**Machine Learning Practical’s**

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

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

iris = pd.read\_csv("Iris.csv") # Reading the dataset “Iris.csv”.

print (iris.head(10))

plt.plot(iris.Id, iris["SepalLengthCm"],"r--")

plt.show #plt.show () will display the current figure that you are working on

iris.plot(kind ="scatter", x ='SepalLengthCm', y ='PetalLengthCm')

plt.grid() # grid () function to add grid lines to the plot

Iris.csv

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species

0 1 5.1 3.5 1.4 0.2 Iris-setosa

1 2 4.9 3.0 1.4 0.2 Iris-setosa

2 3 4.7 3.2 1.3 0.2 Iris-setosa

3 4 4.6 3.1 1.5 0.2 Iris-setosa

4 5 5.0 3.6 1.4 0.2 Iris-setosa

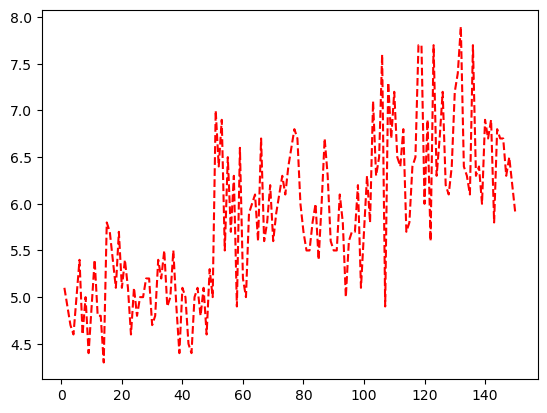
5 6 5.4 3.9 1.7 0.4 Iris-setosa

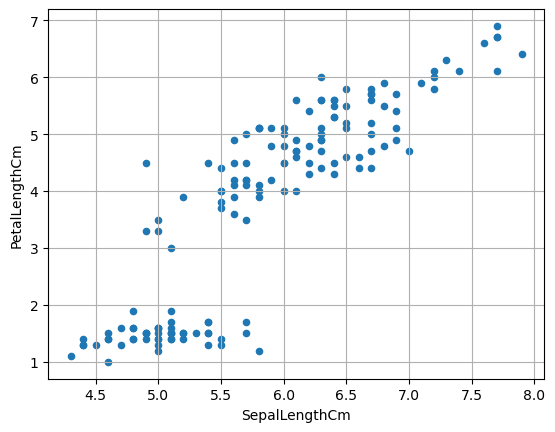
6 7 4.6 3.4 1.4 0.3 Iris-setosa

7 8 5.0 3.4 1.5 0.2 Iris-setosa

8 9 4.4 2.9 1.4 0.2 Iris-setosa

9 10 4.9 3.1 1.5 0.1 Iris-setosa





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

import pandas as pd

import numpy as np

dict={'first score':[100,90,np.nan,95],

      'second score':[30,45,56,np.nan],

      'third score':[np.nan,40,80,98]}

df=pd.DataFrame(dict)

|  |  |  |  |
| --- | --- | --- | --- |
|  | First score | Second score | Third score |
| 0 | 100.0 | 30.0 | NaN |
| 1 | 90.0 | 45.0 | 40.0 |
| 2 | NaN | 56.0 | 80.0 |
| 3 | 95.0 | NaN | 98.0 |

df.isnull()

|  |  |  |  |
| --- | --- | --- | --- |
|  | First score | Second score | Third score |
| 0 | False | False | True |
| 1 | False | False | False |
| 2 | True | False | False |
| 3 | False | False | False |

df.notnull()

|  |  |  |  |
| --- | --- | --- | --- |
|  | First score | Second score | Third score |
| 0 | True | True | False |
| 1 | True | True | True |
| 2 | False | True | True |
| 3 | True | False | True |

