CANARA
ENGINEERING COLLEGE
SUDHINDRA NAGARA, BENJANAPADAVU, BANTWAL TQ, D.K.- 574219
Department Of Computer Science and Design

Machine Learning

Sub Code: **BCM601**

Academic Year:2024-25

Course Outcomes:

CO1	Demonstrate the need for machine learning, its relationship to other fields, and different types of machine learning
CO2	Illustrate the fundamental principles of multivariate data and apply dimensionality reduction techniques.
CO3	Apply similarity-based learning methods and perform linear, polynomial regression analysis
CO4	Apply decision trees for classification and regression problems, and Bayesian models for probabilistic learning
CO5	Analyze the clustering algorithms and reinforce their understanding by applying Q-learning for decision making tasks

Assessment Details (both CIE and SEE):
The weightage of Continuous Internal Evaluation (CIE) is **50%** and for Semester End Exam (SEE) is **50%**. The minimum passing mark for the CIE is **40% of the maximum marks (20 marks out of 50)** and for the SEE minimum passing mark is **35%** of the maximum marks **(18 out of 50 marks).** A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of **40% (40 marks out of 100)** in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

CIE for the theory component of the IPCC (maximum marks 50)
● IPCC means practical portion integrated with the theory of the course.
● CIE marks for the theory component are **25 marks** and that for the practical component is **25 marks**.
● 25 marks for the theory component are split into **15 marks** for two Internal Assessment Tests **(Two Tests, each of 15 Marks with 01-hour duration, are to be conducted)** and **10 marks for other assessment methods** mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second test after covering 85-90% of the syllabus.
● Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theory component of IPCC (that is for **25 marks**).
● The student has to secure **40% of 25 marks to qualify** in the CIE of the theory component of IPCC.

CIE for the practical component of the IPCC
● **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
● On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.
● The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for **10 marks**. Marks of all experiments' write-ups are added and scaled down to **15 marks.**
● The laboratory test **(duration 02/03 hours)** after completion of all the experiments shall be conducted for **50 marks** and scaled down to **10 marks**.
● Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for **25 marks**.
● The student has to secure **40% of 25 marks to qualify** in the CIE of the practical component of the IPCC.

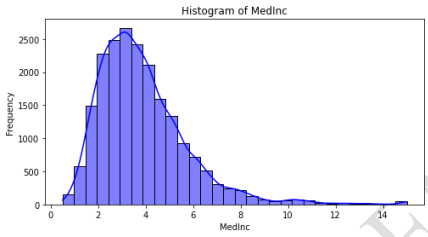
SEE for IPCC
Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the course (duration 03 hours)
1. The question paper will have ten questions. Each question is set for 20 marks.
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), should have a mix of topics under that module.
3. The students have to answer 5 full questions, selecting one full question from each module.
4. Marks scored by the student shall be proportionally scaled down to **50 Marks**

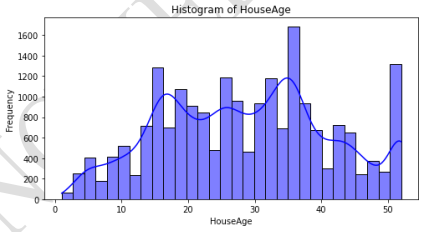
The theory portion of the IPCC shall be for both CIE and SEE, whereas the practical portion will have a CIE component only. Questions mentioned in the SEE paper may include questions from the practical component.

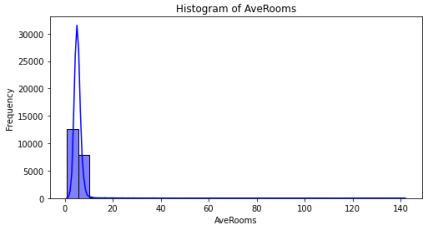
Suggested Learning Resources:
Textbooks:
1. S Sridhar and M Vijayalakshmi, "Machine Learning", Oxford University Press, 2021.
2. M N Murty and Ananthanarayana V S, "Machine Learning: Theory and Practice", Universities Press (India) Pvt. Limited, 2024.
Reference Books:
1. Tom M. Mitchell, "Machine Learning", McGraw-Hill Education, 2013.
2. Miroslav Kubat, "An Introduction to Machine Learning", Springer, 2017.

