1 Instance is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance  2  is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance  3  is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
Specific Bundary after  3 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  3 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]



Instance  4  is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
Specific Bundary after  4 Instance is  ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Boundary after  4 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

# 4. Implementing  Perceptron learning from scratch and showing decision boundary.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
X, y = datasets.make_blobs(n_samples=150, n_features=2, centers=2,
                           cluster_std=1.05, random_state=2)
#Plotting
X.shape
fig = plt.figure(figsize=(5,4))
plt.plot(X[:, 0][y == 0], X[:, 1][y == 0], "r*", markersize=4)
plt.plot(X[:, 0][y == 1], X[:, 1][y == 1], 'bs', markersize=4)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title('Random Classification Data with 2 classes')
```

[1]:
```
Text(0.5, 1.0, 'Random Classification Data with 2 classes')
```

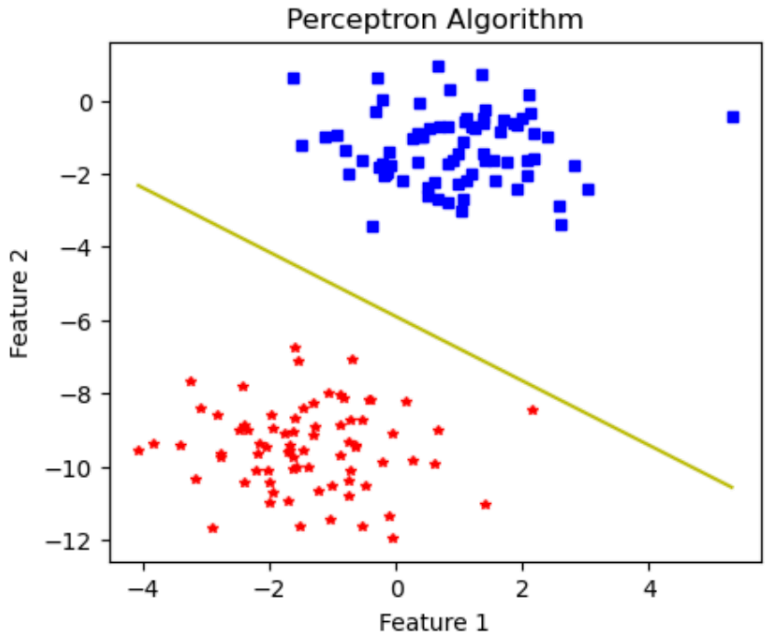



[2]:
```
def perceptron(X, y, lr, epochs):
    m, n = X.shape
    theta = np.zeros((n+1,1))
    n_miss_list = []
    for epoch in range(epochs):
        n_miss = 0
        for idx, x_i in enumerate(X):
            x_i = np.insert(x_i, 0, 1).reshape(-1,1)
            y_hat = step_func(np.dot(x_i.T, theta))
            if (np.squeeze(y_hat) - y[idx]) != 0:
                theta += lr*((y[idx] - y_hat)*x_i)
                n_miss += 1
        n_miss_list.append(n_miss)
    return theta, n_miss_list
```

[3]:
```
def step_func(z):
    return 1.0 if (z>0) else 0.0
```

[4]:
```
def plot_decision_boundary(X, theta):
    x1 = [min(X[:,0]), max(X[:,0])]
    m = -theta[1]/theta[2]
    c = -theta[0]/theta[2]
    x2 = m*x1 + c
    fig = plt.figure(figsize=(5,4))
    plt.plot(X[:, 0][y == 0], X[:, 1][y == 0], "r*", markersize=4)
    plt.plot(X[:, 0][y == 1], X[:, 1][y == 1], 'bs', markersize=4)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title('Perceptron Algorithm')
    plt.plot(x1, x2,"y-")
```

[5]:
```
theta, miss_1 = perceptron(X, y , 0.5, 100)
plot_decision_boundary(X, theta)
```



# 5. Implementing classification using SVM :

## a) For iris dataset using SVC default settings.

[1]:
```
from sklearn.datasets import load_iris
iris = load_iris()
dir(iris)
iris.data
iris.target
iris.target_names
iris.feature_names
```

[1]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

[2]:
```
import pandas as pd
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df.head()
df['target'] = iris.target
df.head()
```

[2]:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 0 |

[3]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:
```
sns.pairplot(df, hue='target', palette = 'brg')
plt.show()
```

[5]:
```
x = df.drop(['target'], axis = 'columns')
y = df.target
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
```

[6]:
```
from sklearn.svm import SVC
model = SVC()
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