#df=pd.DataFrame(dict)

df.fillna(0)

|  |  |  |  |
| --- | --- | --- | --- |
|  | First score | Second score | Third score |
| 0 | 100.0 | 30.0 | 0.0 |
| 1 | 90.0 | 45.0 | 40.0 |
| 2 | 0.0 | 56.0 | 80.0 |
| 3 | 95.0 | 0.0 | 98.0 |

df.fillna(method='pad')

|  |  |  |  |
| --- | --- | --- | --- |
|  | First score | Second score | Third score |
| 0 | 100.0 | 30.0 | NaN |
| 1 | 90.0 | 45.0 | 40.0 |
| 2 | 90.0 | 56.0 | 80.0 |
| 3 | 95.0 | 56.0 | 98.0 |

df.fillna(method='bfill')

|  |  |  |  |
| --- | --- | --- | --- |
|  | First score | Second score | Third score |
| 0 | 100.0 | 30.0 | 40.0 |
| 1 | 95.0 | 45.0 | 40.0 |
| 2 | 95.0 | 56.0 | 80.0 |
| 3 | 95.0 | NaN | 98.0 |

df.replace(to\_replace=np.nan,value=-99)

|  |  |  |  |
| --- | --- | --- | --- |
|  | First score | Second score | Third score |
| 0 | 100.0 | 30.0 | -99.0 |
| 1 | 95.0 | 45.0 | 40.0 |
| 2 | -99.0 | 56.0 | 80.0 |
| 3 | 95.0 | -99.0 | 98.0 |

df.dropna()

|  |  |  |  |
| --- | --- | --- | --- |
|  | First score | Second score | Third score |
| 1 | 90.0 | 45.0 | 40.0 |

df.dropna(axis=1)

|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |

new\_data=df.dropna(axis=0)

new\_data

|  |  |  |  |
| --- | --- | --- | --- |
|  | First score | Second score | Third score |
| 1 | 90.0 | 45.0 | 40.0 |

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

import pandas as pd

import numpy as np

# Define the headers since the data does not have any

headers = ["symboling", "normalized\_losses", "make", "fuel\_type", "aspiration",

           "num\_doors", "body\_style", "drive\_wheels", "engine\_location",

           "wheel\_base", "length", "width", "height", "curb\_weight",

           "engine\_type", "num\_cylinders", "engine\_size", "fuel\_system",

           "bore", "stroke", "compression\_ratio", "horsepower", "peak\_rpm",

           "city\_mpg", "highway\_mpg", "price"]

# Read in the CSV file and convert "?" to NaN

df = pd.read\_csv('imports-85.data',header=None, names=headers, na\_values="?"

df.head()

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

from sklearn.linear\_model import LinearRegression

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

from sklearn.model\_selection import cross\_val\_predict

data = pd.read\_csv(r'kc\_house\_data.csv')

data.head(5)

print(data.shape)

# Make a list of importatnt feature which is needed to be incuding in training data

f = ['price', 'bedrooms', 'bathrooms', 'sqft\_living', 'floors', 'condition', 'sqft\_above', 'sqft\_basement', 'yr\_built',

     'yr\_renovated']

data = data[f]

print(data.shape)

# Drop the missing values

data = data.dropna()

print(data.shape)

# Get the statictial information of the dataset

data.describe()

# Now,Divide the dataset into two parts:independent variable and dependent variable

X = data[f[1:]]

y = data['price']

# Split the dataset into training data and testing data

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.2, random\_state=42)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

# Fit the regression model

lr = LinearRegression() # Create object of linear regression class

lr.fit(X\_train,y\_train) #fit training data

print(lr.coef\_)

# Create the Prediction

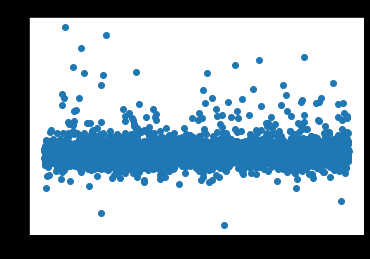
y\_test\_predict = lr.predict(X\_test)

print(y\_test\_predict.shape)

# Plot the error

g=plt.plot((y\_test - y\_test\_predict),marker='o',linestyle='')

(21613, 21) (21613, 10) (21613, 10) (17290, 9) (4323, 9) (17290,) (4323,) [-6.66271486e+04 7.03502003e+04 1.95218823e+02 5.47636142e+04 1.84031121e+04 1.02625904e+02 9.25929193e+01 -3.21077826e+03 2.37927347e+01] (4323,)



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

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('50\_Startups.csv')

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

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

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],

remainder='passthrough')

X = np.array(ct.fit\_transform(X))

print(X)