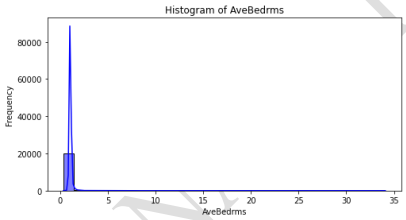
Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset

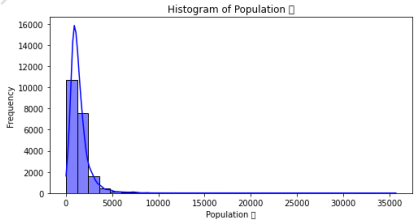
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

# Load the California Housing dataset
#data = fetch_california_housing(as_frame=True) #Use this if dataset is not available locally
#df = data.frame
df=pd.read_csv('dataset/California Housing.csv')
# Create histograms for all numerical features
for column in df.columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[column], kde=True, bins=30, color='blue')
    plt.title(f"Histogram of {column}")
    plt.xlabel(column)
    plt.ylabel("Frequency")
    plt.show()

# Create box plots for all numerical features
for column in df.columns:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[column], color='orange')
    plt.title(f"Box Plot of {column}")
    plt.xlabel(column)
    plt.show()
```

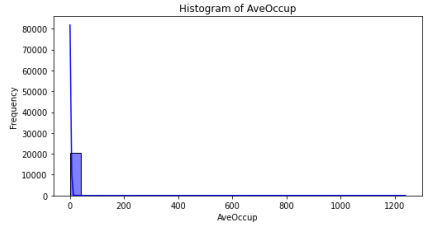


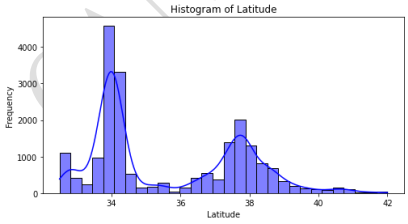


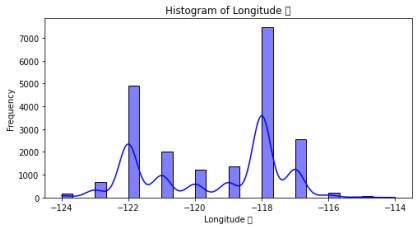


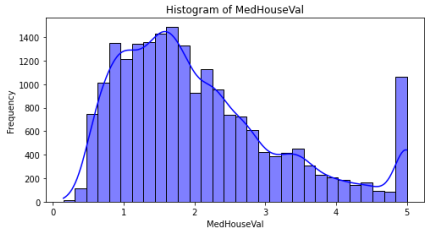


