# DML

|  |  |
| --- | --- |
| Sr. No | List of Practical Experiments |
| 1 | Write a program to implement Simple Linear Regression |
| 2 | Write a program to implement multiple Linear Regression |
| 3 | Write a program to implement K-nearest Neighbors (K-NN)/SVM |
| 4 | Write a program to implement Naïve Bayse / DT |
| 5 | Write a program to implement K-means clustering. |
| 6 | Write a program to implement Hierarchical clustering. |
| 7 | Write a program to build ANN. |
| 8 | Write a program to build CNN. |
| 9 |  |
| 10 |  |

**Practical 1**

**Aim:** Write a program to implement Simple Linear Regression

# Code:

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

dataset = pd.read\_csv("Salary\_Data.csv")

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

print(X) print("\n\n\n") print(y)

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/4, random\_state = 0)

print(X\_train) print("\n\n\n") print(X\_test) print("\n\n\n") print(y\_train) print("\n\n\n") print(y\_test)

from sklearn.linear\_model import LinearRegression linear\_regression = LinearRegression() linear\_regression.fit(X\_train, y\_train)

y\_train\_pred = linear\_regression.predict(X\_train) y\_test\_pred = linear\_regression.predict(X\_test)

plt.scatter(X\_train, y\_train, color = "green", marker = "+", label = "Observed data") plt.plot(X\_train, y\_train\_pred, color = "red", label = "Predicted data") plt.xlabel("Years of experience")

plt.ylabel("Salary")

plt.title("Years of experience v/s Salary (Training dataset)") plt.legend()

plt.show()

plt.scatter(X\_test, y\_test, color = "green", marker = "+", label = "Observed data") plt.plot(X\_test, y\_test\_pred, color = "red", label = "Predicted data") plt.xlabel("Years of experience")

plt.ylabel("Salary")

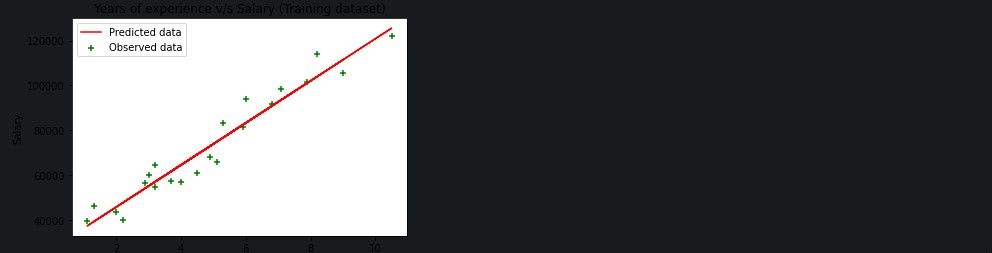
plt.title("Years of experience v/s Salary (Testing dataset)")

plt.legend() plt.show()

# Output:









**Practical 2**

**Aim:** Write a program to implement multiple Linear Regression

# Code:

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

dataset = pd.read\_csv('/content/50\_Startups-2.csv') x = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values print(x)

print(y)

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)

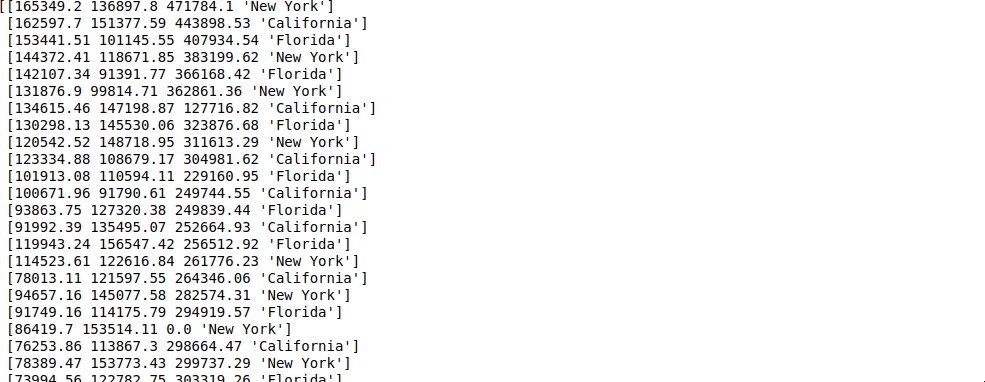
from sklearn.model\_selection import train\_test\_split