Output exceeds the [size limit](command:workbench.action.openSettings?%5B%22notebook.output.textLineLimit%22%5D). Open the full output data [in a text editor](command:workbench.action.openLargeOutput?755bc745-624d-4df4-86ed-c7d10d5d1625)

[[0.0 0.0 1.0 165349.2 136897.8 471784.1] [1.0 0.0 0.0 162597.7 151377.59 443898.53] [0.0 1.0 0.0 153441.51 101145.55 407934.54] [0.0 0.0 1.0 144372.41 118671.85 383199.62] [0.0 1.0 0.0 142107.34 91391.77 366168.42] [0.0 0.0 1.0 131876.9 99814.71 362861.36] [1.0 0.0 0.0 134615.46 147198.87 127716.82] [0.0 1.0 0.0 130298.13 145530.06 323876.68] [0.0 0.0 1.0 120542.52 148718.95 311613.29] [1.0 0.0 0.0 123334.88 108679.17 304981.62] [0.0 1.0 0.0 101913.08 110594.11 229160.95] [1.0 0.0 0.0 100671.96 91790.61 249744.55] [0.0 1.0 0.0 93863.75 127320.38 249839.44] [1.0 0.0 0.0 91992.39 135495.07 252664.93] [0.0 1.0 0.0 119943.24 156547.42 256512.92] [0.0 0.0 1.0 114523.61 122616.84 261776.23] [1.0 0.0 0.0 78013.11 121597.55 264346.06] [0.0 0.0 1.0 94657.16 145077.58 282574.31] [0.0 1.0 0.0 91749.16 114175.79 294919.57] [0.0 0.0 1.0 86419.7 153514.11 0.0] [1.0 0.0 0.0 76253.86 113867.3 298664.47] [0.0 0.0 1.0 78389.47 153773.43 299737.29] [0.0 1.0 0.0 73994.56 122782.75 303319.26] [0.0 1.0 0.0 67532.53 105751.03 304768.73] [0.0 0.0 1.0 77044.01 99281.34 140574.81]

...

[0.0 1.0 0.0 1315.46 115816.21 297114.46] [1.0 0.0 0.0 0.0 135426.92 0.0] [0.0 0.0 1.0 542.05 51743.15 0.0] [1.0 0.0 0.0 0.0 116983.8 45173.06]]

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)

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

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

LinearRegression()

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

df = pd.DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})

df

|  | **Real Values** | **Predicted Values** |
| --- | --- | --- |
| 0 | 78239.91 | 73546.182964 |
| 1 | 105733.54 | 111092.833250 |
| 2 | 146121.95 | 136450.808203 |
| 3 | 155752.60 | 158446.997558 |
| 4 | 42559.73 | 46488.638767 |
| 5 | 97483.56 | 99202.856322 |
| 6 | 191792.06 | 184053.617205 |
| 7 | 111313.02 | 116184.677449 |
| 8 | 96712.80 | 85716.721313 |
| 9 | 35673.41 | 55727.590882 |

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

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

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

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

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

dataset.head(5)

|  | **Position** | **Level** | **Salary** |
| --- | --- | --- | --- |
| 0 | Business Analyst | 1 | 45000 |
| 1 | Junior Consultant | 2 | 50000 |
| 2 | Senior Consultant | 3 | 60000 |
| 3 | Manager | 4 | 80000 |
| 4 | Country Manager | 5 | 110000 |

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

poly\_reg = PolynomialFeatures(degree = 4)

X\_poly = poly\_reg.fit\_transform(X)

lin\_reg = LinearRegression()

lin\_reg.fit(X\_poly, y)

LinearRegression()

y\_pred = lin\_reg.predict(X\_poly)

df = pd.DataFrame({'Real Values':y, 'Predicted Values':y\_pred})

df

|  | **Real Values** | **Predicted Values** |
| --- | --- | --- |
| 0 | 45000 | 53356.643357 |
| 1 | 50000 | 31759.906760 |
| 2 | 60000 | 58642.191142 |
| 3 | 80000 | 94632.867133 |
| 4 | 110000 | 121724.941725 |
| 5 | 150000 | 143275.058275 |
| 6 | 200000 | 184003.496503 |
| 7 | 300000 | 289994.172494 |
| 8 | 500000 | 528694.638695 |
| 9 | 1000000 | 988916.083916 |

X\_grid = np.arange(min(X), max(X), 0.1)