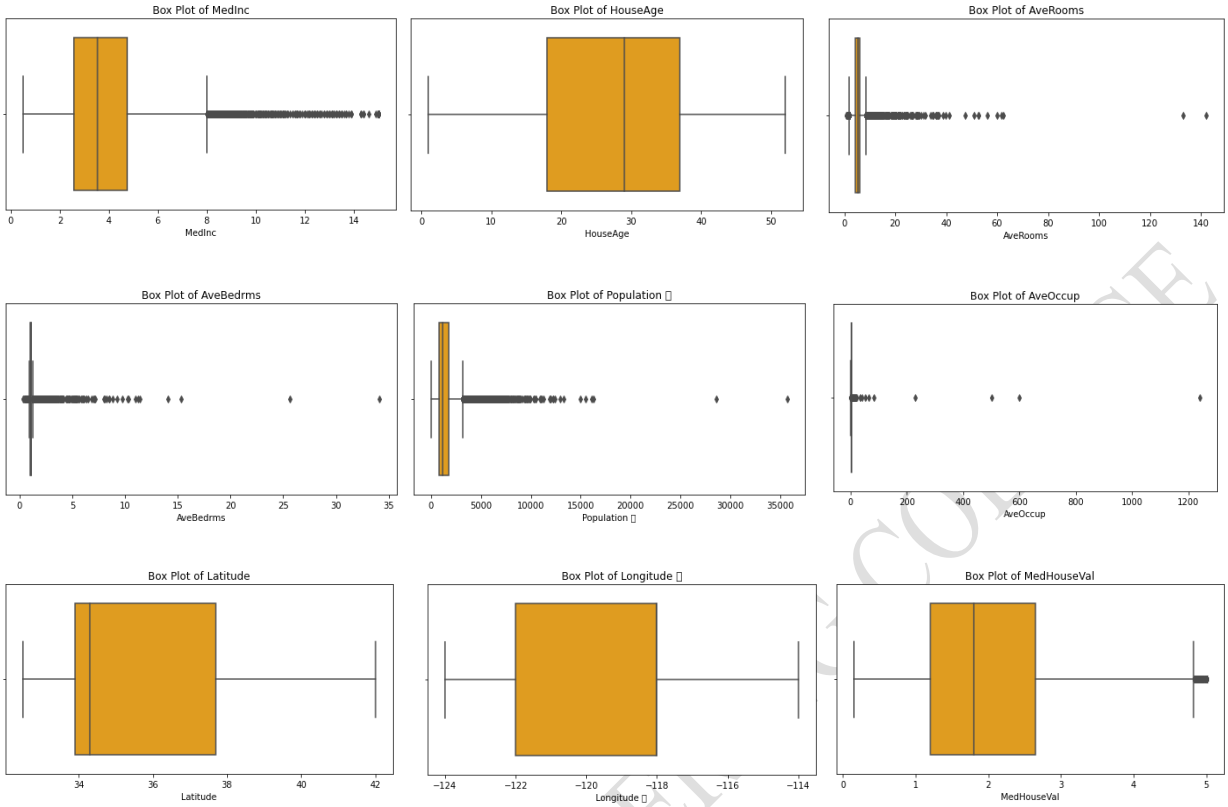








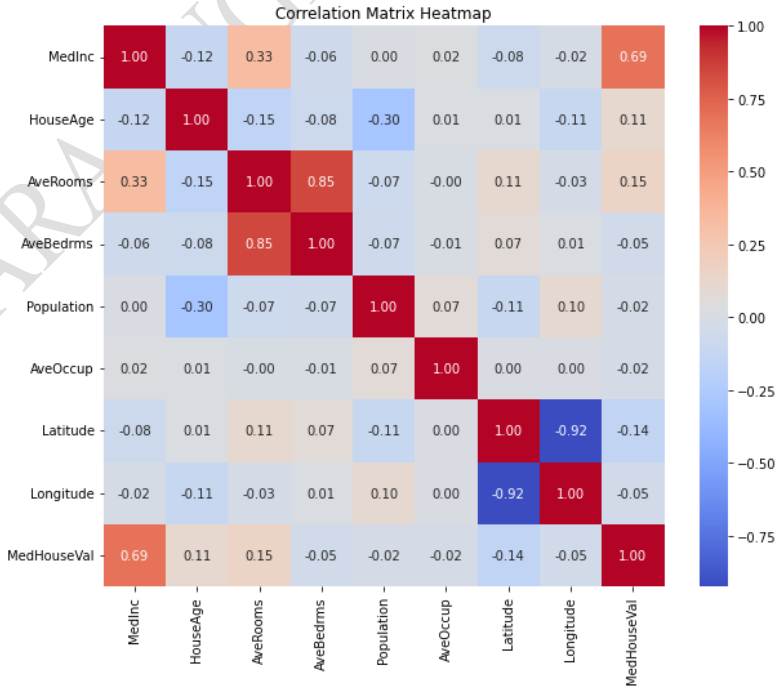




Develop a program to Compute the correlation matrix to understand the relationships between pairs of features. Visualize the correlation matrix using a heatmap to know which variables have strong positive/negative correlations. Create a pair plot to visualize pairwise relationships between features. Use California Housing dataset.

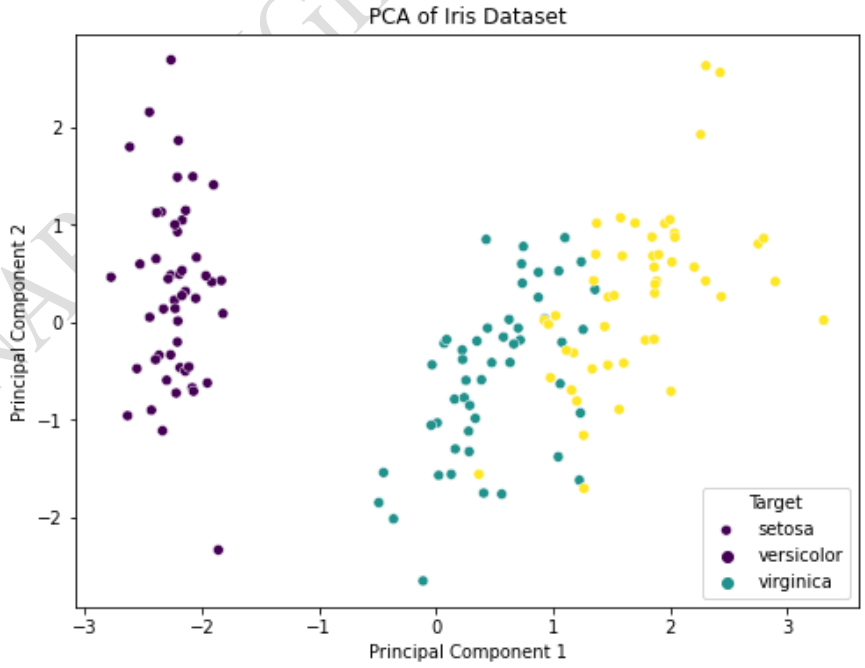
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

# Load the California Housing dataset
data = fetch_california_housing(as_frame=True)
df = data.frame

# Compute the correlation matrix
correlation_matrix = df.corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True, square=True)
plt.title("Correlation Matrix Heatmap")
plt.show()
```

```
Correlation Matrix:
              MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  Longitude  MedHouseVal
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
MedHouseVal 0.688075  0.105623  0.151948  -0.046701   -0.024650 -0.023737 -0.144160  -0.045967     1.000000
```



Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# Load the Iris dataset
iris = load_iris()
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_df_target=pd.DataFrame(iris.target)
# Standardize the data
scaler = StandardScaler()
iris_scaled = scaler.fit_transform(iris_df)

# Perform PCA to reduce dimensions from 4 to 2
pca = PCA(n_components=2)
iris_pca = pca.fit_transform(iris_scaled)

# Create a DataFrame for the PCA result
pca_df = pd.DataFrame(iris_pca, columns=['PC1', 'PC2'])
pca_df['Target'] = iris.target

# Plot the PCA result
plt.figure(figsize=(8, 6))
sns.scatterplot(x='PC1', y='PC2', hue='Target', palette='viridis', data=pca_df, legend="full")
plt.title("PCA of Iris Dataset")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title="Target", labels=iris.target_names)
plt.show()
```



For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
import pandas as pd
data = pd.read_csv('EnjoySport.csv')#ensure csv file exists in the same folder as the python code.
columnLength= data.shape[1]              #obtain number of columns
print (data)
h = ['0']*(columnLength-1)
hp=[]                          #initialize list hp i.e list of hypotheses for positive training examples
hn=[]

for trainingExample in data.values: #this loop is used to build the hypothesis list for every row.
   if trainingExample[-1]!='no':   #if the trainingExample is positive, then it is appended to hp else to hn
      hp.append(list(trainingExample[0:6]))
   else:
      hn.append(list(trainingExample[0:6]))

for i in range (len(hp)):          #update the hypothesis h from most specific to maximally specific
   for j in range(columnLength-1):  #if the hypothesis attribute value is 0, it is updated to the attributes in the first hypothesis
      if (h[j]=='0'):
         h[j]=hp[i][j]
      if (h[j]!=hp[i][j]):       #if the attribute value in the hypothesis is not same as the attribute value in the successive hypotheses
         h[j]='?'              #then it is updated to '?' indicating that any value is acceptable.
      else:                   #if the attribute in the hypothesis is the same as the attribute value in the successive hypotheses
         h[j]=hp[i][j]

print('\nThe positive Hypotheses are')
print(hp)
print('\nThe negative Hypotheses are')
print(hn)
print('\nThe Maximally Specific Hypothesis h is')
print(h)
```

```
The positive Hypotheses are
[['sunny', 'warm', 'normal', 'strong', 'warm', 'same'],
 ['sunny', 'warm', 'high', 'strong', 'warm', 'same'],
 ['sunny', 'warm','high', 'strong', 'cool', 'change']]

The negative Hypotheses are
[['rainy', 'cold', 'high', 'strong', 'warm', 'change']]

The Maximally Specific Hypothesis h is
['sunny', 'warm', '?', 'strong', '?', '?']
```

Develop a program to implement k-Nearest Neighbour algorithm to classify the randomly generated 100 values of x in the range of [0,1]. Perform the following based on dataset generated.