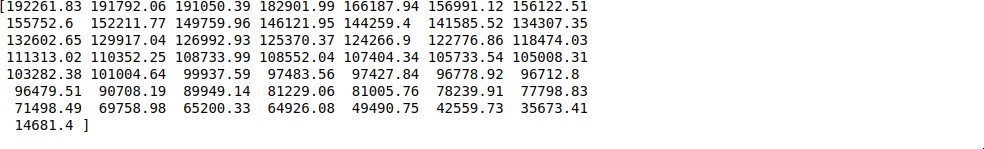
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=0)

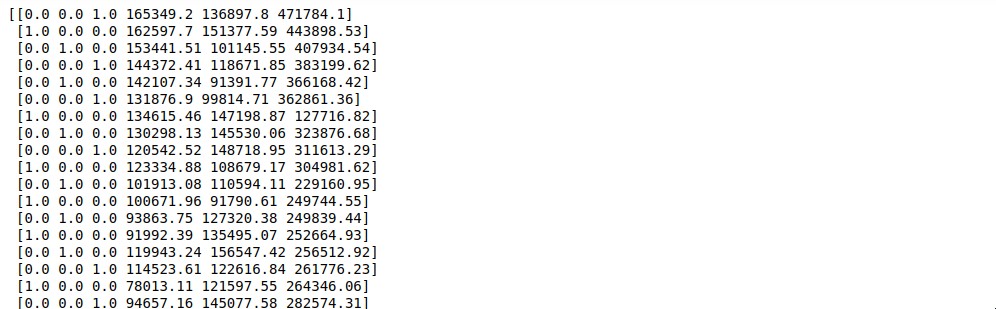
from sklearn.linear\_model import LinearRegression regressor = LinearRegression() regressor.fit(x\_train, y\_train)

y\_pred = regressor.predict(x\_test) np.set\_printoptions(precision=2)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))









# Practical 3

**Aim:** Write a program to implement K-nearest Neighbors (K-NN)/SVM

# Code:

import matplotlib.pyplot as plt import numpy as np

import pandas as pd

dataset = pd.read\_csv('/content/Social\_Network\_Ads.csv') x = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values print(x)

print(y)

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) print(x\_train)

print(y\_train) print(x\_test) print(y\_test)

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

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

print(x\_train) print(x\_test)

from sklearn.svm import SVC

classifier = SVC(kernel='linear', random\_state=0) classifier.fit(x\_train, y\_train)

print(classifier.predict(sc.transform([[30,200000]]))) y\_pred = classifier.predict(x\_test)

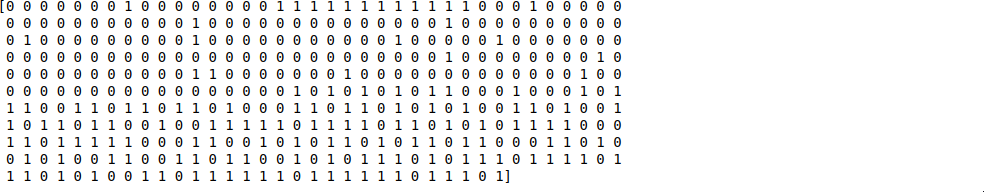
print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

from sklearn.metrics import confusion\_matrix, accuracy\_score cm = confusion\_matrix(y\_test, y\_pred)

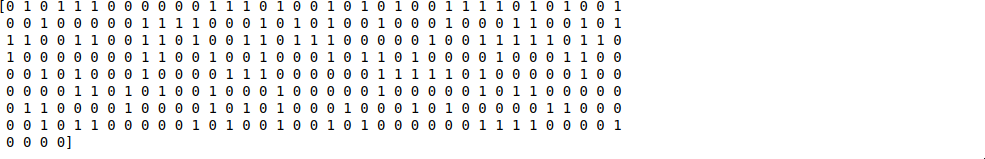
print(cm) accuracy\_score(y\_test, y\_pred)