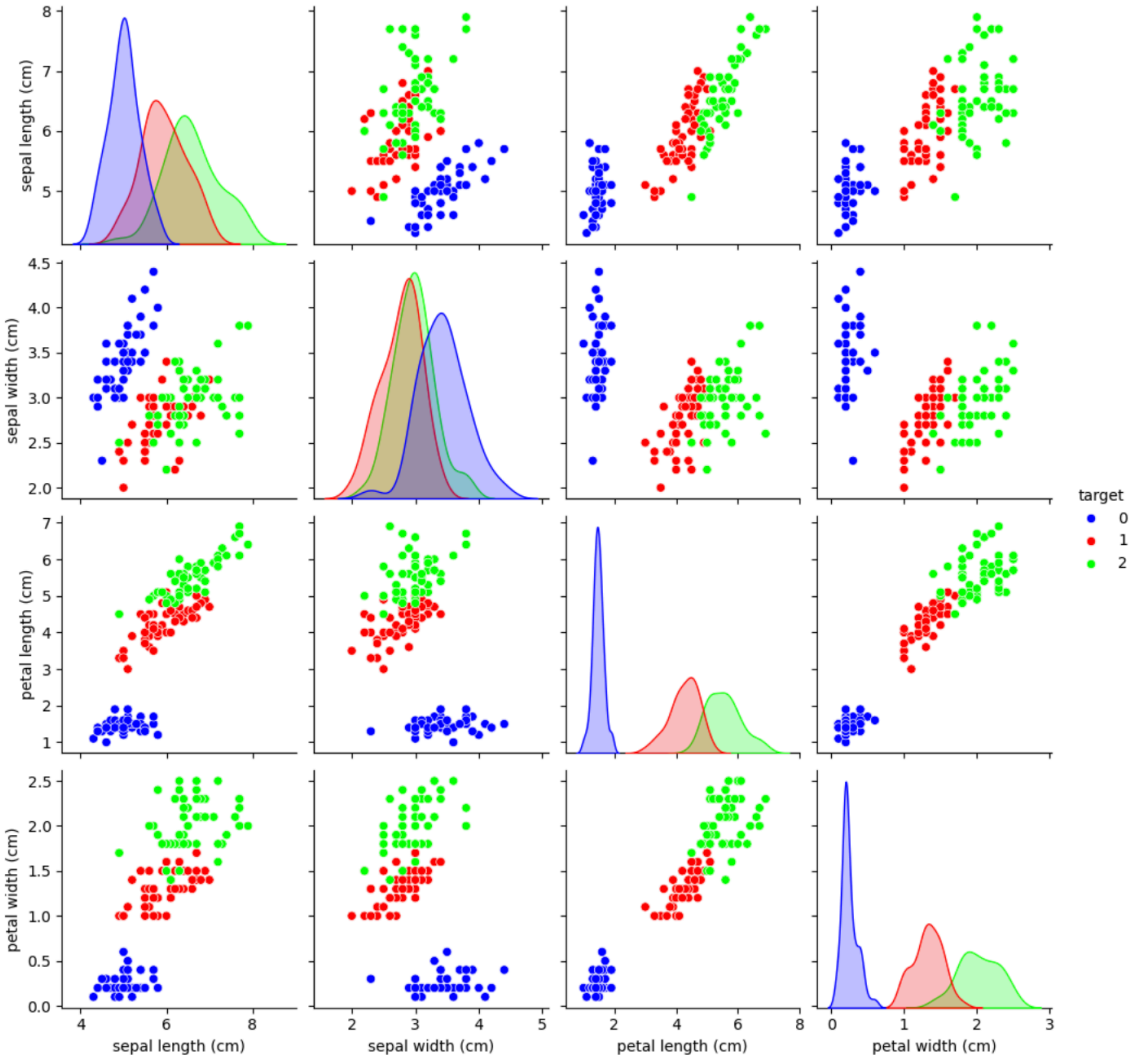
[6]:
```
0.9777777777777777
```

[7]:
```
model.predict(pd.DataFrame([iris.data[50]],columns=iris.feature_names))
```

[7]:
```
array([1])
```

[8]:
```
model.predict(pd.DataFrame([[6.5, 3.0, 5.2, 2.0]],columns=iris.feature_names))
```

[8]:
```
array([2])
```


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
target
0
1
2

## b) For recognition of handwritten digits.

[1]:
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
digits = load_digits()
dir(digits)
```

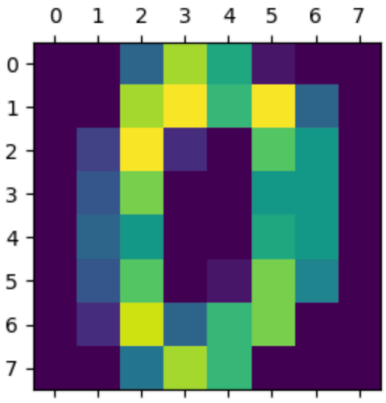
[1]:
```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

[2]:
```
digits.data[0]
```

[2]:
```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

[3]:
```
plt.figure(figsize=(3,3))
plt.matshow(digits.images[0],fignum=1)
digits.target [0]
```

[3]:
```
0
```




[4]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2)
x_train
```

[4]:
```
array([[ 0.,  0.,  7., ..., 16.,  9.,  0.],
       [ 0.,  0., 15., ...,  0.,  0.,  0.],
       [ 0.,  1.,  8., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 14.,  5.,  0.],
