

CL-4 Assignment 1

Linear Regression Machine Learning Project for House Price Prediction

Import Libraries

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

%matplotlib inline
```

Importing Data and Checking out.

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

```
HouseDF.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	\
0	79545.458574	5.682861	7.009188	
1	79248.642455	6.002900	6.730821	
2	61287.067179	5.865890	8.512727	
3	63345.240046	7.188236	5.586729	
4	59982.197226	5.040555	7.839388	

	Avg. Area Number of Bedrooms	Area Population	Price \
0	4.09	23086.800503	1.059034e+06
1	3.09	40173.072174	1.505891e+06
2	5.13	36882.159400	1.058988e+06
3	3.26	34310.242831	1.260617e+06
4	4.23	26354.109472	6.309435e+05

	Address
0	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	USS Barnett\nFPO AP 44820
4	USNS Raymond\nFPO AE 09386


```
HouseDF.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
```

```
Data columns (total 7 columns):
```

Avg. Area Income	5000	non-null	float64
Avg. Area House Age	5000	non-null	float64
Avg. Area Number of Rooms	5000	non-null	float64
Avg. Area Number of Bedrooms	5000	non-null	float64
Area Population	5000	non-null	float64
Price	5000	non-null	float64
Address	5000	non-null	object

dtypes: float64(6), object(1)
memory usage: 273.6+ KB

```
HouseDF.describe()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms
count	5000.000000	5000.000000	5000.000000
mean	68583.108984	5.977222	6.987792
std	10657.991214	0.991456	1.005833
min	17796.631190	2.644304	3.236194
25%	61480.562388	5.322283	6.299250
50%	68804.286404	5.970429	7.002902
75%	75783.338666	6.650808	7.665871
max	107701.748378	9.519088	10.759588

	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5.000000e+03
mean	3.981330	36163.516039	1.232073e+06
std	1.234137	9925.650114	3.531176e+05
min	2.000000	172.610686	1.593866e+04
25%	3.140000	29403.928702	9.975771e+05
50%	4.050000	36199.406689	1.232669e+06
75%	4.490000	42861.290769	1.471210e+06
max	6.500000	69621.713378	2.469066e+06

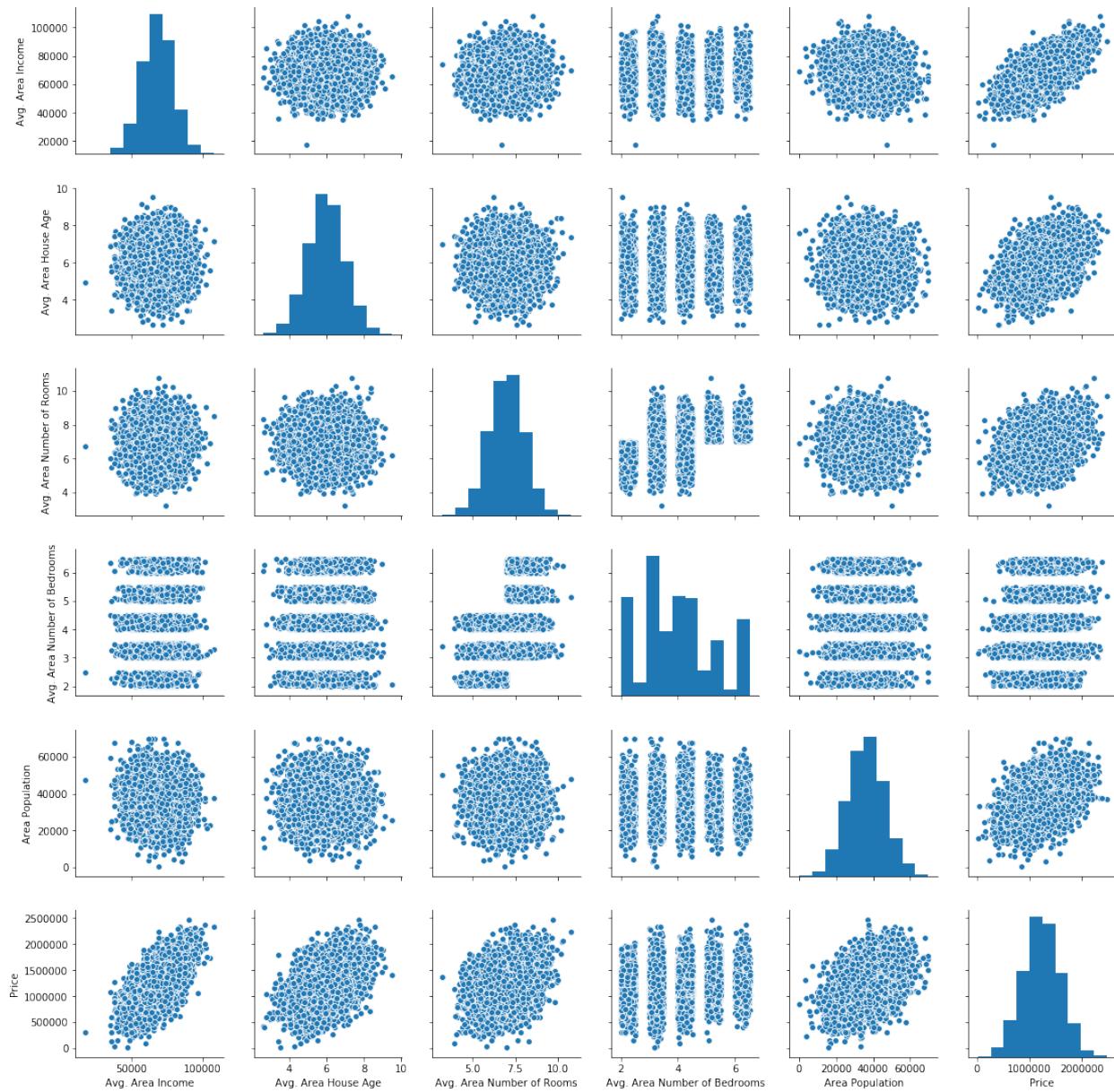
```
HouseDF.columns
```

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

Exploratory Data Analysis for House Price Prediction

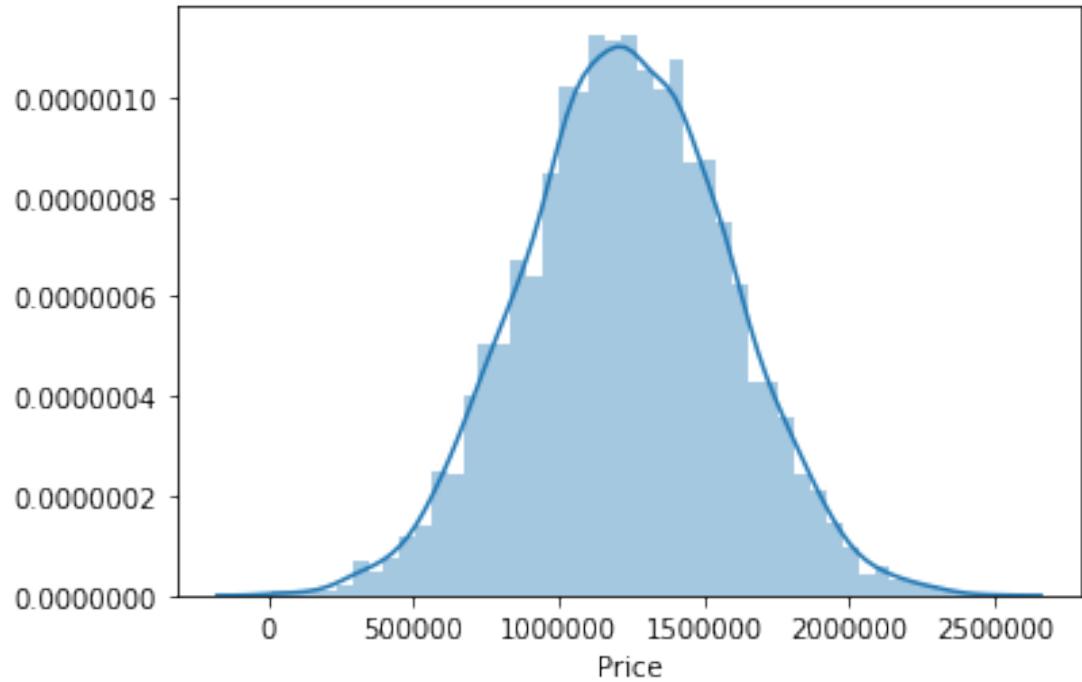
```
sns.pairplot(HouseDF)
```

```
<seaborn.axisgrid.PairGrid at 0x67b60e8d08>
```



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

```
<matplotlib.axes._subplots.AxesSubplot at 0x67b80b5648>
```



```
sns.heatmap(HouseDF.corr(), annot=True)  
<matplotlib.axes._subplots.AxesSubplot at 0x67b868c948>
```



Training a Linear Regression Model

X and y List

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

Split Data into Train, Test

```
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)
```

Creating and Training the LinearRegression Model

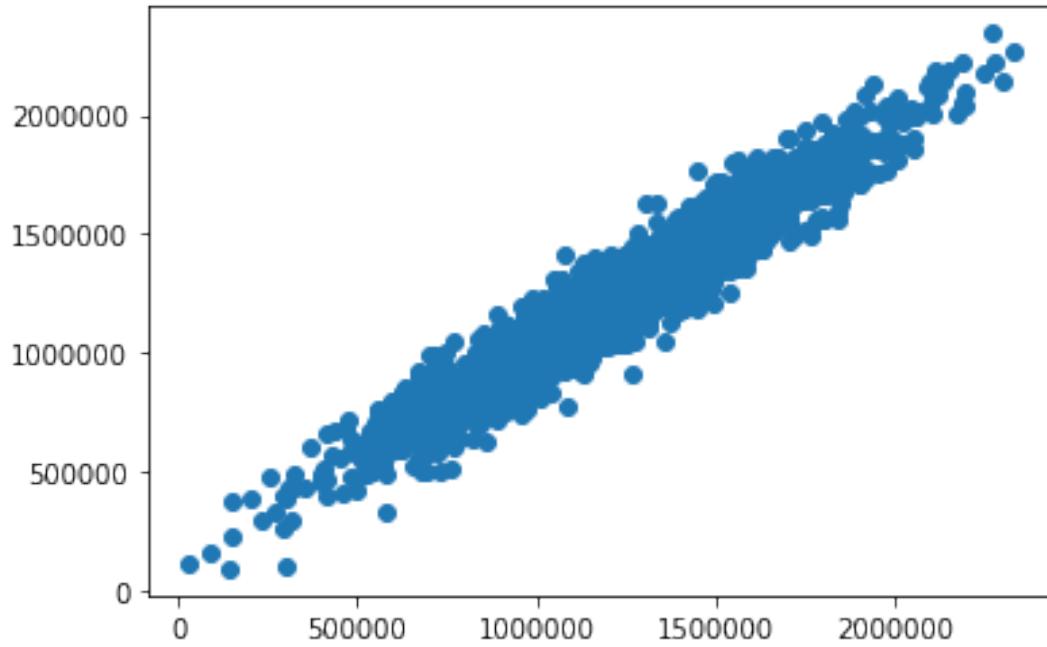
```
from sklearn.linear_model import LinearRegression  
  
lm = LinearRegression()  
  
lm.fit(X_train,y_train)  
  
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,  
normalize=False)
```

LinearRegression Model Evaluation

```
print(lm.intercept_)  
  
-2640159.796851911  
  
coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])  
coeff_df  
  
Avg. Area Income           Coefficient  
Avg. Area House Age        21.528276  
Avg. Area Number of Rooms  164883.282027  
Avg. Area Number of Bedrooms 122368.678027  
Area Population             2233.801864
```

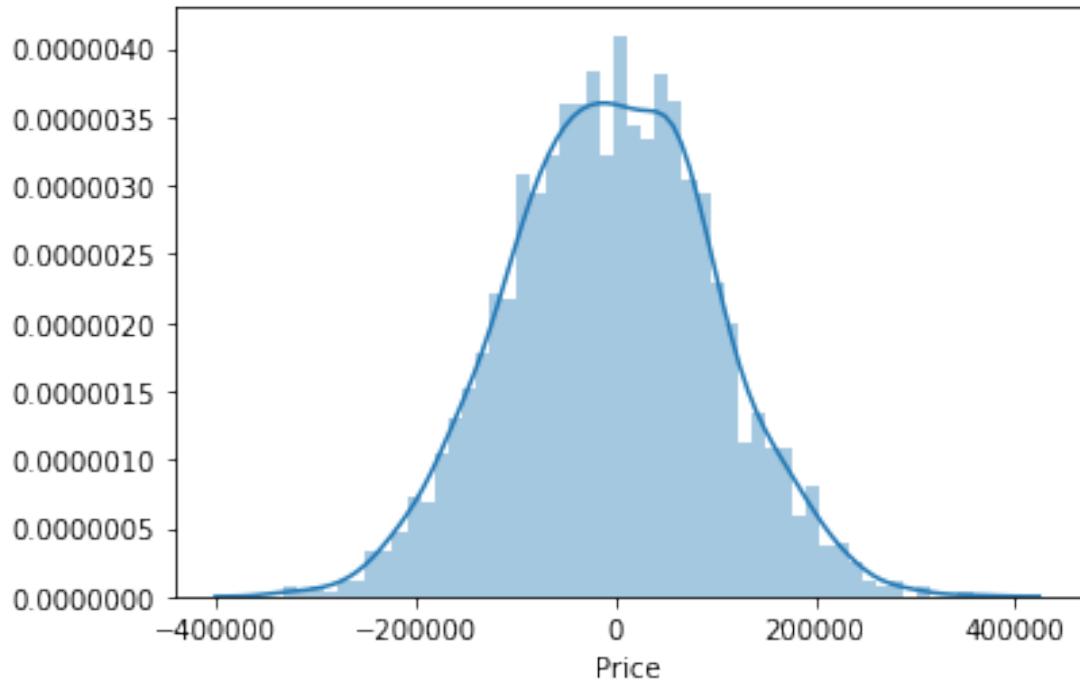
Predictions from our Linear Regression Model

```
predictions = lm.predict(X_test)  
  
plt.scatter(y_test,predictions)  
  
<matplotlib.collections.PathCollection at 0x67b87ccc88>
```



In the above scatter plot, we see data is in line shape, which means our model has done good predictions.

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



In the above histogram plot, we see data is in bell shape (Normally Distributed), which means our model has done good predictions.

Regression Evaluation Metrics

```
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: 82288.22251914957
MSE: 10460958907.209501
RMSE: 102278.82922291153
```

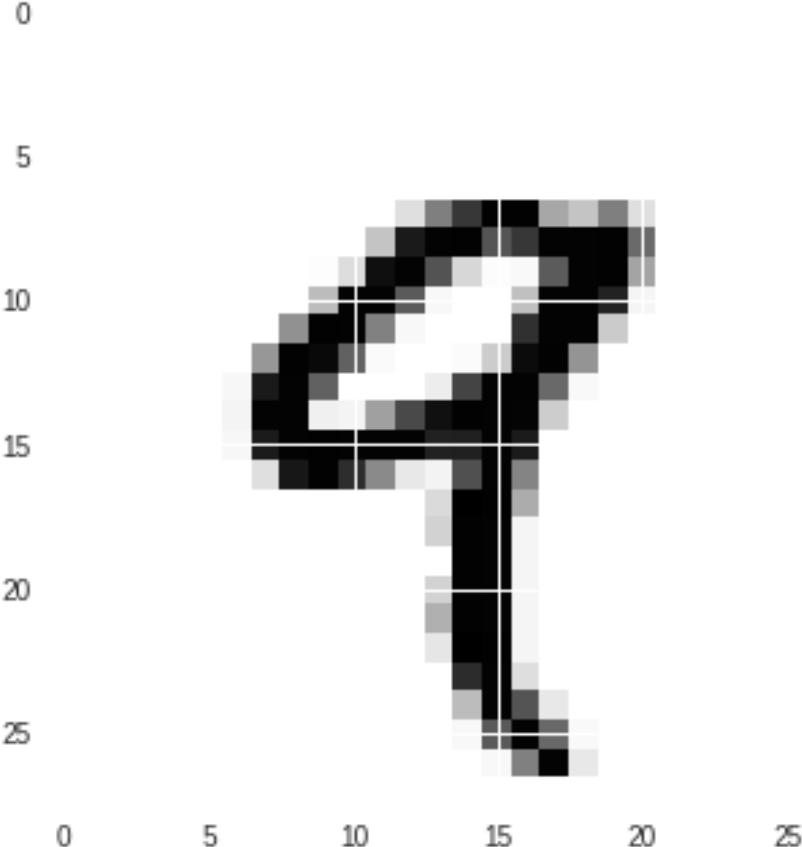
CL-4 Assignment 2

```
#importing the necessary modules
from keras.datasets import mnist
import numpy as np

#loading data
(xtrain,ytrain),(xtest,ytest)=mnist.load_data()

#What mnist images look like
import matplotlib.pyplot as plt
print("Training data:")
plt.imshow(xtrain[4])
plt.show()
print("Label of this image is",ytrain[4])

Training data:
```



```
Label of this image is 9
```

```
#reshaping data as needed by the model
xtrain=np.reshape(xtrain,(-1,28,28,1))
```

```

xtest=np.reshape(xtest,(-1,28,28,1))
xtrain.shape,xtest.shape,ytrain.shape,ytest.shape

((60000, 28, 28, 1), (10000, 28, 28, 1), (60000,), (10000,))

#normalising
xtrain=xtrain/255
xtest=xtest/255

#implementing one hot encoding
from keras.utils.np_utils import to_categorical
y_train = to_categorical(ytrain, num_classes=10)
y_test = to_categorical(ytest, num_classes=10)

#importing the model
from keras.models import Sequential

#creating model object
model=Sequential()

#importing layers
from keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout

#adding layers and forming the model
model.add(Conv2D(32,kernel_size=5,strides=1,padding="Same",activation=
"relu",input_shape=(28,28,1)))
model.add(MaxPooling2D(padding="same"))

model.add(Conv2D(64,kernel_size=5,strides=1,padding="same",activation=
"relu"))
model.add(MaxPooling2D(padding="same"))

model.add(Flatten())

model.add(Dense(1024,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(10,activation="sigmoid"))

#compiling
model.compile(optimizer="adam",loss="categorical_crossentropy",metrics
=[ "accuracy"])

#training the model
model.fit(xtrain,y_train,batch_size=100,epochs=5,validation_data=(xtes
t,y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [=====] - 14s 230us/step - loss:
0.1528 - acc: 0.9534 - val_loss: 0.0411 - val_acc: 0.9865
Epoch 2/5
60000/60000 [=====] - 10s 163us/step - loss:

```

```
0.0429 - acc: 0.9869 - val_loss: 0.0308 - val_acc: 0.9898
Epoch 3/5
60000/60000 [=====] - 10s 163us/step - loss:
0.0308 - acc: 0.9903 - val_loss: 0.0294 - val_acc: 0.9910
Epoch 4/5
60000/60000 [=====] - 10s 162us/step - loss:
0.0218 - acc: 0.9929 - val_loss: 0.0288 - val_acc: 0.9909
Epoch 5/5
60000/60000 [=====] - 10s 163us/step - loss:
0.0160 - acc: 0.9949 - val_loss: 0.0288 - val_acc: 0.9912
<keras.callbacks.History at 0x7f8863897e10>

#model train and test scores
model.evaluate(xtrain,y_train),model.evaluate(xtest,y_test)

60000/60000 [=====] - 6s 108us/step
10000/10000 [=====] - 1s 109us/step
([0.010291422149978097, 0.9966], [0.02878557628018607, 0.9912])
```

```

1 import pandas as pd
2 import numpy as np
3 import tensorflow as tf
4 from tensorflow.keras.preprocessing.text import Tokenizer
5 from tensorflow.keras.preprocessing.sequence import pad_sequences
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
8 from sklearn.model_selection import train_test_split
9 from sklearn.metrics import accuracy_score, classification_report
10
11 df = pd.read_csv("IMBD.csv")
12
13 # Print column names to identify the correct text and label columns
14 print("Dataset Columns:", df.columns)
15
16
17 X = df['title'].values # Change 'text' to the actual column containing reviews
18 y = df['genre'].values # Change 'label' to the actual sentiment column
19
20 # Convert sentiment labels to binary if necessary
21 y = np.where(y == "positive", 1, 0)
22
23 # Tokenization & Padding
24 tokenizer = Tokenizer(num_words=5000, oov_token='<OOV>')
25 tokenizer.fit_on_texts(X)
26 X_sequences = tokenizer.texts_to_sequences(X)
27 X_padded = pad_sequences(X_sequences, maxlen=100, padding='post',
truncating='post')
28
29 # Split Data
30 X_train, X_test, y_train, y_test = train_test_split(X_padded, y, test_size=0.2,
random_state=42)
31
32 # Define LSTM Model
33 model = Sequential([
34     Embedding(input_dim=5000, output_dim=64, input_length=100),
35     LSTM(128, return_sequences=True),
36     LSTM(64),
37     Dropout(0.5),
38     Dense(32, activation='relu'),
39     Dense(1, activation='sigmoid')
40 ])
41
42 # Compile Model
43 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=
['accuracy'])
44
45 # Train Model
46 model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test,
y_test))
47
48 # Predictions
49 y_pred = (model.predict(X_test) > 0.5).astype("int32")
50
51 # Evaluate Model
52 accuracy = accuracy_score(y_test, y_pred)
53 print(f'Accuracy: {accuracy:.4f}')
54 print(classification_report(y_test, y_pred))

```

→ Dataset Columns: Index(['title', 'year', 'certificate', 'duration', 'genre', 'rating',
 'description', 'stars', 'votes'],
 dtype='object')

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Ju
 warnings.warn(
Epoch 1/5
125/125 54s 368ms/step - accuracy: 0.9812 - loss: 0.0864 - val_accuracy: 1.0000 - val_loss: 2.8386e-06
Epoch 2/5
125/125 95s 478ms/step - accuracy: 1.0000 - loss: 1.7248e-05 - val_accuracy: 1.0000 - val_loss: 1.6532e-06
Epoch 3/5
125/125 68s 370ms/step - accuracy: 1.0000 - loss: 1.1244e-05 - val_accuracy: 1.0000 - val_loss: 1.0741e-06
Epoch 4/5
125/125 83s 377ms/step - accuracy: 1.0000 - loss: 8.7805e-06 - val_accuracy: 1.0000 - val_loss: 7.3702e-07
Epoch 5/5
125/125 83s 381ms/step - accuracy: 1.0000 - loss: 6.1761e-06 - val_accuracy: 1.0000 - val_loss: 5.3914e-07
63/63 5s 72ms/step
Accuracy: 1.0000

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1992
accuracy			1.00	1992
macro avg	1.00	1.00	1.00	1992
weighted avg	1.00	1.00	1.00	1992


```
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split

# Load and preprocess the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

x_train = x_train.astype('float32') / 255.0
x_val = x_val.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

y_train = to_categorical(y_train, 10)
y_val = to_categorical(y_val, 10)
y_test = to_categorical(y_test, 10)

# Define a function to create a CNN model
def create_cnn_model(learning_rate=0.001, filter_size=32, num_layers=3, dropout_rate=0.5, optimizer='ada'):
    model = models.Sequential()
    model.add(layers.Conv2D(filter_size, (3, 3), activation='relu', input_shape=(32, 32, 3)))
    model.add(layers.MaxPooling2D((2, 2)))

    for _ in range(num_layers - 1):
        model.add(layers.Conv2D(filter_size * 2, (3, 3), activation='relu'))
        model.add(layers.MaxPooling2D((2, 2)))
        filter_size *= 2

    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dropout(dropout_rate))
    model.add(layers.Dense(10, activation='softmax'))

    if optimizer == 'adam':
        opt = optimizers.Adam(learning_rate=learning_rate)
    elif optimizer == 'sgd':
        opt = optimizers.SGD(learning_rate=learning_rate)
    else:
        raise ValueError("Unsupported optimizer type")

    model.compile(optimizer=opt,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Create and train the model
model = create_cnn_model(learning_rate=0.001, filter_size=32, num_layers=3, dropout_rate=0.5, optimizer='ada')
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_data=(x_val, y_val))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')
```

→ Downloading data from <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>
170498071/170498071 ————— 2s 0us/step

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
625/625 68s 100ms/step - accuracy: 0.2452 - loss: 2.0125 - val_accuracy: 0.4871
Epoch 2/10
625/625 78s 94ms/step - accuracy: 0.4476 - loss: 1.5264 - val_accuracy: 0.5523
Epoch 3/10
625/625 57s 91ms/step - accuracy: 0.5139 - loss: 1.3589 - val_accuracy: 0.5578
Epoch 4/10
625/625 83s 93ms/step - accuracy: 0.5543 - loss: 1.2558 - val_accuracy: 0.6018
Epoch 5/10
625/625 84s 96ms/step - accuracy: 0.5946 - loss: 1.1465 - val_accuracy: 0.6212
Epoch 6/10
625/625 81s 95ms/step - accuracy: 0.6172 - loss: 1.0875 - val_accuracy: 0.6504
Epoch 7/10
625/625 57s 91ms/step - accuracy: 0.6460 - loss: 1.0157 - val_accuracy: 0.6658
Epoch 8/10
625/625 64s 102ms/step - accuracy: 0.6693 - loss: 0.9486 - val_accuracy: 0.6799
Epoch 9/10
625/625 77s 95ms/step - accuracy: 0.6889 - loss: 0.9056 - val_accuracy: 0.6940
Epoch 10/10
596/625 2s 87ms/step - accuracy: 0.7012 - loss: 0.8574
```



```
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f'Test accuracy: {test_acc:.4f}')

# Make predictions on the test set
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Print classification report
print("Classification Report:")
print(classification_report(y_true, y_pred_classes))

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_true, y_pred_classes))
```

→ Test accuracy: 0.6962
313/313 5s 17ms/step

Classification Report:				
	precision	recall	f1-score	support
0	0.78	0.69	0.73	1000
1	0.80	0.88	0.84	1000
2	0.56	0.59	0.57	1000
3	0.54	0.43	0.48	1000
4	0.58	0.68	0.62	1000
5	0.57	0.68	0.62	1000
6	0.79	0.76	0.77	1000
7	0.71	0.74	0.73	1000
8	0.84	0.78	0.81	1000
9	0.85	0.75	0.80	1000

accuracy			0.70	10000
macro avg	0.70	0.70	0.70	10000
weighted avg	0.70	0.70	0.70	10000

Confusion Matrix:

```
[[685  34 113  14  25   6  11  13  70  29]
 [ 17 881   9  12   9   4   5   2  17  44]
 [ 43   5 587  42 127  93  53  38   9   3]
 [ 12  11  69 426 100 258  61  47   7   9]
 [ 10   2  87  42 678  32  40  98  11   0]
 [  5   3  65 103  64 675  15  64   1   5]
 [  4   4  61  67  73  23 760   6   1   1]
 [  7   4  34  32  85  80   2 745   3   8]
 [ 75  45  20  18  11  11   7   4 776  33]
 [ 21 112  11  27   6   8  10  27  29 749]]
```

Start coding or generate with AI.