X\_grid = X\_grid.reshape((len(X\_grid), 1))

plt.scatter(X, y, color = 'red')

plt.scatter(X, y\_pred, color = 'green')

plt.plot(X\_grid,

lin\_reg.predict(poly\_reg.fit\_transform(X\_grid)), color =

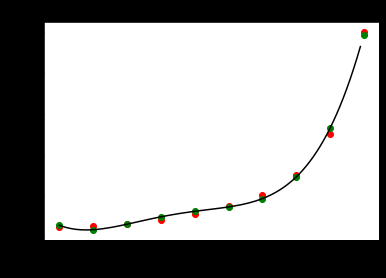
'black')

plt.title('Polynomial Regression')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()



7. Write a python program to Implement Naïve Bayes.

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

9. Write a python program to implement linear SVM.

10. Write a python program to find Decision boundary by using a neural network with 10 hidden units on two moons dataset

import numpy as np

from sklearn.datasets import make\_moons

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

import matplotlib.pyplot as plt

np.random.seed(0)

X, Y = make\_moons(500, noise=0.1)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

    X, Y, test\_size=0.25, random\_state=73)

plt.figure(figsize=(12,8))

plt.scatter(X\_train[:,0], X\_train[:,1], c=Y\_train,

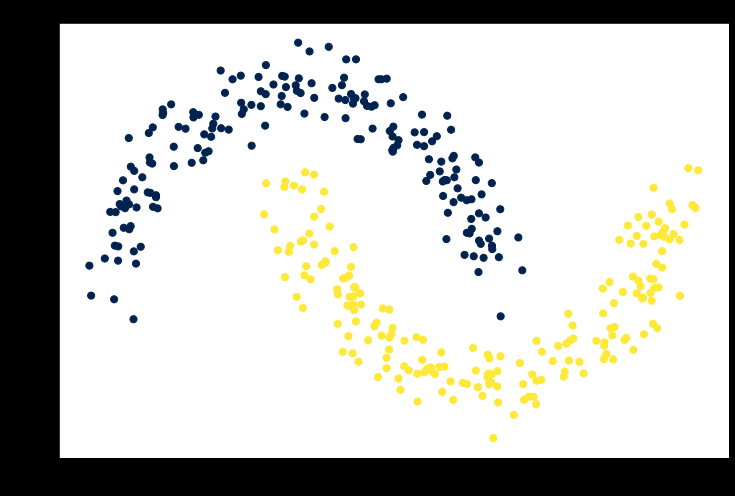
            cmap=plt.cm.cividis, s=50)

plt.xlabel('X1')

plt.ylabel('X2')

plt.title('Random Training Data')

plt.show()



n\_inputs = X\_train.shape[0]

n\_inputs\_dim = X\_train.shape[1]

n\_h = 4 # Number of hidden nodes

n\_out = 1 # Number of output nodes = for binary classifier

c = np.sqrt(3 / (0.5 + n\_inputs\_dim + n\_out))

# Initialize weights and bias

W1 = np.random.uniform(low=-c, high=c,

                    size=(n\_inputs\_dim, n\_h))

b1 = np.zeros((1, n\_h))

W2 = np.random.uniform(low=-c, high=c,

                       size=(n\_h, n\_out))

b2 = np.zeros((1, n\_out))

def Elu(x, a=2):

    """

    Compute the ELU output of x

    """

    return np.where(x<=0, a \* (np.exp(x) - 1), x)

def sigmoid(x):

    """

    Compute sigmoid of array x

    """

    return 1 / (1 + np.exp(-x))

Z1 = np.dot(X\_train, W1) + b1

A1 = Elu(Z1)

Z2 = np.dot(A1, W2) + b2

A2 = sigmoid(Z2)

Y\_train = Y\_train.reshape(-1,1)

log\_probs = (np.multiply(np.log(A2), Y\_train) +

        np.multiply(np.log(1 - A2), (1 - Y\_train)))

loss = -1 / n\_inputs \* np.sum(log\_probs)

loss

def dElu(x, a=2):

    return np.where(x<=0,a \* np.exp(x), 0)

m = 1 / n\_inputs

dZ2 = A2 - Y\_train

dW2 = m \* np.dot(A1.T, dZ2)

db2 = m \* np.sum(dZ2, axis=0, keepdims=True)

dZ1 = m \* np.dot(dZ2, W2.T) \* dElu(Z1)

dW1 = m \* np.dot(X\_train.T, dZ1)

db1 = m \* np.sum(dZ1, axis=0, keepdims=True)

learning\_rate = 0.01

W2 -= learning\_rate \* dW2

b2 -= learning\_rate \* db2

W1 -= learning\_rate \* dW1

b1 -= learning\_rate \* db1

class network():

    def \_\_init\_\_(self, X, Y):

        self.X = X

        self.Y = Y

        self.n\_input\_dim = X.shape[1]

        self.n\_output\_dim = 1

        self.n\_inputs = X.shape[0]

    def initialize(self, n\_hidden, seed=1):

        self.n\_hidden = n\_hidden

        np.random.seed(seed)

        c = np.sqrt(3 / (0.5 + self.n\_input\_dim + self.n\_output\_dim))

        W1 = np.random.uniform(low=-c, high=c,

                    size=(self.n\_input\_dim, self.n\_hidden))

        b1 = np.zeros((1, self.n\_hidden))

        W2 = np.random.uniform(low=-c, high=c,

                              size=(self.n\_hidden, self.n\_output\_dim))

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

        self.params = {'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2}

        self.cache = {}

    def Elu(self, x, a=2):

        return np.where(x<=0, (a \* (np.exp(x) - 1)), x)

    def dElu(self, x, a=2):

        return np.where(x<=0, a \* np.exp(x), 1)

    def sigmoid(self, x):

        return 1. / (1 + np.exp(-x))

    def forward\_prop(self, X=None, cache=True):

        W1 = self.params['W1']

        b1 = self.params['b1']

        W2 = self.params['W2']

        b2 = self.params['b2']

        if X is None:

            X = self.X.copy()

        Z1 = X.dot(W1) + b1

        A1 = self.Elu(Z1)

        Z2 = np.dot(A1, W2) + b2

        A2 = self.sigmoid(Z2)

        probs = A2

        if cache:

            self.cache = {'Z1': Z1, 'A1': A1, 'Z2': Z2,

                          'A2': A2, 'probs': probs}

        else:

            return probs

    def back\_prop(self):

        # Import parameters and cached values

        A1 = self.cache['A1']

        A2 = self.cache['A2']

        W1 = self.params['W1']

        b1 = self.params['b1']

        W2 = self.params['W2']

        b2 = self.params['b2']

        # Calculate derivatives

        m = 1 / self.n\_inputs

        dZ2 = A2 - self.Y.reshape(-1,1)

        dW2 = m \* A1.T.dot(dZ2)

        db2 = m\* np.sum(dZ2, axis=0, keepdims=True)

        dZ1 = m \* dZ2.dot(W2.T) \* self.dElu(A1)

        dW1 = m \* np.dot(self.X.T, dZ1)

        db1 = m \* np.sum(dZ1, axis=0)

        # Apply gradient descent updates

        W1 -= self.learning\_rate \* dW1

        b1 -= self.learning\_rate \* db1

        W2 -= self.learning\_rate \* dW2

        b2 -= self.learning\_rate \* db2

        # Store updated network parameters

        self.params = {'W1': W1, 'b1': b1,

                       'W2': W2, 'b2': b2}

    def train(self, learning\_rate=1e-2,

              n\_iters=10000, log\_loss=False):

        self.learning\_rate = learning\_rate

        loss = []

        # Train the network

        for i in range(n\_iters):

            self.forward\_prop()

            self.back\_prop()

            # Calculate the loss value to track progress

            if log\_loss:

                loss.append(self.calculate\_loss())

        if log\_loss:

            return loss

    def predict(self, X):

        probs = self.forward\_prop(X, cache=False)

        return np.where(probs<0.5,0,1)

    def calculate\_loss(self):

        probs = self.cache['probs']

        W1 = self.params['W1']

        W2 = self.params['W2']

        Y = self.Y.reshape(-1,1)

        loss = (np.multiply(np.log(probs), Y) +

                np.multiply(np.log(1 - probs), (1 - Y)))

        return -1 / self.n\_inputs \* np.sum(loss)

    def train\_accuracy(self):

        probs = self.cache['probs']

        clf = np.where(probs<0.5, 0, 1)

        return np.sum(self.Y.reshape(-1,1)==clf) / self.n\_inputs

    # Call this function to view the decision boundary

    def plot\_decision\_boundary(self):

        # Determine grid range in x and y directions

        x\_min, x\_max = self.X[:, 0].min()-0.1, self.X[:, 0].max()+0.1

        y\_min, y\_max = self.X[:, 1].min()-0.1, self.X[:, 1].max()+0.1

        # Set grid spacing parameter

        spacing = min(x\_max - x\_min, y\_max - y\_min) / 100

        # Create grid

        XX, YY = np.meshgrid(np.arange(x\_min, x\_max, spacing),

                       np.arange(y\_min, y\_max, spacing))

        # Concatenate data to match input

        data = np.hstack((XX.ravel().reshape(-1,1),

                          YY.ravel().reshape(-1,1)))

        # Pass data to predict method

        clf = self.predict(data)

        Z = clf.reshape(XX.shape)

        plt.figure(figsize=(10,8))

        plt.contourf(XX, YY, Z, cmap=plt.cm.Spectral, alpha=0.8)

        plt.scatter(self.X[:,0], self.X[:,1], c=self.Y,

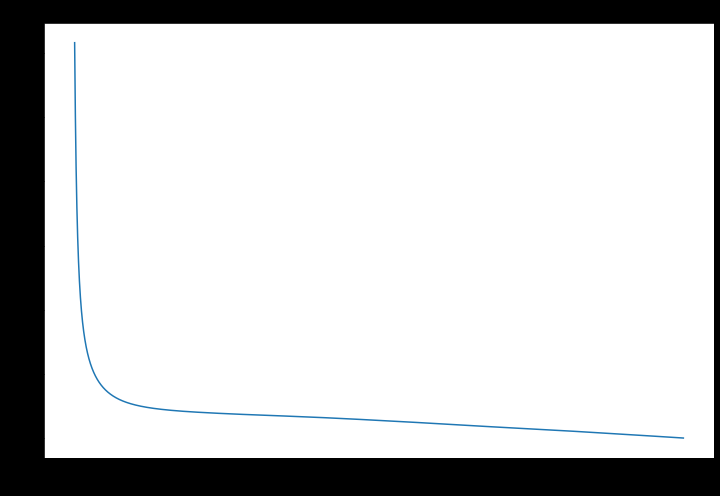
                    cmap=plt.cm.cividis, s=50)

        plt.show()

# In[14]:

net = network(X\_train, Y\_train)

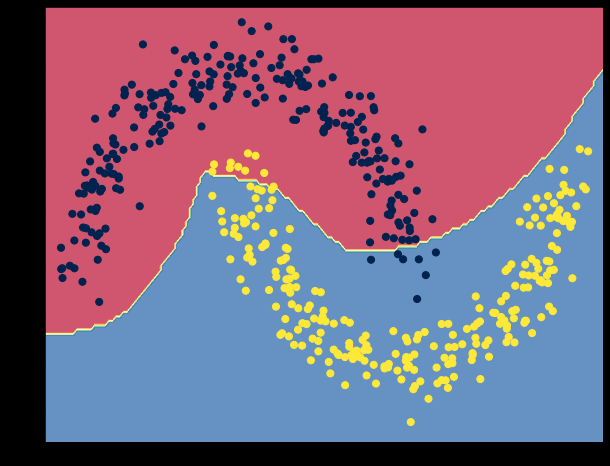
net.initialize(n\_hidden=3)



pred = net.predict(X\_test)

np.sum(Y\_test.reshape(-1,1)==pred) / len(Y\_test)

net.plot\_decision\_boundary()



n\_hidden = [10]

for n in n\_hidden:

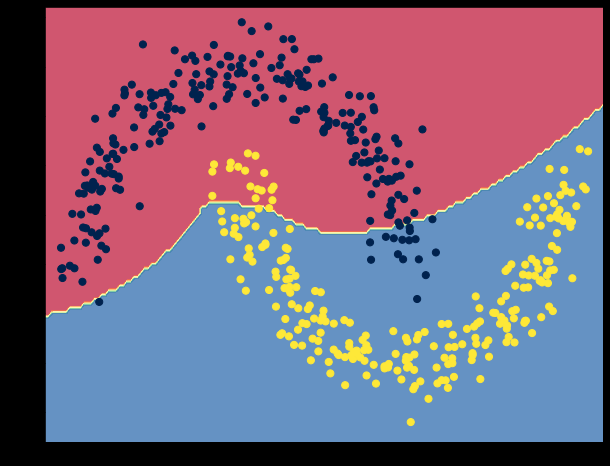
    net = network(X, Y)

    net.initialize(n\_hidden=n)

    net.train(learning\_rate=0.1, log\_loss=False)

    print("%d Hidden Nodes" %n)

    net.plot\_decision\_boundary()