       [ 0.,  0.,  0., ..., 15.,  8.,  0.]])
```

[5]:
```
from sklearn.svm import SVC
model = SVC()
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

[5]:
```
0.9888888888888889
```

If we want to use different kernels and find the accuracy levels, we can use the following code :

[7]:
```
# C=1.0, kernel='linear', gamna='scale'
model = SVC(kernel='linear')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

[7]:
```
0.9944444444444445
```

[8]:
```
# C=1.0, kernel='sigmoid', gamma='scale'
model = SVC(kernel='sigmoid')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

[8]: 0.9055555555555556

[9]:
```
# C=1.0, kernel='poly', gamna='scale'
model = SVC(kernel='poly')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

[9]: 0.9944444444444445

[10]:
```
# C=1.0, kernel='poly', gamma='auto'
model = SVC(kernel='poly', gamma='auto')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

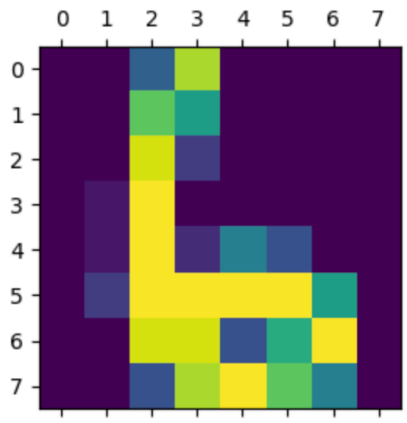
[10]: 0.9944444444444445

[11]:
```
x_test[100]
print(f'Predicted Digit: {model.predict([x_test[100]])} \nActual Digit : {y_test[100]}')
```

