**Vivian Joseph**

**(20BCE0777)**

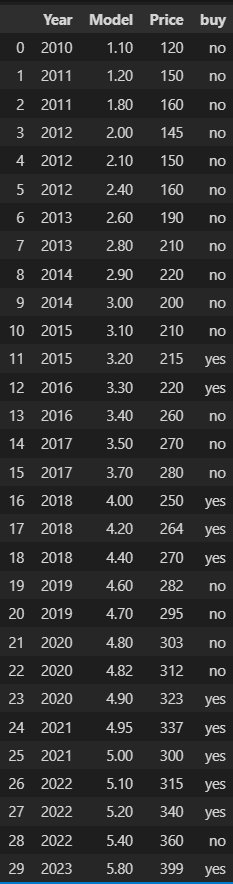
**Machine Learning**

**Lab Da-3**

**Single Layer Perceptron**

**and KNN Classifier**

**Dataset:**

****

**Q1) Apply KNN classification on the dataset and visualize the same**

**Making a function to check the distances of an input point from all points in the dataset**

def knn(year,price):

    df['diff1']=df['Year']-year

    df['diff2']=df['Price']-price

    df['d1sqr']=df['diff1']\*\*2

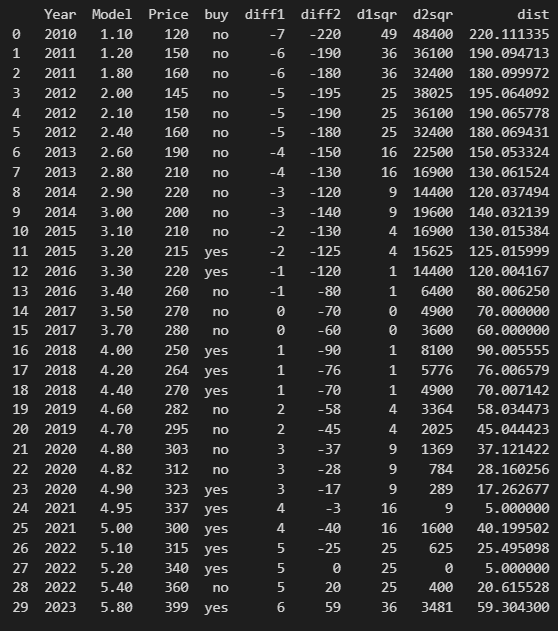
    df['d2sqr']=df['diff2']\*\*2

    df['dist']=np.sqrt(df['d2sqr']+df['d1sqr'])

    print(df)

**eg:**

knn(2017,340)

****

**Make function that take input values and k value and gives the classification of that point according to k nearest neighbours**

def knn\_classify(year,price,k):

    df['diff1']=df['Year']-year

    df['diff2']=df['Price']-price

    df['d1sqr']=df['diff1']\*\*2

    df['d2sqr']=df['diff2']\*\*2

    df['distance']=np.sqrt(df['d2sqr']+df['d1sqr'])

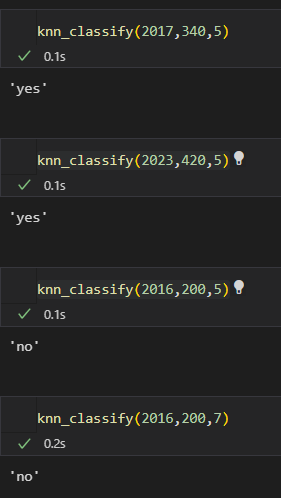
    top\_k = df.nsmallest(k, 'distance')

    prediction = top\_k['buy'].value\_counts().head(1).index[0]

    return prediction

**returns yes/no**

knn\_classify(2017,340,5)

****

**Plotting the dataset and plotting regions according to what classification they would give**

k=5

xx, yy = np.meshgrid(np.arange(2009, 2024, 0.5), np.arange(100, 450, 1))

zz = np.array([knn\_classify(x, y,k)=='yes' for x, y in zip(xx.ravel(), yy.ravel())])

zz = zz.reshape(xx.shape)



# print(zz)

fig, ax = plt.subplots()

ax.contourf(xx, yy, zz, cmap='coolwarm', alpha=0.5)

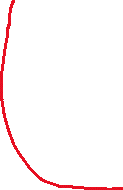
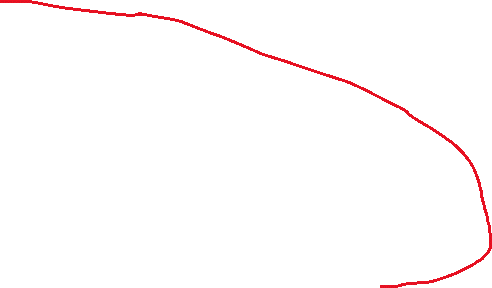


colors = {'yes': 'red', 'no': 'blue'}

markers = {'yes': 'o', 'no': 's'}

for i, row in df.iterrows():

    ax.scatter(row['Year'], row['Price'], c=colors[knn\_classify(row['Year'], row['Price'],k)], marker=markers[knn\_classify(row['Year'], row['Price'],k)], s=30)