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, _), (_, _) = tf.keras.datasets.mnist.load_data()
x_train = x_train.astype('float32') / 255.0
x_train = np.expand_dims(x_train, axis=-1)

# Define the generator model
def build_generator():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7 * 7 * 256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    return model

# Define the discriminator model
def build_discriminator():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same', input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))
    return model

# Define the loss and optimizers
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)

generator = build_generator()
```

```
discriminator = build_discriminator()

generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)

# Training loop
EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16

seed = tf.random.normal([num_examples_to_generate, noise_dim])

@tf.function
def train_step(images):
    noise = tf.random.normal([images.shape[0], noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)

        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

        gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
        gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

        generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
        discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))

    return gen_loss, disc_loss

def train(dataset, epochs):
    for epoch in range(epochs):
        for image_batch in dataset:
            train_step(image_batch)

            # Produce images for the GIF as we go
            generate_and_save_images(generator, epoch + 1, seed)

    # Generate after the final epoch
    generate_and_save_images(generator, epochs, seed)

def generate_and_save_images(model, epoch, test_input):
    predictions = model(test_input, training=False)

    fig = plt.figure(figsize=(4, 4))

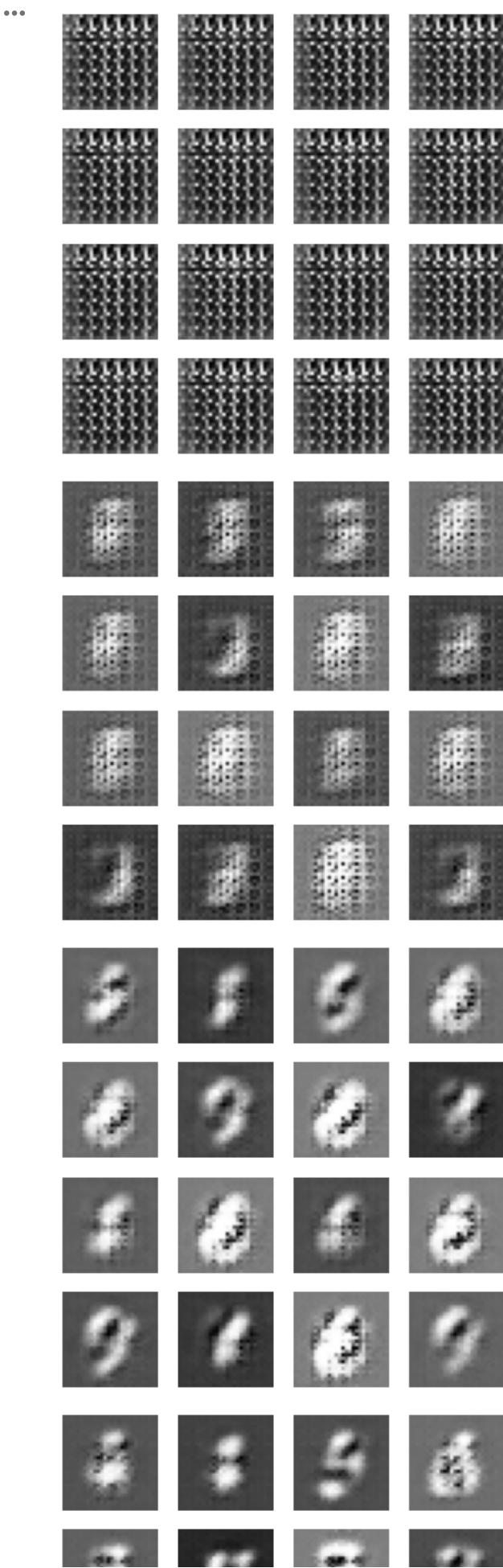
    for i in range(predictions.shape[0]):
        plt.subplot(4, 4, i + 1)
        plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
        plt.axis('off')

    plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
    plt.show()

# Prepare the dataset
BUFFER_SIZE = 60000
BATCH_SIZE = 256
```

```
train_dataset = tf.data.Dataset.from_tensor_slices(x_train).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

# Train the DCGAN
train(train_dataset, EPOCHS)
```



```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence

# Load the IMDb dataset
max_features = 10000 # Number of words to consider as features
 maxlen = 500 # Cut texts after this number of words (among top max_features most common words)

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Pad sequences to ensure uniform input size
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

# Build the RNN model
model = models.Sequential()
model.add(layers.Embedding(max_features, 128, input_length=maxlen))
model.add(layers.SimpleRNN(128, return_sequences=False))
model.add(layers.Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train,
                      epochs=10,
                      batch_size=32,
                      validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')
```

→ Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz>
17464789/17464789 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument
warnings.warn(
Epoch 1/10
625/625 ━━━━━━━━━━ 145s 228ms/step - accuracy: 0.5479 - loss: 0.6746 - val_accuracy: 0.667
Epoch 2/10
625/625 ━━━━━━ 139s 222ms/step - accuracy: 0.7430 - loss: 0.5133 - val_accuracy: 0.630
Epoch 3/10
625/625 ━━━━━━ 146s 229ms/step - accuracy: 0.7253 - loss: 0.5503 - val_accuracy: 0.658
Epoch 4/10
625/625 ━━━━━━ 144s 230ms/step - accuracy: 0.7765 - loss: 0.4676 - val_accuracy: 0.708
Epoch 5/10
625/625 ━━━━━━ 203s 233ms/step - accuracy: 0.8436 - loss: 0.3718 - val_accuracy: 0.733
Epoch 6/10
625/625 ━━━━━━ 201s 232ms/step - accuracy: 0.8782 - loss: 0.3109 - val_accuracy: 0.752
Epoch 7/10
625/625 ━━━━━━ 201s 230ms/step - accuracy: 0.8970 - loss: 0.2705 - val_accuracy: 0.738
Epoch 8/10
625/625 ━━━━━━ 203s 232ms/step - accuracy: 0.8921 - loss: 0.2701 - val_accuracy: 0.750
Epoch 9/10

```
625/625 ━━━━━━━━━━ 144s 231ms/step - accuracy: 0.9248 - loss: 0.2002 - val_accuracy: 0.763
Epoch 10/10
625/625 ━━━━━━━━━━ 201s 230ms/step - accuracy: 0.9436 - loss: 0.1607 - val_accuracy: 0.766
782/782 ━━━━━━━━ 40s 51ms/step - accuracy: 0.7638 - loss: 0.6777
Test accuracy: 0.7666
```



Start coding or generate with AI.



```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

↳ Accuracy: 1.0000
Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Confusion Matrix:

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

Start coding or generate with AI.


```

import pandas as pd

import sqlalchemy
sqlalchemy.__version__
'1.4.5'

# Connection with dataset

connection_uri = "sqlite:///Chinook.sqlite"
#connection_uri = "postgresql://user:password@localhost:5432/Chinook"

# Create engine: db_engine
db_engine = sqlalchemy.create_engine(connection_uri)

```

ETL

1. Preprocessing of Extract

```

# Names of the tables
# with sqlalchemy v1.3
# table_names = db_engine.table_names()
# table_names

# with sqlalchemy v1.4
inspector = sqlalchemy.inspect(db_engine)
table_names = inspector.get_table_names()
table_names

['Album',
 'Artist',
 'Customer',
 'Disaster_ETL2',
 'Employee',
 'Genre',
 'Invoice',
 'InvoiceLine',
 'MediaType',
 'Playlist',
 'PlaylistTrack',
 'Track']

# Open engine connection: con
con = db_engine.connect()

# Perform some query
qr_invoice = con.execute("SELECT * FROM Invoice")

type(qr_invoice)

```

```

sqlalchemy.engine.cursor.LegacyCursorResult

qr_invoice.keys()

RMKeyView(['InvoiceId', 'CustomerId', 'InvoiceDate', 'BillingAddress',
'BillingCity', 'BillingState', 'BillingCountry', 'BillingPostalCode',
'Total'])

df = pd.DataFrame(qr_invoice.fetchall())
df.columns = qr_invoice.keys()

df

      InvoiceId CustomerId      InvoiceDate \
0              1           2  2009-01-01 00:00:00
1              2           4  2009-01-02 00:00:00
2              3           8  2009-01-03 00:00:00
3              4          14  2009-01-06 00:00:00
4              5          23  2009-01-11 00:00:00
..            ...
407            408          25  2013-12-05 00:00:00
408            409          29  2013-12-06 00:00:00
409            410          35  2013-12-09 00:00:00
410            411          44  2013-12-14 00:00:00
411            412          58  2013-12-22 00:00:00

                                BillingAddress BillingCity BillingState \
\
0             Theodor-Heuss-Straße 34     Stuttgart        None
1             Ullevålsveien 14             Oslo        None
2             Grétrystraat 63         Brussels        None
3             8210 111 ST NW       Edmonton         AB
4             69 Salem Street          Boston         MA
..            ...
407          319 N. Frances Street        Madison         WI
408          796 Dundas Street West      Toronto         ON
409  Rua dos Campeões Europeus de Viena, 4350        Porto        None
410          Porthaninkatu 9        Helsinki        None
411          12,Community Centre        Delhi        None

      BillingCountry BillingPostalCode   Total

```

```

0          Germany      70174  1.98
1          Norway       0171   3.96
2          Belgium      1000   5.94
3          Canada       T6G 2C7  8.91
4          USA          2113   13.86
..
407         ..          ...    ...
408         USA          53703  3.96
409         Canada      M6J 1V1  5.94
409         Portugal     None   8.91
410         Finland     00530  13.86
411         India        110017 1.99

```

[412 rows x 9 columns]

2. Preprocessing of Transforming

2.1. Calculating average billing per Invoice by Country.

```
# Calculating average billing by Country
df_g = df.groupby(['BillingCountry'])[['Total']].mean()
df_g
```

BillingCountry	Total
Argentina	5.374286
Australia	5.374286
Austria	6.088571
Belgium	5.374286
Brazil	5.431429
Canada	5.427857
Chile	6.660000
Czech Republic	6.445714
Denmark	5.374286
Finland	5.945714
France	5.574286
Germany	5.588571
Hungary	6.517143
India	5.789231
Ireland	6.517143
Italy	5.374286
Netherlands	5.802857
Norway	5.660000
Poland	5.374286
Portugal	5.517143
Spain	5.374286
Sweden	5.517143
USA	5.747912
United Kingdom	5.374286

```

df_g = df_g.reset_index()
df_g.rename(columns = {"Total":"Average"}, inplace=True)
df_g.head(5)

   BillingCountry    Average
0      Argentina  5.374286
1     Australia  5.374286
2      Austria  6.088571
3     Belgium  5.374286
4      Brazil  5.431429

# Joining calculated Average into Dataset
df = df.merge(df_g, how="left", left_on = "BillingCountry", right_on =
"BillingCountry")
df.head(5)

   InvoiceId CustomerId        InvoiceDate          BillingAddress
0           1            2  2009-01-01 00:00:00  Theodor-Heuss-Straße 34
1           2            4  2009-01-02 00:00:00       Ullevålsveien 14
2           3            8  2009-01-03 00:00:00  Grétrystraat 63
3           4           14  2009-01-06 00:00:00  8210 111 ST NW
4           5           23  2009-01-11 00:00:00      69 Salem Street

   BillingCity BillingState BillingCountry BillingPostalCode  Total
Average
0      Stuttgart        None      Germany          70174  1.98
5.588571
1         Oslo        None      Norway          0171  3.96
5.660000
2    Brussels        None      Belgium          1000  5.94
5.374286
3    Edmonton        AB      Canada          T6G 2C7  8.91
5.427857
4     Boston        MA       USA          2113 13.86
5.747912

```

2.2. Processing of Null values in features fill with None and '99999'.

```

# Print out the number of missing values per column of Each Features
display('The Number of Null for Each Features', df.isnull().sum())

'The Number of Null for Each Features'

InvoiceId          0
CustomerId          0

```

```

InvoiceDate      0
BillingAddress   0
BillingCity      0
BillingState     202
BillingCountry   0
BillingPostalCode 28
Total            0
Average          0
dtype: int64

df= df.fillna({"BillingState": "None", "BillingPostalCode":"99999"})

display('The Number of Null for Each Features after transformed: ',
df.isnull().sum())

'The Number of Null for Each Features after transformed: '

InvoiceId      0
CustomerId      0
InvoiceDate    0
BillingAddress  0
BillingCity     0
BillingState    0
BillingCountry  0
BillingPostalCode 0
Total           0
Average         0
dtype: int64

df

   InvoiceId  CustomerId      InvoiceDate \
0          1            2  2009-01-01 00:00:00
1          2            4  2009-01-02 00:00:00
2          3            8  2009-01-03 00:00:00
3          4           14  2009-01-06 00:00:00
4          5           23  2009-01-11 00:00:00
..          ...
407        408           25  2013-12-05 00:00:00
408        409           29  2013-12-06 00:00:00
409        410           35  2013-12-09 00:00:00
410        411           44  2013-12-14 00:00:00
411        412           58  2013-12-22 00:00:00

   BillingAddress BillingCity BillingState \
0  Theodor-Heuss-Straße 34      Stuttgart      None
1  Ullevålsveien 14             Oslo        None
2  Grétrystraat 63            Brussels      None

```

3	8210 111 ST NW	Edmonton	AB	
4	69 Salem Street	Boston	MA	
..	
407	319 N. Frances Street	Madison	WI	
408	796 Dundas Street West	Toronto	ON	
409	Rua dos Campeões Europeus de Viena, 4350	Porto	None	
410	Porthaninkatu 9	Helsinki	None	
411	12,Community Centre	Delhi	None	
	BillingCountry	BillingPostalCode	Total	Average
0	Germany	70174	1.98	5.588571
1	Norway	0171	3.96	5.660000
2	Belgium	1000	5.94	5.374286
3	Canada	T6G 2C7	8.91	5.427857
4	USA	2113	13.86	5.747912
..
407	USA	53703	3.96	5.747912
408	Canada	M6J 1V1	5.94	5.427857
409	Portugal	99999	8.91	5.517143
410	Finland	00530	13.86	5.945714
411	India	110017	1.99	5.789231

[412 rows x 10 columns]

3. Preprocessing of Loading

- Loading finally project to sql

```
con

<sqlalchemy.engine.base.Connection at 0x1d18db24408>

sqlite_table = "New_Invoice"
df.to_sql(sqlite_table, con, if_exists='fail')

con.close()
```

4. Creating functions for ETL

4.1. Creating functions for Extract:

Function name: **extract_table_to_pandas**

```

# Function to extract for taken engine and connection from database
def extract_database(path):
    ...
    Input:
        path: Path of the database

    Output:
        return
        db_engine : The engine is get database
        db_connect: The connect is get connection with engine
    ...
    db_engine = sqlalchemy.create_engine(path)
    db_connect = db_engine.connect()

    return db_engine, db_connect

# Function to extract table to a pandas DataFrame

def extract_table_to_pandas(tablename, db_connect):
    ...
    Input:
        Table Name: Name of the table to be extracted
        db_connect: The connect to get connection with engine
    Output:
        return
        df          : The dataframe to be transformed
    ...
    query = "SELECT * FROM {}".format(tablename)
    result = db_connect.execute(query)

    df = pd.DataFrame(result.fetchall())
    df.columns = result.keys()

    return df

```

4.2. Creating functions for Transform:

Function name:

- `transform_avg_billing`
- `transform_fill_null`

```

# The transformation function of Average
def transform_avg_billing(data):
    ...
    Group by Country and extract average billing per Invoice
    Input:
        data : The dataframe to get to transform
    Output:
        return dataframe transformed

```

```

    ...
# Calculation of average by Country
df_g = data.groupby(['BillingCountry'])[['Total']].mean()
df_g = df_g.reset_index()
df_g.rename(columns = {"Total":"Average"}, inplace=True)

df = data.merge(df_g, how="left", left_on = "BillingCountry",
right_on = "BillingCountry")

return df

# The transformation function of filling in the missing values
def transform_fill_null(data):
    ...
    Some processing about Null values
    "BillingState" > "None"
    "BillingPostalCode" > "99999"
    Input:
        data : The dataframe to get to transform
    Output:
        return dataframe transformed
    ...

    # Processing of filling in the missing values
    data = data.fillna({"BillingState": "None",
"BillingPostalCode": "99999"})

    return data

```

4.3. Creating functions for Loading:

```

# Function to loading a pandas dataframe into table in database
def loading_to_sql(data, connect, sqlite_table='New_Table'):
    ...
    Input:
        data      : The dataframe to get to loading
        connect   : Name of the engine connection
        sqlite_table: Table name for loading
    ...

    # Processing of filling in the missing values

    data.to_sql(sqlite_table, connect, if_exists='fail')
    connect.close()
    return 'Loading is Done'

```

4.4. Testing functions of ETL

```

path =
"sqlite:///D:/01_DS_Document/Data_Sample/Chinook_SQLite/Chinook_Test.sqlite"

# Extracting

```

```

extract_db = extract_database(path)

tablename = 'Invoice'
engine = extract_db[0]
extract = extract_table_to_pandas(tablename, engine)

# Transformation
transform = transform_avg_billing(extract)
transform = transform_fill_null(transform)

# Loading
data = transform
connect = extract_db[1]
sqlite_table
loading_to_sql(data, connect, sqlite_table)

'Loading is Done'

# Print results
transform

```

	InvoiceId	CustomerId	InvoiceDate	\
0	1	2	2009-01-01 00:00:00	
1	2	4	2009-01-02 00:00:00	
2	3	8	2009-01-03 00:00:00	
3	4	14	2009-01-06 00:00:00	
4	5	23	2009-01-11 00:00:00	
..	
407	408	25	2013-12-05 00:00:00	
408	409	29	2013-12-06 00:00:00	
409	410	35	2013-12-09 00:00:00	
410	411	44	2013-12-14 00:00:00	
411	412	58	2013-12-22 00:00:00	
			BillingAddress	BillingCity
\			Theodor-Heuss-Straße 34	Stuttgart
0				None
1			Ullevålsveien 14	Oslo
2			Grétrystraat 63	Brussels
3			8210 111 ST NW	Edmonton
4			69 Salem Street	Boston
..		
407			319 N. Frances Street	Madison
408			796 Dundas Street West	Toronto
				ON

409	Rua dos Campeões Europeus de Viena, 4350	Porto	None
410	Porthaninkatu 9	Helsinki	None
411	12,Community Centre	Delhi	None


```

BillingCountry  BillingPostalCode  Total  Average
0              Germany          70174  1.98   5.588571
1              Norway           0171   3.96   5.660000
2              Belgium          1000   5.94   5.374286
3              Canada           T6G 2C7  8.91   5.427857
4              USA              2113   13.86  5.747912
..
407             USA              53703  3.96   5.747912
408             Canada          M6J 1V1  5.94   5.427857
409             Portugal         99999  8.91   5.517143
410             Finland          00530  13.86  5.945714
411             India            110017 1.99   5.789231

```

[412 rows x 10 columns]

```

transform.isnull().sum()

InvoiceId      0
CustomerId      0
InvoiceDate     0
BillingAddress   0
BillingCity      0
BillingState     0
BillingCountry    0
BillingPostalCode 0
Total           0
Average          0
dtype: int64

```

Finally block will help with **always closing** the connection to DB even in case of error.

```

# finally block will help with always closing the connection to DB
# even in case of error.
# sqlite_connection.close()

con.close()

```

```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Creating time series data
months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec"]
rainfall_values = [799.0, 1174.8, 865.1, 1334.6, 635.4, 918.5, 685.5,
998.6, 784.2, 985.0, 882.8, 1071.0]
time = np.arange(2012, 2013, 1/12) # Monthly intervals for 2012

# Creating a DataFrame
df = pd.DataFrame({"Month": months, "Year": 2012, "Rainfall (mm)": rainfall_values})

# Encoding month names to numerical values
label_encoder = LabelEncoder()
df['Month_encoded'] = label_encoder.fit_transform(df['Month'])

# Defining features and labels
X = df[['Month_encoded']]
y = (df['Rainfall (mm)'] > np.mean(rainfall_values)).astype(int) # Classify as high(1) or low(0) rainfall

# Splitting dataset
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Applying classification model
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

# Displaying accuracy
print(f'Classification Accuracy: {accuracy_score(y_test, y_pred):.2f}')

# Displaying data in tabular format
print("Output:")
print("When we execute the above code, it produces the following result and chart -\n")
print(df.to_string(index=False)) # Print table without index

# Creating a time series plot
plt.figure(figsize=(6, 5))
plt.plot(time, rainfall_values, marker='.', linestyle='-' ,

```

```

color='black')

# Labels
plt.xlabel("Time")
plt.ylabel("rainfall/timeseries")
plt.title("Rainfall Time Series")

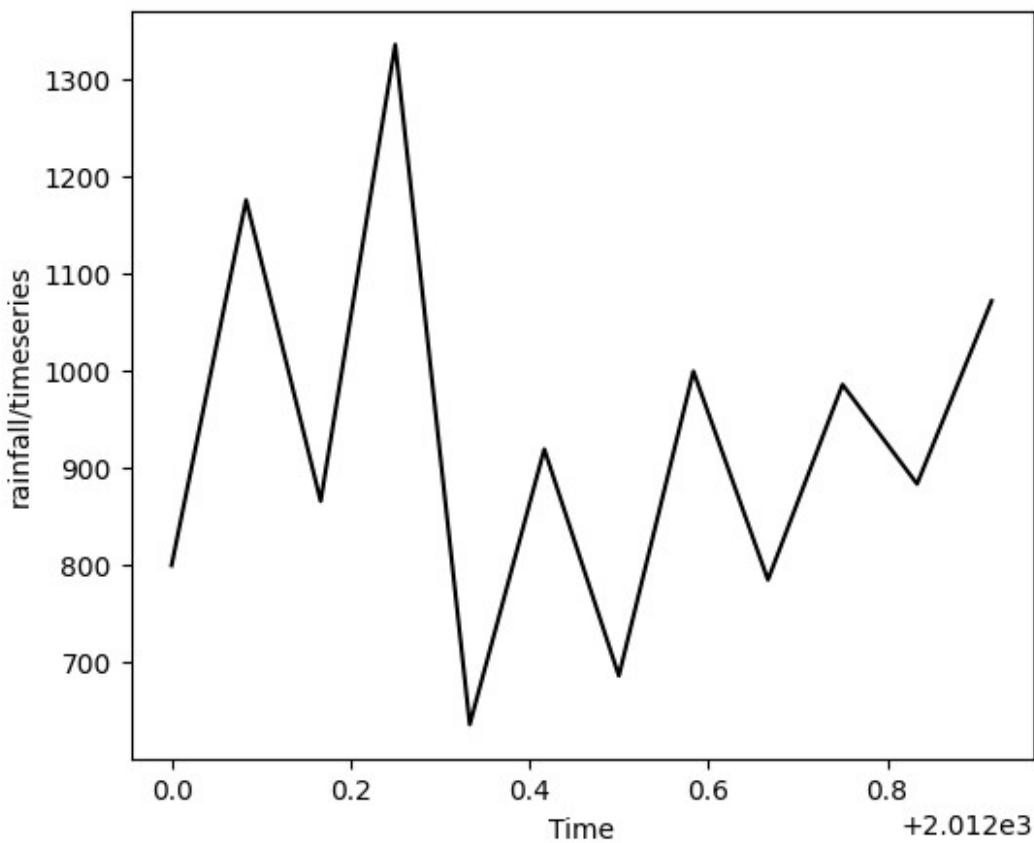
# Show plot
plt.show()

Classification Accuracy: 0.33
Output:
When we execute the above code, it produces the following result and
chart –

```

Month	Year	Rainfall (mm)	Month_encoded
Jan	2012	799.0	4
Feb	2012	1174.8	3
Mar	2012	865.1	7
Apr	2012	1334.6	0
May	2012	635.4	8
Jun	2012	918.5	6
Jul	2012	685.5	5
Aug	2012	998.6	1
Sep	2012	784.2	11
Oct	2012	985.0	10
Nov	2012	882.8	9
Dec	2012	1071.0	2

Rainfall Time Series



```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import networkx as nx
import requests
from bs4 import BeautifulSoup

# Generate sample data for clustering
def generate_data():
    X, _ = make_blobs(n_samples=300, centers=4, cluster_std=1.0,
random_state=42)
    return X

# Perform K-Means Clustering
def perform_kmeans(X, n_clusters=4):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X)
    return labels, kmeans.cluster_centers_

# Visualize Clustering Results
def visualize_clusters(X, labels, centers):
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels,
palette='viridis', alpha=0.7)
    plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='X',
s=200, label='Centroids')
    plt.title('K-Means Clustering')
    plt.legend()
    plt.show()

# PageRank Algorithm Implementation
def get_links(url):
    try:
        response = requests.get(url)
        soup = BeautifulSoup(response.text, 'html.parser')
        links = [a['href'] for a in soup.find_all('a', href=True) if
a['href'].startswith('http')]
        return links
    except:
        return []

# Build a web graph and compute PageRank
def compute_pagerank(start_url, max_pages=5):
    G = nx.DiGraph()
    urls_to_visit = [start_url]
    visited = set()