```
Predicted Digit: [6]
Actual Digit : 6
```

[12]:
```
plt.figure(figsize=(3,3))
plt.matshow(digits.images[67],fignum=1)
model.predict([digits.data[67]])
digits.target[67]
```

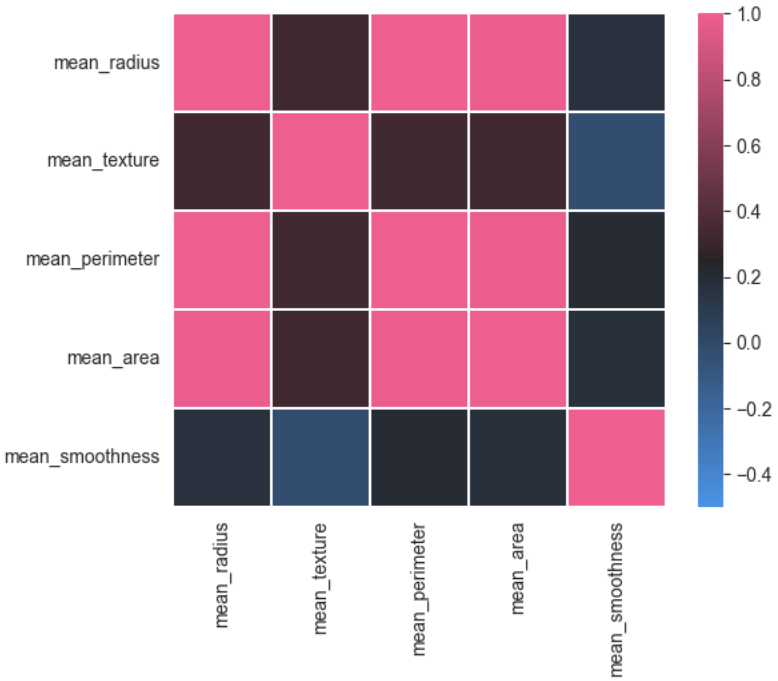
[12]: 6

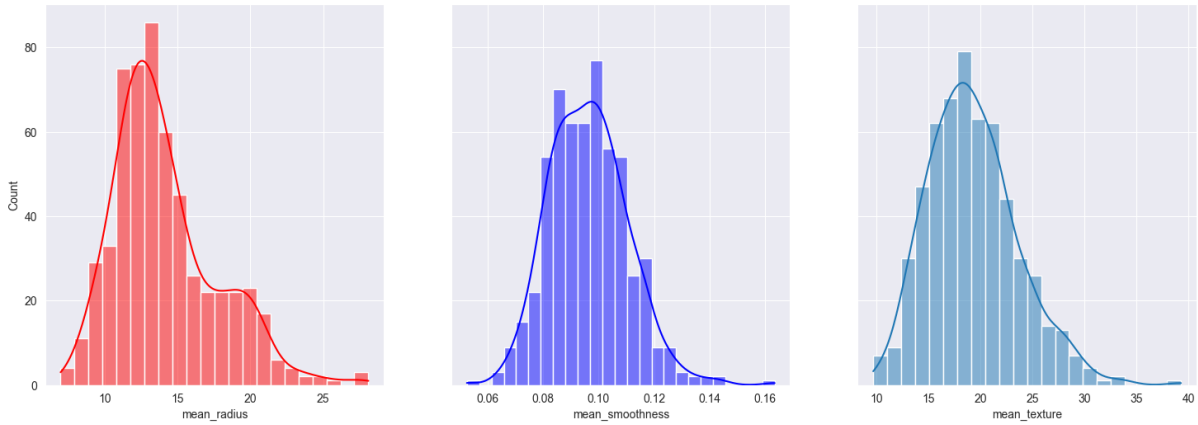


# 6. Implement Naïve Bayes Classifier

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
data = pd.read_csv("breast-cancer.csv")
data.head(10)

# Convert diagnosis column to numerical values
data["diagnosis"] = data["diagnosis"].map({'M': 1, 'B': 0})

data["diagnosis"].hist()
corr = data.iloc[:,:-1].corr(method="pearson")
cmap = sns.diverging_palette(250, 354, 80, 60, center = "dark", as_cmap=True)
sns.heatmap(corr, vmax=1, vmin=-.5, cmap=cmap, square=True, linewidths=.2)
data = data[["mean_radius", "mean_texture", "mean_smoothness", "diagnosis"]]
data.head(10)
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
sns.histplot(data, ax=axes[0], x="mean_radius", kde=True, color='r')
sns.histplot(data, ax=axes[1], x="mean_smoothness", kde=True, color='b')
sns.histplot(data, ax=axes[2], x="mean_texture", kde=True)
```