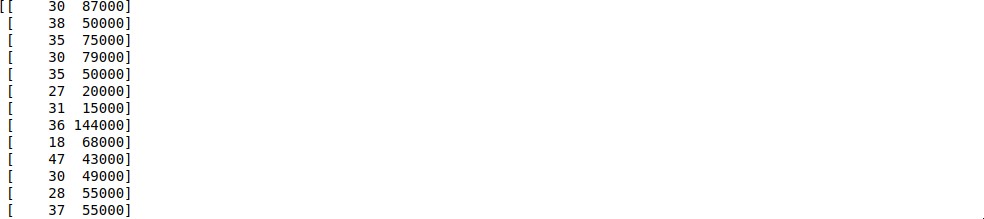
# Output:





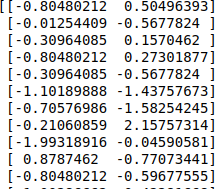




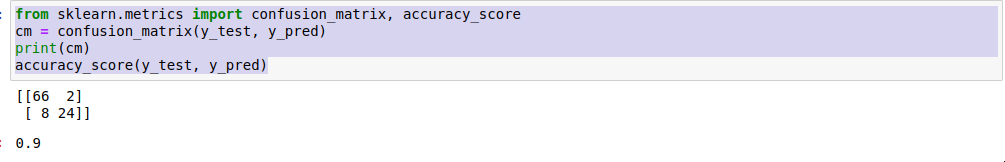












**Practical 4**

**Aim:** Write a program to implement Naïve Bayse / DT

# Code:

import matplotlib.pyplot as plt import pandas as pd

import numpy as np

dataset = pd.read\_csv('/content/Social\_Network\_Ads.csv') x = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values print(x)

print(y)

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) print(x\_train)

print(y\_train) print(x\_test) print(y\_test)

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

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

print(x\_train) print(x\_test)

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=5, metric='minkowski', p=2) classifier.fit(x\_train, y\_train)

print(classifier.predict(sc.transform([[40, 200000]]))) y\_pred = classifier.predict(x\_test)

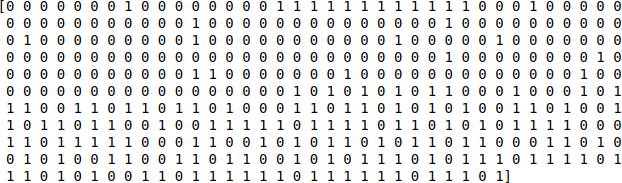
print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1)) from sklearn.metrics import confusion\_matrix, accuracy\_score

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

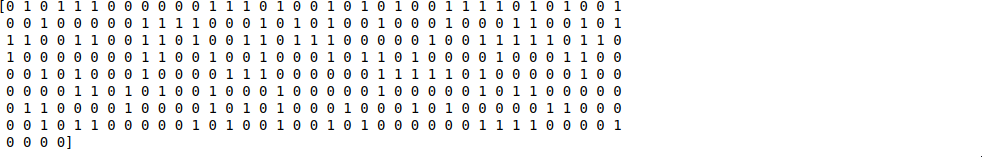
accuracy\_score(y\_test, y\_pred)

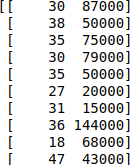
# Output:







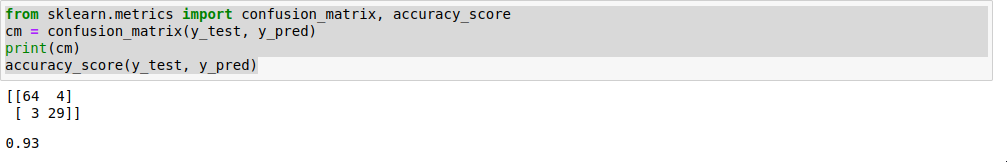












**Practical 5**

**Aim:** Write a program to implement K-means clustering.

# Code:

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

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

X = dataset.iloc[:, [3,4]].values print(X)

from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n\_clusters=5, affinity='euclidean', linkage='ward') y\_hc = hc.fit\_predict(X)

print(y\_hc)