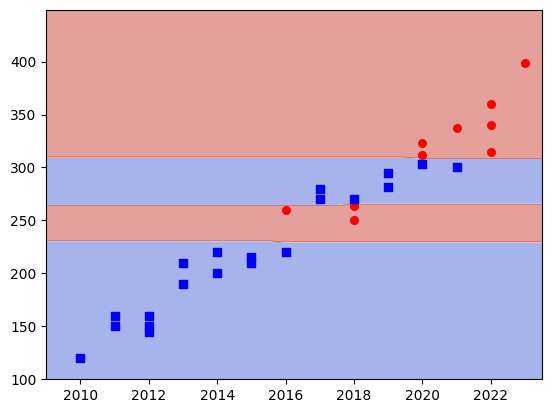


plt.show()

plot the contour

Make contours according to classification and color them accordingly

**Red=”Yes” Blue=”No”**

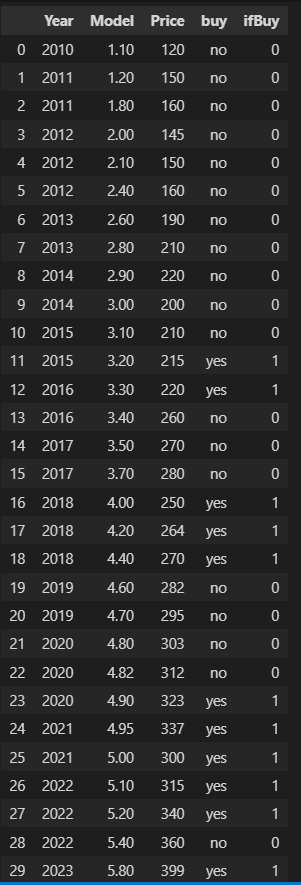
****



**Q2) Pass the dataset through a single layer perceptron to find the weights of each feature and the bias**

**Converting class labels “yes” and “no” to ordinal 0 and 1**

df['ifBuy']=(df['buy'].map({'yes':1,'no':0}))

****

**Running a single layer perceptron with all weights starting at 1, alpha as 0.005 an for 10000 epochs**

w0=1

w1=1



w2=1

w3=1

alpha=0.005

epochs=10000

for i in range(0,epochs):

    for j in range(0,df.shape[0]):

        sum=w0\*(-100)+w1\*df['Year'][j]+w2\*df['Model'][j]+w3\*df['Price'][j]

        if sum>=0:

            pred=1



        else:

            pred=0

        delta=alpha\*(df['ifBuy'][j]-pred)

        w0=w0-delta

        w1=w1+delta\*df['Year'][j]

        w2=w2+delta\*df['Model'][j]



        w3=w3+delta\*df['Price'][j]



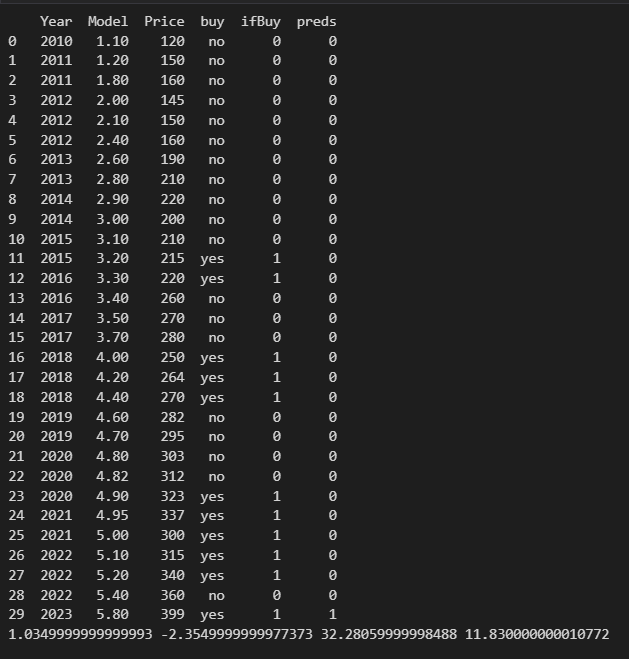
df['preds']=w0+w1\*df['Year']+w2\*df['Model']+w3\*df['Price']



df['preds']=(df['preds'] >=0).astype(int)

print(df)

print(w0,w1,w2,w3)

****



**Clearly, a single layer perceptron provides a very bad fit for the given dataset even after 10000 epochs, as it has 3 features and 30 samples.**

**More layers would be required to get any substantial fit in a neural network.**