[9] ✓ 0.3s






```python
def calculate_prior(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior = []
    for i in classes:
        prior.append(len(df[df[Y]==i])/len(df))
    return prior
```

[7] ✓ 0.0s

```python
def calculate_likelihood_gaussian(df, feat_name, feat_val, Y, label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    mean, std = df[feat_name].mean(), df[feat_name].std()
    p_x_given_y = (1 / (np.sqrt(2 * np.pi) * std)) *  np.exp(-((feat_val-mean)**2 / (2 * std**2 )))
    return p_x_given_y
```

[8] ✓ 0.0s

```python
def naive_bayes_gaussian(df, X, Y):
    # get feature names
    features = list(df.columns)[:-1]

    # calculate prior
    prior = calculate_prior(df, Y)

    Y_pred = []
    # loop over every data sample
    for x in X:
        # calculate likelihood
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calculate_likelihood_gaussian(df, features[i], x[i], Y, labels[j])

        # calculate posterior probability (numerator only)
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)
```

[9] ✓ 0.0s

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
# Encode the target variable in the original dataset
le = LabelEncoder()
data["diagnosis"] = le.fit_transform(data["diagnosis"])  # Converts 'B' -> 0, 'M' -> 1

# Split data
train, test = train_test_split(data, test_size=0.2, random_state=41)

# Prepare test data
X_test = test.iloc[:, :-1].values
Y_test = test.iloc[:, -1].values  # These are already encoded as integers

# Predictions (Gaussian or Categorical)
Y_pred = naive_bayes_gaussian(train, X=X_test, Y="diagnosis")  # For Gaussian
# Y_pred = naive_bayes_categorical(train, X=X_test, Y="diagnosis")  # For Categorical

# Evaluate metrics
from sklearn.metrics import confusion_matrix, f1_score
print("Confusion Matrix:")
print(confusion_matrix(Y_test, Y_pred))
print("F1 Score:", f1_score(Y_test, Y_pred))
```

[10] ✓ 0.1s

```
Confusion Matrix:
[[74  0]
 [ 4 36]]
F1 Score: 0.9473684210526315
```

```
data["cat_mean_radius"] = pd.cut(data["mean_radius"].values, bins = 3, labels = [0,1,2])
data["cat_mean_texture"] = pd.cut(data["mean_texture"].values, bins = 3, labels = [0,1,2])
data["cat_mean_smoothness"] = pd.cut(data["mean_smoothness"].values, bins = 3, labels = [0,1,2])

data = data.drop(columns=["mean_radius", "mean_texture", "mean_smoothness"])
data = data[["cat_mean_radius", "cat_mean_texture", "cat_mean_smoothness", "diagnosis"]]
data.head(10)
```

[11] ✓ 0.0s

| | cat_mean_radius | cat_mean_texture | cat_mean_smoothness | diagnosis |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 2 | 1 | 1 | 1 | 1 |
| 3 | 0 | 1 | 2 | 1 |
| 4 | 1 | 0 | 1 | 1 |
| 5 | 0 | 0 | 2 | 1 |
| 6 | 1 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 |

```
def calculate_likelihood_categorical(df, feat_name, feat_val, Y, label):
    df = df[df[Y] == label]
    if len(df) == 0:
        return 1e-6
    p_x_given_y = len(df[df[feat_name] == feat_val]) / len(df)
    return p_x_given_y
```

[12] ✓ 0.0s