```

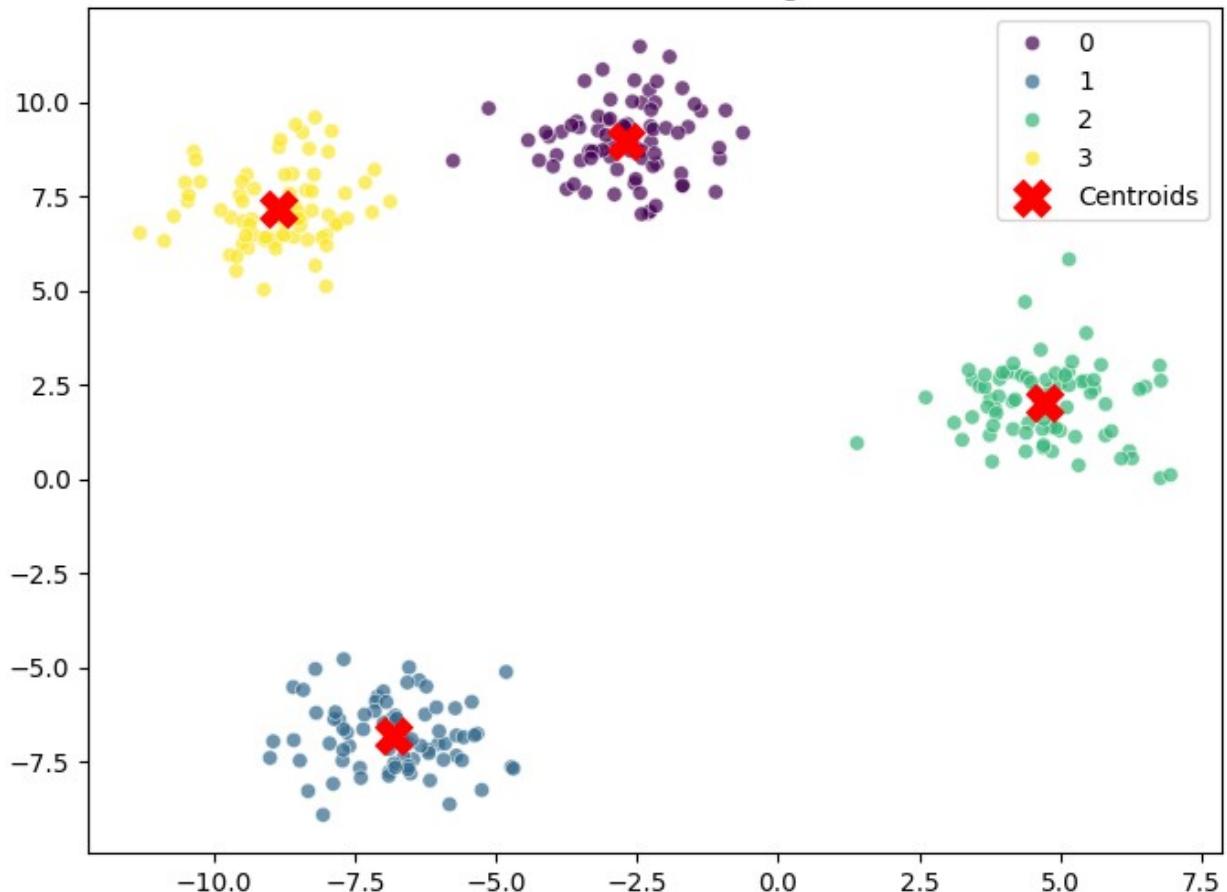
```
while urls_to_visit and len(visited) < max_pages:
    url = urls_to_visit.pop(0)
    if url in visited:
        continue
    visited.add(url)
    links = get_links(url)
    for link in links:
        G.add_edge(url, link)
        if link not in visited:
            urls_to_visit.append(link)

pagerank = nx.pagerank(G)
return pagerank

# Run Clustering
X = generate_data()
labels, centers = perform_kmeans(X)
visualize_clusters(X, labels, centers)

# Run PageRank
start_url = "https://www.kaggle.com/" # Replace with a real website
pagerank_scores = compute_pagerank(start_url)
print("PageRank Scores:", pagerank_scores)
```

K-Means Clustering



```
PageRank Scores: {'https://www.example.com': 0.012224236680506699,
'https://www.iana.org/domains/example': 0.022614813635585124,
'http://pti.icann.org': 0.017029713263097775, 'http://www.icann.org/':
0.017029713263097775, 'https://www.icann.org/privacy/policy':
0.01723314561716004, 'https://www.icann.org/privacy/tos':
0.01723314561716004, 'https://www.cloudflare.com/5xx-error-landing':
0.026698399503563727, 'https://account.icann.org/authorize?
redirect_uri=https://www.icann.org/
oauth&client_id=0oa9o9k9rp8VUcyJL2p7&nonce=6f5398ba-8a62-4370-99a8-
e00d981b692d&state=https://www.icann.org/': 0.012427669034568965,
'https://account.icann.org/registeraccount?redirect_uri=https://www.ic
ann.org/oauth&client_id=0oa9o9k9rp8VUcyJL2p7&nonce=6f5398ba-8a62-4370-
99a8-e00d981b692d&state=https://www.icann.org/': 0.012427669034568965,
'https://www.icann.org/history': 0.012427669034568965,
'https://info.icann.org/LP---Regional-Newsletter.html':
0.012427669034568965, 'https://subscribe.icann.org/':
0.012427669034568965, 'https://www.icann.org/resources/pages/regional-
reports-2021-2021-08-24-en': 0.012427669034568965,
'https://www.icann.org/en/public-comment?sort-param=upcoming-
date&sort-direction=asc&upcoming-sort-param=upcoming-date&upcoming-
```

sort-direction=desc&open-sort-param=open-closed-date&open-sort-direction=asc&pending-sort-param=pending-report-due-date&pending-sort-direction=desc': 0.012427669034568965,
'https://www.icann.org/en/public-comment/upcoming-proceedings?sort-param=upcoming-date&sort-direction=asc&page=1': 0.012427669034568965,
'https://www.icann.org/en/public-comment/closed-proceedings?page=1&sort-param=closed-date&sort-direction=desc':
0.012427669034568965, 'https://www.icann.org/en/public-comment/about':
0.012427669034568965, 'https://www.icann.org/en/public-comment/other-public-consultations': 0.012427669034568965,
'https://icann.org/en/board-activities-and-meetings':
0.012427669034568965, 'https://www.icann.org/complaints-office':
0.012427669034568965, 'https://www.icann.org/resources/pages/domain-name-registrants-2017-06-20-en': 0.012427669034568965,
'https://www.iana.org/': 0.012427669034568965,
'https://www.icann.org/en/icann-acronyms-and-terms':
0.012427669034568965, 'https://newgtldprogram.icann.org/':
0.012427669034568965, 'https://pti.icann.org/': 0.012427669034568965,
'https://opendata.icann.org/': 0.012427669034568965,
'https://www.icann.org/compliance/complaint': 0.012427669034568965,
'https://www.icann.org/technology': 0.012427669034568965,
'https://www.icann.org/careers': 0.012427669034568965,
'http://www.iana.org/dnssec': 0.012427669034568965,
'http://stats.research.icann.org/dns/tld_report/':
0.012427669034568965, 'https://www.iana.org/dnssec/ceremonies':
0.012427669034568965,
'http://gnso.icann.org/en/group-activities/inactive/2012/improvements':
0.012427669034568965,
'https://lists.icann.org/hyperkitty/list/ecadmin@icann.org/':
0.012427669034568965, 'http://aoctracking.icann.org/':
0.012427669034568965, 'https://gac.icann.org': 0.012427669034568965,
'https://atlarge.icann.org/': 0.012427669034568965,
'https://aso.icann.org/': 0.012427669034568965,
'https://ccnso.icann.org/': 0.012427669034568965,
'https://gnso.icann.org/en/': 0.012427669034568965,
'https://www.icann.org/nomcom': 0.012427669034568965,
'https://www.icann.org/csc': 0.012427669034568965,
'https://www.icann.org/rzerc': 0.012427669034568965,
'https://features.icann.org/compliance': 0.012427669034568965,
'https://www.icann.org/registrar-library': 0.012427669034568965,
'https://www.icann.org/en/system/files/files/registrar-billing-faq-21dec21-en.pdf': 0.012427669034568965,
'https://www.icann.org/registrar-reports/accreditation-qualified-list.html': 0.012427669034568965,
'https://meetings.icann.org/en/calendar ': 0.012427669034568965,
'https://newgtlds.icann.org/en/': 0.012427669034568965,
'https://newgtlds.icann.org/en/announcements-and-media/case-studies ':
0.012427669034568965,
'https://www.icann.org/resources/registries/gtlds/v1/newgtlds.csv':

0.012427669034568965, 'https://newgtlds.icann.org/en/about/trademark-clearinghouse': 0.012427669034568965,
'http://www.iana.org/domains/root/db': 0.012427669034568965,
'https://www.icann.org/public-comments/archive': 0.012427669034568965,
'https://www.icann.org/en/help/name-collision/report-problems':
0.012427669034568965, 'https://www.icann.org/resources/pages/privacy-policy-2021-03-05-ar': 0.012427669034568965,
'https://www.icann.org/resources/pages/privacy-policy-2021-03-05-es':
0.012427669034568965, 'https://www.icann.org/resources/pages/privacy-policy-2021-03-05-fr': 0.012427669034568965,
'https://www.icann.org/resources/pages/privacy-policy-2021-03-05-pt':
0.012427669034568965, 'https://www.icann.org/resources/pages/privacy-policy-2021-03-05-ru': 0.012427669034568965,
'https://www.icann.org/resources/pages/privacy-policy-2021-03-05-zh':
0.012427669034568965, 'https://www.youtube.com/icannnews':
0.012427669034568965, 'https://www.twitter.com/icann':
0.012427669034568965, 'https://www.linkedin.com/company/icann':
0.012427669034568965, 'http://www.flickr.com/photos/icann':
0.012427669034568965, 'https://www.facebook.com/icannorg':
0.012427669034568965, 'https://www.icann.org/resources/pages/global-newsletter': 0.012427669034568965, 'https://community.icann.org':
0.012427669034568965, 'https://www.icann.org/blog':
0.012427669034568965, 'http://learn.icann.org/': 0.012427669034568965,
'https://www.icann.org/resources/pages/groups-2012-02-06-en':
0.012427669034568965, 'https://www.icann.org/presidentsandceo-corner':
0.012427669034568965, 'https://www.icann.org/dprd':
0.012427669034568965, 'https://lookup.icann.org/':
0.012427669034568965, 'https://rdrs.icann.org/': 0.012427669034568965,
'https://www.icann.org/privacy': 0.012427669034568965,
'https://www.icann.org/privacy/cookies': 0.012427669034568965}

ASSIGNMENT No: 01

Title:-

Implement a Linear Regression Model to predict house prices for regions in the USA using the provided dataset.

Mapping with Syllabus -

Unit 1

Objective -

Develop a model to estimate house prices based on relevant features using Linear Regression.

Outcome -

- Apply Linear Regression in a real-world scenario.
- Understand the implementation of a regression model for house price prediction.

Software Requirements -

- Python (3.x recommended)
- Jupyter Notebook or any Python IDE

Hardware Requirements -

A machine with sufficient RAM and processing power for model training (8GB RAM recommended)

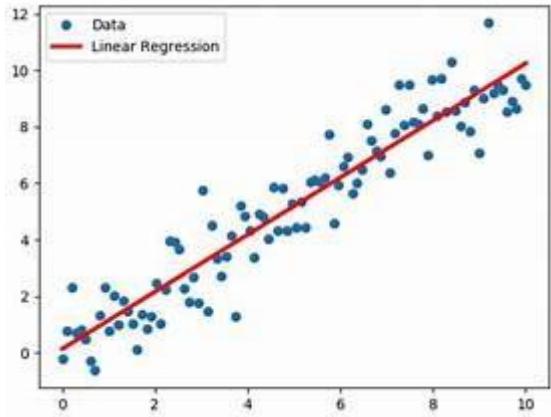
Prerequisites -

- Basic understanding of Python programming
- Familiarity with the concepts of supervised learning

Dataset -

https://github.com/huzaifsayed/Linear-Regression-Model-for-House-Price-Prediction/blob/master/USA_Housing.csv

LINEAR REGRESSION MODEL -



Libraries or Modules Used -

- NumPy
- pandas
- scikit-learn
- Matplotlib
- Seaborn

Theory -

Linear Regression –

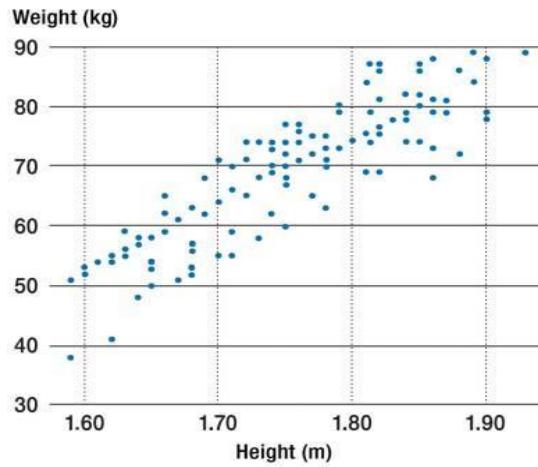
WHAT IS LINEAR REGRESSION ?

When we see a relationship in a scatterplot, we can use a line to summarize the relationship in the data. We can also use that line to make predictions in the data. This process is called **linear regression**. Linear Regression is a supervised learning algorithm used for predicting a continuous outcome, typically represented by the target variable. In the context of this assignment, we aim to predict house prices based on various features such as area, income, house age, number of rooms, and others.

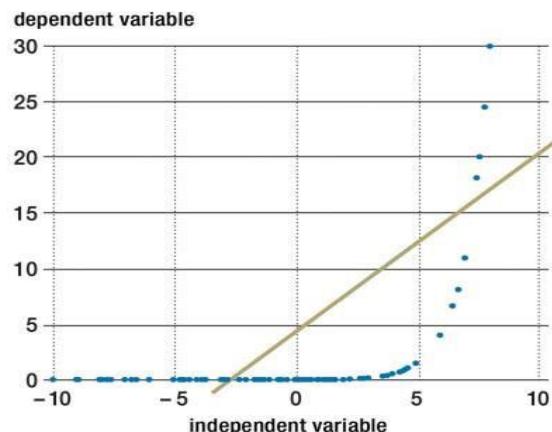
Linear regression is used to study the linear relationship between a dependent variable Y (blood pressure) and one or more independent variables X (age, weight, sex).

The dependent variable Y must be continuous, while the independent variables may be either continuous (age), binary (sex), or categorical (social status). The initial judgment of a possible relationship between two continuous variables should always be made on the basis of a scatter

plot (scatter graph). This type of plot will show whether the relationship is linear ([figure 1](#)) or nonlinear ([figure 2](#)).



[Figure 1](#)



[Figure 2](#)

A scatter plot showing an exponential relationship. In this case, it would not be appropriate to compute a coefficient of

Simple linear regression formula-

The formula for a simple linear regression is:

$$y = \beta_0 + \beta_1 X + \epsilon$$

- y is the predicted value of the dependent variable (y) for any given value of the

independent variable (X).

- B_0 is the **intercept**, the predicted value of Y when the X is 0.
- B_1 is the regression coefficient – how much we expect Y to change as X increases.
- X is the independent variable (the variable we expect is influencing Y).
- e is the **error** of the estimate, or how much variation there is in our estimate of the regression coefficient.

Linear regression finds the line of best fit line through your data by searching for the regression coefficient (B_1) that minimizes the total error (e) of the model.

While you can perform a linear regression by hand, this is a tedious process, so most people use statistical programs to help them quickly analyze the data.

Model Training –

Training a regression model involves teaching the model to predict continuous values based on input features. Here's a brief explanation of the process, along with some images to illustrate key concepts.

Regression Model Training:

1. Data Collection: Gather a dataset with input features (independent variables) and corresponding target values (dependent variable).
2. Data Splitting: Split the dataset into training and testing sets. The training set is used to train the model, and the testing set is used to evaluate its performance.
3. Model Selection: Choose a regression model architecture. Common choices include linear regression, decision trees, or more complex models like neural networks.
4. Feature Scaling: Normalize or standardize the input features to ensure that they are on a similar scale. This helps the model converge faster during training.
5. Model Training: Feed the training data into the chosen model and adjust the model's parameters to minimize the difference between predicted and actual target values.
6. Loss Function: Use a loss function to measure the difference between predicted and actual values. The goal is to minimize this loss during training.
7. Gradient Descent: Use optimization algorithms like gradient descent to iteratively update the model parameters and reduce the loss.
8. Model Evaluation: Evaluate the trained model on the testing set to assess its performance on unseen data.
9. Prediction: Once satisfied with the model's performance, use it to make predictions on new, unseen data.
- 10 Model Deployment: If the model performs well, deploy it to production for making real-world predictions.

Evaluation Metrics –

The performance of the model will be assessed using various evaluation metrics, such as Mean Squared Error (MSE) and R-squared. These metrics provide insights into how well the model

generalizes to unseen data.

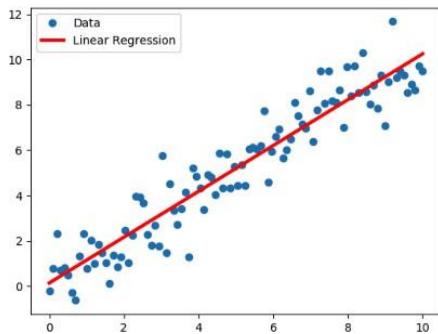


Figure: Linear Regression Model

Algorithm –

Simple Linear Regression Algorithm –

You'll start with the simplest case, which is simple linear regression. There are five basic steps when you're implementing linear regression:

1. Import the packages and classes that you need.
2. Provide data to work with, and eventually do appropriate transformations.
3. Create a regression model and fit it with existing data.
4. Check the results of model fitting to know whether the model is satisfactory.
5. Apply the model for predictions.

These steps are more or less general for most of the regression approaches and implementations. Throughout the rest of the tutorial, you'll learn how to do these steps for several different scenarios.

Step 1: Import packages and classes

The first step is to import the package numpy and the class LinearRegression from sklearn.linear_model:

Python

```
>>> import numpy as np  
>>> from sklearn.linear_model import LinearRegression
```

Now, you have all the functionalities that you need to implement linear regression.

The fundamental data type of NumPy is the array type called `numpy.ndarray`. The rest of this tutorial uses the term **array** to refer to instances of the type `numpy.ndarray`.

You'll use the class `sklearn.linear_model.LinearRegression` to perform linear and polynomial regression and make predictions accordingly.

Step 2: Provide data

The second step is defining data to work with. The inputs (regressors, x) and output

(response, y) should be arrays or similar objects. This is the simplest way of providing data for regression:

Python

```
>>> x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))

>>> y = np.array([5, 20, 14, 32, 22, 38])
```

Now, you have two arrays: the input, x , and the output, y . You should call `.reshape()` on x because this array must be **two-dimensional**, or more precisely, it must have **one column** and **as many rows as necessary**

. That's exactly what the argument $(-1, 1)$ of `.reshape()` specifies.

This is how x and y look now:

Python

```
>>> x

array([[ 5],
       [15],
       [25],
       [35],
       [45],

>>> y

array([ 5, 20, 14, 32, 22, 38])
```

As you can see, x has two dimensions, and $x.shape$ is $(6, 1)$, while y has a single dimension, and $y.shape$ is $(6,)$.

Step 3: Create a model and fit it

The next step is to create a linear regression model and fit it using the existing data.

Create an instance of the class `LinearRegression`, which will represent the regression model:

Python

```
>>> model = LinearRegression()
```

This statement creates the `variable` model as an instance of `LinearRegression`. You can provide several optional parameters to `LinearRegression`:

- `fit_intercept` is a `Boolean` that, if True, decides to calculate the intercept b_0 or, if False, considers it equal to zero. It defaults to True.
- `normalize` is a Boolean that, if True, decides to normalize the input variables. It defaults to False, in which case it doesn't normalize the input variables.
- `copy_X` is a Boolean that decides whether to copy (True) or overwrite the input variables (False). It's True by default.
- `n_jobs` is either an integer or None. It represents the number of jobs used in parallel computation. It defaults to None, which usually means one job. -1 means to use all available processors.

Your model as defined above uses the default values of all parameters. It's time to start

using the model. First, you need to call `.fit()` on model:

Python

```
>>> model.fit(x, y) LinearRegression()
```

With `fit()`, you calculate the optimal values of the weights b_0 and b_1 , using the existing input

and output, x and y, as the arguments. In other words, .fit() **fits the model**. It returns self, which is the variable model itself. That's why you can replace the last two statements with this one:

Python

```
>>> model = LinearRegression().fit(x, y)
```

This statement does the same thing as the previous two. It's just shorter.

Step 4: Get results

Once you have your model fitted, you can get the results to check whether the model works satisfactorily and to interpret it.

You can obtain the coefficient of determination, R^2 , with .score() called on model:

Python

```
>>> r_sq = model.score(x, y)
```

```
>>> print(f"coefficient of determination: {r_sq}")
```

When you're applying .score(), the arguments are also the predictor x and response y, and the

return value is R^2 .

The attributes of model are .intercept_, which represents the coefficient b_0 , and .coef_, which

represents b_1 :

Python

```
>>> print(f"intercept: {model.intercept_}")
```

```
intercept: 5.63333333333329
```

```
>>> print(f"slope: {model.coef_}")
```

```
slope: [0.54]
```

The code above illustrates how to get b_0 while .coef_ is an array.

The value of b_0 is approximately 5.63. This illustrates that your model predicts the response

5.63 when b_1 is zero. The value $b_1 = 0.54$ means that the predicted response rises by 0.54 when

x is increased by one.

You'll notice that you can provide y as a two-dimensional array as well. In this case, you'll get a similar result. This is how it might look:

Python

```
>>> new_model = LinearRegression().fit(x, y.reshape((-1, 1)))

>>> print(f"intercept: {new_model.intercept_}") intercept:
[5.63333333]

>>> print(f"slope: {new_model.coef_}")

slope: [[0.54]]
```

As you can see, this example is very similar to the previous one, but in this case, `.intercept_` is a one-dimensional array with the single element b_0 , and `.coef_` is a two-dimensional array with the single element b_1 .

Step 5: Predict response

Once you have a satisfactory model, then you can use it for predictions with either existing or new data. To obtain the predicted response, use `.predict()`:

Python

```
>>> y_pred = model.predict(x)

>>> print(f"predicted response:\n{y_pred}")
```

predicted response:

```
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```

When applying .predict(), you pass the regressor as the argument and get the corresponding predicted response. This is a nearly identical way to predict the response:

Python

```
>>> y_pred = model.intercept_ + model.coef_* x  
  
>>> print(f"predicted response:\n{y_pred}")  
predicted  
response:  
  
[[ 8.3333333]  
  
[13.7333333]  
  
[19.1333333]
```

In this case, you multiply each element of x with model.coef_ and add model.intercept_ to the product.

The output here differs from the previous example only in dimensions. The predicted response is now a two-dimensional array, while in the previous case, it had one dimension.

If you reduce the number of dimensions of x to one, then these two approaches will yield the same result. You can do this by replacing x with x.reshape(-1), x.flatten(), or x.ravel() when multiplying it with model.coef_.

In practice, regression models are often applied for forecasts. This means that you can use fitted models to calculate the outputs based on new inputs:

Python

```
>>> x_new = np.arange(5).reshape((-1, 1))  
  
>>> x_new  
array([[0],  
       [1],  
       [2],  
       [3],  
       [4]]  
  
>>> y_new = model.predict(x_new)
```

```
>>> y_new  
array([5.63333333, 6.17333333, 6.71333333, 7.25333333, 7.79333333])
```

Here .predict() is applied to the new regressor x_new and yields the response y_new. This example conveniently uses `arange()` from numpy to generate an array with the elements from 0, inclusive, up to but excluding 5 that is, 0, 1, 2, 3, and 4.

Application -

Real estate pricing strategies

1. Assisting clients in making informed decisions
2. Market analysis for regions in the USA
3. Predictive tool for estimating house prices

Inference -

The process involves exploring the data, data collection, model fitting, and interpretation of regression results. Practical skills in using statistical software for regression analysis are likely to be developed, enhancing students ability to analyze relationships between variables and make predictions based on linear model.

Code -

```
import pandas as pd import  
numpy as np  
from sklearn.model_selection import train_test_split from  
sklearn.linear_model import LinearRegression  
from sklearn.metrics import mean_squared_error, r2_score import  
matplotlib.pyplot as plt  
  
# Load the dataset  
df = pd.read_csv('USA_Housing.csv') #  
Explore the dataset print(df.head())  
print(df.info())  
# Handle missing values if any df =  
df.dropna()  
  
# Select relevant features and target variable  
X = df[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of  
Bedrooms', 'Area Population']]  
y = df['Price']  
  
# Split the dataset into training and testing sets  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
  
# Initialize the Linear Regression model model =  
LinearRegression()
```

```
# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set y_pred =
model.predict(X_test)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred) r2 =
r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

Output -

```
Avg. Area Income Avg. Area House Age Avg. Area Number of Rooms \
0 79545.458574      5.682861      7.009188
1 79248.642455      6.002900      6.730821
2 61287.067179      5.865890      8.512727
3 63345.240046      7.188236      5.586729
4 59982.197226      5.040555      7.839388
```

```
Avg. Area Number of Bedrooms Area Population      Price \
0 4.09            23086.800503  1.059034e+06
1 3.09            40173.072174  1.505891e+06
2 5.13            36882.159400  1.058988e+06
3 3.26            34310.242831  1.260617e+06
4 4.23            26354.109472  6.309435e+05
```

Address

```
0 208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1 188 Johnson Views Suite 079\nLake Kathleen, CA...
2 9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3           USS Barnett\nFPO AP 44820
4           USNS Raymond\nFPO AE 09386
```

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 5000 entries, 0 to 4999 Data

columns (total 7 columns):

#	Column	Non-Null Count	Dtype
0	Avg. Area Income	5000	non-null float64
1	Avg. Area House Age	5000	non-null float64
2	Avg. Area Number of Rooms	5000	non-null float64
3	Avg. Area Number of Bedrooms	5000	non-null float64
4	Area Population	5000	non-null float64
5	Price	5000	non-null float64
6	Address	5000	non-null object

dtypes: float64(6), object(1)

memory usage: 273.6+ KB None

Mean Squared Error: 10089009300.894518 R-

squared: 0.9179971706834289

References – https://en.wikipedia.org/wiki/Linear_regression

<https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/introduction-to-trend-lines/a/linear-regression-review>

<https://github.com/huzafsayed/Linear-Regression-Model-for-House-Price-Prediction/blob/master/linear-regression-model.jpg>

Conclusion: Thus Implemented a Linear Regression Model to predict house prices for regions in the USA using the provided dataset.