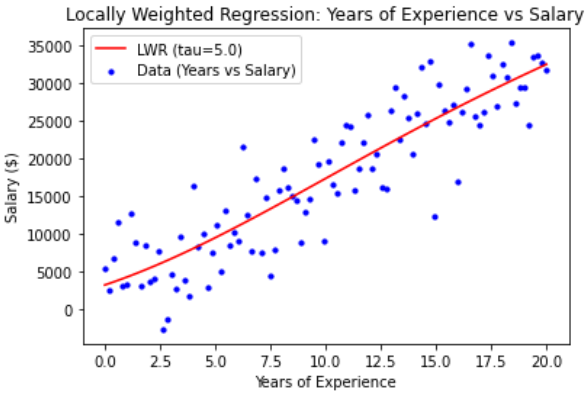
a. **Label the first 50 points $\{x_1,\ldots\ldots,x_{50}\}$ as follows: if $(x_i \leq 0.5)$, then $x_i \in$ Class1, else $x_i \notin$ Class1**
b. **Classify the remaining points, $x_{51},\ldots\ldots,x_{100}$ using KNN. Perform this for k=1,2,3,4,5,20,30**

```
import random
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import pandas as pd
# Generate 100 random values of x in the range [0, 1]
random_values = [random.uniform(0, 1) for _ in range(100)]
x_train=[]
x_test=[]
for i in range(0,len(random_values)):
    if(i<50):
        x_train.append(random_values[i])
    else:
        x_test.append(random_values[i])
y_train=[]
for item in x_train:
    if(item<=0.5):
        y_train.append(0)
    else:
        y_train.append(1)
y_test=[]
for item in x_test:
    if(item<=0.5):
        y_test.append(0)
    else:
        y_test.append(1)

train = pd.DataFrame(x_train, columns=['values'])
train.insert(1, "labels", y_train, True)
test = pd.DataFrame(x_test, columns=['values'])
test.insert(1, "labels", y_test, True)
for k in [1,2,3,4,5,20,30]:
    model=KNeighborsClassifier(n_neighbors=k,p=2,metric='minkowski')
    model.fit(np.array(x_train).reshape(-1,1),y_train)
    y_pred=model.predict(np.array(x_test).reshape(-1,1))
    test.insert(2, "predictions"+"k="+str(k), y_pred, True)
    from sklearn.metrics import confusion_matrix, accuracy_score
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    acc=accuracy_score(y_test, y_pred)
    print(acc)
```

```
[[28  0]
 [ 1 21]]
0.98
[[28  0]
 [ 1 21]]
0.98
[[28  0]
 [ 0 22]]
1.0
[[28  0]
 [ 1 21]]
0.98
[[28  0]
 [ 1 21]]
0.98
[[28  0]
 [ 1 21]]
0.98
[[28  0]
 [ 1 21]]
0.98
```

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

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

def gaussian_kernel(x, x_i, tau):
    return np.exp(-(x - x_i) ** 2 / (2 * tau ** 2))

def locally_weighted_regression(x_query, X, y, tau):
    weights = np.array([gaussian_kernel(x_query, x_i, tau) for x_i in X])

    # Compute weighted sums for slope and intercept
    weighted_sum_x = np.sum(weights * X)
    weighted_sum_y = np.sum(weights * y)
    weighted_sum_xy = np.sum(weights * X * y)
    weighted_sum_xx = np.sum(weights * X * X)
    total_weight = np.sum(weights)

    # Compute slope and intercept
    slope = (weighted_sum_xy - (weighted_sum_x * weighted_sum_y / total_weight)) / \
            (weighted_sum_xx - (weighted_sum_x ** 2 / total_weight))
    intercept = (weighted_sum_y / total_weight) - slope * (weighted_sum_x / total_weight)

    # Predict the value for the query point
    return slope * x_query + intercept

def predict(X_query, X, y, tau):
    predictions = []
    for x_query in X_query:
        predictions.append(locally_weighted_regression(x_query, X, y, tau))
    return np.array(predictions)