```python
def naive_bayes_categorical(df, X, Y):
    features = list(df.columns)[:-1]
    prior = calculate_prior(df, Y)
    Y_pred = []

    for x in X:
        labels = sorted(df[Y].unique())
        likelihood = [1] * len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calculate_likelihood_categorical(df, features[i], x[i], Y, labels[j])
        post_prob = [likelihood[j] * prior[j] for j in range(len(labels))]
        Y_pred.append(np.argmax(post_prob))
    return np.array(Y_pred)
```

[13] ✓ 0.0s

```python
from sklearn.model_selection import train_test_split
train_test = train_test_split(data, test_size=.2, random_state=41)

X_test = test.iloc[:,:-1].values
Y_test = test.iloc[:,-1].values
Y_pred = naive_bayes_categorical(train, X=X_test, Y="diagnosis")

from sklearn.metrics import confusion_matrix, f1_score
print(confusion_matrix(Y_test, Y_pred))
print(f1_score(Y_test, Y_pred))
```

[14] ✓ 0.1s

```
[[74  0]
 [40  0]]
0.0
```

# 7. Implementing PCA on Iris Dataset.

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.datasets import load_iris

# Load the iris dataset
iris = load_iris()

# Take Independent features as x and dependent feature as y
x = pd.DataFrame(data=iris.data, columns=iris.feature_names)
y = iris.target
x
```

[1] ✓ 0.6s

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) |
|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 |
| ... | ... | ... | ... | ... |
| 145 | 6.7 | 3.0 | 5.2 | 2.3 |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 |

150 rows × 4 columns

```
# Use StandardScaler to scale the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
x_scaled
```

[2] ✓ 0.0s

```
# Use StandardScaler to scale the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
x_scaled
```

[2] ✓ 0.0s

```
array([[-9.00681170e-01,  1.01900435e+00, -1.34022653e+00,
        -1.31544430e+00],
       [-1.14301691e+00, -1.31979479e-01, -1.34022653e+00,
        -1.31544430e+00],
       [-1.38535265e+00,  3.28414053e-01, -1.39706395e+00,
        -1.31544430e+00],
       [-1.50652052e+00,  9.82172869e-02, -1.28338910e+00,
        -1.31544430e+00],
       [-1.02184904e+00,  1.24920112e+00, -1.34022653e+00,
        -1.31544430e+00],
       [-5.37177559e-01,  1.93979142e+00, -1.16971425e+00,
        -1.05217993e+00],
       [-1.50652052e+00,  7.88807586e-01, -1.34022653e+00,
        -1.18381211e+00],
       [-1.02184904e+00,  7.88807586e-01, -1.28338910e+00,
        -1.31544430e+00],
```

```
# Display coverience matrix
# The columns in the x_scaled will become the rows in the coverience matrix
rows = x_scaled.T
coverience_matrix = np.cov(rows)
print(coverience_matrix)
```

[3] ✓ 0.0s

```
[[ 1.00671141 -0.11835884  0.87760447  0.82343066]
 [-0.11835884  1.00671141 -0.43131554 -0.36858315]
 [ 0.87760447 -0.43131554  1.00671141  0.96932762]
 [ 0.82343066 -0.36858315  0.96932762  1.00671141]]
```

```
# Eigen values and Eigen vectors
# Eigen vectors (principal components) determine the directions and
# Eigen values determine the magnitude
eigen_values,eigen_vectors = np.linalg.eig(coverience_matrix)
print("Eigen Vectors :", eigen_vectors)
print("Eigen Values :", eigen_values)

# Let us see which eigen value has maximum varience
for i in range(len(eigen_values)):
    print(eigen_values[i]/sum(eigen_values))

pc1 = x_scaled.dot(eigen_vectors.T[0])
pc2 = x_scaled.dot(eigen_vectors.T[1])

# Now compose the data frame for our analysis
# Take pc1 on x-axis and pc2 on y-axis
df = pd.DataFrame(data = pc1, columns = ['PC1'])
df['PC2'] = pc2
df['target'] = y
df
```