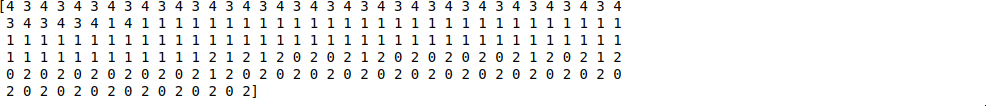
plt.scatter(X[y\_hc==0,0], X[y\_hc==0,1], s=100, c='red', label='Cluster 1') plt.scatter(X[y\_hc==1,0], X[y\_hc==1,1], s=100, c='green', label='Cluster 2') plt.scatter(X[y\_hc==2,0], X[y\_hc==2,1], s=100, c='blue', label='Cluster 3') plt.scatter(X[y\_hc==3,0], X[y\_hc==3,1], s=100, c='cyan', label='Cluster 4') plt.scatter(X[y\_hc==4,0], X[y\_hc==4,1], s=100, c='magenta', label='Cluster 5') plt.title('Clusters of customers')

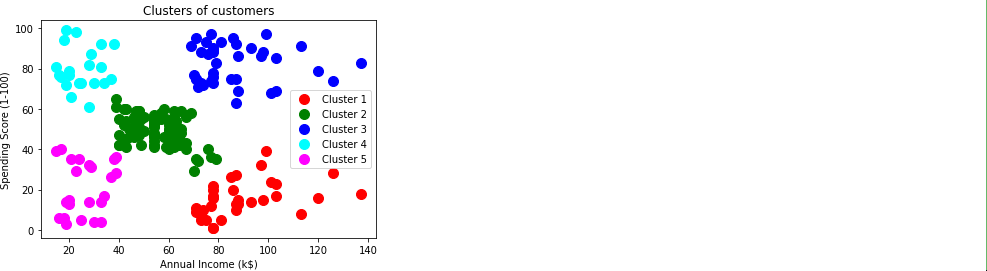
plt.xlabel('Annual Income (k$)') plt.ylabel('Spending Score (1-100)') plt.legend()

plt.show()

# Output:







**Practical 6**

**Aim:** Write a program to implement Hierarchical clustering.

# Code:

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

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

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

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters = 5, init = 'k-means++', random\_state = 42) y\_kmeans = kmeans.fit\_predict(X)

print(y\_kmeans)

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

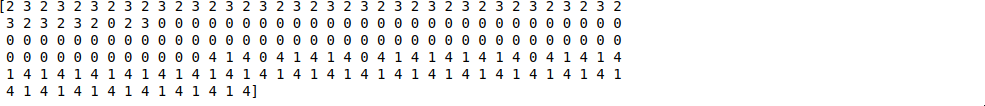
plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

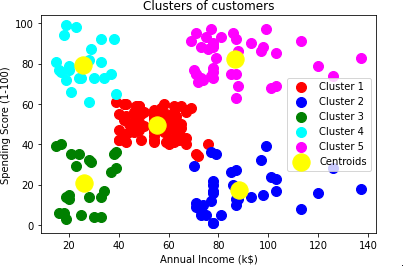
plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

plt.title('Clusters of customers') plt.xlabel('Annual Income (k$)') plt.ylabel('Spending Score (1-100)') plt.legend()

plt.show()

# Output:



**Practical 7 ** **Aim:** Write a program to build ANN.

# Code:

import numpy as np import pandas as pd import tensorflow as tf

dataset = pd.read\_csv('Churn\_Modelling.csv')

X = dataset.iloc[:, 3:-1].values y = dataset.iloc[:, -1].values

print(X) print(y)

from sklearn.preprocessing import LabelEncoder le = LabelEncoder()

X[:, 2] = le.fit\_transform(X[:, 2]) print(X)

from sklearn.compose import ColumnTransformer from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')

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

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)

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

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

ann = tf.keras.models.Sequential() ann.add(tf.keras.layers.Dense(units=6, activation='relu')) ann.add(tf.keras.layers.Dense(units=6, activation='relu')) ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

ann.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy']) ann.fit(X\_train, y\_train, batch\_size = 32, epochs = 100) print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])) >0.5)

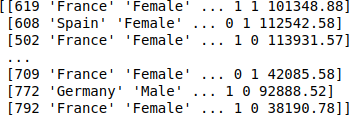
y\_pred = ann.predict(X\_test) y\_pred = (y\_pred > 0.5)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