ASSIGNMENT No: 02

Title:- Build a Multiclass classifier using the CNN model. Use MNIST or any other suitable dataset.

- a. Perform Data Pre-processing
- b. Define Model and perform training
- c. Evaluate Results using confusion matrix.

Mapping with Syllabus -

Unit 3

Objective -

Building a multiclass classifier using a Convolutional Neural Network (CNN) using MNIST or any other suitable dataset. It involves several steps, including data pre-processing, defining the model architecture, training the model, and evaluating its performance using a confusion matrix.

Outcome -

Implement the technique of Convolution neural network (CNN)

Software Requirements -

- Python (3.x recommended)
- TensorFlow (Deep learning framework for building CNNs)
- Jupyter Notebook, any Python IDE, or Google Colab (for running Python code)

Hardware Requirements -

- A machine with at least 8GB of RAM is recommended for model training.
- A multi-core CPU is suitable, and for faster training, a GPU (Graphics Processing Unit) is highly recommended.

Prerequisites -

- Basic understanding of Python programming
- Familiarity with the concepts of Neural Networks, especially Convolutional Neural Networks (CNNs)

Dataset -

<https://github.com/AmritK10/MNIST-CNN>

A 10x10 grid of handwritten digits from 0 to 9, representing the MNIST dataset. The digits are arranged in a grid where each row contains digits from 0 to 9. The digits are written in a cursive style.

0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9

Libraries or Modules Used -

- Numpy - for linear algebra.
- Pandas - for data analysis.
- Matplotlib - for data visualization.
- Tensorflow - for neural networks.

Theory -

ANN or Artificial Neural Network is a multi-layer fully-connected neural net that consists of many layers, including an input layer, multiple hidden layers, and an output layer. This is a very popular deep learning algorithm used in various classification tasks like audio and words. Similarly, we have Convolutional Neural Networks(CNNs) for image classification.

CNN is basically a model known to be **Convolutional Neural Network** and in recent times it has gained a lot of popularity because of its usefulness. CNN uses multilayer perceptrons to do computational works. CNN uses relatively little pre-processing compared to other image classification algorithms. This means the network learns through filters that in traditional algorithms were hand-engineered. So, for the image processing tasks CNNs are the best-suited option.

Applying a Convolutional Neural Network (CNN) on the MNIST dataset is a popular way to learn about and demonstrate the capabilities of CNNs for image classification tasks. The

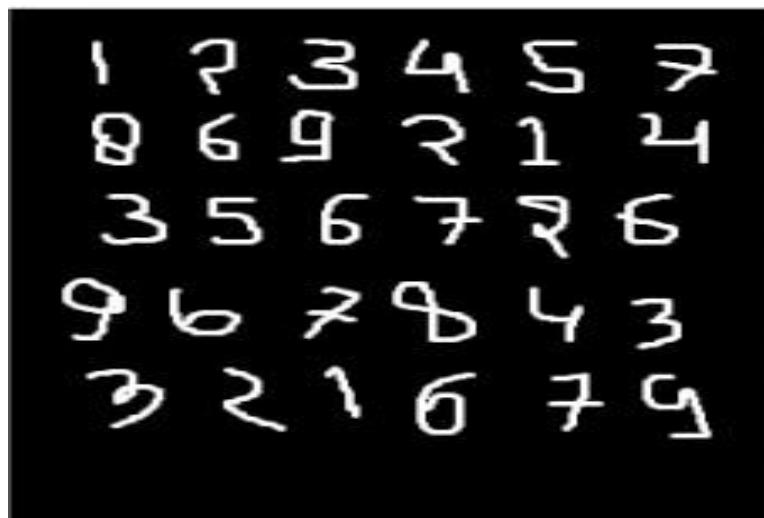
MNIST dataset consists of 28×28 grayscale images of hand-written digits (0-9), with a training set of 60,000 examples and a test set of 10,000 examples.

Here is a basic approach to applying a CNN on the MNIST dataset using the Python programming language and the Keras library:

1. Load and preprocess the data: The MNIST dataset can be loaded using the Keras library, and the images can be normalized to have pixel values between 0 and 1.
2. Define the model architecture: The CNN can be constructed using the Keras Sequential API, which allows for easy building of sequential models layer-by-layer. The architecture should typically include convolutional layers, pooling layers, and fully-connected layers.
3. Compile the model: The model needs to be compiled with a loss function, an optimizer, and a metric for evaluation.
4. Train the model: The model can be trained on the training set using the Keras fit() function. It is important to monitor the training accuracy and loss to ensure the model is converging properly.
5. Evaluate the model: The trained model can be evaluated on the test set using the Keras evaluate() function. The evaluation metric typically used for classification tasks is accuracy.

MNIST dataset:

mnist dataset is a dataset of handwritten images as shown below in the image



We can get 99.06% accuracy by using CNN(Convolutional Neural Network) with a functional model. The reason for using a functional model is to maintain easiness while connecting the layers.

Firstly, include all necessary libraries

Python3:

```
import numpy as np
import keras
from keras.datasets import mnist
from keras.models import Model
from keras.layers import Dense, Input
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten
from keras import backend as k
```

Create the train data and test data

- **Test data:** Used for testing the model that how our model has been trained. Used to
- Train data:** train our model.

Python3:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

While proceeding further, **img_rows** and **img_cols** are used as the image dimensions. In mnis dataset, it is 28 and 28. We also need to check the data format i.e. ‘channels_first’ or ‘channels_last’. In CNN, we can normalize data before hands such that large terms of the calculations can be reduced to smaller terms. Like, we can normalize the x_train and x_test data by dividing it by 255.

Checking data-format:

Python3:

```
img_rows, img_cols=28, 28
```

```
if k.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    inpx = (1, img_rows, img_cols)
```

else:

```
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
inpx = (img_rows, img_cols, 1)
```

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
```

x_test /= 255

Since the output of the model can comprise any of the digits between 0 to 9. so, we need 10 classes in output. To

Description of the output classes:

make output for 10 classes, use keras.utils.to_categorical function, which will provide the 10 columns. Out of these 10 columns, only one value will be one and the rest 9 will be zero and this one value of the output will denote the class of the digit.

Python3:

```
y_train = keras.utils.to_categorical(y_train) y_test =
keras.utils.to_categorical(y_test)
```

- Now, the dataset is ready so let's move towards the CNN model :

Python3:

```
inpx = Input(shape=inpx)
```

```
layer1 = Conv2D(32, kernel_size=(3, 3), activation='relu')(inpx) layer2 =
Conv2D(64, (3, 3), activation='relu')(layer1)
layer3 = MaxPooling2D(pool_size=(3, 3))(layer2) layer4 =
Dropout(0.5)(layer3)
layer5 = Flatten()(layer4)
layer6 = Dense(250, activation='sigmoid')(layer5) layer7 = Dense(10,
activation='softmax')(layer6)
```

- Explanation of the working of each layer in the CNN model:**

layer1 is the Conv2d layer which convolves the image using 32 filters each of size (3*3). layer2 is again a Conv2D layer which is also used to convolve the image and is using 64 filters each of size (3*3).

layer3 is the MaxPooling2D layer which picks the max value out of a matrix of size (3*3).

layer4 is showing Dropout at a rate of 0.5.

layer5 is flattening the output obtained from layer4 and this flattens output is passed to layer6.

layer6 is a hidden layer of a neural network containing 250 neurons.

layer7 is the output layer having 10 neurons for 10 classes of output that is using the softmax function.

- Calling compile and fit function:**

Python3:

```
model = Model([inpx], layer7) model.compile(optimizer=
keras.optimizers.Adadelta(),
loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])

model.fit(x_train, y_train, epochs=12, batch_size=500)
```

```

Epoch 1/12
60000/60000 [=====] - 968s 16ms/step - loss: 0.7357 - acc: 0.7749
Epoch 2/12
60000/60000 [=====] - 955s 16ms/step - loss: 0.2087 - acc: 0.9413
Epoch 3/12
60000/60000 [=====] - 968s 16ms/step - loss: 0.1287 - acc: 0.9631
Epoch 4/12
60000/60000 [=====] - 968s 16ms/step - loss: 0.0948 - acc: 0.9728
Epoch 5/12
60000/60000 [=====] - 956s 16ms/step - loss: 0.0780 - acc: 0.9774
Epoch 6/12
60000/60000 [=====] - 915s 15ms/step - loss: 0.0655 - acc: 0.9807
Epoch 7/12
60000/60000 [=====] - 907s 15ms/step - loss: 0.0575 - acc: 0.9829
Epoch 8/12
60000/60000 [=====] - 914s 15ms/step - loss: 0.0498 - acc: 0.9852
Epoch 9/12
60000/60000 [=====] - 917s 15ms/step - loss: 0.0468 - acc: 0.9861
Epoch 10/12
60000/60000 [=====] - 912s 15ms/step - loss: 0.0420 - acc: 0.9873
Epoch 11/12
60000/60000 [=====] - 967s 16ms/step - loss: 0.0405 - acc: 0.9880
Epoch 12/12
60000/60000 [=====] - 993s 17ms/step - loss: 0.0371 - acc: 0.9888
<keras.callbacks.History at 0x21ce04bb6a0>

```

- Firstly, we made an object of the model as shown in the above-given lines, where [inpx] is the input in the model and layer7 is the output of the model. We compiled the model using the required optimizer, loss function and printed the accuracy and at the last model.fit was called along with parameters like x_train(means image vectors), y_train(means the label), number of epochs, and the batch size. Using fit function x_train, y_train dataset is fed to model in particular batch size.

Evaluate function:

model.evaluate provides the score for the test data i.e. provided the test data to the model. Now, the model will predict the class of the data, and the predicted class will be matched with the y_test label to give us the accuracy.

Python3:

```

score = model.evaluate(x_test, y_test, verbose=0) int('loss',
score[0])
print('accuracy=', score[1])

```

Output:

loss= 0.0295960184669

accuracy = 0.991

Algorithm -

1. Import Libraries:

- Import necessary libraries, including TensorFlow and Keras.

2. Load and Pre-process the MNIST Dataset:

- Load the MNIST dataset, which consists of 28x28 grayscale images of handwritten digits (0 through 9).
- Pre-process the data by normalizing pixel values (between 0 and 1), reshaping images, and one-hot encoding labels.

3. Define CNN Model Architecture:

- Design the CNN architecture with convolutional layers, pooling layers, and fully connected layers.
- Use activation functions like ReLU to introduce non-linearity.
- The final layer has 10 units with softmax activation for multiclass classification.

4. Compile the Model:

- Specify the optimizer (e.g., 'adam'), loss function (e.g., 'categorical_crossentropy' for multiclass classification), and evaluation metric (e.g., 'accuracy').

5. Train the Model:

- Train the CNN using the training dataset.
- Specify the number of epochs (passes through the entire dataset) and batch size.

6. Evaluate the Model:

- Evaluate the trained CNN on the test dataset to assess its performance.
- Measure metrics such as accuracy to understand how well the model generalizes to unseen data.

These steps provide a comprehensive overview of the process involved in building and training a CNN for image classification using the MNIST dataset. Adjustments to these steps can be made based on specific model requirements and task objectives.

Application -

1. Handwritten Digit Recognition:

- Recognizes handwritten digits (0-9) with applications in automated systems.

2. Automatic Check Processing:

- Processes checks by recognizing handwritten amounts and account numbers.

3. Postal Code Recognition:

- Recognizes postal codes on envelopes for automated mail sorting.

4. Document Classification:

- Classifies documents based on handwritten patterns or characters.

5. Medical Imaging:

- Analyzes medical images for detecting anomalies or identifying patterns.

6. Character Recognition in Forms:

- Recognizes handwritten characters in forms for efficient data entry

7. Gesture Recognition:

- Recognizes hand gestures for sign language translation or device control

8. Product Label Recognition:

- Reads and interprets product labels for inventory or quality control.

9. Traffic Sign Recognition:

- Identifies handwritten or printed characters on traffic signs for intelligent transportation.

10. Historical Document Analysis:

- Analyzes historical handwritten documents for digitization and preservation.

Inference –

CNN excels in diverse applications: finance (check processing), healthcare (image analysis), retail (label recognition), and transportation (sign recognition). It aids digitization efforts and ensures cultural heritage preservation. Overall, CNN offers valuable automation and efficiency across industries.

References:

<https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/>

<https://www.youtube.com/watch?v=9cPMFTwBdM4>

Conclusion: Thus I Build a Multiclass classifier using the CNN model using MNIST or any other suitable dataset by Performing Data Pre-processing, Defining Model and perform training and Evaluating Results using confusion matrix.

ASSIGNMENT No: 03

Title:-

Design RNN or its variant including LSTM or GRU

- a) Select a suitable time series dataset. E.g - Predict sentiments based on product reviews.
- b) Apply for prediction

Mapping with Syllabus -

Unit 4

Objective -

Implement a Recurrent Neural Network (RNN) or its variant (LSTM or GRU) on a selected time series dataset, such as predicting sentiments based on product reviews, to develop a predictive model for sentiment analysis.

Outcome -

Solve the language translation problem by Recurrent neural network(RNN)

Software Requirements -

- Python (3.x recommended)
- Jupyter Notebook or any Python IDE or Google Colab

Hardware Requirements -

A machine with sufficient RAM and processing power for model training (8GB RAM recommended)

Prerequisites -

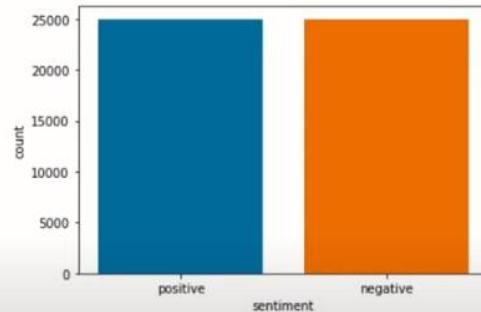
- Basic understanding of Python programming
- Familiarity with the concepts of Neural Networks

Dataset -

Inbuilt tensorflow-keras-imdb dataset

IMDb Movie Reviews Dataset

review	sentiment
50k unique values	2 unique values
One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. The...	positive
A wonderful little production. The filming technique is very unassuming- very old-time-B...	positive
I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air con...	positive
Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet	negative



Libraries or Modules Used

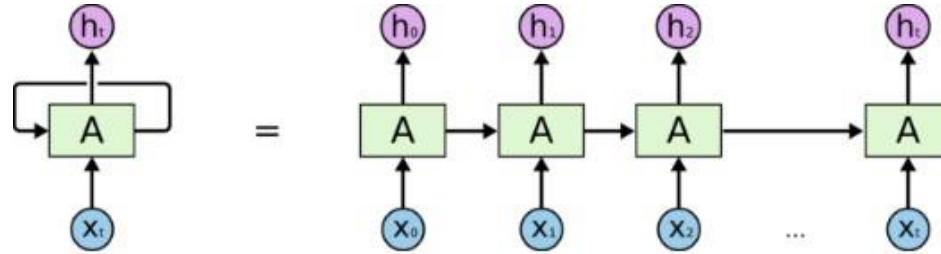
- Keras
- Tensorflow

Theory -

Recurrent Neural Network (RNN)

Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input.

Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other.



An unrolled recurrent neural network.

First, it takes the $X(0)$ from the sequence of input and then it outputs $h(0)$ which together with $X(1)$ is the input for the next step. So, the $h(0)$ and $X(1)$ is the input for the next step. Similarly, $h(1)$ from the next is the input with $X(2)$ for the next step and so on. This way, it keeps remembering the context while training
The formula for the current state is

$$h_t = f(h_{t-1}, x_t)$$

Applying Activation Function:

$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$

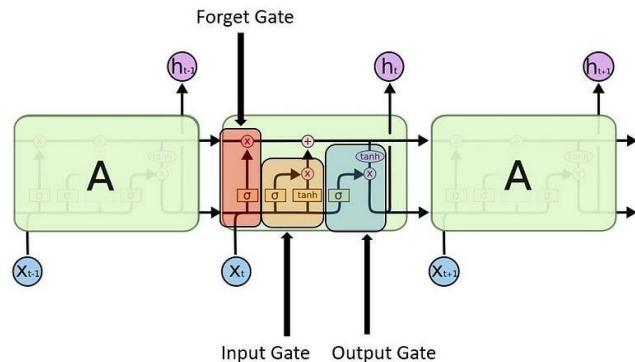
W is weight, h is the single hidden vector, Whh is the weight at previous hidden state, Whx is the weight at current input state, \tanh is the Activation function, that implements a Non-linearity that squashes the activations to the range[-1,1]

$$y_t = W_{hy}h_t$$

y_t is the output state. Why is the weight at the output state.

Long Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present:



- 1) Input gate - discover which value from input should be used to modify the memory. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- 2) Forget gate - discover what details to be discarded from the block. It is decided by the sigmoid function. it looks at the previous state(h_{t-1}) and the content input(x_t) and outputs a number between 0(omit this)and 1(keep this)for each number in the cell state C_{t-1} .

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

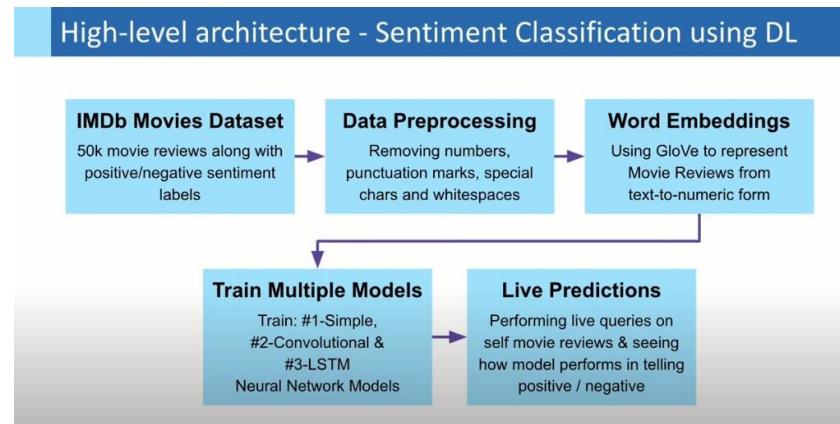
- 3) Output gate — the input and the memory of the block is used to decide the output. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1 and multiplied with output of Sigmoid.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Algorithm -

- 1) Load IMDb Movie Reviews dataset (50,000 reviews)



- 2) Pre-process dataset by removing special characters, numbers, etc. from user reviews + convert sentiment labels positive & negative to numbers 1 & 0, respectively

Data Preprocessing

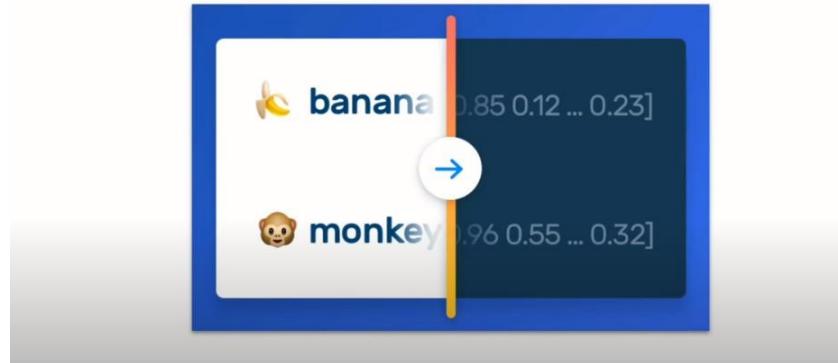
Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time.

This movie is slower than a soap opera... and suddenly, Jake decides to become Rambo and kill the zombie.

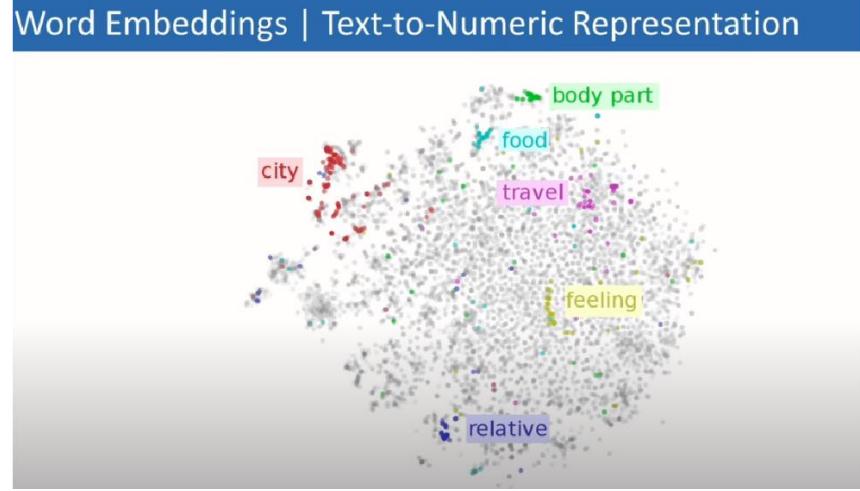
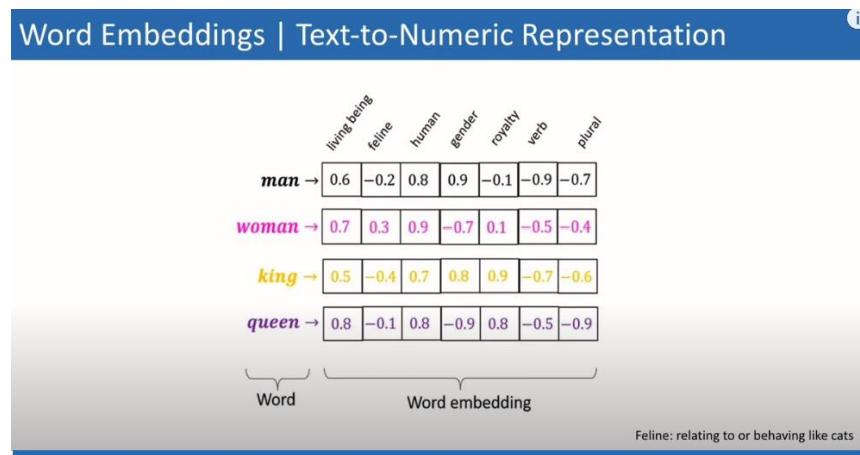
OK, first of all when you're going to make a film you must Decide if its a thriller or a drama! As a drama the movie is watchable. Parents are divorcing & arguing like in real life. And then we have Jake with his closet which totally ruins all the film! I expected to see a BOOGEYMAN similar movie, and instead i watched a drama with some meaningless thriller spots.

3 out of 10 just for the well playing parents & descent dialogs. As for the shots with Jake: just ignore them.

Transforming text into numbers



3) Import GloVe Word Embedding to build Embedding Dictionary + Use this to build Embedding Matrix for our Corpus



4) Model Training using Deep Learning in Keras for separate: Simple Neural Net, CNN and LSTM Models and analyse model performance and results
 5) Perform predictions on real IMDb movie reviews

Application -

1) Product Review Sentiment Analysis:

Predict sentiment (positive, negative, neutral) from user reviews for product improvement insights.

2) Customer Feedback Analysis:

Analyze sentiments in customer feedback to understand overall satisfaction and identify areas for improvement.

3) Brand Monitoring:

Monitor social media for product mentions and analyze sentiments to assess brand perception.

4) Market Research:

Analyze sentiments in market surveys to gauge consumer opinions about specific products or features.

5) Quality Assurance in E-commerce:

Automatically categorize and flag reviews with negative sentiments to improve product quality.

Inference -

The process involves data preparation, embedding, model design, training, and evaluation.

The use of different architectures such as Simple Neural Net, CNN, and LSTM allows for comparison and analysis of their performance on sentiment prediction for IMDb movie reviews. The GloVe Word Embedding enhances the models' understanding of the textual data.

Finally, predictions are made on real IMDb movie reviews to assess the models' applicability and accuracy.

Code -

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Set the parameters
max_features = 10000 # Number of words to consider as features
maxlen = 100 # Cut texts after this number of words (among top max_features most common words)
batch_size = 32

# Load the IMDB dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features) # Pad
# sequences to have a consistent length for the input to the RNN
```

```
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

# Build the RNN model with LSTM
model = Sequential()
model.add(Embedding(max_features, 128))
model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy',
              optimizer='adam', metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train,
          batch_size=batch_size, epochs=5,
          validation_data=(x_test, y_test))

# Evaluate the model
score, acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print(f'Test score: {score}')
print(f'Test accuracy: {acc}')
```