[4] ✓ 0.0s

```
Eigen Vectors : [[ 0.52106591 -0.37741762 -0.71956635  0.26128628]
 [-0.26934744 -0.92329566  0.24438178 -0.12350962]
 [ 0.5804131  -0.02449161  0.14212637 -0.80144925]
 [ 0.56485654 -0.06694199  0.63427274  0.52359713]]
Eigen Values : [2.93808505 0.9201649  0.14774182 0.02085386]
0.7296244541329987
0.2285076178670176
0.03668921889282877
0.0051787091071548
```

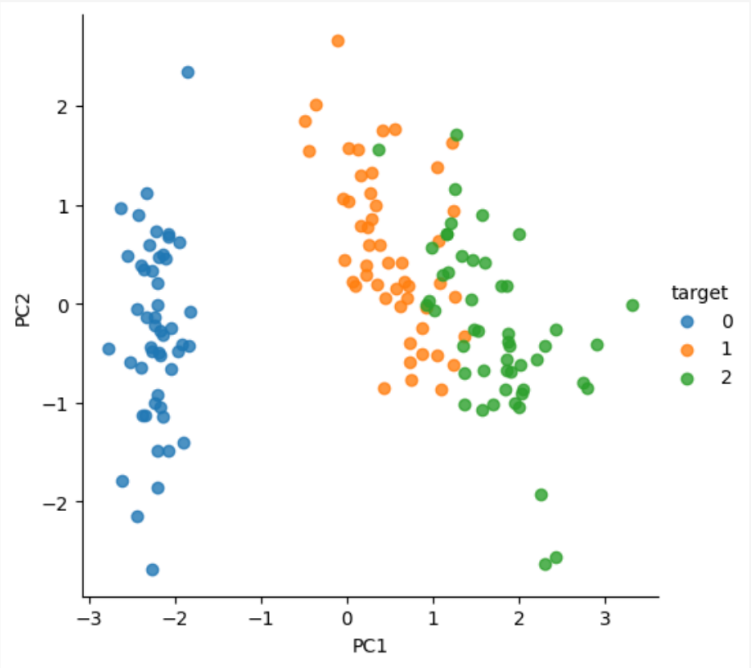
| | PC1 | PC2 | target |
|---|---|---|---|
| 0 | -2.264703 | -0.480027 | 0 |
| 1 | -2.080961 | 0.674134 | 0 |
| 2 | -2.364229 | 0.341908 | 0 |
| 3 | -2.299384 | 0.597395 | 0 |
| 4 | -2.389842 | -0.646835 | 0 |
| ... | ... | ... | ... |
| 145 | 1.870503 | -0.386966 | 2 |
| 146 | 1.564580 | 0.896687 | 2 |
| 147 | 1.521170 | -0.269069 | 2 |
| 148 | 1.372788 | -1.011254 | 2 |
| 149 | 0.960656 | 0.024332 | 2 |

150 rows × 3 columns

```
# Plot the data
sns.lmplot(data = df, x = 'PC1', y = 'PC2', hue = 'target', fit_reg = False)
```

[5] ✓ 0.1s

```
<seaborn.axisgrid.FacetGrid at 0x107dff770>
```




```
# Method 2 : using PCA class in sklearn
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
projected = pca.fit_transform(x_scaled)

# From projected data, take 0th column as pc1 and 1st column as pc2
pc1 = projected[:,0]
pc2 = projected[:,1]

# create a data frame
df1 = pd.DataFrame(pc1, columns = ['PC1'])
df1['PC2'] = pc2
df1['target'] = y
df1
```

[6]

| | PC1 | PC2 | target |
|---|---|---|---|
| 0 | -2.264703 | 0.480027 | 0 |
| 1 | -2.080961 | -0.674134 | 0 |
| 2 | -2.364229 | -0.341908 | 0 |
| 3 | -2.299384 | -0.597395 | 0 |
| 4 | -2.389842 | 0.646835 | 0 |
| ... | ... | ... | ... |
| 145 | 1.870503 | 0.386966 | 2 |
| 146 | 1.564580 | -0.896687 | 2 |
| 147 | 1.521170 | 0.269069 | 2 |
| 148 | 1.372788 | 1.011254 | 2 |
| 149 | 0.960656 | -0.024332 | 2 |

150 rows × 3 columns

```python
# remove the target column and take remaining dataframe
df1 = df1.drop('target', axis=1)

# divide the data into two parts
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(df1, y, test_size = 0.2)

# Train the model using svm
from sklearn.svm import SVC
model = SVC()
model.fit(x_train, y_train)

# Find the accuracy level
model.score(x_test, y_test)
```

[8] ✓ 0.0s

```
1.0
```

```python
# Enter the new data for prediction
input_data = pd.DataFrame([[6.6, 3.0, 4.4, 1.4]], columns=iris.feature_names)

# convert the inputs to scaled data
scaled = scaler.transform(input_data)
print(scaled)

#apply pca and transform the data
pca_data = pca.transform(scaled)

# predict the data
model.predict(pca_data)
```

[9] ✓ 0.0s

```
[[ 0.91683689 -0.13197948  0.36489628  0.26414192]]
/opt/homebrew/lib/python3.13/site-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names,
  warnings.warn(

array([1])
```

# 8. Implementing Gradient Descent Algorithm

```python
import numpy as np

# Generate synthetic data
np.random.seed(42)
X = np.random.rand(100, 1)  # 100 samples, 1 feature
y = 4 + 3 * X + np.random.randn(100, 1) * 0.2  # Linear relation with noise

# Normalize input data
X = (X - np.mean(X)) / np.std(X)

# Neural Network Parameters
input_dim = 1  # Single feature
output_dim = 1  # Single target
learning_rate = 0.01
n_iterations = 1000

# Initialize weights and bias
W = np.random.randn(input_dim, output_dim)
b = np.zeros((1, output_dim))
```

[1] ✓ 0.0s

```python
# Define functions for forward and backward pass
def forward(X, W, b):
    """Forward pass: Compute predictions."""
    return np.dot(X, W) + b

def compute_loss(y, y_pred):
    """Mean Squared Error Loss."""
    return np.mean((y - y_pred) ** 2)

def backward(X, y, y_pred):
    """Backward pass: Compute gradients."""
    m = len(y)
    dW = -(2 / m) * np.dot(X.T, (y - y_pred))
    db = -(2 / m) * np.sum(y - y_pred, axis=0, keepdims=True)
    return dW, db
```

[2] ✓ 0.0s

```python
# Training loop
loss_history = []
for iteration in range(n_iterations):
    # Forward pass
    y_pred = forward(X, W, b)

    # Compute loss
    loss = compute_loss(y, y_pred)
    loss_history.append(loss)

    # Backward pass
    dW, db = backward(X, y, y_pred)

    # Update parameters
    W -= learning_rate * dW
    b -= learning_rate * db

    # Print progress every 100 iterations
    if iteration % 100 == 0:
        print(f"Iteration {iteration}, Loss: {loss:.4f}")
```

[3] ✓ 0.0s

```
Iteration 0, Loss: 30.0226
Iteration 100, Loss: 0.5597
Iteration 200, Loss: 0.0415
Iteration 300, Loss: 0.0324
Iteration 400, Loss: 0.0323
Iteration 500, Loss: 0.0323
Iteration 600, Loss: 0.0323
Iteration 700, Loss: 0.0323
Iteration 800, Loss: 0.0323
Iteration 900, Loss: 0.0323
```

```python
# Output final weights and bias
print("Trained Weights (W):", W)
print("Trained Bias (b):", b)

# Plot the loss over iterations
import matplotlib.pyplot as plt
plt.plot(range(n_iterations), loss_history, label="Loss")
plt.title("Loss Curve")
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.legend()
plt.show()

# Visualize predictions
plt.scatter(X, y, label="True Data")
plt.plot(X, forward(X, W, b), color='red', label="Predicted Line")
plt.title("Linear Regression Fit with Neural Network")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```

[4] ✓ 0.2s

```
Trained Weights (W): [[0.86077627]]
Trained Bias (b): [[5.41032615]]
```