**Md Zahid Khan**

**Machine Learning Practical**

**Machine Learning Practical’s**

1. Write a python program to Prepare Scatter Plot (Use Forge Dataset / Iris Dataset)

import matplotlib.pyplot as plt

import pandas as pd

# Load data

data = pd.read\_csv('IRIS.csv')

sepal\_length = data['sepal\_length']

petal\_length = data['petal\_length']

x = []

y = []

x = list(sepal\_length)

y = list(petal\_length)

# Plot

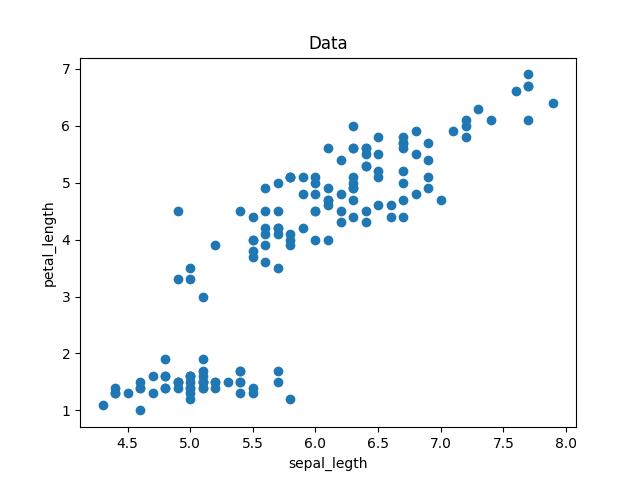
plt.scatter(x, y)

plt.xlabel('sepal\_legth')

plt.ylabel('petal\_length')

plt.title('Data')

plt.show()



2. Write a python program to find all null values in a given data set and remove them.

import pandas as pd

dataset = pd.read\_csv('titanic.csv')

dataset.shape

print("Info:")

dataset.info()

dataset.head()

dataset.isnull()

dataset.isnull().sum()

dataset.drop('Cabin', axis=1, inplace=True)

dataset.dropna(inplace=True)

1. Write a python program the Categorical values in numeric format for a given dataset.

#import pandas

import pandas as pd

# read csv file

df = pd.read\_csv("./data3.csv")

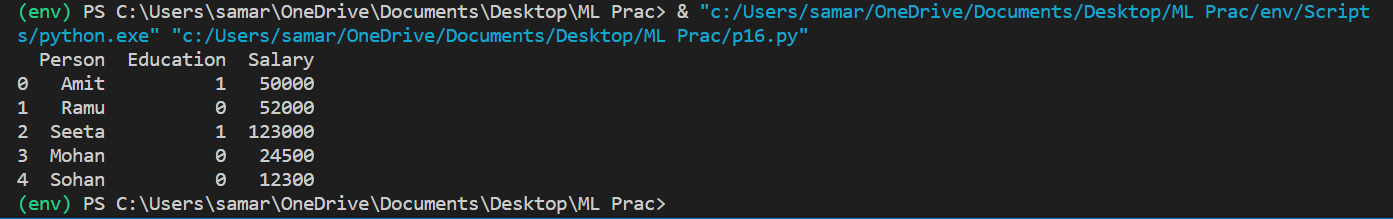
# replacing values

df['Education'].replace(['UG', 'PG'],

[0, 1], inplace=True)

print(df)

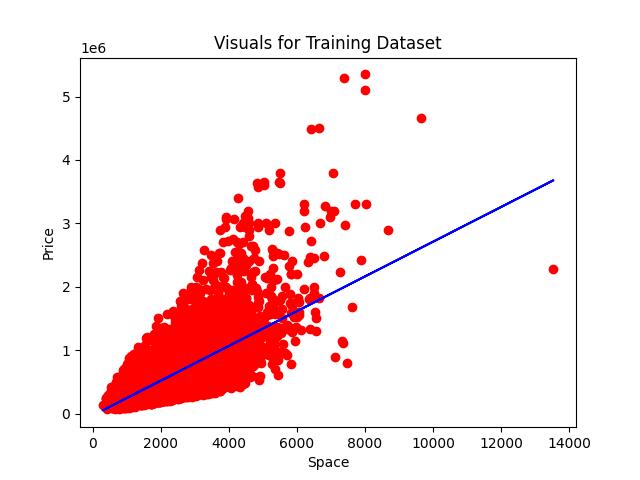
Output:



4. Write a python program to implement simple Linear Regression for predicting house

Price.

import numpy as np   
import pandas as pd   
import matplotlib.pyplot as plt   
import statsmodels.api as sm  
  
from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import train\_test\_split  
  
dataset =pd.read\_csv("C://Users//ADMIN/Desktop//Python//kc\_house\_data.csv")  
print("data Frame\n",dataset)  
space=dataset['sqft\_living']  
price=dataset['price']  
  
x = np.array(space).reshape(-1, 1)  
y = np.array(price)  
  
  
#Splitting the data into Train and Test  
xtrain, xtest, ytrain, ytest = train\_test\_split(x,y,test\_size=1/3, random\_state=0)  
#Fitting simple linear regression to the Training Set  
regressor = LinearRegression()  
regressor.fit(xtrain, ytrain)  
  
  
#Predicting the prices  
pred = regressor.predict(xtest)  
#Visualizing the training Test Results   
plt.scatter(xtrain, ytrain, color= 'red')  
plt.plot(xtrain, regressor.predict(xtrain), color = 'blue')  
plt.title ("Visuals for Training Dataset")  
plt.xlabel("Space")  
plt.ylabel("Price")  
plt.show()  
  
#Visualizing the Test Results   
plt.scatter(xtest, ytest, color= 'green')  
plt.plot(xtrain, regressor.predict(xtrain), color = 'blue')  
plt.title("Visuals for Test DataSet")  
plt.xlabel("Space")  
plt.ylabel("Price")  
plt.show()



5. Write a python program to implement multiple Linear Regression for a given dataset.

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

# Reading the dataset

dataset = pd.read\_csv("advertising.csv")

dataset.head()

# Setting the value for X and Y

x = dataset[['TV', 'Radio', 'Newspaper']]

y = dataset['Sales']

#Splitting the dataset

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=100)

# Fitting the Multiple Linear Regression model

mlr = LinearRegression()

mlr.fit(x\_train, y\_train)

#Intercept and Coefficient

# Printing the model coefficients

print(mlr.intercept\_)

# pair the feature names with the coefficients

list(zip(x, mlr.coef\_))

# Predicting the Test and Train set result

y\_pred\_mlr = mlr.predict(x\_test)

x\_pred\_mlr = mlr.predict(x\_train)

print("Prediction for test set: {}".format(y\_pred\_mlr))

# Actual value and the predicted value

mlr\_diff = pd.DataFrame(

{'Actual value': y\_test, 'Predicted value': y\_pred\_mlr})

mlr\_diff

# Predict for any value

mlr.predict([[56, 55, 67]])

# print the R-squared value for the model

print('R squared value of the model: {:.2f}'.format(mlr.score(x, y)\*100))

# 0 means the model is perfect. Therefore the value should be as close to 0 as possible

meanAbErr = metrics.mean\_absolute\_error(y\_test, y\_pred\_mlr)

meanSqErr = metrics.mean\_squared\_error(y\_test, y\_pred\_mlr)

rootMeanSqErr = np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred\_mlr))

print('Mean Absolute Error:', meanAbErr)

print('Mean Square Error:', meanSqErr)

print('Root Mean Square Error:', rootMeanSqErr)

6. Write a python program to implement Polynomial Regression for given dataset.

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

data\_set = pd.read\_csv('Position\_Salaries.csv')

x = data\_set.iloc[:, 1:2].values

y = data\_set.iloc[:, 2].values

data\_set.head()

lin\_regs = LinearRegression()

lin\_regs.fit(x, y)

LinearRegression(copy\_X = True, fit\_intercept =True, n\_jobs=None, normalize=False )

mtp.scatter(x,y,color="blue")

mtp.plot(x, lin\_regs.predict(x), color="red")

mtp.title("Salary estimation model using Linear Regression")

mtp.xlabel("Postion Levels")

mtp.ylabel("Salary")

mtp.show()

#Fitting the Polynomial regression of degree-2 to the dataset

poly\_regs= PolynomialFeatures(degree= 2)

x\_poly = poly\_regs.fit\_transform(x)

lin\_reg\_2 =LinearRegression()

lin\_reg\_2.fit(x\_poly, y)

#visulaizing the result for Polynomial Regression of degree-2 mtp.scatter(x,y,color="blue")

mtp.plot(x, lin\_reg\_2.predict(poly\_regs.fit\_transform(x)), color="red")

mtp.title("Salary estimation model Polynomial Regression of degree=2")

mtp.xlabel("Position Levels")

mtp.ylabel("Salary")

mtp.show()

#Fitting the Polynomial regression of degree-3 to the dataset

poly\_regs= PolynomialFeatures(degree= 2)

x\_poly = poly\_regs.fit\_transform(x)

lin\_reg\_3 =LinearRegression()

lin\_reg\_3.fit(x\_poly, y)

#visulaizing the result for Polynomial Regression of degree-3 mtp.scatter(x,y,color="blue")

mtp.plot(x, lin\_reg\_3.predict(poly\_regs.fit\_transform(x)), color="red")

mtp.title("Salary estimation model Polynomial Regression of degree=3")

mtp.xlabel("Position Levels")

mtp.ylabel("Salary")

mtp.show()

#Fitting the Polynomial regression of degree-4 to the dataset

poly\_regs=PolynomialFeatures(degree= 2)

x\_poly = poly\_regs.fit\_transform(x)

lin\_reg\_4 =LinearRegression()

lin\_reg\_4.fit(x\_poly, y)

#visulaizing the result for Polynomial Regression of degree-4 mtp.scatter(x,y,color="blue")

mtp.plot(x, lin\_reg\_4.predict(poly\_regs.fit\_transform(x)), color="red")

mtp.title("Salary estimation model Polynomial Regression of degree=4")

mtp.xlabel("Position Levels")

mtp.ylabel("Salary")

mtp.show()

lin\_pred = lin\_regs.predict([[6.5]])

print(lin\_pred)

poly\_pred =lin\_reg\_2(poly\_regs.fit\_transform([[6.5]]))