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

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

import seaborn as sns

from sklearn.linear\_model import LinearRegression

from sklearn.decomposition import PCA

data=pd.read\_csv('auto-mpg (1).csv')

data.head()

data=data.drop(['car name','origin'],axis=1)

data.head()

hpisdigit=pd.DataFrame(data.horsepower.str.isdigit())

data[hpisdigit['horsepower']==False]

data=data.replace('?',np.nan)

data[hpisdigit['horsepower']==False]

data.median()

medianfiller=lambda x:x.fillna(x.median())

data=data.apply(medianfiller,axis=0)

data['horsepower']=data['horsepower'].astype('float64')

x=data.drop(['mpg'],axis=1)

y=data[['mpg']]

sns.pairplot(x)

from scipy.stats import zscore

Xscaled=x.apply(zscore)

Xscaled.head()

Xscaled.head()

print(covmatrix)

pca=PCA(n\_components=5)

pca.fit(Xscaled)

print(pca.explained\_variance\_)

print(pca.components)

print(pca.explained\_variance\_ratio\_)

xpca=pca.transform(Xscaled)

regression\_model=LinearRegression()

regression\_model.fit(Xscaled,y)

regression\_model.score(Xscaled,y)

regression\_model\_pca=LinearRegression()

regression\_model\_pca.fit(xpca,y)

regression\_model\_pca.score(xpca,y)

pca=PCA(n\_components=3)

pca.fit(Xscaled)

print(pca.explained\_variance\_)

print(pca.components\_)

print(pca.explained\_variance\_ratio\_)

xpca=pca.transform(Xscaled)

regression\_model=LinearRegression()

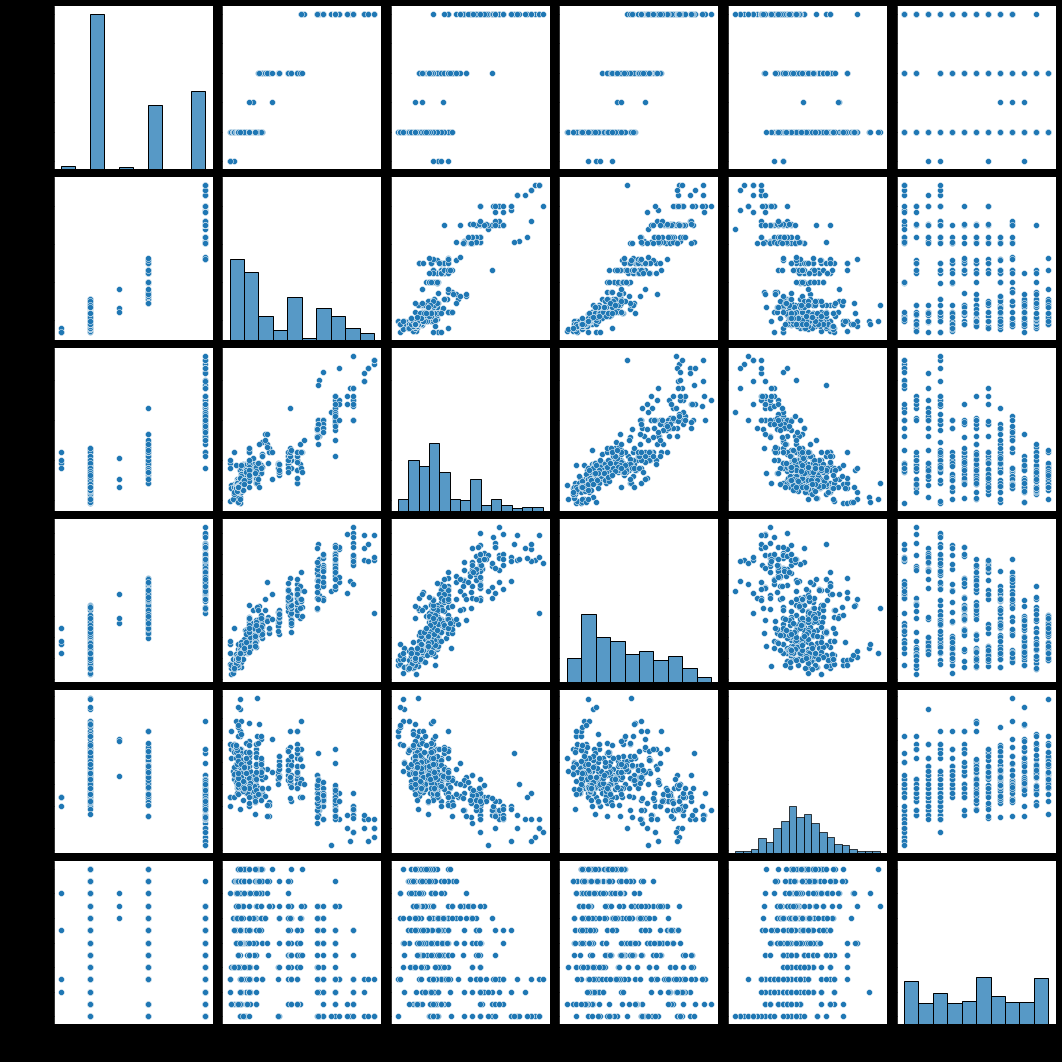
regression\_model.fit(Xscaled,y)

regression\_model.score(Xscaled,y)

regression\_model\_pca=LinearRegression()

regression\_model\_pca.fit(xpca,y)

regression\_model\_pca.score(xpca,y)



--------------------------------------------------------------------------- NameError Traceback (most recent call last) ~\AppData\Local\Temp/ipykernel\_14668/874798960.py in <module> 33 Xscaled.head() 34 ---> 35 print(covmatrix) 36 37 pca=PCA(n\_components=5) NameError: name 'covmatrix' is not defined

|  | **mpg** | **cylinders** | **displacement** | **horsepower** | **weight** | **acceleration** | **model year** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 18.0 | 8 | 307.0 | 130 | 3504 | 12.0 | 70 |
| 1 | 15.0 | 8 | 350.0 | 165 | 3693 | 11.5 | 70 |
| 2 | 18.0 | 8 | 318.0 | 150 | 3436 | 11.0 | 70 |
| 3 | 16.0 | 8 | 304.0 | 150 | 3433 | 12.0 | 70 |
| 4 | 17.0 | 8 | 302.0 | 140 | 3449 | 10.5 | 70 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 393 | 27.0 | 4 | 140.0 | 86 | 2790 | 15.6 | 82 |
| 394 | 44.0 | 4 | 97.0 | 52 | 2130 | 24.6 | 82 |
| 395 | 32.0 | 4 | 135.0 | 84 | 2295 | 11.6 | 82 |
| 396 | 28.0 | 4 | 120.0 | 79 | 2625 | 18.6 | 82 |
| 397 | 31.0 | 4 | 119.0 | 82 | 2720 | 19.4 | 82 |

398

12. Write a python program to implement k-nearest Neighbors ML algorithm to build prediction model (Use Forge Dataset)

# 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)')

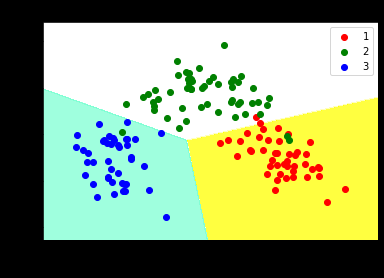
plt.xlabel('PC1') # for Xlabel

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)')

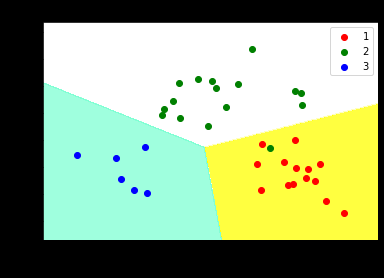
plt.xlabel('PC1') # for Xlabel

plt.ylabel('PC2') # for Ylabel

plt.legend()

# show scatter plot

plt.show()



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