# Generate some sample data (Years of Experience vs Salary)
np.random.seed(42)
X = np.linspace(0, 20, 100)  # Years of experience
y = 3000 + 1500 * X + np.random.normal(scale=5000, size=X.shape[0])  # Salary with noise

 # Define query points
X_query = np.linspace(0, 20, 100)

# Perform predictions
tau = 5.0  # Bandwidth parameter
y_pred = predict(X_query, X, y, tau)

# Plot the results
plt.scatter(X, y, label="Data (Years vs Salary)", color="blue", s=10)
plt.plot(X_query, y_pred, label=f"LWR (tau={tau})", color="red")
plt.xlabel("Years of Experience")
plt.ylabel("Salary ($)")
plt.title("Locally Weighted Regression: Years of Experience vs Salary")
plt.legend()
plt.show()
```



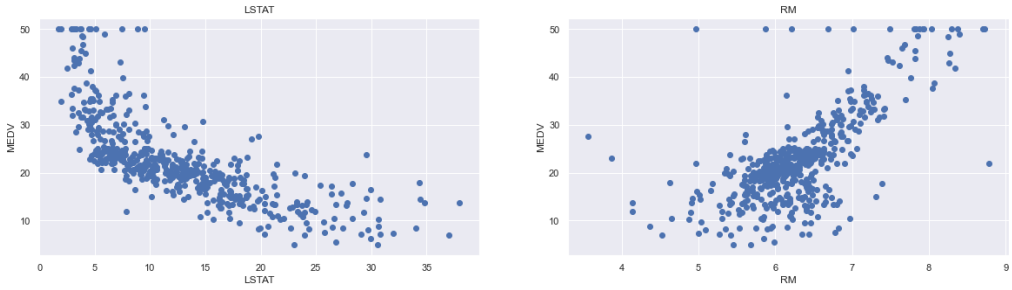
Develop a program to demonstrate the working of Linear Regression and Polynomial Regression. Use Boston Housing Dataset for Linear Regression and Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression.

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
# Linear Regression using Boston Housing Dataset
# Load dataset
df=pd.read_csv("Boston_Housing.csv")
df.isnull().sum()

sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.distplot(df['MEDV'], bins=30)
plt.show()

correlation_matrix = df.corr().round(2)
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)

X = pd.DataFrame(np.c_[df['LSTAT'], df['RM']], columns = ['LSTAT','RM'])
Y = df['MEDV']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)
model = LinearRegression()
model.fit(X_train, Y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
# model evaluation for training set
y_train_predict = model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# model evaluation for testing set
y_test_predict = model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))

coef_df=pd.DataFrame(model.coef_,X_test.columns,columns=['Coefficients'])
print(coef_df)
```

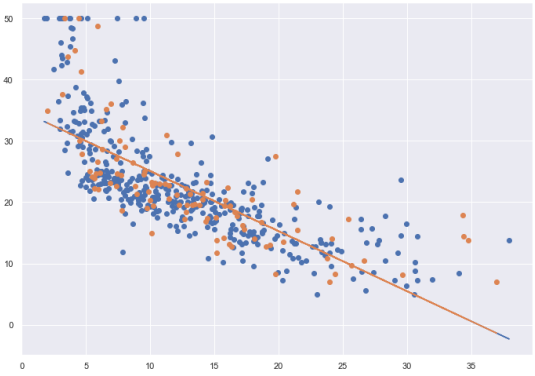
```
The model performance for training set
--------------------------------------
RMSE is 5.6371293350711955
R2 score is 0.6300745149331701


The model performance for testing set
--------------------------------------
RMSE is 5.137400784702911
R2 score is 0.6628996975186952
```

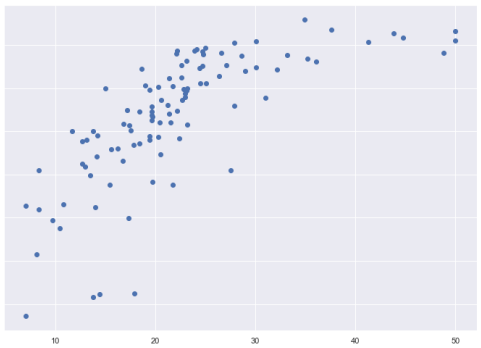
	Coefficients
LSTAT	-0.717230
RM	4.589388

```
#Extra: Simple Linear Regression using only one feature
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']
target = df['MEDV']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = df[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```