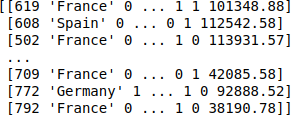
from sklearn.metrics import confusion\_matrix, accuracy\_score cm = confusion\_matrix(y\_test, y\_pred)

print(cm) accuracy\_score(y\_test, y\_pred)

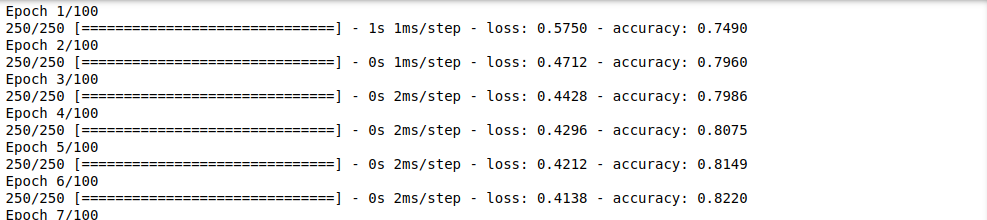
# Output:

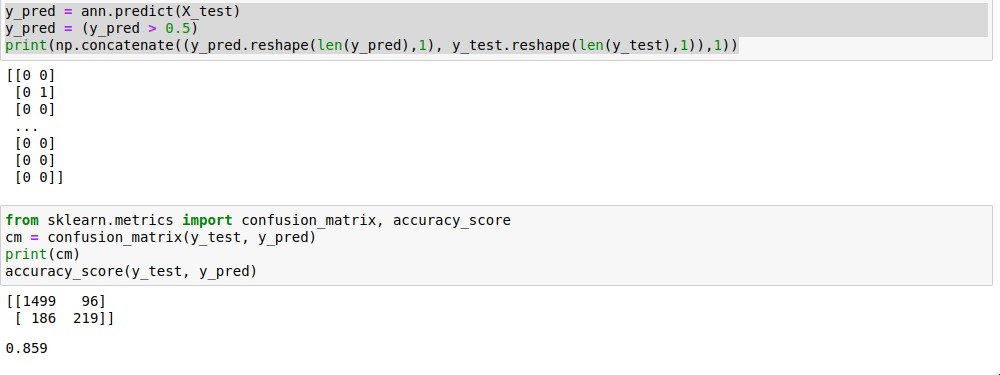












**Practical 8**

**Aim:** Write a program to build CNN.

# Code:

import tensorflow as tf

from keras.preprocessing.image import ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale=1./255, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True)

training\_set = train\_datagen.flow\_from\_directory('/content/drive/MyDrive/small\_dataset/training\_set', target\_size=(64,64), batch\_size=32, class\_mode='binary')

train\_datagen = ImageDataGenerator(rescale=1./255, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True)

test\_set = train\_datagen.flow\_from\_directory('/content/drive/MyDrive/small\_dataset/test\_set', target\_size=(64,64), batch\_size=32, class\_mode='binary')

cnn = tf.keras.models.Sequential()

cnn.add(tf.keras.layers.Conv2D(filters=32, kernel\_size=3, activation='relu', input\_shape=[64,64,3]))

cnn.add(tf.keras.layers.MaxPool2D(pool\_size=2, strides=2))

cnn.add(tf.keras.layers.Conv2D(filters=32, kernel\_size=3, activation='relu')) cnn.add(tf.keras.layers.MaxPool2D(pool\_size=2, strides=2))

cnn.add(tf.keras.layers.Flatten()) cnn.add(tf.keras.layers.Dense(units=128, activation='relu')) cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

cnn.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy']) cnn.fit(x=training\_set, validation\_data=test\_set, epochs=25)

import numpy as np

from keras.preprocessing import image test\_image=image.load\_img('/content/drive/MyDrive/small\_dataset/single\_prediction/cat\_or\_dog\_1

.jpg', target\_size=(64,64)) test\_image=image.img\_to\_array(test\_image) test\_image=np.expand\_dims(test\_image, axis=0) result=cnn.predict(test\_image) training\_set.class\_indices

if result[0][0]==1: prediction='dog' else: prediction='cat'

print(prediction)

# Output:

