```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline



HouseDF = pd.read_csv('USA_Housing.csv')
HouseDF.head()
```

```
HouseDF.info()
```

```
HouseDF.describe()
```

```
HouseDF.columns
```

```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

```
sns.pairplot(HouseDF)
```

```
sns.distplot(HouseDF['Price'])
```

```
sns.heatmap(HouseDF.corr(),annot=True)
```

```
X = HouseDF[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
               'Avg. Area Number of Bedrooms', 'Area Population']]

y = HouseDF['Price']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=101)
```

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
```

```
print(lm.intercept_)
```

```
-2640159.7968526953
```

```
coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
coeff_df
```

```
predictions = lm.predict(X_test)
plt.scatter(y_test,predictions)
```

```
sns.distplot((y_test-predictions),bins=50);
```

```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 82288.22251914942
MSE: 10460958907.208977
RMSE: 102278.82922290897
```

Task_2

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("Mall_Customers.csv")
data.head()
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   CustomerID              200 non-null    int64 
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64 
 3   Annual Income (k$)      200 non-null    int64 
 4   Spending Score (1-100)  200 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
data.describe()
```

```
data.columns
```

```
Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)'],
      dtype='object')
```

```
data.duplicated().sum()
```

```
0
```

```
data.isnull().sum()
```

```
X = data.iloc[:, [3, 4]].values


from sklearn.cluster import KMeans
# Find optimal number of clusters using the elbow method

wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)


plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')  # Within cluster sum of squares
plt.show()
```

```
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42)
y_kmeans = kmeans.fit_predict(X)


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

Task_3

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle
```

```
cp: cannot stat 'kaggle.json': No such file or directory
```

```
!kaggle datasets download -d stefancomanita/cats-and-dogs-40
```

```
Dataset URL: https://www.kaggle.com/datasets/stefancomanita/cats-and-dogs-40
License(s): CC0-1.0
Downloading cats-and-dogs-40.zip to /content
  0% 0.00/445k [00:00<?, ?B/s]
100% 445k/445k [00:00<00:00, 97.9MB/s]
```

```
import zipfile
zip_ref=zipfile.ZipFile('/content/cats-and-dogs-40.zip','r')
```

```
zip_ref.extractall('/content')
zip_ref.close()


import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten


from skimage.io import imread
from skimage.transform import resize
import os
import numpy as np

Categories = ['cat', 'dog']
flat_data_arr = []  # input array
target_arr = []  # output array

# Specify the main directory containing both training and testing subdirectories
main_datadir = 'catsAndDogs40/'

# Iterate through both training and testing categories
for category in Categories:
    print(f'loading... category: {category}')

    # Specify the subdirectories for training and testing
    train_or_test_dirs = ['train', 'test']

    for train_or_test in train_or_test_dirs:
        # Form the complete path
        path = os.path.join(main_datadir, train_or_test, category)

        for img in os.listdir(path):
            img_array = imread(os.path.join(path, img))
            img_resized = resize(img_array, (40, 40, 3))  # Adjust the size as needed
            flat_data_arr.append(img_resized.flatten())
            target_arr.append(Categories.index(category))

        print(f'loaded {train_or_test} category: {category} successfully')

flat_data = np.array(flat_data_arr)
target = np.array(target_arr)
```

```
loading... category: cat
loaded train category: cat successfully
loaded test category: cat successfully
loading... category: dog
loaded train category: dog successfully
loaded test category: dog successfully
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
df=pd.DataFrame(flat_data)
```

```
df['Target']=target
df.shape
```

```
(80, 4801)
```

```
#input data
x=df.iloc[:,:-1]
#output data
y=df.iloc[:,-1]
```

```
# Splitting the data into training and testing sets
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,
                                               random_state=77,
                                               stratify=y)
```

```
# Defining the parameters grid for GridSearchCV
param_grid={'C':[0.1,1,10,100],
            'gamma':[0.0001,0.001,0.1,1],
            'kernel':['rbf','poly']}

# Creating a support vector classifier
svc=svm.SVC(probability=True)

# Creating a model using GridSearchCV with the parameters grid
model=GridSearchCV(svc,param_grid)
```

```
model.fit(x_train,y_train)
```

```
# Testing the model using the testing data
y_pred = model.predict(x_test)

# Calculating the accuracy of the model
accuracy = accuracy_score(y_pred, y_test)

# Print the accuracy of the model
print(f"The model is {accuracy*100}% accurate")
```

```
The model is 50.0% accurate
```

```
print(classification_report(y_test, y_pred, target_names=['cat', 'dog']))
```

```
              precision    recall  f1-score   support

         cat       0.50      0.88      0.64         8
         dog       0.50      0.12      0.20         8

    accuracy                           0.50        16
```

```
       macro avg       0.50      0.50      0.42        16
    weighted avg       0.50      0.50      0.42        16
```

```
path='catsAndDogs40/test/cat/5.jpg'
img=imread(path)
plt.imshow(img)
plt.show()
img_resize=resize(img,(40,40,3))
l=[img_resize.flatten()]
print("The predicted image is : "+Categories[model.predict(l)[0]])
```

```
path = 'catsAndDogs40/test/dog/2.jpg'
img = imread(path)
plt.imshow(img)
plt.axis('off')  # Hide axes
plt.show()

# Resize and preprocess the image
img_resize = resize(img, (40, 40, 3))  # Ensure the shape is (40,40,3)
img_resize = img_resize.flatten().reshape(1, -1)  # Flatten and reshape for prediction

# Make prediction
prediction = model.predict(img_resize)
predicted_class = np.argmax(prediction)  # Get the class index

# Display predicted class
print(f"The predicted image is: {Categories[predicted_class]}")
```

Task_4

Task_5

```
import os
import torch
import pandas as pd
import numpy as np
from sklearn.utils import shuffle
from PIL import Image
import torchvision
from torchvision import datasets, transforms
from torch.utils.data import Dataset, DataLoader
import requests as reqs
import torch.nn as nn
from torchvision import models
from tqdm import tqdm
from torch.utils.checkpoint import checkpoint  # For gradient checkpointing
from torch.cuda.amp import autocast, GradScaler  # For mixed-precision trainin
import matplotlib.pyplot as plt
import torch.utils.checkpoint as checkpoint
from albumentations.pytorch import ToTensorV2
import torch.optim.lr_scheduler as lr_scheduler
import torch.optim as optim
import zipfile
import urllib.request
import shutil
import random
from torch.optim.lr_scheduler import CosineAnnealingLR, SequentialLR, LinearLR
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
```

```
# Setting the device for PyTorch; use CUDA if available, else CPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cpu
```

```
classes = open("food-101/meta/classes.txt", 'r').read().splitlines()
classes_21 = classes[:20] + ['other']
classes_21, len(classes_21)

# Defining a custom label encoder for the classes
class Label_encoder:
    def __init__(self, labels):
        self.labels = {label: idx for idx, label in enumerate(labels)}
    def get_label(self, idx):
        return list(self.labels.keys())[idx]
    def get_idx(self, label):
        return self.labels.get(label)

# Initializing label encoder with 21 classes and testing its functionality
encoder_21 = Label_encoder(classes_21)
encoder_21.get_label(0), encoder_21.get_idx( encoder_21.get_label(0) )

# Printing each class with its corresponding index
for i in range(21):
    print(encoder_21.get_label(i), encoder_21.get_idx( encoder_21.get_label(i) ))

# Defining a custom dataset class for handling image data
class Food21(Dataset):
```