```
X = pd.DataFrame(np.c_[df['LSTAT']], columns = ['LSTAT',])
Y = df['MEDV']
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=5)
model = LinearRegression()
model.fit(X_train, Y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

a=model.coef_[0]
b=model.intercept_
X=X_train["LSTAT"].tolist()
Y=[]
for i in range(len(X)):
    Y.append((a*X[i])+b)
plt.scatter(X_train,Y_train)
plt.plot(X,Y)

X=X_test["LSTAT"].tolist()
Y=[]
for i in range(len(X)):
    Y.append((a*X[i])+b)
plt.scatter(X_test,Y_test)
plt.plot(X,Y)


plt.scatter(Y_test,y_test_pred)


rmse = (np.sqrt(mean_squared_error(Y_train, y_train_pred)))
r2 = r2_score(Y_train, y_train_pred)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# model evaluation for testing set
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_pred)))
r2 = r2_score(Y_test, y_test_pred)
print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
coef_df=pd.DataFrame(model.coef_,X_test.columns,columns=['Coefficients'])
print(coef_df)
```





```
The model performance for training set
--------------------------------------
RMSE is 6.201452973865344
R2 score is 0.5523019908037391


The model performance for testing set
--------------------------------------
RMSE is 6.2307165730986815
R2 score is 0.5041523728903132
```

	Coefficients
LSTAT	-0.979812

```
#Extra: Simple Linear Regression using only one feature
X = pd.DataFrame(np.c_[df['RM']], columns = ['RM',])
Y = df['MEDV']
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=5)
model = LinearRegression()
model.fit(X_train, Y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

a=model.coef_[0]
b=model.intercept_
X=X_train["RM"].tolist()
Y=[]
for i in range(len(X)):
    Y.append((a*X[i])+b)
plt.scatter(X_train,Y_train)
plt.plot(X,Y)

X=X_test["RM"].tolist()
Y=[]
for i in range(len(X)):
    Y.append((a*X[i])+b)
plt.scatter(X_test,Y_test)
plt.plot(X,Y)

plt.scatter(Y_test,y_test_pred)

rmse = (np.sqrt(mean_squared_error(Y_train, y_train_pred)))
r2 = r2_score(Y_train, y_train_pred)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# model evaluation for testing set
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_pred)))
r2 = r2_score(Y_test, y_test_pred)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))

coef_df=pd.DataFrame(model.coef_,X_test.columns,columns=['Coefficients'])
print(coef_df)
```

```
The model performance for training set
--------------------------------------
RMSE is 6.972277149440585
R2 score is 0.4340897790637215


The model performance for testing set
--------------------------------------
RMSE is 4.895963186952216
R2 score is 0.6938399401553497
```

	Coefficients
RM	8.823456

#Polynomial Regression

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
# Linear Regression using Boston Housing Dataset
# Load dataset
df=pd.read_csv("autompg.csv")
df.isnull().sum()

df = df.drop('car name', axis=1)
# Also replacing the categorical var with actual values
df['origin'] = df['origin'].replace({1: 'america', 2: 'europe', 3: 'asia'})
df.head()

df = pd.get_dummies(df, columns=['origin'])
df.head()

df.describe()

# Missing values have a'?"
# Replace missing values with NaN
df = df.replace('?', np.nan)

median_fill = lambda x: x.fillna(x.median())
df = df.apply(median_fill,axis=0)
# converting the hp column from object / string type to float
df['horsepower'] = df['horsepower'].astype('float64')

df_plot = df.iloc[:, 0:7]
sns.pairplot(df_plot, diag_kind='kde')

from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model

X = df.drop(['mpg','origin_europe'], axis=1)
# the dependent variable
y = df[['mpg']]
# Split X and y into training and test set in 70:30 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1)
poly = PolynomialFeatures(degree=2, interaction_only=True)
X_train2 = poly.fit_transform(X_train)
X_test2 = poly.fit_transform(X_test)
poly_regr = linear_model.LinearRegression()
poly_regr.fit(X_train2, y_train)
y_pred = poly_regr.predict(X_test2)
#print(y_pred)
#In sample (training) R^2 will always improve with the number of variables!
print(poly_regr.score(X_train2, y_train))
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```

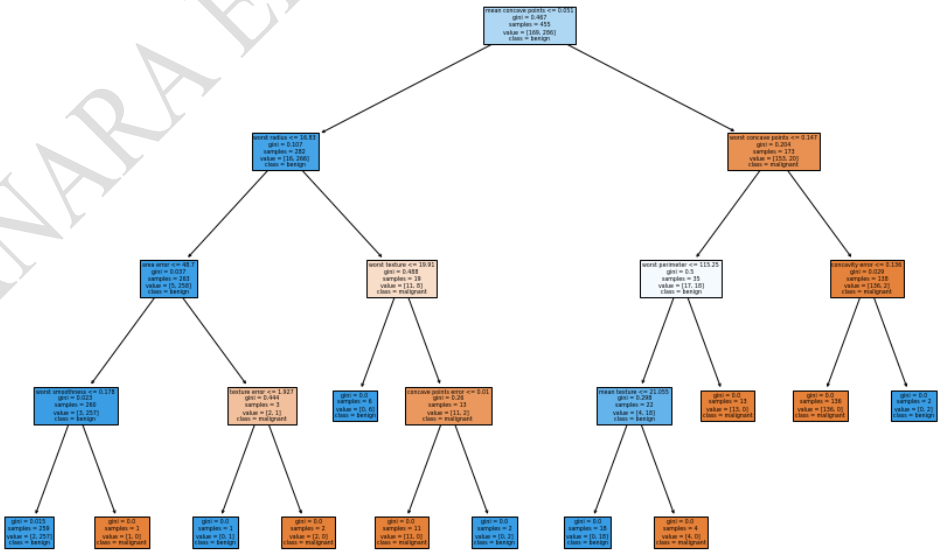
```
0.9015975294369648
Mean Squared Error (MSE): 0.47
R-squared (R2): 0.84
```

Develop a program to demonstrate the working of the decision tree algorithm. Use Breast Cancer Data set for building the decision tree and apply this knowledge to classify a new sample.

```
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
from sklearn import tree

# Load the Breast Cancer dataset
data = load_breast_cancer()
X, y = data.data, data.target

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

# Create and train the Decision Tree model
clf = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Visualizing the Decision Tree
plt.figure(figsize=(15, 10))
tree.plot_tree(clf, filled=True, feature_names=data.feature_names, class_names=data.target_names)
plt.show()

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

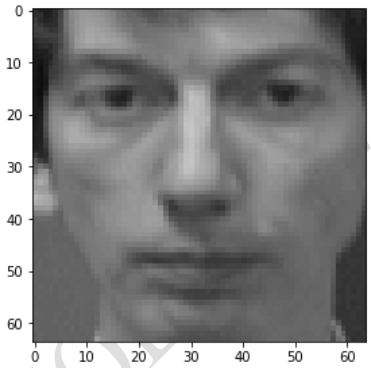
```
Accuracy: 0.9474
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        43
           1       0.96      0.96      0.96        71

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```

```
[[40  3]
 [ 3 68]]
```



Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data sets.

```
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import numpy as np

# Load the Olivetti Faces dataset
dataset = fetch_olivetti_faces(shuffle=True, random_state=42)
#39 Faces with 10 samples per face

#Display Data Sample
import matplotlib.pyplot as plt
from skimage.io import imshow
imshow(dataset.data[1].reshape(64,64))
```




```
X, y = dataset.data, dataset.target

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

# Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Naive Bayes Classifier Accuracy: {accuracy:.4f}')
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Naive Bayes Classifier Accuracy: 0.7750
[[2 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 ...
 [0 0 0 ... 2 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 4]]
```

Develop a program to implement k-means clustering using Wisconsin Breast Cancer data set and visualize the clustering result.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer

# Load the dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)

# Standardizing the data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans.fit(df_scaled)
labels = kmeans.labels_

# Reduce dimensions using PCA for visualization
pca = PCA(n_components=2)
pca_result = pca.fit_transform(df_scaled)
df_pca = pd.DataFrame(pca_result, columns=['PCA1', 'PCA2'])
df_pca['Cluster'] = labels

# Plot the clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df_pca['PCA1'], y=df_pca['PCA2'], hue=df_pca['Cluster'], palette='coolwarm', alpha=0.7)
plt.title('K-Means Clustering on Breast Cancer Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.show()
```