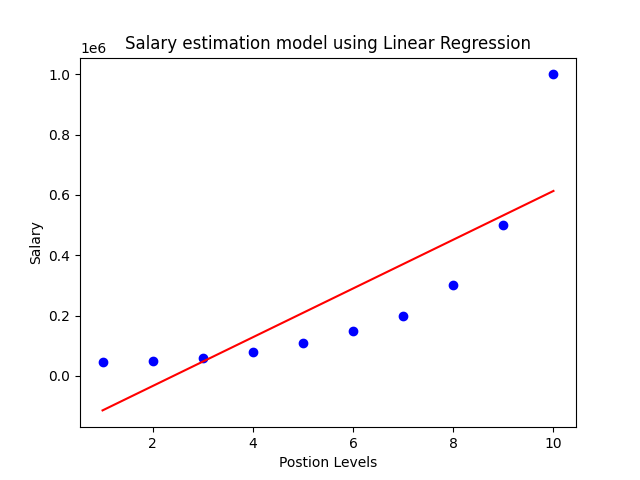
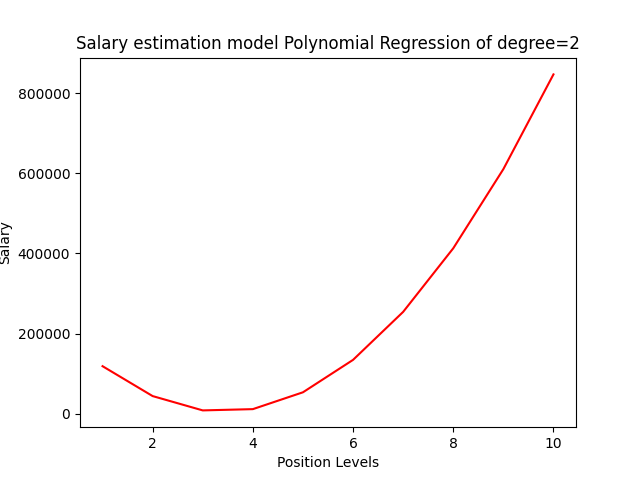
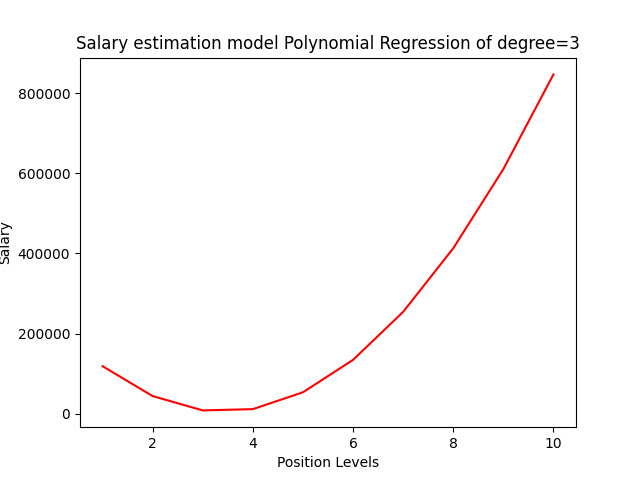
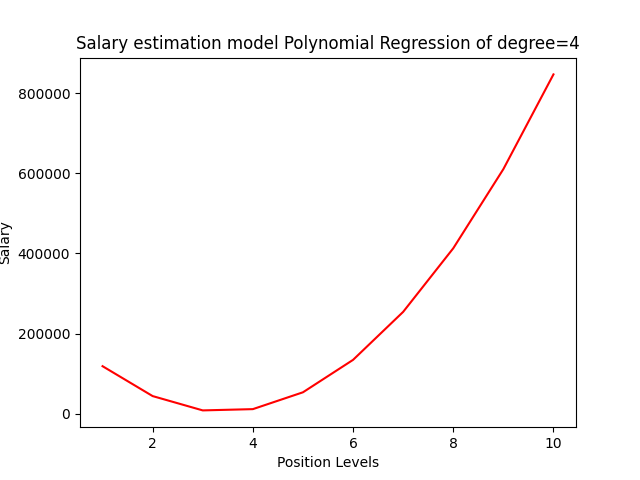
print(poly\_pred)

poly\_pred =lin\_reg\_3(poly\_regs.fit\_transform([[6.5]]))

print(poly\_pred)

poly\_pred =lin\_reg\_4(poly\_regs.fit\_transform([[6.5]]))

print(poly\_pred)



7. Write a python program to Implement Naïve Bayes.

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import confusion\_matrix, accuracy\_score

from matplotlib.colors import ListedColormap

dataset = pd.read\_csv('suv\_data.csv')

x = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

x, y, test\_size=0.25, random\_state=0)

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.transform(x\_test)

classifier = GaussianNB()

classifier.fit(x\_train, y\_train)

GaussianNB(priors=None, var\_smoothing=1e-09)

y\_pred = classifier.predict(x\_test)

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

print("Accuracy = ", accuracy\_score(y\_test, y\_pred))

x\_set, y\_set = x\_train, y\_train

X1, X2 = nm.meshgrid(nm.arange(start=x\_set[:, 0].min() - 1, stop=x\_set[:, 0].max() + 1, step=0.01),

nm.arange(start=x\_set[:, 1].min() - 1, stop=x\_set[:, 1].max() + 1, step=0.01))

mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(

X1.shape), alpha=0.75, cmap=ListedColormap(('white', 'grey')))

mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c=ListedColormap(('purple', 'green'))(i), label=j)

mtp.title("Naive Bayes(Training set)")

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

x\_set, y\_set = x\_train, y\_train

X1, X2 = nm.meshgrid(nm.arange(start=x\_set[:, 0].min() - 1, stop=x\_set[:, 0].max(

) + 1, step=0.01), nm.arange(start=x\_set[:, 1].min() - 1, stop=x\_set[:, 1].max() + 1, step=0.01))

mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(

X1.shape), alpha=0.75, cmap=ListedColormap(('white', 'grey')))

mtp.xlim(X1.min(), X1.max())

mtp.ylim(X2.min(), X2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c=ListedColormap(('purple', 'green'))(i), label=j)

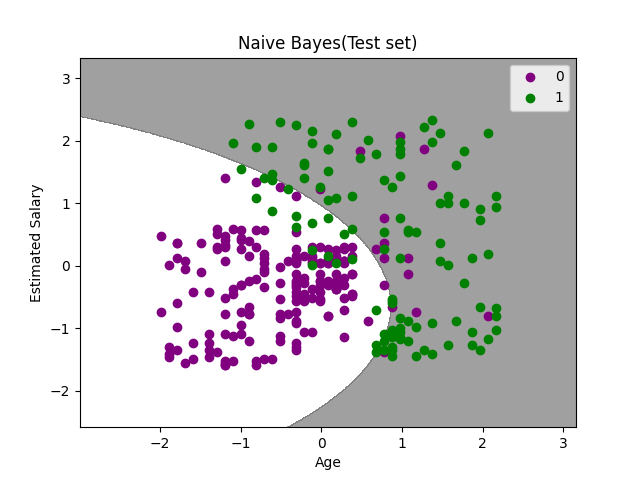
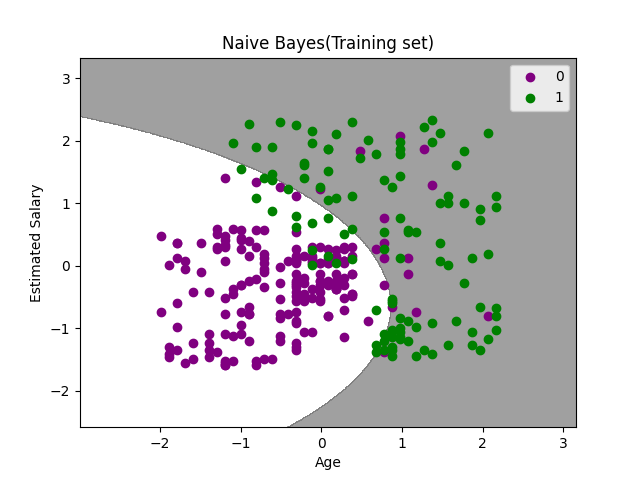
mtp.title("Naive Bayes(Test set)")

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()



8. Write a python program to Implement Decision Tree whether or not to play tennis.

import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn import tree

#Loading the dataset

PlayTennis = pd.read\_csv("play\_tennis.csv")

#Before LabelEncoding

print(PlayTennis)

Le = LabelEncoder()

PlayTennis['day'] = Le.fit\_transform(PlayTennis['day'])

PlayTennis['outlook'] = Le.fit\_transform(PlayTennis['outlook'])

PlayTennis['temp'] = Le.fit\_transform(PlayTennis['temp'])

PlayTennis['humidity'] = Le.fit\_transform(PlayTennis['humidity'])

PlayTennis['wind'] = Le.fit\_transform(PlayTennis['wind'])

PlayTennis['play'] = Le.fit\_transform(PlayTennis['play'])

#After LabelEncoding

print(PlayTennis)

#Determining Target variabel and independent variable

X = PlayTennis.drop(['play'],axis=1) #Set of input variables of outlook, temperature, humidity, windy

y = PlayTennis['play'] #The target variable play

#Decision tree from sklearn

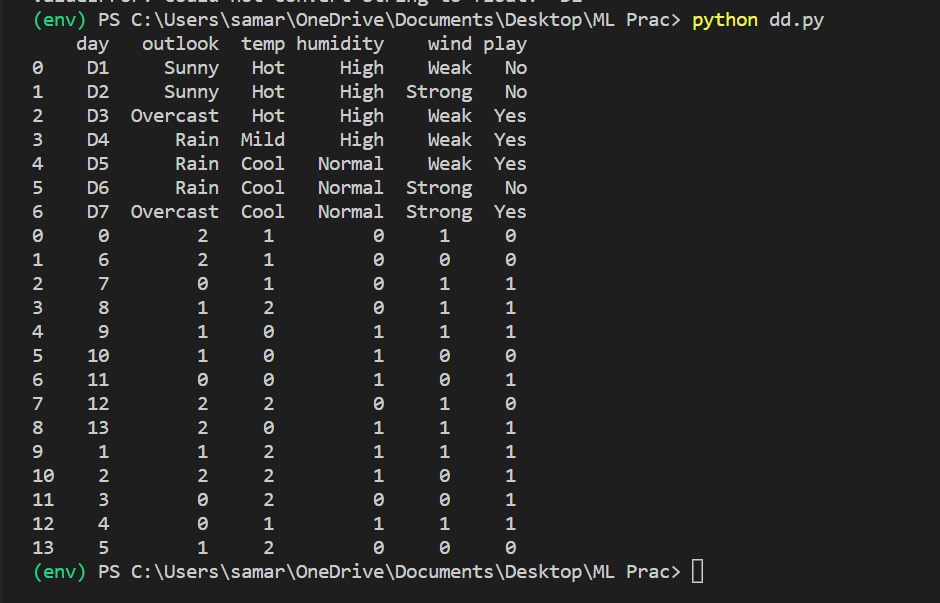
clf = tree.DecisionTreeClassifier(criterion = 'entropy')

clf = clf.fit(X, y)

X\_pred = clf.predict(X)

# verifying if the model has predicted it all right.

X\_pred == y



9. Write a python program to implement linear SVM.

#Data Pre-processing Step

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets

data\_set= pd.read\_csv('User\_Data.csv')

#Extracting Independent and dependent Variable

x= data\_set.iloc[:, [2,3]].values

y= data\_set.iloc[:, 4].values

# Splitting the dataset into training and test set.

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

#feature Scaling

from sklearn.preprocessing import StandardScaler

st\_x= StandardScaler()

x\_train= st\_x.fit\_transform(x\_train)

x\_test= st\_x.transform(x\_test)

from sklearn.svm import SVC # "Support vector classifier"

classifier = SVC(kernel='linear', random\_state=0)

classifier.fit(x\_train, y\_train)

#Predicting the test set result

y\_pred= classifier.predict(x\_test)

#Creating the Confusion matrix

from sklearn.metrics import confusion\_matrix

cm= confusion\_matrix(y\_test, y\_pred)

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_train, y\_train

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

mtp.title('SVM classifier (Training set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

#Visulaizing the test set result

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_test, y\_test

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('red','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

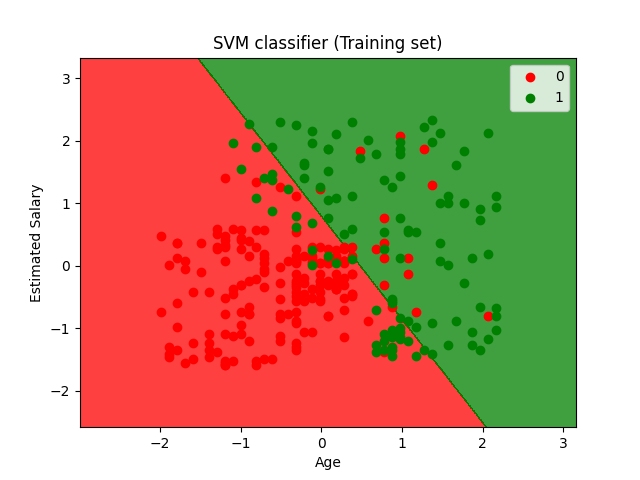
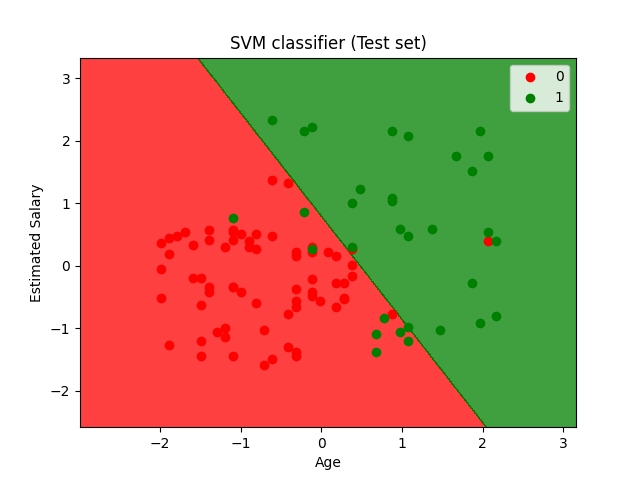
mtp.title('SVM classifier (Test set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()



10. Write a python program to find Decision boundary by using a neural network with 10

hidden units on two moons dataset

# %% 1

# Package imports

import matplotlib.pyplot as plt

import numpy as np

import sklearn

import sklearn.datasets

import sklearn.linear\_model

import matplotlib

# Display plots inline and change default figure size

#%matplotlib inline

matplotlib.rcParams['figure.figsize'] = (10.0, 8.0)

# %% 2

np.random.seed(3)

X, y = sklearn.datasets.make\_moons(200, noise=0.20)

plt.scatter(X[:,0], X[:,1], s=40, c=y, cmap=plt.cm.Spectral)

# %% 3

# Train the logistic rgeression classifier

clf = sklearn.linear\_model.LogisticRegressionCV()

clf.fit(X, y)

# %% 4

# Helper function to plot a decision boundary.

# If you don't fully understand this function don't worry, it just generates the contour plot below.

def plot\_decision\_boundary(pred\_func):

# Set min and max values and give it some padding

x\_min, x\_max = X[:, 0].min() - .5, X[:, 0].max() + .5

y\_min, y\_max = X[:, 1].min() - .5, X[:, 1].max() + .5

h = 0.01

# Generate a grid of points with distance h between them

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

# Predict the function value for the whole gid

Z = pred\_func(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

# Plot the contour and training examples

plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)

# %% 12

# Plot the decision boundary

plot\_decision\_boundary(lambda x: clf.predict(x))

plt.title("Logistic Regression")

# %% 15

num\_examples = len(X) # training set size

nn\_input\_dim = 2 # input layer dimensionality

nn\_output\_dim = 2 # output layer dimensionality

# Gradient descent parameters (I picked these by hand)

epsilon = 0.01 # learning rate for gradient descent

reg\_lambda = 0.01 # regularization strength

# %% 7

# Helper function to evaluate the total loss on the dataset

def calculate\_loss(model):

W1, b1, W2, b2 = model['W1'], model['b1'], model['W2'], model['b2']

# Forward propagation to calculate our predictions

z1 = X.dot(W1) + b1

a1 = np.tanh(z1)

z2 = a1.dot(W2) + b2

exp\_scores = np.exp(z2)

probs = exp\_scores / np.sum(exp\_scores, axis=1, keepdims=True)

# Calculating the loss

corect\_logprobs = -np.log(probs[range(num\_examples), y])

data\_loss = np.sum(corect\_logprobs)

# Add regulatization term to loss (optional)

data\_loss += reg\_lambda/2 \* (np.sum(np.square(W1)) + np.sum(np.square(W2)))

return 1./num\_examples \* data\_loss

# %% 8

# Helper function to predict an output (0 or 1)

def predict(model, x):

W1, b1, W2, b2 = model['W1'], model['b1'], model['W2'], model['b2']

# Forward propagation

z1 = x.dot(W1) + b1

a1 = np.tanh(z1)

z2 = a1.dot(W2) + b2

exp\_scores = np.exp(z2)

probs = exp\_scores / np.sum(exp\_scores, axis=1, keepdims=True)

return np.argmax(probs, axis=1)

# %% 16

# This function learns parameters for the neural network and returns the model.

# - nn\_hdim: Number of nodes in the hidden layer

# - num\_passes: Number of passes through the training data for gradient descent

# - print\_loss: If True, print the loss every 1000 iterations

def build\_model(nn\_hdim, num\_passes=20000, print\_loss=False):

# Initialize the parameters to random values. We need to learn these.

np.random.seed(0)

W1 = np.random.randn(nn\_input\_dim, nn\_hdim) / np.sqrt(nn\_input\_dim)

b1 = np.zeros((1, nn\_hdim))

W2 = np.random.randn(nn\_hdim, nn\_output\_dim) / np.sqrt(nn\_hdim)

b2 = np.zeros((1, nn\_output\_dim))

# This is what we return at the end

model = {}

# Gradient descent. For each batch...

for i in range(0, num\_passes):

# Forward propagation

z1 = X.dot(W1) + b1

a1 = np.tanh(z1)

z2 = a1.dot(W2) + b2

exp\_scores = np.exp(z2)

probs = exp\_scores / np.sum(exp\_scores, axis=1, keepdims=True)

# Backpropagation

delta3 = probs

delta3[range(num\_examples), y] -= 1

dW2 = (a1.T).dot(delta3)

db2 = np.sum(delta3, axis=0, keepdims=True)

delta2 = delta3.dot(W2.T) \* (1 - np.power(a1, 2))

dW1 = np.dot(X.T, delta2)

db1 = np.sum(delta2, axis=0)

# Add regularization terms (b1 and b2 don't have regularization terms)

dW2 += reg\_lambda \* W2

dW1 += reg\_lambda \* W1

# Gradient descent parameter update

W1 += -epsilon \* dW1

b1 += -epsilon \* db1

W2 += -epsilon \* dW2

b2 += -epsilon \* db2

# Assign new parameters to the model

model = { 'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2}

# Optionally print the loss.

# This is expensive because it uses the whole dataset, so we don't want to do it too often.

if print\_loss and i % 1000 == 0:

print("Loss after iteration %i: %f" %(i, calculate\_loss(model)))

return model

# %% 17

# Build a model with a 3-dimensional hidden layer

model = build\_model(10, print\_loss=True)

# Plot the decision boundary

plot\_decision\_boundary(lambda x: predict(model, x))

plt.title("Decision Boundary for hidden layer size 10")

Output:

Loss after iteration 0: 0.594909

Loss after iteration 1000: 0.044892

Loss after iteration 2000: 0.038739

Loss after iteration 3000: 0.036230

Loss after iteration 4000: 0.035161

Loss after iteration 5000: 0.034530

Loss after iteration 6000: 0.034043

Loss after iteration 7000: 0.033568

Loss after iteration 8000: 0.033012

Loss after iteration 9000: 0.032285

Loss after iteration 10000: 0.031309

Loss after iteration 11000: 0.030272

Loss after iteration 12000: 0.029468

Loss after iteration 13000: 0.028961

Loss after iteration 14000: 0.028679

Loss after iteration 15000: 0.028532

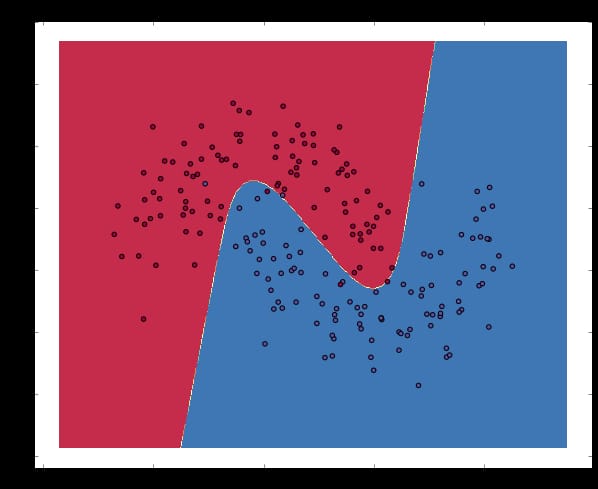
Loss after iteration 16000: 0.028449

Loss after iteration 17000: 0.028395

Loss after iteration 18000: 0.028357

Loss after iteration 19000: 0.028327

Text(0.5, 1.0, 'Decision Boundary for hidden layer size 10')



11. Write a python program to transform data with Principal Component Analysis (PCA)

# importing required libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# importing or loading the dataset

dataset = pd.read\_csv('wine.csv')

# distributing the dataset into two components X and Y

X = dataset.iloc[:, 0:13].values

y = dataset.iloc[:, 13].values

# Splitting the X and Y into the

# Training set and Testing set

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 = 0)

# performing preprocessing part

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Applying PCA function on training

# and testing set of X component

from sklearn.decomposition import PCA

pca = PCA(n\_components = 2)

X\_train = pca.fit\_transform(X\_train)

X\_test = pca.transform(X\_test)

explained\_variance = pca.explained\_variance\_ratio\_

# Fitting Logistic Regression To the training set

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

classifier.fit(X\_train, y\_train)

# Predicting the test set result using

# predict function under LogisticRegression

y\_pred = classifier.predict(X\_test)

# making confusion matrix between

# test set of Y and predicted value.

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Predicting the training set

# result through scatter plot

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1,

stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1,

stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),

X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,

cmap = ListedColormap(('yellow', 'white', 'aquamarine')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green', 'blue'))(i), label = j)

plt.title('Logistic Regression (Training set)')

plt.xlabel('PC1') # for Xlabel

plt.ylabel('PC2') # for Ylabel

plt.legend() # to show legend

# show scatter plot

plt.show()

# Visualising the Test set results through scatter plot

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1,

stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1,

stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),

X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,

cmap = ListedColormap(('yellow', 'white', 'aquamarine')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green', 'blue'))(i), label = j)

# title for scatter plot

plt.title('Logistic Regression (Test set)')

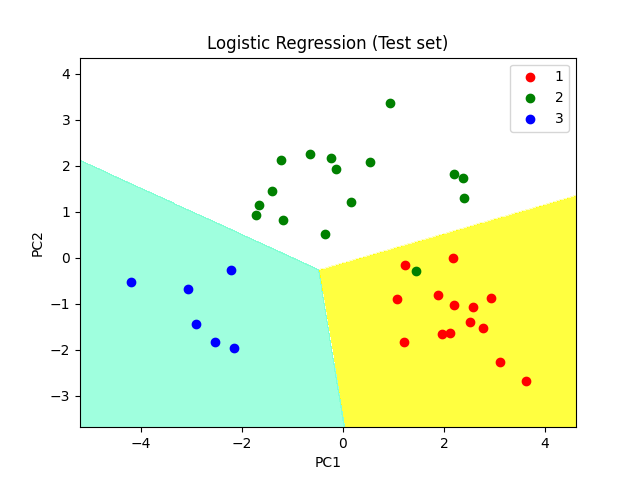
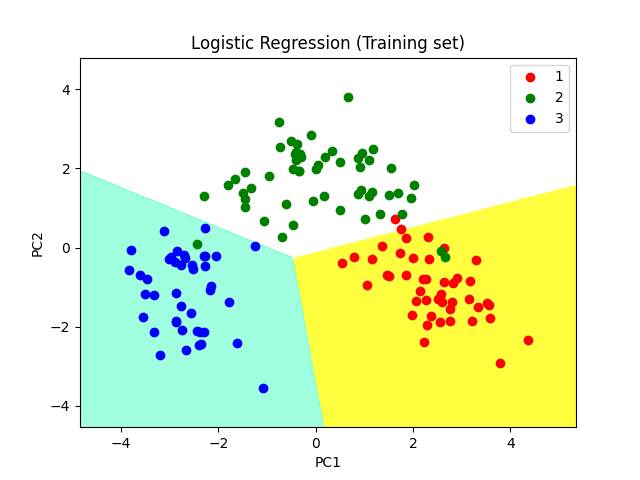
plt.xlabel('PC1') # for Xlabel

plt.ylabel('PC2') # for Ylabel

plt.legend()

# show scatter plot

plt.show()



12. Write a python program to implement k-nearest Neighbors ML algorithm to build

prediction model (Use Forge Dataset)

# Step 1 - Load Data

import pandas as pd

dataset = pd.read\_csv("iphone\_purchase\_records.csv")

X = dataset.iloc[:,:-1].values

y = dataset.iloc[:, 3].values

# Step 2 - Convert Gender to number

from sklearn.preprocessing import LabelEncoder

labelEncoder\_gender = LabelEncoder()

X[:,0] = labelEncoder\_gender.fit\_transform(X[:,0])

# Optional - if you want to convert X to float data type

import numpy as np

X = np.vstack(X[:, :]).astype(float)

# Step 3 - Split into training and test data

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=0)

# Step 4 - Feature Scaling

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

# Step 5 - Fit KNN Classifier

from sklearn.neighbors import KNeighborsClassifier

# metric = minkowski and p=2 is Euclidean Distance

# metric = minkowski and p=1 is Manhattan Distance

classifier = KNeighborsClassifier(n\_neighbors=5, metric="minkowski",p=2)

classifier.fit(X\_train, y\_train)

# Step 5 - Make Prediction

y\_pred = classifier.predict(X\_test)

# Step 6 - Confusion Matrix

from sklearn import metrics

cm = metrics.confusion\_matrix(y\_test, y\_pred) ## 4,3 errors

accuracy = metrics.accuracy\_score(y\_test, y\_pred) ## 0.93

precision = metrics.precision\_score(y\_test, y\_pred) ## 0.87

recall = metrics.recall\_score(y\_test, y\_pred) ## 0.90

# Step 7 - Confusion Matrix

from sklearn import metrics

cm = metrics.confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

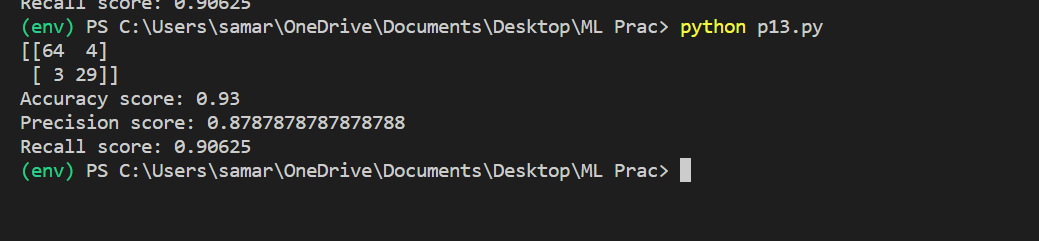
print("Accuracy score:",accuracy)

precision = metrics.precision\_score(y\_test, y\_pred)

print("Precision score:",precision)

recall = metrics.recall\_score(y\_test, y\_pred)

print("Recall score:",recall)



13. Write a python program to implement k-means algorithm on a synthetic dataset.

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Mall\_Customers.csv')

x = dataset.iloc[:, [3, 4]].values

from sklearn.cluster import KMeans

wcss\_list= [] #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state= 42)

kmeans.fit(x)

wcss\_list.append(kmeans.inertia\_)

mtp.plot(range(1, 11), wcss\_list)

mtp.title('The Elobw Method Graph')

mtp.xlabel('Number of clusters(k)')

mtp.ylabel('wcss\_list')

mtp.show()

#training the K-means model on a dataset

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state= 42)

y\_predict= kmeans.fit\_predict(x)

mtp.scatter(x[y\_predict == 0, 0], x[y\_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster

mtp.scatter(x[y\_predict == 1, 0], x[y\_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster

mtp.scatter(x[y\_predict== 2, 0], x[y\_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster

mtp.scatter(x[y\_predict == 3, 0], x[y\_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster

mtp.scatter(x[y\_predict == 4, 0], x[y\_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster

mtp.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroid')

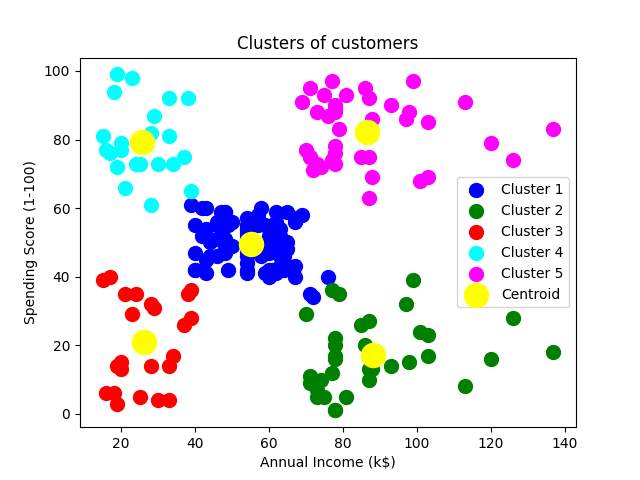
mtp.title('Clusters of customers')

mtp.xlabel('Annual Income (k$)')

mtp.ylabel('Spending Score (1-100)')

mtp.legend()

mtp.show()



14. Write a python program to implement Agglomerative clustering on a synthetic dataset.

# Importing the libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Mall\_Customers.csv')

dataset.head()

dataset.shape

x = dataset.iloc[:, [3, 4]].values

#Finding the optimal number of clusters using the dendrogram

import scipy.cluster.hierarchy as shc

dendro = shc.dendrogram(shc.linkage(x, method="ward"))

mtp.title("Dendrogrma Plot")

mtp.ylabel("Euclidean Distances")

mtp.xlabel("Customers")

mtp.show()

#training the hierarchical model on dataset

from sklearn.cluster import AgglomerativeClustering

hc= AgglomerativeClustering(n\_clusters=5, affinity='euclidean', linkage='ward')

y\_pred= hc.fit\_predict(x)

#visulaizing the clusters

mtp.scatter(x[y\_pred == 0, 0], x[y\_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')

mtp.scatter(x[y\_pred == 1, 0], x[y\_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')

mtp.scatter(x[y\_pred== 2, 0], x[y\_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')

mtp.scatter(x[y\_pred == 3, 0], x[y\_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

mtp.scatter(x[y\_pred == 4, 0], x[y\_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

mtp.title('Clusters of customers')

mtp.xlabel('Annual Income (k$)')

mtp.ylabel('Spending Score (1-100)')

mtp.legend()

mtp.show()