Output -

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [=====] - 0s 0us/step
Epoch 1/5
782/782 [=====] - 319s 401ms/step - loss: 0.4161 - accuracy: 0.8074 - val_loss: 0.3585 - val_accuracy: 0.8412 Epoch 2/5
782/782 [=====] - 288s 368ms/step - loss: 0.2625 - accuracy: 0.8950 - val_loss: 0.3482 - val_accuracy: 0.8454 Epoch 3/5
782/782 [=====] - 284s 363ms/step - loss: 0.1931 - accuracy: 0.9244 - val_loss: 0.4158 - val_accuracy: 0.8375 Epoch 4/5
782/782 [=====] - 285s 365ms/step - loss: 0.1431 - accuracy: 0.9472 - val_loss: 0.4504 - val_accuracy: 0.8412 Epoch 5/5
782/782 [=====] - 287s 367ms/step - loss: 0.1093 - accuracy: 0.9606 - val_loss: 0.4790 - val_accuracy: 0.8413
782/782 [=====] - 25s 32ms/step - loss: 0.4790 - accuracy:
```

0.8413

Test score: 0.47897636890411377

Test accuracy: 0.8413199782371521

References:

<https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e>

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

https://youtu.be/oWo9SNcyxII?si=0OzO6SUYZ_FxbTgY

Conclusion: Thus Designed RNN or its variant including LSTM or GRU

ASSIGNMENT No: 04

Title:-

Design and implement CNN for image classification.

- a) Select a suitable image classification dataset (medical engineering, agricultural, etc.).
- b) Optimized with different hyper-parameters including learning rate, filter size, no. of layers, optimizers, dropouts, etc.

Mapping with Syllabus -

Unit 3

Objective -

Design and implement a Convolutional Neural Network (CNN) for image classification on a selected image classification dataset, such as medical engineering, agricultural to optimize with different hyper-parameters including learning rate, filter size, no. of layers, optimizers, dropouts, etc.

Outcome -

Implement the technique of Convolution neural network (CNN)

Software Requirements -

- Python (3.x recommended)
- Jupyter Notebook or any Python IDE or Google Colab

Hardware Requirements -

A machine with sufficient RAM and processing power for model training (8GB RAM recommended)

Prerequisites -

- Basic understanding of Python programming
- Familiarity with the concepts of Neural Networks

Dataset -

Inbuilt dataset - MNIST

<https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

Libraries or Modules Used -

- Keras (for building and training neural network models)
- NumPy (for numerical operations)
- Matplotlib (for plotting images)
- TensorFlow (deep learning framework)
- Adam (optimizer for training)

Theory -

Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNN) are a powerful type of deep learning model specifically designed for processing and analyzing visual data, such as images and videos. They have revolutionized the field of Computer Vision, enabling remarkable advancements in tasks like Image Recognition, Object Detection, and Image Segmentation.

To grasp the essence of Convolutional Neural Networks (CNNs), it is essential to have a solid understanding of the basics of Deep Learning and acquaint yourself with the terminology and principles of neural networks. If you're new to this, don't fret! I have previously covered these fundamentals in my blog posts, serving as primers to help you lay a strong foundation.

Basic Architecture

The architecture of Convolutional Neural Networks is meticulously designed to extract meaningful features from complex visual data. This is achieved through the use of specialized layers within the network architecture. It comprises of three fundamental layer types:

1. Convolutional Layers
2. Pooling Layers
3. Fully-Connected Layers

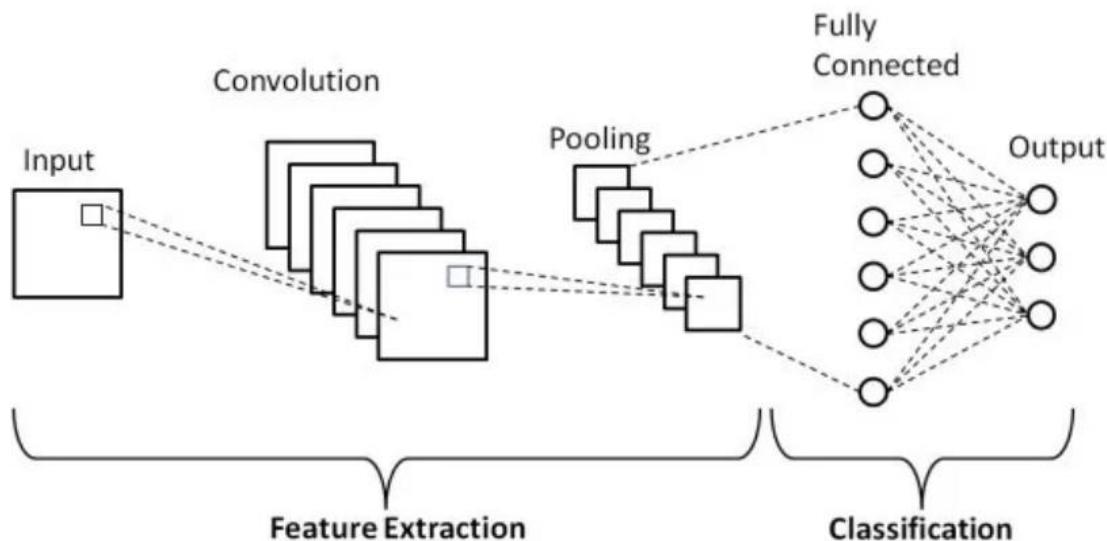


Image Source: [Research Gate](#)

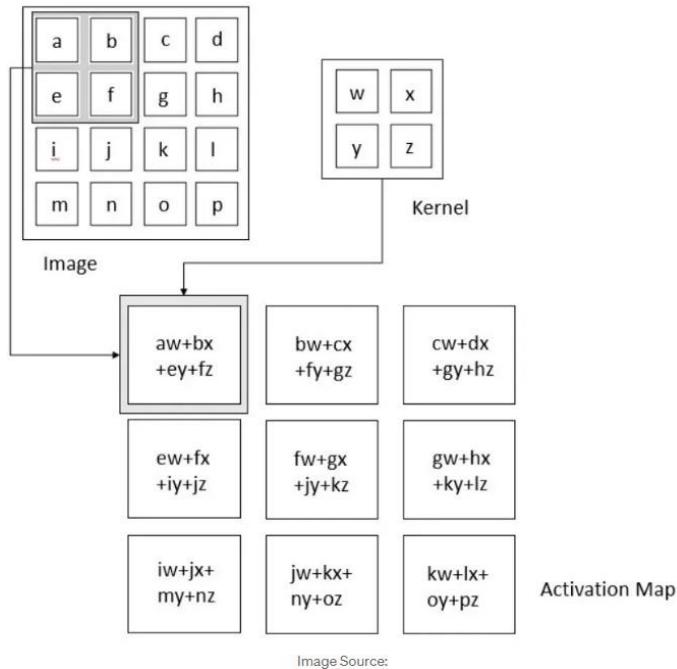
Now, let's delve into each of these layers in detail to gain a deeper understanding of their role and significance in Convolutional Neural Networks (CNN).

The Convolution Layer:

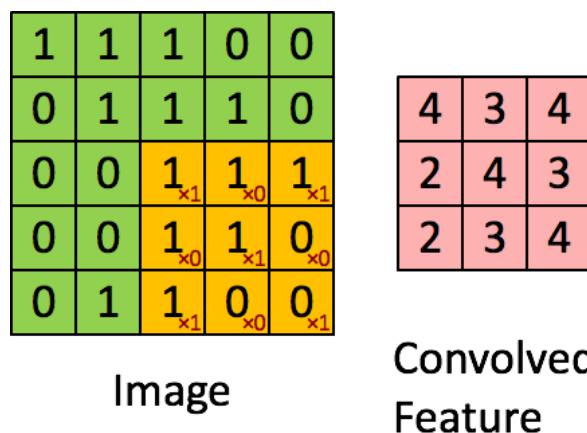
The convolutional layer serves as the fundamental building block within a Convolutional Neural Network (CNN), playing a central role in performing the majority of computations. It relies on several key components, including input data, filters, and feature maps.

Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. It is a tensor operation (dot product) where two tensors serve as input, and a resulting tensor is generated as the output. This layer employs a tile-like filtering approach on an input tensor using a small window known as a kernel. The kernel specifies the specific characteristics that the convolution operation seeks to filter, generating a significant response when it detects the desired features. To explore further details about various kernels and their functionalities, refer here.

The convolutional layer computes a dot product between the filter value and the image pixel values, and the matrix formed by sliding the filter over the image is called the **Convolved Feature, Activation Map, or Feature Map**.



Each element from one tensor (image pixel) is multiplied by the corresponding element (the element in the same position) of the second tensor(kernel value), and then all the values are summed to get the result.



The output size of the convoluted layer is determined by several factors, including the input size, kernel size, stride, and padding. The formula to calculate the output size is as follows:

$$H_{out} = 1 + \frac{H_{in} + (2 \cdot pad) - K_{height}}{s}, \quad W_{out} = 1 + \frac{W_{in} + (2 \cdot pad) - K_{width}}{s}$$

Image By Author: Output size of the convolution image

Let's take an example to better understand this concept. Imagine we have an input image with dimensions of 6x6 pixels. For the convolutional operation, we use a kernel with dimensions of 3x3 pixels, a stride of 1, and no padding (padding of 0).

To calculate the output size of the convoluted image, we can apply the following formula: $\text{output_size} = 1 + (\text{input_size} * \text{kernel_size} + (2 * \text{padding})) / \text{stride}$.

Plugging in the values, we get: $\text{output_size} = 1 + (6 * 3 + (2 * 0)) / 1 = 1 + (3 / 1) = 1 + 3 = 4$.

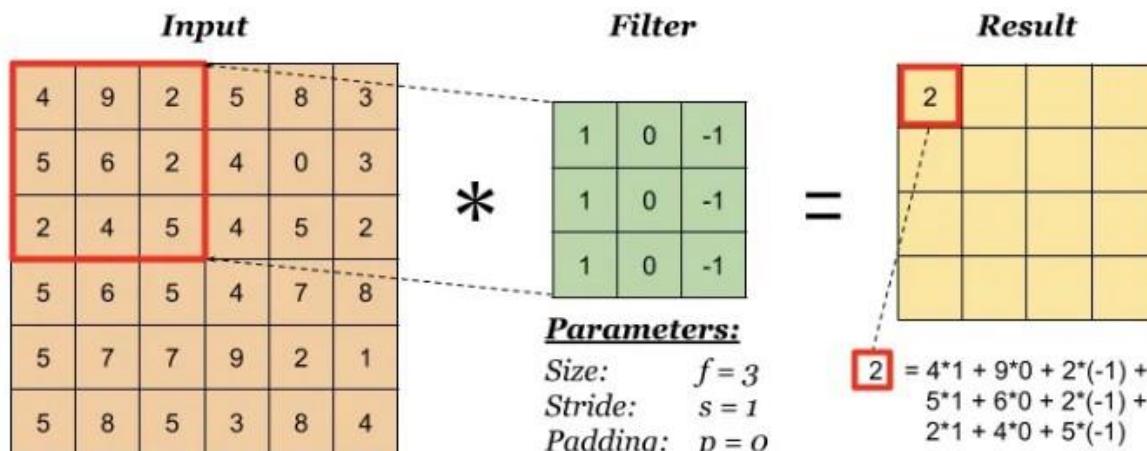


Image By Author: Convolution on 2D Image / Single Channel

Hence, the resulting convoluted image will have dimensions of 4x4 pixels.

When the input has more than one channel (e.g. an RGB image), the filter should have a matching number of channels. To calculate one output cell, perform convolution on each matching channel, then add the result together.

After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, introducing nonlinearity to the model.

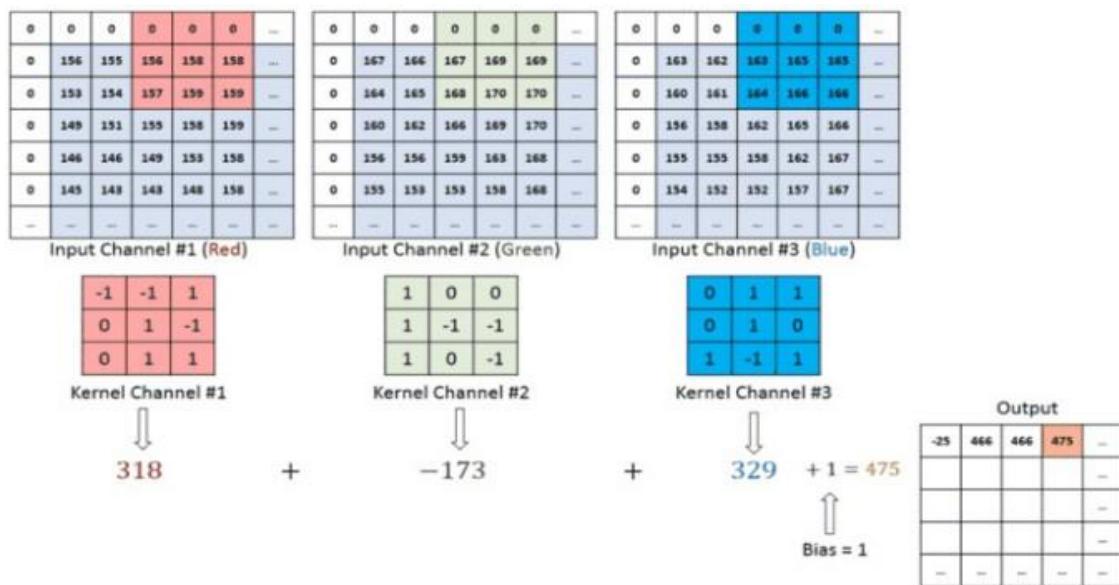


Image By Author: Convolution on RGB Image

The Pooling Layer:

Pooling layers, also referred to as downsampling, serve to reduce the dimensionality of the input, thereby decreasing the number of parameters. Similar to convolutional layers, pooling operations involve traversing a filter across the input. However, unlike convolutional layers, the pooling filter does not possess weights. Instead, the filter applies an aggregation function to the values within its receptive field, generating the output array. Two primary types of pooling are commonly employed:

Max Pooling: It selects the pixel with the maximum value to send to the output array.

Average pooling: It calculates the average value within the receptive field to send to the output array.

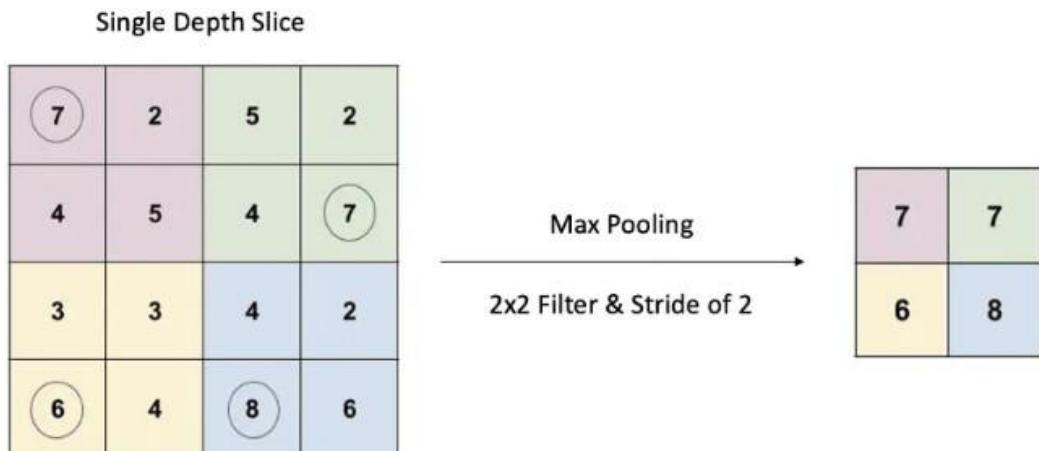
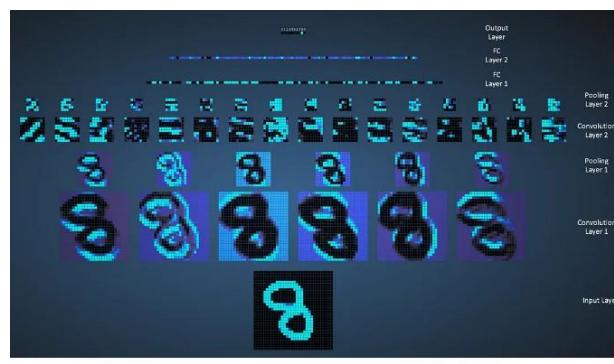


Image By Author: Max Pooling

Pooling offers a significant advantage in that it does not require learning any parameters. However, this attribute also presents a potential drawback as pooling may discard crucial information. While pooling serves to reduce dimensionality and extract key features, there is a possibility that important details can be lost during this process.

Fully-Connected Layer:

The Fully Connected Layer i.e dense layer aims to provide global connectivity between all neurons in the layer. Unlike convolutional and pooling layers, which operate on local spatial regions, the fully connected layer connects every neuron to every neuron in the previous and subsequent layers.

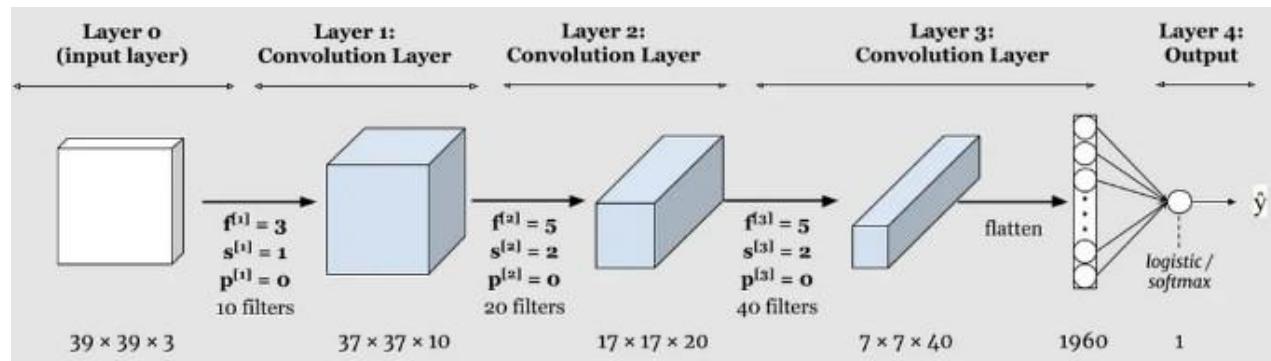


The fully connected layer typically appears at the end of the ConvNet architecture, taking the flattened feature maps from the preceding convolutional and pooling layers as input. Its purpose is to combine and transform these high-level features into the final output, such as

class probabilities or regression values, depending on the specific task. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

This layer converts a three-dimensional layer in the network into a one-dimensional vector to fit the input of a fully-connected layer for classification. For example, a $5 \times 5 \times 2$ tensor would be converted into a vector of size 50. This part is in principle the same as a regular Neural Network.

Now that we have explored the concepts of convolution, pooling, and fully connected layers individually, let's combine them to understand the basic architecture of a Convolutional Neural Network (CNN). In a typical CNN, the input data passes through a series of convolutional layers, which extract features using filters. The output of each convolutional layer is then downsampled using pooling layers to reduce dimensionality and capture the most salient information. Finally, the resulting feature maps are flattened and fed into one or more fully connected layers, which perform the classification or regression tasks.



A Sample Convnet: [Resource](#)

This combination of convolution, pooling, and fully connected layers forms the core structure of a CNN and enables it to learn and recognize complex patterns in images or other data.

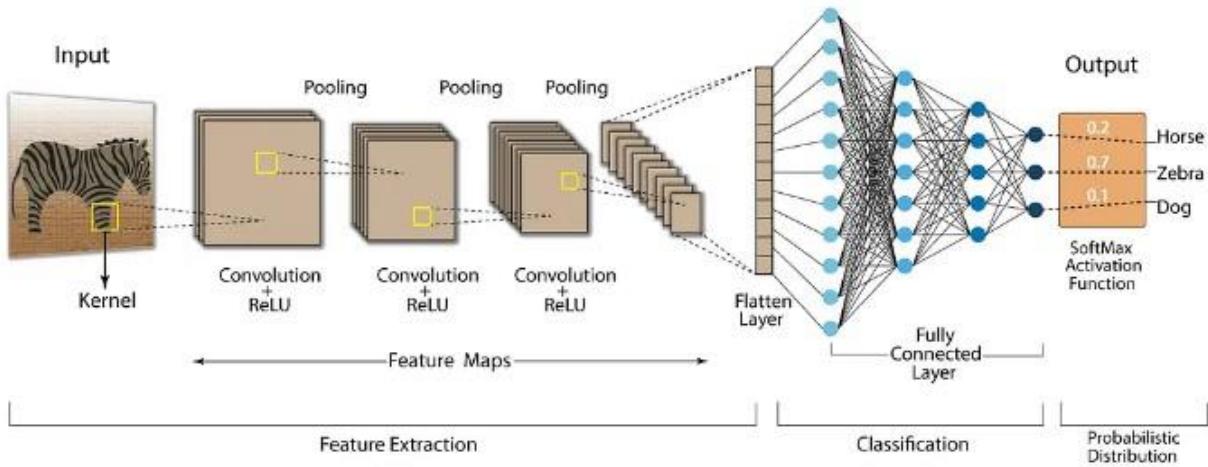


Image Source: [Analytics Vidhya](#)

Algorithm -

1. Load and Prepare Dataset:

Load the chosen image classification dataset (e.g., medical, agricultural) either from a local directory or using TensorFlow Datasets.

Split the dataset into training and validation sets.

Preprocess the images (resize, normalization) and preprocess labels (if necessary).

2. Define CNN Architecture:

Design the CNN architecture using TensorFlow's Keras API.

Construct the model with convolutional layers, pooling layers, fully connected layers, and appropriate activation functions.

Specify input shape, number of classes, and layer configurations.

3. Compile the Model:

Compile the model by specifying the optimizer, loss function, and evaluation metrics. Choose an appropriate optimizer (e.g., Adam) and a suitable loss function (e.g., sparse categorical crossentropy for multi-class classification).

4. Hyperparameter Tuning and Training:

Set hyperparameters like learning rate, batch size, and number of epochs. Train the CNN model on the training set using the fit function.

Utilize techniques like early stopping to prevent overfitting and monitor validation loss/accuracy.

5. Evaluate and Test:

Evaluate the trained model's performance on the validation set to assess its accuracy and generalization.

Once satisfied, use the model to predict and evaluate its accuracy on the test set to assess its real-world performance.

Application -

1. Medical Imaging:

Disease Detection: Identifying tumors, lesions, or anomalies in MRI, X-ray, or CT scans for conditions like cancer, fractures, or internal organ abnormalities.

Diagnosis Assistance: Analyzing retinal scans for diabetic retinopathy or identifying skin conditions through dermatology images.

2. Agriculture:

Crop Disease Identification: Classifying plant images to detect diseases, nutrient deficiencies, or pest infestations in crops using systems like Plant Village.

Weed Detection: Identifying and distinguishing weeds from crops to enable targeted herbicide application.

3. Autonomous Vehicles:

Object Recognition: Recognizing pedestrians, traffic signs, and other vehicles for safe navigation and decision-making in self-driving cars.

Lane Detection: Identifying lane markings for autonomous vehicle guidance.

4. Retail and E-commerce:

Product Classification: Categorizing products for inventory management and cataloging in e-commerce platforms.

Visual Search: Enabling visual search capabilities to find similar products using images captured by users.

5. Security and Surveillance:

Facial Recognition: Verifying identities or identifying individuals in security systems or surveillance footage.

Anomaly Detection: Spotting suspicious activities or objects in surveillance images or videos.

Inference -

The process involves training the model, preprocessing image, prediction, evaluation and iteration.

The use of different architectures such as Simple Neural Network, CNN allows for comparison and analysis of different images for image classification.

Optimization with different hyper-parameters including learning rate, filter size, no. of layers, optimizers, dropouts, etc. are understood in this practical.

Code -

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0

# Add channel dimension to the images
train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

# Split the dataset into training and validation sets
train_images, val_images, train_labels, val_labels = train_test_split(
    train_images, train_labels, test_size=0.1, random_state=42
)

# Data augmentation for training images
datagen = ImageDataGenerator(rotation_range=10, zoom_range=0.1,
                             width_shift_range=0.1,
                             height_shift_range=0.1)
datagen.fit(train_images)

# Create a CNN model with hyperparameter tuning and regularization
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model with data augmentation
history = model.fit(datagen.flow(train_images, train_labels, batch_size=64),
```

```
epochs=20, validation_data=(val_images, val_labels)) # Evaluate the
model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"Test Accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Output -

```
11490434/11490434 [=====] - 0s 0us/step
Epoch 1/20
844/844 [=====] - 87s 100ms/step - loss: 0.3852 - accuracy:
0.8760 - val_loss: 0.0543 - val_accuracy: 0.9838 Epoch
2/20
844/844 [=====] - 83s 98ms/step - loss: 0.1320 - accuracy:
0.9589 - val_loss: 0.0491 - val_accuracy: 0.9847 Epoch
3/20
844/844 [=====] - 72s 85ms/step - loss: 0.0998 - accuracy:
0.9693 - val_loss: 0.0361 - val_accuracy: 0.9892 Epoch
4/20
844/844 [=====] - 70s 83ms/step - loss: 0.0812 - accuracy:
0.9747 - val_loss: 0.0391 - val_accuracy: 0.9875 Epoch
5/20
844/844 [=====] - 71s 84ms/step - loss: 0.0727 - accuracy:
0.9775 - val_loss: 0.0308 - val_accuracy: 0.9910 Epoch
6/20
844/844 [=====] - 70s 83ms/step - loss: 0.0639 - accuracy:
0.9804 - val_loss: 0.0317 - val_accuracy: 0.9907 Epoch
7/20
844/844 [=====] - 70s 83ms/step - loss: 0.0563 - accuracy:
0.9826 - val_loss: 0.0370 - val_accuracy: 0.9890 Epoch
8/20
844/844 [=====] - 70s 83ms/step - loss: 0.0557 - accuracy:
0.9829 - val_loss: 0.0283 - val_accuracy: 0.9920 Epoch
9/20
844/844 [=====] - 71s 84ms/step - loss: 0.0507 - accuracy:
0.9847 - val_loss: 0.0269 - val_accuracy: 0.9927 Epoch
10/20
```

844/844 [=====] - 74s 87ms/step - loss: 0.0497 - accuracy: 0.9845 - val_loss: 0.0300 - val_accuracy: 0.9922 Epoch

11/20

844/844 [=====] - 71s 84ms/step - loss: 0.0465 - accuracy: 0.9859 - val_loss: 0.0259 - val_accuracy: 0.9928 Epoch

12/20

844/844 [=====] - 72s 85ms/step - loss: 0.0432 - accuracy: 0.9870 - val_loss: 0.0336 - val_accuracy: 0.9915 Epoch

13/20

844/844 [=====] - 71s 84ms/step - loss: 0.0427 - accuracy: 0.9863 - val_loss: 0.0300 - val_accuracy: 0.9925 Epoch

14/20

844/844 [=====] - 70s 83ms/step - loss: 0.0429 - accuracy: 0.9872 - val_loss: 0.0254 - val_accuracy: 0.9923 Epoch

15/20

844/844 [=====] - 70s 83ms/step - loss: 0.0380 - accuracy: 0.9880 - val_loss: 0.0321 - val_accuracy: 0.9913 Epoch

16/20

844/844 [=====] - 68s 81ms/step - loss: 0.0385 - accuracy: 0.9880 - val_loss: 0.0283 - val_accuracy: 0.9922 Epoch

17/20

844/844 [=====] - 72s 85ms/step - loss: 0.0357 - accuracy: 0.9884 - val_loss: 0.0243 - val_accuracy: 0.9932 Epoch

18/20

844/844 [=====] - 70s 83ms/step - loss: 0.0381 - accuracy: 0.9878 - val_loss: 0.0264 - val_accuracy: 0.9937 Epoch

19/20

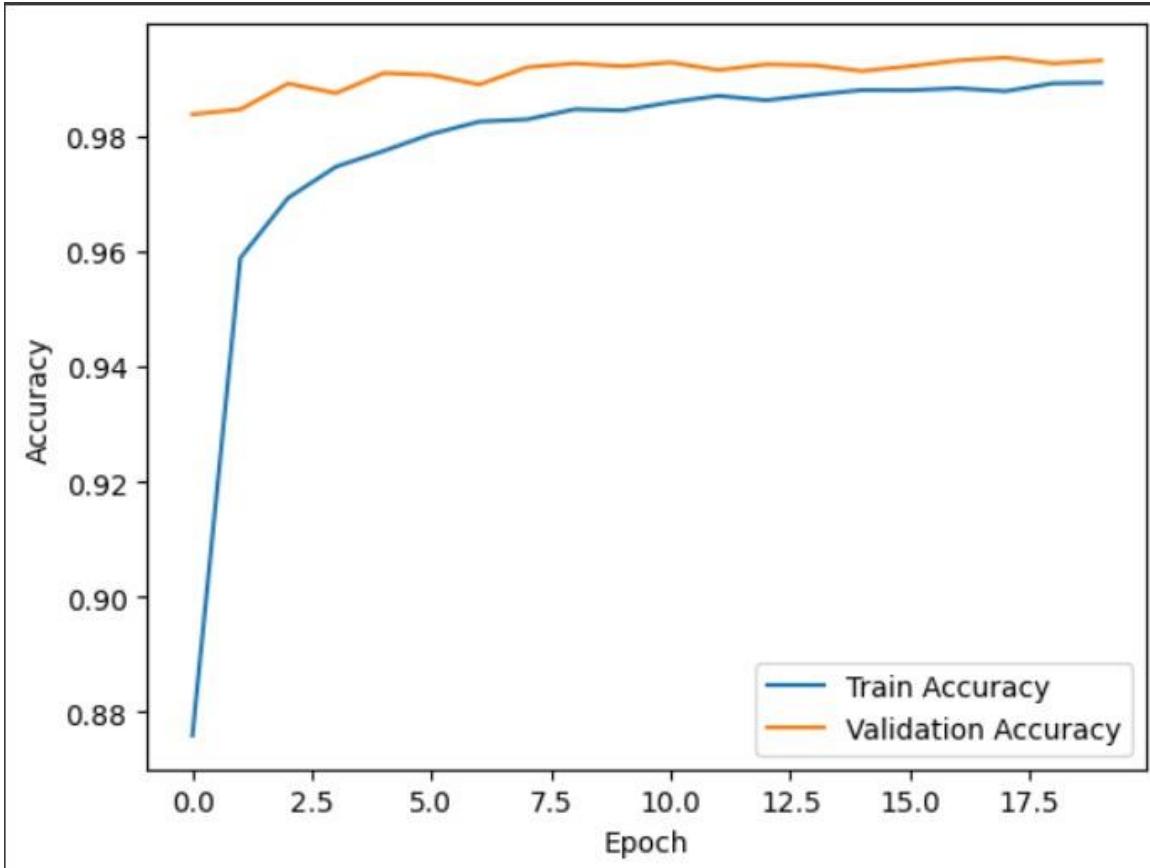
844/844 [=====] - 70s 82ms/step - loss: 0.0352 - accuracy: 0.9892 - val_loss: 0.0259 - val_accuracy: 0.9927 Epoch

20/20

844/844 [=====] - 71s 84ms/step - loss: 0.0351 - accuracy: 0.9893 - val_loss: 0.0233 - val_accuracy: 0.9932

313/313 [=====] - 4s 12ms/step - loss: 0.0162 - accuracy: 0.9951

Test Accuracy: 0.9951000213623047



References:

<https://medium.com/codex/understanding-convolutional-neural-networks-a-beginners-journey-into-the-architecture-aab30dfac10>

<https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

Conclusion: Thus Designed and implemented a CNN for Image Classification.

ASSIGNMENT No: 05

Title: Design and implement Deep Convolutional GAN to generate images of faces/digits from a set of given images.

Problem Statement: Design and implement Deep Convolutional GAN to generate images of faces/digits from a set of given images.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn Design and implement Deep Convolutional GAN to generate images of faces/digits from a set of given images.

Outcomes:

After completion of this assignment students are able to Design and implement Deep Convolutional GAN to generate images of faces/digits from a set of given images.

Theory:

Deep Convolutional Generative Adversarial Network (DCGAN):

Deep Convolutional Generative Adversarial Network (DCGAN) represents a breakthrough in generative models, specifically designed for image generation tasks. DCGANs are an extension of the traditional GAN architecture, tailored for generating high-quality, coherent images.

Architecture:

Generator:

The generator is responsible for synthesizing realistic images from random noise. It employs a series of transposed convolutional layers to transform the input noise into a complex image.

The generator's architecture typically consists of fractional-strided convolutions, batch normalization, and rectified linear unit (ReLU) activations. These architectural choices aid in preventing issues like mode collapse and vanishing gradients.

Discriminator:

The discriminator is a binary classifier tasked with distinguishing between real and generated images. It utilizes convolutional layers to extract hierarchical features from the input images. Similar to the generator, batch normalization and Leaky ReLU activations are commonly used in the discriminator to ensure stable training.

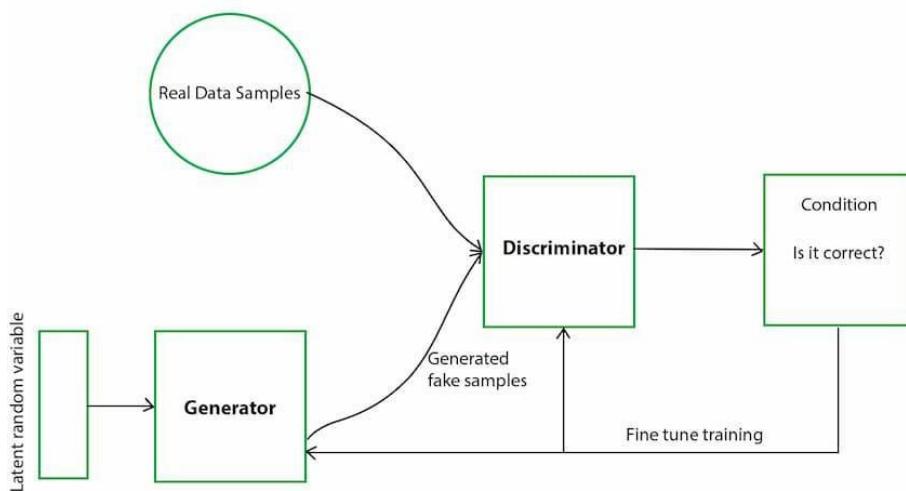
Key Design Principles:

Strided convolutions: Enable the network to learn spatial hierarchies effectively.

Batch normalization: Promotes stable and accelerated training by normalizing the input of each layer.

Leaky ReLU activations: Prevents the issue of "dying ReLU" by allowing a small, non-zero gradient for negative input values.

Transposed convolutions: Essential for upsampling the input noise and generating high-resolution images.



Training Process:

Input Noise:

The generator takes random noise as input, typically sampled from a Gaussian distribution. This noise is transformed into a synthetic image.

Adversarial Training:

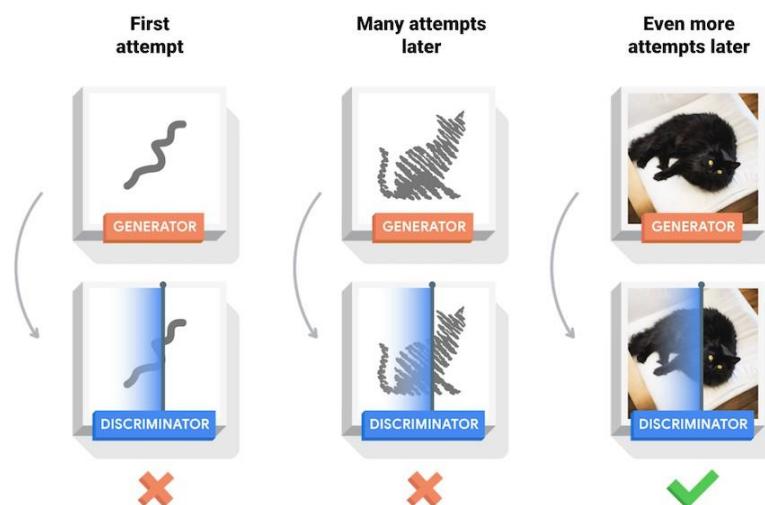
The discriminator evaluates both real and generated images, providing feedback to the generator. The generator aims to create images that are indistinguishable from real ones.

Discriminator Feedback:

The discriminator is trained to correctly classify real and generated images. It learns to distinguish subtle patterns and features in the images.

Generator Improvement:

The generator adjusts its parameters based on the feedback from the discriminator. This adversarial process continues iteratively, leading to the refinement of both networks.

**Algorithm -**

1. Initialize the generator and discriminator models with the specified architectures
2. Prepare the dataset for training, ensuring proper normalization and preprocessing.
3. Train the discriminator using real and generated images, updating its parameters.
4. Train the generator to produce realistic images that can deceive the discriminator
5. Iterate between discriminator and generator training to refine their capabilities over epochs.

Application -

1. Artistic image generation.
2. Image-to-image translation tasks.
3. Data augmentation processes.

Inference -

The GAN experiment involves training a generator and discriminator, optimizing their performance. The generator creates synthetic images, while the discriminator distinguishes between real and fake. Training alternates, aiming for the generator to produce realistic images. The final model can generate diverse images from noise, showcasing the GAN's ability in image synthesis.

Code -

```
from keras.models import Model from
keras.layers import Input,Dense import
numpy as np
import pandas as pd import
keras.backend as K
import matplotlib.pyplot as plt from
keras import preprocessing
from keras.models import Sequential #from
keras.layers import
Conv2D,Dropout,Dense,Flatten,Conv2DTranspose,BatchNormalization,LeakyReLU,Reshape import
tensorflow as tf
from keras.layers import *
from keras.datasets import fashion_mnist
(train_x, train_y), (val_x, val_y) = fashion_mnist.load_data() train_x
= train_x/255.
val_x = val_x/255.
train_x=train_x.reshape(-1,28,28,1)
print(train_x.shape)
#train_x = train_x.reshape(-1, 784)
#val_x = val_x.reshape(-1, 784)
fig,axe=plt.subplots(2,2)
idx = 0
for i in range(2): for
    j in range(2):
        axe[i,j].imshow(train_x[idx].reshape(28,28),cmap='gray') idx+=1
train_x = train_x*2 - 1
print(train_x.max(),train_x.min()) generator
= Sequential()
generator.add(Dense(512,input_shape=[100]))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))
generator.add(Dense(256))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))
generator.add(Dense(128))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))
generator.add(Dense(784)) generator.add(Reshape([28,28,1]))
generator.summary()
discriminator = Sequential()
discriminator.add(Dense(1,input_shape=[28,28,1]))
```

```
discriminator.add(Flatten()) discriminator.add(Dense(256))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.5))
discriminator.add(Dense(128))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.5))
discriminator.add(Dense(64))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.5))
discriminator.add(Dense(1,activation='sigmoid'))
discriminator.summary()
GAN =Sequential([generator,discriminator])
discriminator.compile(optimizer='adam',loss='binary_crossentropy')
discriminator.trainable = False
GAN.compile(optimizer='adam',loss='binary_crossentropy') GAN.summary()
epochs = 30
batch_size = 100
noise_shape=100
with tf.device('/gpu:0'):
    for epoch in range(epochs):
        print(f"Currently on Epoch {epoch+1}")

        for i in range(train_x.shape[0]/batch_size): if
            (i+1)% 100 == 0:
                print(f"\tCurrently on batch number {i+1} of {train_x.shape[0]/batch_size}")

        noise=np.random.normal(size=[batch_size,noise_shape])

        gen_image = generator.predict_on_batch(noise)

        train_dataset = train_x[i*batch_size:(i+1)*batch_size]

#training discriminator on real images
train_label=np.ones(shape=(batch_size,1))
#train_label=np.ones((batch_size, 1))
discriminator.trainable = True
#train_dataset=train_x[idx]
d_loss_real=discriminator.train_on_batch(train_dataset,train_label) #training
discriminator on fake images train_label=np.zeros(shape=(batch_size,1))
d_loss_fake=discriminator.train_on_batch(gen_image,train_label)

#training generator
noise=np.random.normal(size=[batch_size,noise_shape])
train_label=np.ones(shape=(batch_size,1))
discriminator.trainable = False

d_g_loss_batch =GAN.train_on_batch(noise, train_label)

#plotting generated images at the start and then after every 10 epoch if epoch
% 10 == 0:
```

```

samples = 10
x_fake = generator.predict(np.random.normal(loc=0, scale=1, size=(samples, 100)))

for k in range(samples):
    plt.subplot(2, 5, k+1)
    plt.imshow(x_fake[k].reshape(28, 28), cmap='gray')
    plt.xticks([])
    plt.yticks([])

plt.tight_layout()
plt.show()

print('Training is complete')
noise=np.random.normal(size=[10,noise_shape]) gen_image
= generator.predict(noise) plt.imshow(noise)
plt.title('How the noise looks')
fig,axe=plt.subplots(2,5)
fig.suptitle('Generated Images from Noise using GANs') idx=0
for i in range(2): for
    j in range(5):
        axe[i,j].imshow(gen_image[idx].reshape(28,28),cmap='gray') idx+=1

```

Output –

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 28, 28, 1)	2
flatten (Flatten)	(None, 784)	0
dense_5 (Dense)	(None, 256)	200960
leaky_re_lu_3 (LeakyReLU)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 128)	32896
leaky_re_lu_4 (LeakyReLU)	(None, 128)	0
dropout_1 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 64)	8256

leaky_re_lu_5 (LeakyReLU)	(None, 64)	0
dropout_2 (Dropout)	(None, 64)	0
dense_8 (Dense)	(None, 1)	65

Total params: 242179 (946.01 KB)
Trainable params: 242179 (946.01 KB)
Non-trainable params: 0 (0.00 Byte)

Model: "sequential_2"

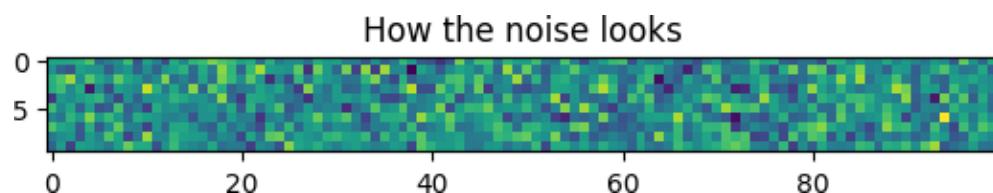
Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 28, 28, 1)	320656
sequential_1 (Sequential)	(None, 1)	242179

Total params: 562835 (2.15 MB)
Trainable params: 318864 (1.22 MB)
Non-trainable params: 243971 (953.01 KB)

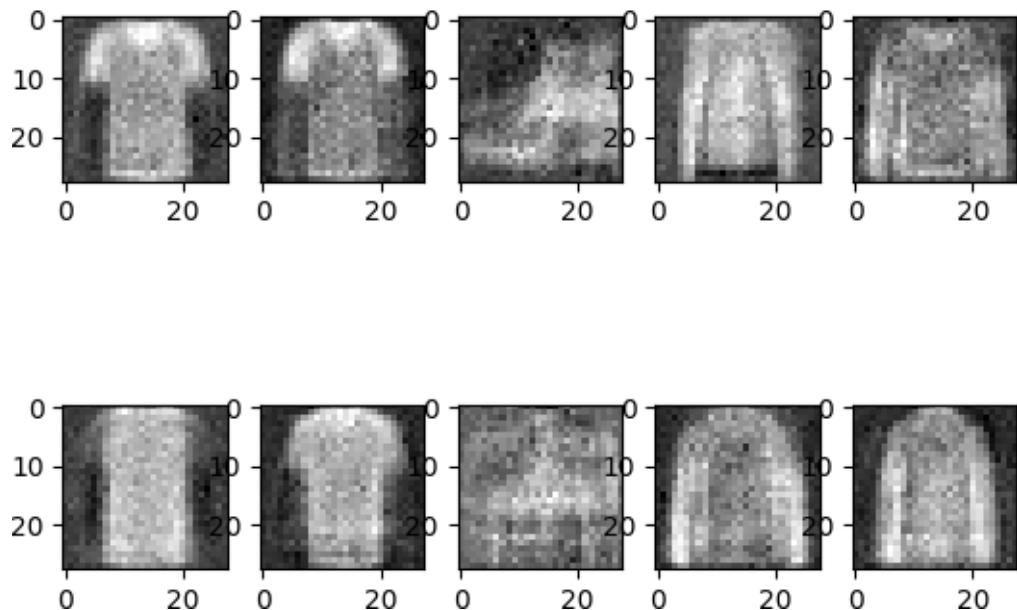
Currently on batch number 600 of 600
Currently on Epoch 30

Currently on batch number 100 of 600
Currently on batch number 200 of 600
Currently on batch number 300 of 600
Currently on batch number 400 of 600
Currently on batch number 500 of 600
Currently on batch number 600 of 600

Training is complete



Generated Images from Noise using GANs



References-

<https://www.tensorflow.org/tutorials/generative/dcgan>

Conclusion: Thus Designed and implemented Deep Convolutional GAN to generate images of faces/digits from a set of given images.

ASSIGNMENT No: 06

Title:- Perform sentiment analysis with a recurrent neural networks RNN

Mapping with Syllabus -

Unit 4

Objective -

Implement a Recurrent Neural Network (RNN) on a network graph for sentiment analysis.

Outcome -

Solve the language translation problem by Recurrent neural network(RNN)

Software Requirements -

- Python (3.x recommended)
- Jupyter Notebook or any Python IDE

Hardware Requirements -

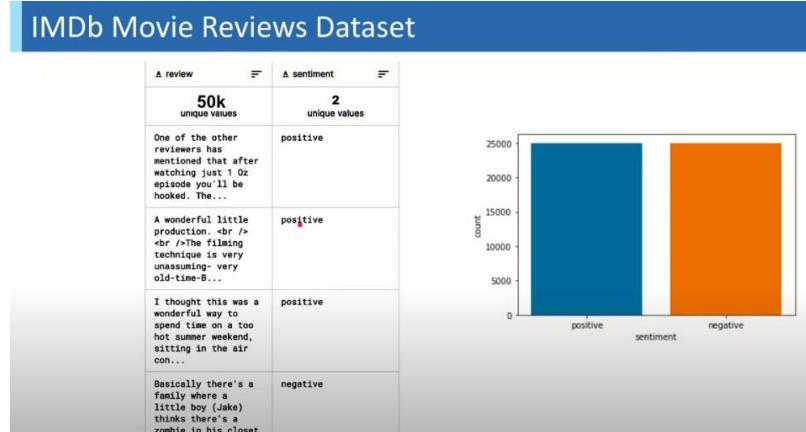
A machine with sufficient RAM and processing power for model training (8GB RAM recommended)

Prerequisites -

- Basic understanding of Python programming
- Familiarity with the concepts of Neural Networks

Dataset -

https://github.com/skillcate/sentiment-analysis-with-deep-neural-networks/blob/main/a1_IMDB_Dataset.csv

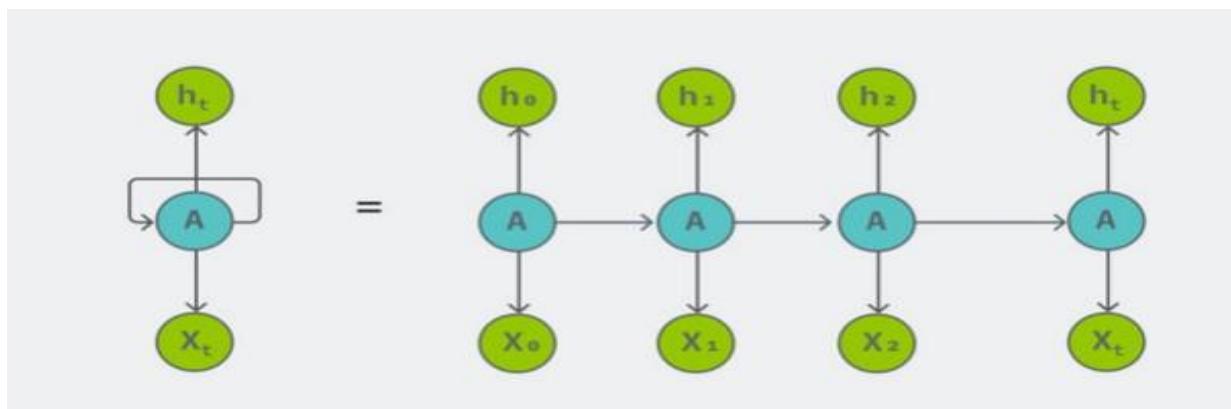


Libraries or Modules Used-

- TensorFlow or PyTorch
- NumPy
- Matplotlib

Theory -

A Recurrent Neural Network (RNN) represents a sophisticated extension of the traditional feedforward neural network by incorporating an internal memory mechanism. In contrast to feedforward networks, RNNs possess a recurrent nature, executing the same operation for each input data while considering the outcome of the preceding computation. This iterative process involves generating an output, which is then replicated and fed back into the recurrent network. In making decisions, the RNN takes into account both the current input and the knowledge acquired from the previous input.



RNNs, uniquely suited for tasks like unsegmented connected handwriting recognition or speech

recognition, leverage their internal state (memory) to effectively process sequences of inputs. Unlike other neural networks where inputs are treated independently, RNNs establish a relationship among all inputs in a sequence.

The sequence begins with the initial input, $X(0)$, producing an output, $h(0)$, which, together with $X(1)$, becomes the input for the subsequent step. This process continues iteratively, with each hidden state (h) from the previous step being combined with the current input (X) for the next computation. This sequential progression ensures that the network retains contextual information throughout the training process.

The formula for the current state involves weight matrices (W) for the current input (W_{hx}) and the previous hidden state (W_{hh}), alongside an activation function, typically \tanh . This activation function introduces non-linearity, compressing activations within the range of $[-1, 1]$. The final output state (Y_t) is determined by the weight matrix (W_{hy}) at the output layer.

$$h_t = f(h_{t-1}, x_t)$$

Applying Activation Function:

$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

y_t is the output state. W_{hy} is the weight at the output state.

In essence, RNNs excel at learning and remembering dependencies between inputs across sequential data, making them invaluable for tasks that involve context and temporal relationships.

Algorithm –

Applying Recurrent Neural Networks (RNNs) to network graphs for sentiment analysis involves a step-by-step process. Here's a high-level algorithm:

1. Data Preparation:

- Collect a dataset with network graph data and sentiment labels for each node or edge.
- Represent the graph as an adjacency matrix or an edge list.
- Assign sentiment labels to nodes or edges in the graph.

2. Node/Edge Embedding:

- Represent each node or edge in the graph using embeddings. You can use techniques like Word2Vec, GloVe, or Graph Embeddings (e.g., GraphSAGE) to convert nodes into fixed-size vectors.

3. Sequence Generation:

- Create sequences of node or edge embeddings for each path or subgraph in the network. The sequence length can vary based on the application and context.

4. Labeling:

- Assign sentiment labels to the sequences based on the sentiment of the nodes or edges in the sequence. This is your target variable for training the RNN.

5. RNN Model Architecture:

- Choose an RNN architecture suitable for sequence processing. Long Short-

rm Memory (LSTM) or Gated Recurrent Unit (GRU) cells are often used due to their ability to capture long-range dependencies.

- Define the input layer to accept sequences of node or edge embeddings.
- Optionally, include additional layers like Bidirectional RNNs or attention mechanisms for improved performance.

6. Training:

- Split the dataset into training, validation, and test sets.
- Train the RNN model on the training set using backpropagation through time (BPTT) or other sequence training methods.
- Optimize the model using an appropriate loss function (e.g., categorical cross-entropy) and an optimizer (e.g., Adam).

7. Evaluation:

- Evaluate the trained model on the validation set to tune hyperparameters and prevent overfitting.
- Test the model on the test set to assess its performance on unseen data.

8. Inference:

- Use the trained RNN model for sentiment analysis on new or unseen network graphs.

9. Post-Processing:

- Analyze the results, and if needed, perform post-processing or fine-tuning to improve the model's accuracy.

10. Visualization (Optional):

- Visualize the sentiment predictions on the network graph to gain insights into sentiment patterns and relationships.

Remember that the specifics of the algorithm may vary based on the exact nature of your network graph data and the sentiment analysis task at hand. Adjustments may be needed based on the characteristics of your dataset and the desired outcomes.

Let's understand using an example:

Let's break down the algorithm with a simplified example of sentiment analysis on a network graph. In this example, we'll consider a social network where nodes represent users, and edges represent connections between users. The task is to predict the sentiment (positive or negative) of interactions between users.

Example:

1. Data Preparation:

- Collect a dataset with a social network graph, where each node represents a user and each edge represents a connection.
- Assign sentiment labels (positive or negative) to edges based on the sentiment of interactions between users.

2. Node Embedding:

- Represent each user (node) with Word2Vec embeddings based on their interactions or posts in the social network.

3. Sequence Generation:

- Create sequences of node embeddings for each path of interactions between users. For example, if we have the following interactions: A -> B (positive), B -> C (negative), C -> D (positive), the sequence for user A could be [A_embedding, B_embedding].

4. Labeling:

- Assign sentiment labels to the sequences. For instance, if the sequence is [A_embedding, B_embedding] and the sentiment of the interaction B -> C is negative, then the label for this sequence is negative.

5. RNN Model Architecture:

- Use an LSTM-based RNN architecture for sequence processing.
- Define the input layer to accept sequences of user embeddings.
- Add LSTM layers to capture the temporal dependencies in the sequence.

6. Training:

- Split the dataset into training, validation, and test sets.
- Train the RNN model on the training set using backpropagation through time (BPTT).
- Optimize the model using categorical cross-entropy as the loss function and the Adam optimizer.

7. Evaluation:

- Evaluate the trained model on the validation set to tune hyperparameters and prevent overfitting.
- Test the model on the test set to assess its performance on unseen interactions.

8. Inference:

- Use the trained RNN model for sentiment analysis on new interactions between users.
9. Post-Processing:
- Analyze the results, and if needed, perform post-processing or fine-tuning to improve the model's accuracy.
10. Visualization (Optional):
- Visualize the predicted sentiments on the social network graph to understand sentiment patterns and relationships between users.

This example outlines a simplified algorithm for sentiment analysis on a social network graph using an RNN. Keep in mind that real-world scenarios may involve more complex data preprocessing, model tuning, and feature engineering based on the characteristics of your dataset.

Application -

1. Social Media Sentiment Analysis:
 - ❖ Analyzing sentiments in interactions between users on social media platforms.
 - ❖ Identifying positive or negative sentiments in comments, replies, and discussions.
2. Online Community Monitoring:
 - ❖ Monitoring sentiments within online communities or forums.
 - ❖ Identifying potential issues, conflicts, or positive interactions within community discussions.
3. Brand Reputation Management:
 - ❖ Analyzing sentiments related to a brand by examining interactions between users mentioning the brand.
 - ❖ Monitoring sentiment trends over time and identifying areas for improvement.
4. Customer Feedback Analysis:
 - ❖ Analyzing sentiments in customer reviews and feedback.
 - ❖ Understanding the overall sentiment regarding a product or service.
5. Collaborative Filtering in Recommendation Systems:
 - ❖ Utilizing sentiments in user-item interactions to improve collaborative filtering recommendation systems.
 - ❖ Personalizing recommendations based on both historical interactions and sentiments.
6. Financial News Analysis:
 - ❖ Analyzing sentiments in financial news articles and discussions between investors.
 - ❖ Understanding market sentiment and potential impacts on stock prices.

7. Healthcare Interactions:

- ❖ Analyzing sentiments in interactions between patients and healthcare professionals.
- ❖ Identifying trends in patient satisfaction or areas for improvement in healthcare services.

8. Educational Platforms:

- ❖ Analyzing sentiments in interactions between students and instructors on educational platforms.
- ❖ Monitoring student engagement and identifying potential issues or areas for improvement.

9. Political Discourse Analysis:

- ❖ Analyzing sentiments in interactions between individuals discussing political topics.
- ❖ Identifying trends in public opinion and potential areas of concern.

10. Influencer Marketing:

- ❖ Analyzing sentiments in interactions between influencers and their followers.
- ❖ Understanding the impact of influencer content on audience sentiment.

These applications highlight the versatility of sentiment analysis using RNNs on network graphs, where the sequential nature of interactions and relationships plays a crucial role in understanding sentiments in various domains.

Inference -

For sentiment analysis with RNNs, new data is processed by a pre-trained model, generating predictions. Post-processing and analysis follow, with optional visualizations. A feedback loop may improve the model, allowing real-time sentiment analysis in production.

Code -

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Set the parameters
max_features = 10000 # Number of words to consider as features
maxlen = 100 # Cut texts after this number of words (among top max_features most common words)
batch_size = 32

# Load the IMDB dataset
```

```
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Pad sequences to have a consistent length for the input to the RNN x_train =
pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

# Build the RNN model with LSTM model
= Sequential()
model.add(Embedding(max_features, 128)) model.add(LSTM(64, dropout=0.2,
recurrent_dropout=0.2)) model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])

# Train the model model.fit(x_train,
y_train,
batch_size=batch_size, epochs=5,
validation_data=(x_test, y_test))

# Evaluate the model
score, acc = model.evaluate(x_test, y_test, batch_size=batch_size) print(f'Test score: {score}')
print(f'Test accuracy: {acc}')
```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [=====] - 0s 0us/step
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.

Epoch 1/5
782/782 [=====] - 319s 401ms/step - loss: 0.4161 - accuracy: 0.8074 - val_loss: 0.3585 - val_accuracy: 0.8412

Epoch 2/5
782/782 [=====] - 288s 368ms/step - loss: 0.2625 - accuracy: 0.8950 - val_loss: 0.3482 - val_accuracy: 0.8454

Epoch 3/5
782/782 [=====] - 284s 363ms/step - loss: 0.1931 - accuracy: 0.9244 - val_loss: 0.4158 - val_accuracy: 0.8375

Epoch 4/5
782/782 [=====] - 285s 365ms/step - loss: 0.1431 - accuracy: 0.9472 - val_loss: 0.4504 - val_accuracy: 0.8412

Epoch 5/5
782/782 [=====] - 287s 367ms/step - loss: 0.1093 - accuracy: 0.9606 - val_loss: 0.4790 - val_accuracy: 0.8413

782/782 [=====] - 25s 32ms/step - loss: 0.4790 - accuracy: 0.8413

Test score: 0.47897636890411377
Test accuracy: 0.8413199782371521

References -

<https://www.geeksforgeeks.org/what-is-sentiment-analysis/amp/>

<https://www.google.com/amp/s/www.geeksforgeeks.org/introduction-to-network/amp/> recurrent-neural-

<https://www.analyticsvidhya.com/blog/2022/01/sentiment-analysis-with-lstm/>

Conclusion: Thus Performed Sentiment Analysis in the network graph using RNN.

ASSIGNMENT No: 07

Title: Import Data from different Sources such as (Excel, Sql Server, Oracle etc.) and load in targeted system.

Problem Statement: Import Data from different Sources such as (Excel, Sql Server, Oracle etc.) and load in targeted system.

Prerequisite:

Basics of Python

Software Requirements: Power BI Tool

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to import data from different sources such as(Excel, Sql Server, Oracle etc.) and load in targeted system.

Outcomes:

After completion of this assignment students are able to understand how to import data from different sources such as(Excel, Sql Server, Oracle etc.) and load in targeted system.

Theory:

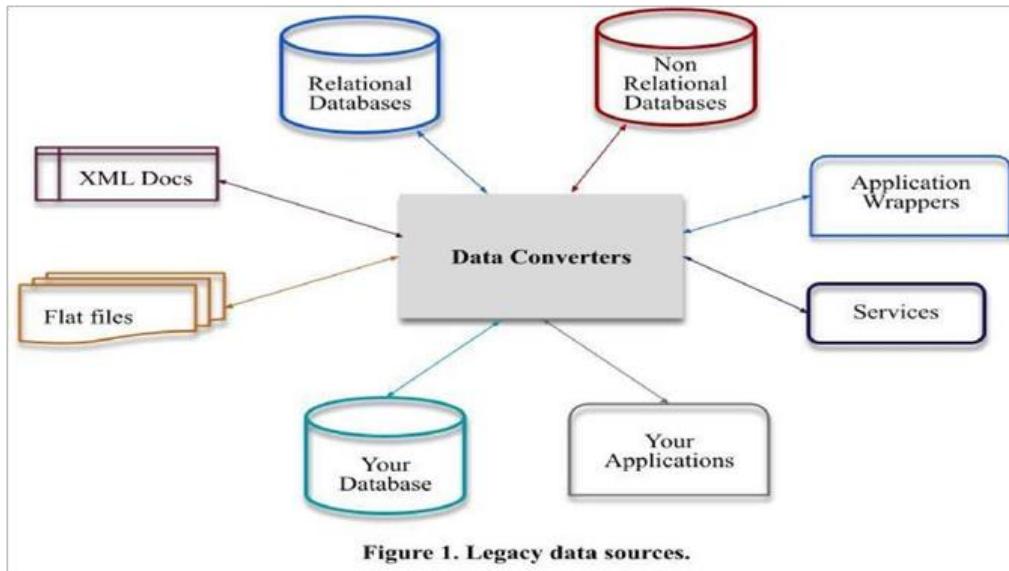
What is Legacy Data?

Legacy data, according to Business Dictionary, is "information maintained in an old or out-of-date format or computer system that is consequently challenging to access or handle."

Sources of Legacy Data

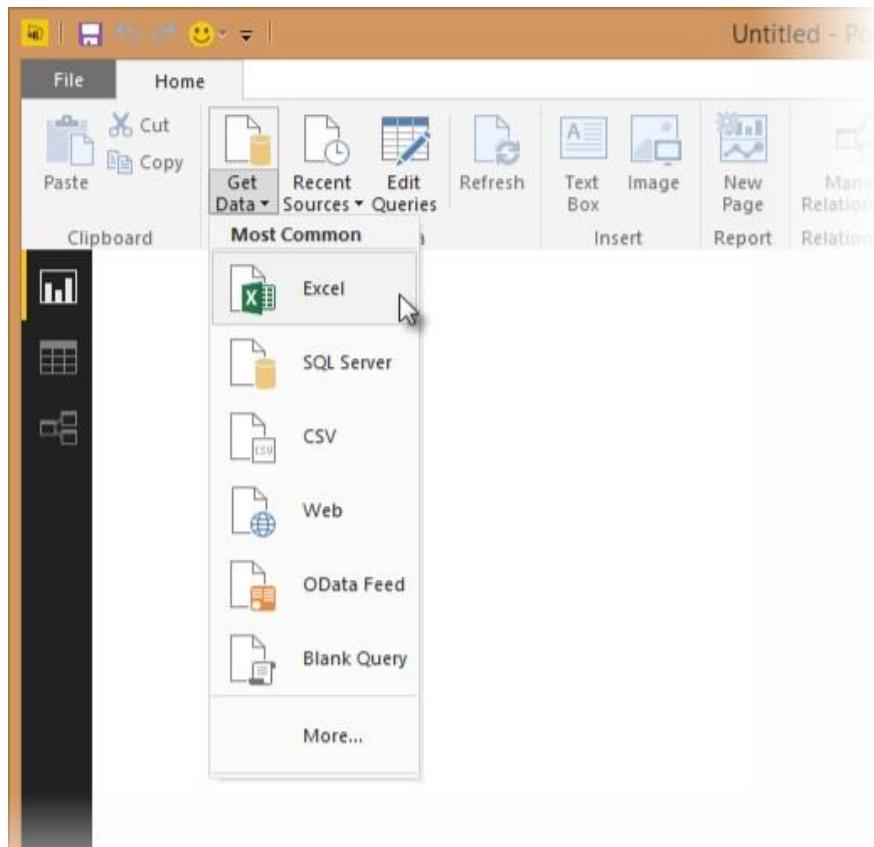
Where does legacy data come from? Virtually everywhere. Figure 1 indicates that there are many sources from which you may obtain legacy data. This includes existing databases, often relational, although non-RDBs such as hierarchical, network, object, XML, object/relational databases, and NoSQL databases. Files, such as XML documents or "flat files" such as configuration files and comma-delimited text files, are also common sources of legacy data. Software, including legacy applications that have been wrapped (perhaps via CORBA) and legacy services such as web services or CICS transactions, can also provide

access to existing information. The point to be made is that there is often far more to gaining access to legacy data than simply writing an SQL query against an existing relational database.



Importing Excel Data

- 1) Launch Power BI Desktop.
- 2) From the Home ribbon, select Get Data. Excel is one of the Most Common data connections, so you can select it directly from the Get Data menu.



- 3) If you select the Get Data button directly, you can also select File > Excel and select Connect.
- 4) In the Open File dialog box, select the Products.xlsx file.
- 5) In the Navigator pane, select the Products table and then select Edit.

The screenshot shows the Power BI Desktop interface. On the left, the Navigator pane displays a list of files: 'Products.xlsx [2]' (selected), 'Products' (unchecked), and 'Sheet1'. The main area is titled 'Products' and shows a preview of the data with the message 'Preview downloaded on Friday'. A table is displayed with the following columns: ProductID, ProductName, SupplierID, CategoryID, and Unit. The data consists of 22 rows of product information. A pink arrow points from the text 'In the Navigator pane, select the Orders table, and then select Edit.' to the 'Edit' button at the bottom right of the table preview.

ProductID	ProductName	SupplierID	CategoryID	Unit
1	Chai	1	1	10
2	Chang	1	1	24
3	Aniseed Syrup	1	2	15
4	Chef Anton's Cajun Seasoning	2	2	48
5	Chef Anton's Gumbo Mix	2	2	36
6	Grandma's Boysenberry Spread	3	2	14
7	Uncle Bob's Organic Dried Pears	3	7	12
8	Northwoods Cranberry Sauce	3	2	12
9	Mishi Kobe Niku	4	6	18
10	Ikura	4	8	12
11	Queso Cabrales	5	4	1
12	Queso Manchego La Pastora	5	4	10
13	Konbu	6	8	2
14	Tofu	6	7	4
15	Gement Shouyu	6	2	24
16	Pavlova	7	3	34
17	Alice Mutton	7	5	20
18	Carnarvon Tigers	7	8	14
19	Teatime Chocolate Biscuits	8	3	10
20	Sir Rodney's Marmalade	8	3	30
21	Sir Rodney's Scones	8	3	24
22	Gustaf's Känckebroöd	9	5	24

Importing Data from OData Feed

In this task, you'll bring in order data. This step represents connecting to a sales system. You import data into Power BI Desktop from the sample Northwind OData feed at the following URL, which you can copy (and then paste) in the steps below:

<http://services.odata.org/V3/Northwind/Northwind.svc/>

Connect to an OData feed:

- 1) From the Home ribbon tab in Query Editor, select Get Data.
- 2) Browse to the OData Feed data source.
- 3) In the OData Feed dialog box, paste the URL for the Northwind OData feed.
- 4) Select OK.

- 5) In the Navigator pane, select the Orders table, and then select Edit.

The screenshot shows a 'Navigator' window with a tree view on the left and a grid view on the right.

Left Panel (Navigator Tree):

- Show All | Show Selected [1]
- http://services.odata.org/V3/Northwind/Nort...
- Alphabetical_list_of_products
- Categories
- Category_Sales_for_1997
- Current_Product_Lists
- Customer_and_Suppliers_by_Cities
- CustomerDemographics
- Customers
- Employees
- Invoices
- Order_Details
- Order_Details_Extendeds
- Order_Subtotals
- Orders
- Orders_Qries
- Product_Sales_for_1997
- Products
- Products_Above_Average_Prices
- Products_by_Categories
- Regions

Right Panel (Orders Table):

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate
10248	VINET	5	7/4/1996 12:00:00 AM	8/1/1996
10249	TOMSP	6	7/5/1996 12:00:00 AM	8/16/1996
10250	HANAR	4	7/8/1996 12:00:00 AM	8/5/1996
10251	VICTE	3	7/8/1996 12:00:00 AM	8/6/1996
10252	SUPRD	4	7/9/1996 12:00:00 AM	8/6/1996
10253	HANAR	3	7/10/1996 12:00:00 AM	7/24/1996
10254	CHOPS	5	7/11/1996 12:00:00 AM	8/8/1996
10255	RICSU	9	7/12/1996 12:00:00 AM	8/9/1996
10256	WELLI	3	7/15/1996 12:00:00 AM	8/12/1996
10257	HILAA	4	7/16/1996 12:00:00 AM	8/13/1996
10258	ERNSH	1	7/17/1996 12:00:00 AM	8/14/1996
10259	CENTC	4	7/18/1996 12:00:00 AM	8/15/1996
10260	OTTIK	4	7/19/1996 12:00:00 AM	8/16/1996
10261	QUEDE	4	7/22/1996 12:00:00 AM	8/19/1996
10262	RATTI	8	7/22/1996 12:00:00 AM	8/19/1996
10263	ERNSH	9	7/23/1996 12:00:00 AM	8/20/1996
10264	POLKO	6	7/24/1996 12:00:00 AM	8/21/1996
10265	BLOTP	2	7/25/1996 12:00:00 AM	8/22/1996
10266	WARTH	3	7/26/1996 12:00:00 AM	8/6/1996
10267	FRANK	4	7/29/1996 12:00:00 AM	8/26/1996
10268	GROSR	8	7/30/1996 12:00:00 AM	8/27/1996
10269	WHITC	5	7/31/1996 12:00:00 AM	8/14/1996
10270	WARTH	1	8/1/1996 12:00:00 AM	8/29/1996

OK Cancel

Conclusion: - This way, Implemented a program for inverted files.

ASSIGNMENT No: 08

Title: Data Visualization from Extraction Transformation and Loading (ETL) Process.

Problem Statement: Data Visualization from Extraction Transformation and Loading (ETL) Process.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn Data Visualization from Extraction Transformation and Loading (ETL) Process

Outcomes:

After completion of this assignment students are able to understand how Data Visualization is done through Extraction Transformation and Loading (ETL) Process

Theory:

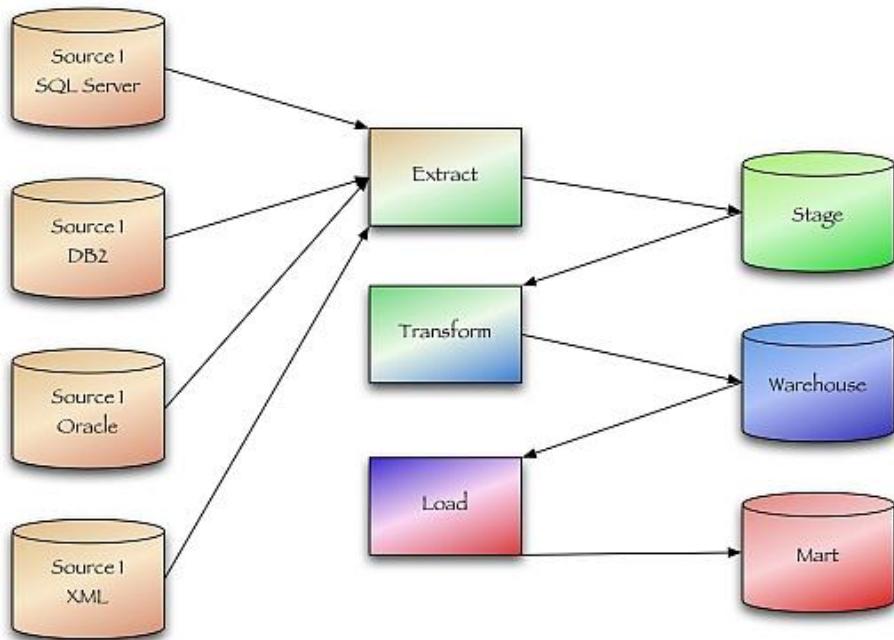
Extract, transform, and load (ETL) are 3 data processes, followed after data collection. Extraction takes data, collected in data sources like flat files, databases (relational, hierarchical etc.), transactional datastores, semi-structured repositories (e.g. email systems or document libraries) with different structure and format, pre-validating extracted data and parsing valid data to destination (e.g. staging database)

Transformation takes extracted data and applies predefined rules and functions to it, including selection (e.g. ignore or remove NULLs), data cleansing, encoding (e.g. mapping “Male” to “M”), deriving (e.g. calculating designated value as a product of extracted value and predefined constant), sorting, joining data from multiple sources (e.g. lookup or merge), aggregation (e.g. summary for each month), transposing (columns to rows or vice versa), splitting, disaggregation, lookups (e.g. validation through dictionaries), predefined validation etc. which may lead to rejection of some data. Transformed data can be stored into Data Warehouse (DW).

Load takes transformed data and places it into end target, in most cases called Data Mart (sometimes they called Data Warehouse too). Load can append, refresh or/and overwrite preexisting data, apply constraints

and execute appropriate triggers (to enforce data integrity, uniqueness, mandatory fields, provide log etc.) and may start additional processes, like data backup or replication.

ETL Workflow



Extract



RDBMS querying
XML/JSON/CSV
Web App APIs
pdf/xlsx/pptx

Transform



Power Query
Insomnia
Data cleaning and transforming

Load



Loading structured and unstructured data into SharePoint folder

Analyze



Dashboarding and analyzing data using Power BI

Conclusion:- This way Data Visualization from Extraction Transformation and Loading (ETL) Process is done.

ASSIGNMENT No: 09

Title: Perform the Extraction Transformation and Loading (ETL) process to construct the database in the Sql server / Power BI.

Problem Statement: Perform the Extraction Transformation and Loading (ETL) process to construct the database in the Sql server / Power BI.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to Perform the Extraction Transformation and Loading (ETL) process to construct the database in the Sql server / Power BI.

Outcomes:

After completion of this assignment students are able to understand how to Perform the Extraction Transformation and Loading (ETL) process to construct the database in the Sql server / Power BI.

Theory:

Step 1 : Data Extraction :

The data extraction is first step of ETL. There are 2 Types of Data Extraction

1. Full Extraction : All the data from source systems or operational systems gets extracted to staging area. (Initial Load)
2. Partial Extraction : Sometimes we get notification from the source system to update specific date. It is called as Delta load.

Source System Performance: The Extraction strategies should not affect source system performance.

Step 2 : Data Transformation :

The data transformation is second step. After extracting the data there is big need to do the transformation as per the target system. I would like to give you some bullet points of Data Transformation.

- Data Extracted from source system is in to Raw format. We need to transform it before loading in to target server.
- Data has to be cleaned, mapped and transformed.
- There are following important steps of Data Transformation :

1.Selection : Select data to load in target

2.Matching : Match the data with target system

3.Data Transforming : We need to change data as per target table structures

Real life examples of Data Transformation :

- Standardizing data : Data is fetched from multiple sources so it needs to be standardized as per the target system.
- Character set conversion : Need to transform the character sets as per the target systems. (Firstname and last name example)
- Calculated and derived values: In source system there is first val and second val and in target we need the calculation of first val and second val.
- Data Conversion in different formats : If in source system date in in DDMMYYY format and in target the date is in DDMONYYYY format then this transformation needs to be done at transformation phase.

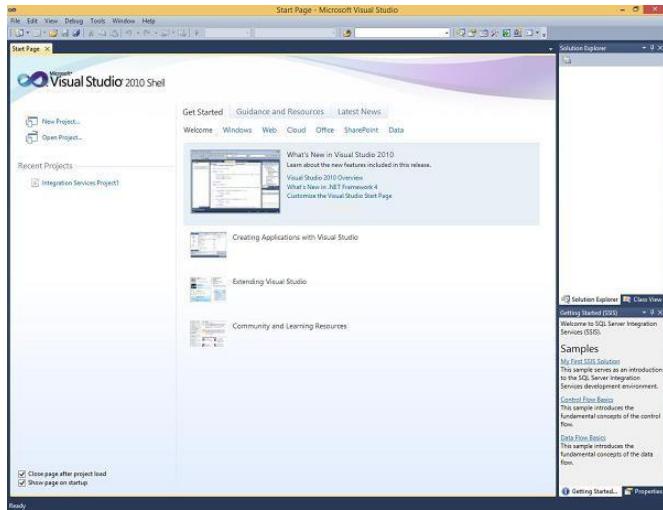
Step 3 : Data Loading

- Data loading phase loads the prepared data from staging tables to main tables.

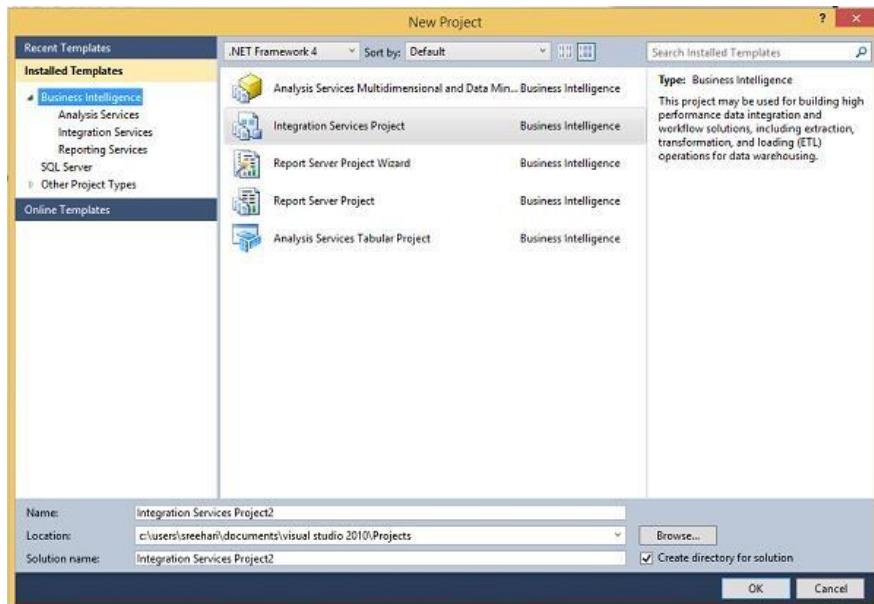
ETL process in SQL Server:

Following are the steps to open BIDS\SSDT.

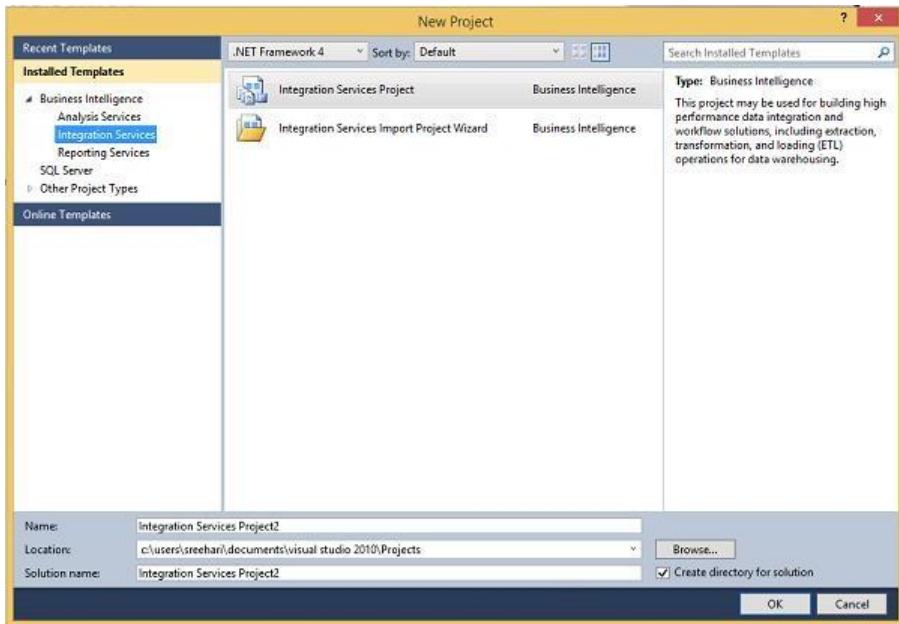
Step 1 – Open either BIDS\SSDT based on the version from the Microsoft SQL Server programs group. The following screen appears.



Step 2 – The above screen shows SSDT has opened. Go to file at the top left corner in the above image and click New. Select project and the following screen opens.



Step 3 – Select Integration Services under Business Intelligence on the top left corner in the above screen to get the following screen.



Step 4 – In the above screen, select either Integration Services Project or Integration Services Import Project Wizard based on your requirement to develop\create the package.

Modes

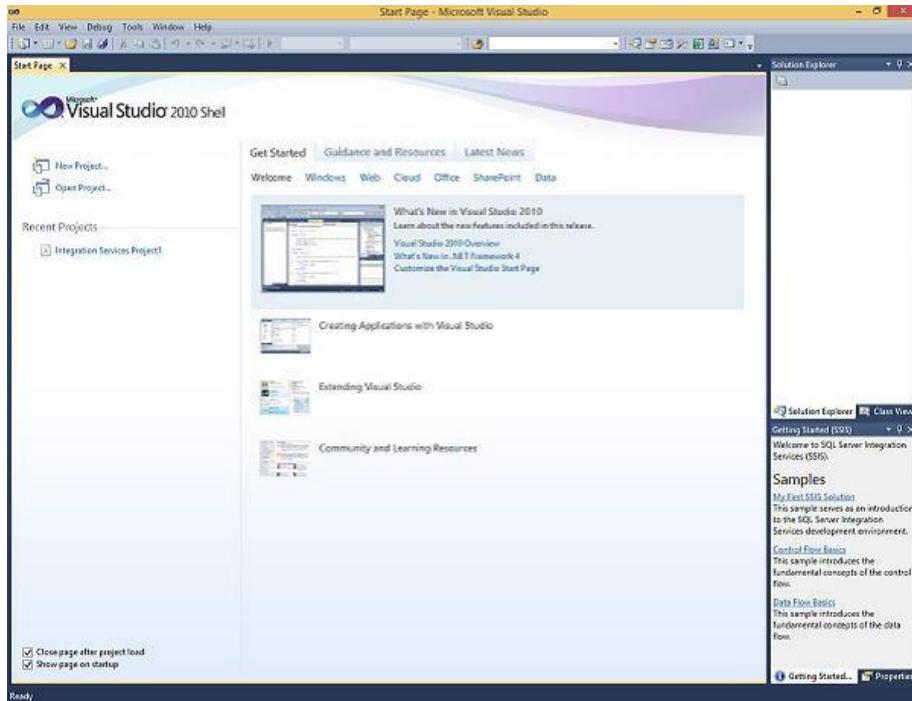
There are two modes – Native Mode (SQL Server Mode) and Share Point Mode.

Models

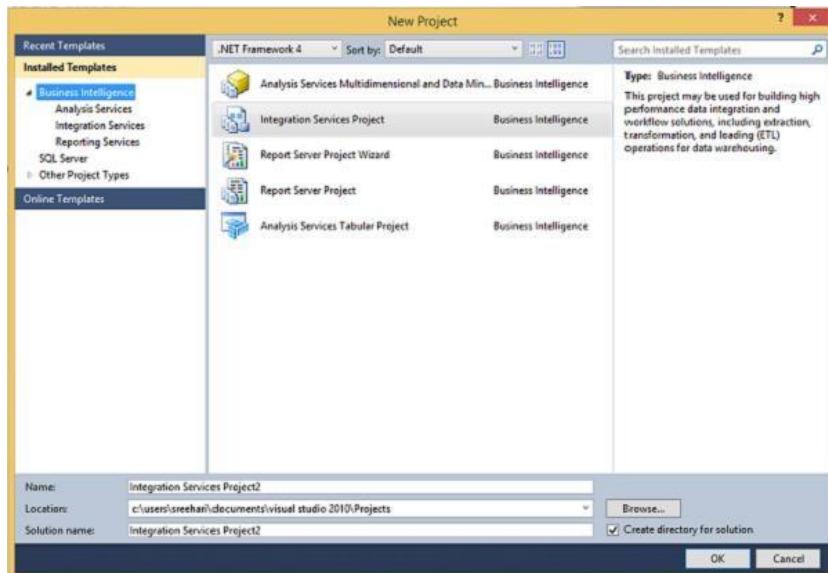
There are two models – Tabular Model (For Team and Personal Analysis) and Multi Dimensions Model (For Corporate Analysis).

The BIDS (Business Intelligence Studio till 2008 R2) and SSDT (SQL Server Data Tools from 2012) are environments to work with SSAS.

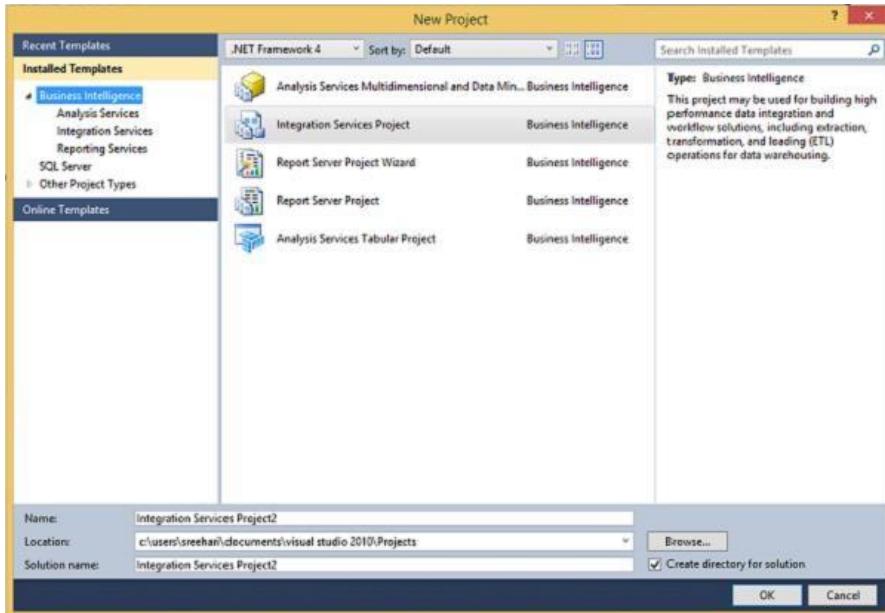
Step 1 – Open either BIDS\SSDT based on the version from the Microsoft SQL Server programs group. The following screen will appear.



Step 2 – The above screen shows SSDT has opened. Go to file on the top left corner in the above image and click New. Select project and the following screen opens.



Step 3 – Select Analysis Services in the above screen under Business Intelligence as seen on the top left corner. The following screen pops up.



Step 4 – In the above screen, select any one option from the listed five options based on your requirement to work with Analysis services.

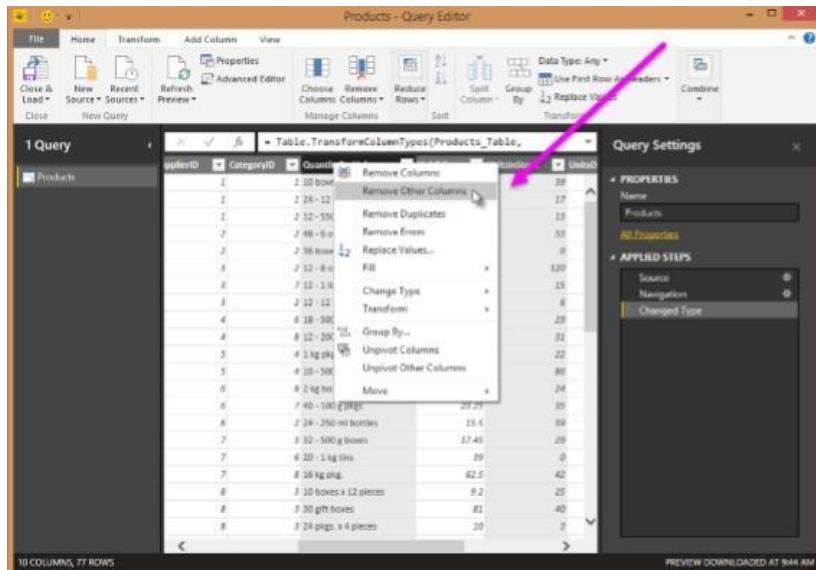
ETL Process in Power BI

1) Remove other columns to only display columns of interest

In this step you remove all columns except **ProductID**, **ProductName**, **UnitsInStock**, and **QuantityPerUnit**

Power BI Desktop includes Query Editor, which is where you shape and transform your data connections. Query Editor opens automatically when you select **Edit** from Navigator. You can also open the Query Editor by selecting **Edit Queries** from the Home ribbon in Power BI Desktop. The following steps are performed in Query Editor.

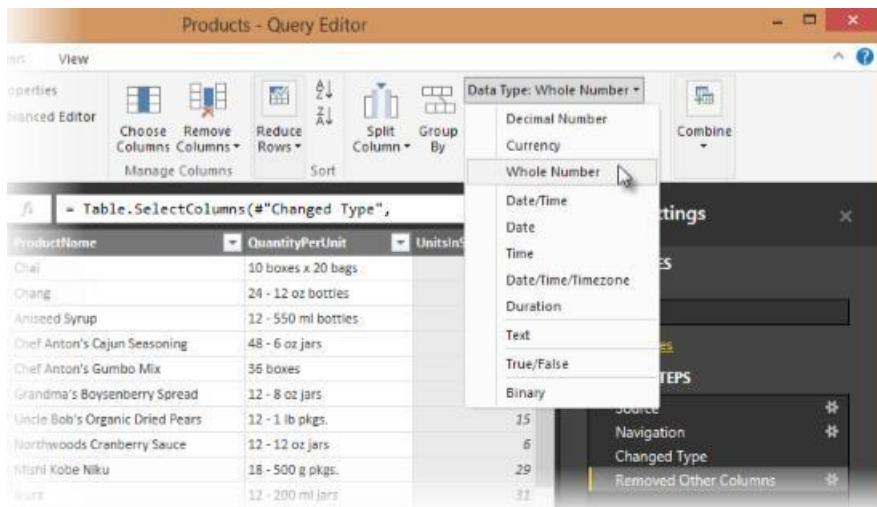
1. In **Query Editor**, select the **ProductID**, **ProductName**, **QuantityPerUnit**, and **UnitsInStock** columns (use **Ctrl+Click** to select more than one column, or **Shift+Click** to select columns that are beside each other).
2. Select **Remove Columns > Remove Other Columns** from the ribbon, or right-click on a column header and click **Remove Other Columns**.



3. Change the data type of the UnitsInStock column

When Query Editor connects to data, it reviews each field and to determine the best data type. For the Excel workbook, products in stock will always be a whole number, so in this step you confirm the **UnitsInStock** column's datatype is Whole Number.

1. Select the **UnitsInStock** column.
2. Select the **Data Type drop-down button** in the **Home** ribbon.
3. If not already a Whole Number, select **Whole Number** for data type from the drop down (the Data Type: button also displays the data type for the current selection).



3. Expand the Order_Details table

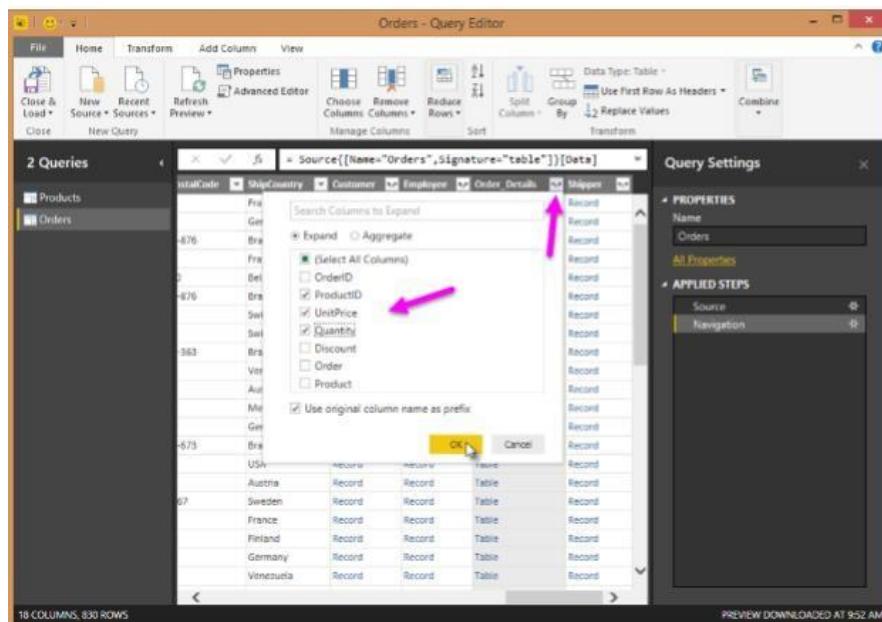
The Orders table contains a reference to a Details table, which contains the individual products that were included in each Order. When you connect to data sources with multiple tables (such as a relational database) you can use these references to build up your query.

In this step, you expand the **Order_Details** table that is related to the Orders table, to combine the **ProductID**, **UnitPrice**, and **Quantity** columns from **Order_Details** into the **Orders** table. This is a representation of the data in these tables:

The Expand operation combines columns from a related table into a subject table. When the query runs, rows from the related table (**Order_Details**) are combined into rows from the subject table (**Orders**).

After you expand the Order_Details table, three new columns and additional rows are added to the Orders table, one for each row in the nested or related table.

1. In the Query View, scroll to the Order_Details column.
2. In the Order_Details column, select the expand icon ().
3. In the Expand drop-down:
 - a. Select (Select All Columns) to clear all columns.
 - b. Select ProductID, UnitPrice, and Quantity.
 - c. Click OK.



4. Calculate the line total for each Order_Details row

Power BI Desktop lets you to create calculations based on the columns you are importing, so you can enrich the data that you connect to. In this step, you create a Custom Column to calculate the line total for each Order_Details row.

Calculate the line total for each Order_Details row:

1. In the Add Column ribbon tab, click Add Custom Column.

	LineTotal	ShipCountry
	168	France
	98	France
	174	France
	167.4	Germany
	1696	Germany
	77	Brazil
	1484	Brazil
	252	Brazil
	100.8	France

2. In the Add Custom Column dialog box, in the Custom Column Formula textbox, enter [Order_Details.UnitPrice] * [Order_Details.Quantity].
3. In the New column name textbox, enter LineTotal.
4. Click OK.



5. Rename and reorder columns in the query

In this step you finish making the model easy to work with when creating reports, by renaming the final columns and changing their order.

1. In Query Editor, drag the LineTotal column to the left, after ShipCountry.

The screenshot shows the Power BI Query Editor interface. On the left, there are two queries: 'Products' and 'Orders'. The 'Orders' query is currently selected. In the main preview area, a table is displayed with four columns: 'ShipCity', 'LineTotal', 'Order_Details.ProductID', and 'Order_Details.UnitPrice'. The data in the preview includes rows for cities like 'AM- Reims', 'AM- Münster', and 'AM- Rio de Janeiro' with their corresponding 'LineTotal' values. To the right of the preview is the 'Query Settings' pane. It contains sections for 'PROPERTIES' (Name: Orders) and 'APPLIED STEPS' (Source, Navigation, Expanded Order_Details, Removed Other Columns, Added Custom). A tooltip at the bottom right of the preview area says 'PREVIEW DOWNLOADED AT 9:52 AM'.

2.Remove the Order_Details. prefix from the Order_Details.ProductID, Order_Details.UnitPrice and Order_Details.Quantity columns, by double-clicking on each column header, and then deleting that text from the column name.

6. Combine the Products and Total Sales queries

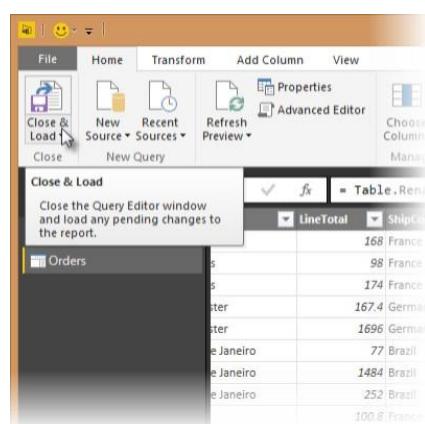
Power BI Desktop does not require you to combine queries to report on them. Instead, you can create Relationships between datasets. These relationships can be created on any column that is common to your datasets

we have Orders and Products data that share a common 'ProductID' field, so we need to ensure there's a relationship between them in the model we're using with Power BI Desktop. Simply specify in Power BI Desktop that the columns from each table are related (i.e. columns that have the same values). Power BI Desktop works out the direction and cardinality of the relationship for you. In some cases, it will even detect the relationships automatically.

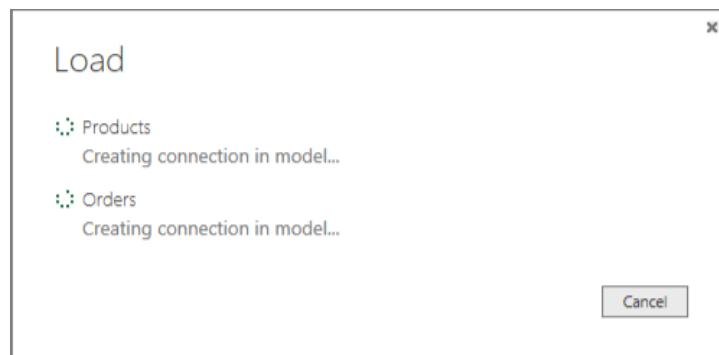
In this task, you confirm that a relationship is established in Power BI Desktop between the Products and Total Sales queries

Step 1: Confirm the relationship between Products and Total Sales

1. First, we need to load the model that we created in Query Editor into Power BI Desktop. From the Home ribbon of Query Editor, select Close & Load.



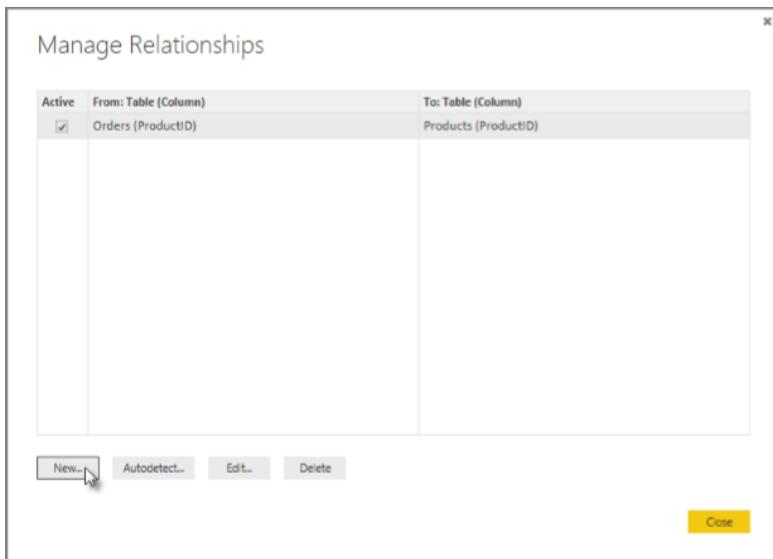
1. Power BI Desktop loads the data from the two queries.



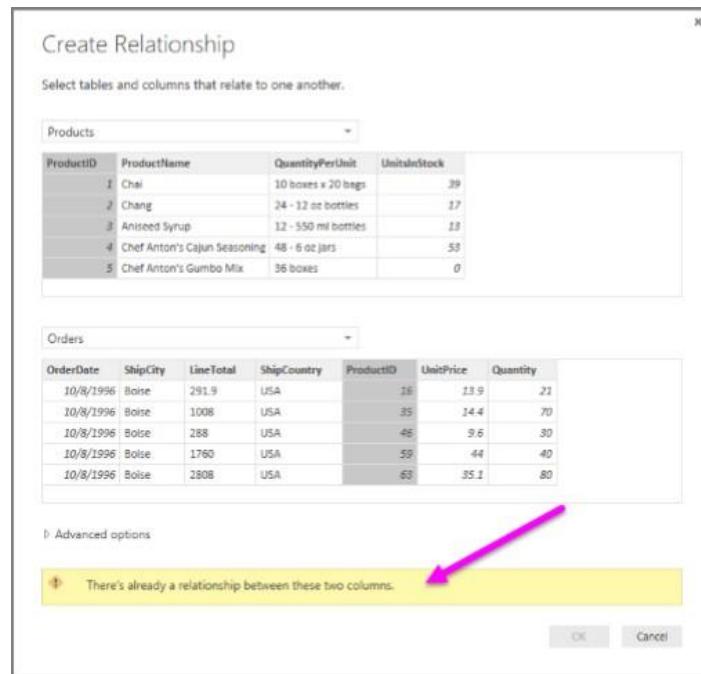
2. Once the data is loaded, select the Manage Relationships button Home ribbon.



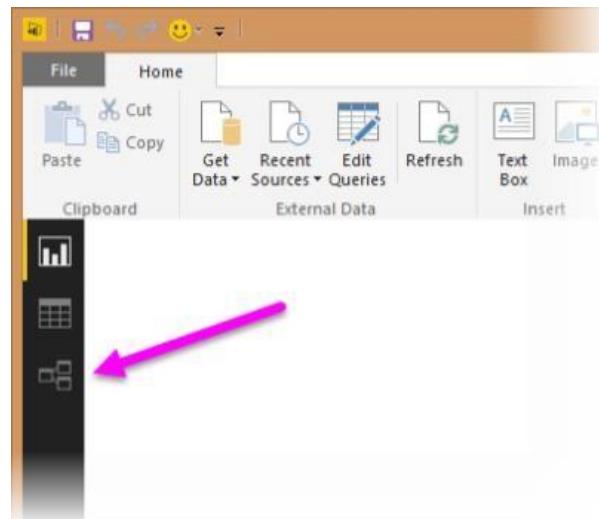
3. Select the New... button



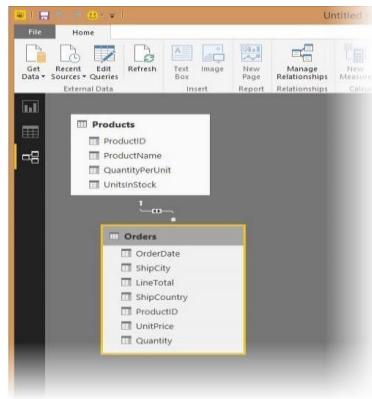
4. When we attempt to create the relationship, we see that one already exists! As shown in the Create Relationship dialog (by the shaded columns), the ProductsID fields in each query already have an established relationship.



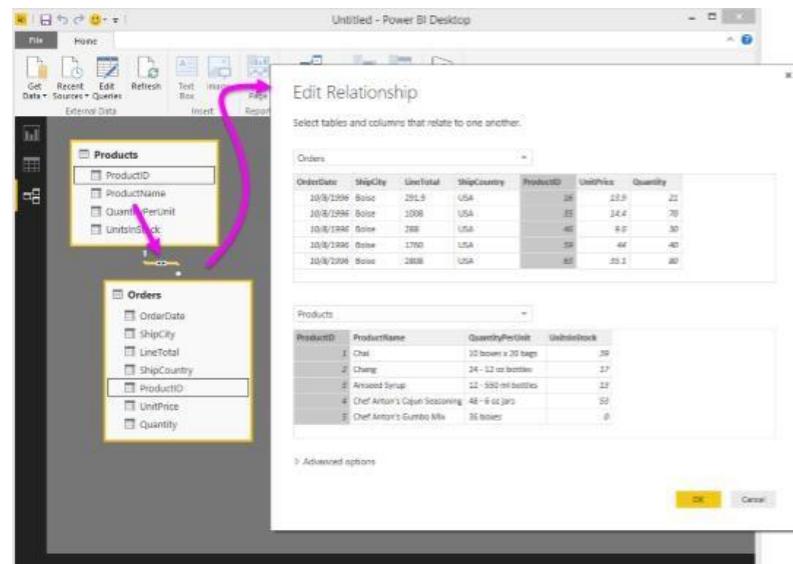
5. Select Cancel, and then select Relationship view in Power BI Desktop.



6. We see the following, which visualizes the relationship between the queries.



- When you double-click the arrow on the line that connects the two queries, an EditRelationship dialog appears.



No need to make any changes, so we'll just select Cancel to close the EditRelationship dialog

Conclusion: Thus Performed Extraction Transformation and Loading (ETL) process to construct the database in the Sql server / Power BI.

ASSIGNMENT No: 10

Title: Data Analysis and Visualization using Advanced Excel.

Problem Statement: Data Analysis and Visualization using Advanced Excel.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to Perform Data Analysis and Visualization using Advanced Excel.

Outcomes:

After completion of this assignment students are able to understand Data Analysis and Visualization using Advanced Excel.

Theory:

Defining Data Visualization

We'll first start by defining what data visualization is. Data visualization is a graphical representation of data. By utilizing charts, graphs, maps, etc., we can provide a simple and accessible way to understand our data and identify trends and outliers within our datasets. Note that Excel uses the term "chart" to mean a "plot". For example, a bar plot is called a bar chart in Excel terminology.

The purpose of this tutorial is to walk you through some basic charts to visualize your data before jumping into more advanced techniques later on. We highly recommend you check out our Data Visualization cheat sheet to learn more about the most common visualizations and when to use them.

Example Dataset

We first need a dataset to work with before creating any visualizations. This tutorial will use a simple dataset containing sales data for a local electronics store. The dataset contains information on the number of units sold for various product types in 2022 and totals for columns and rows.

Month	TVs	Mobile Phones	Laptops	Total
1/1/2022	145	335	82	562
2/1/2022	145	362	126	633
3/1/2022	105	311	95	511
4/1/2022	171	259	93	523
5/1/2022	178	277	107	562
6/1/2022	167	292	145	604
7/1/2022	200	385	77	662
8/1/2022	181	388	78	647

9/1/2022	152	291	83	526
10/1/2022	143	345	102	590
11/1/2022	114	399	99	612
12/1/2022	109	250	101	460
Total	1810	3894	1188	

We'll be working with this dataset throughout this tutorial. You can download the data file from GitHub.

Alternatively, you can import the dataset using the following steps:

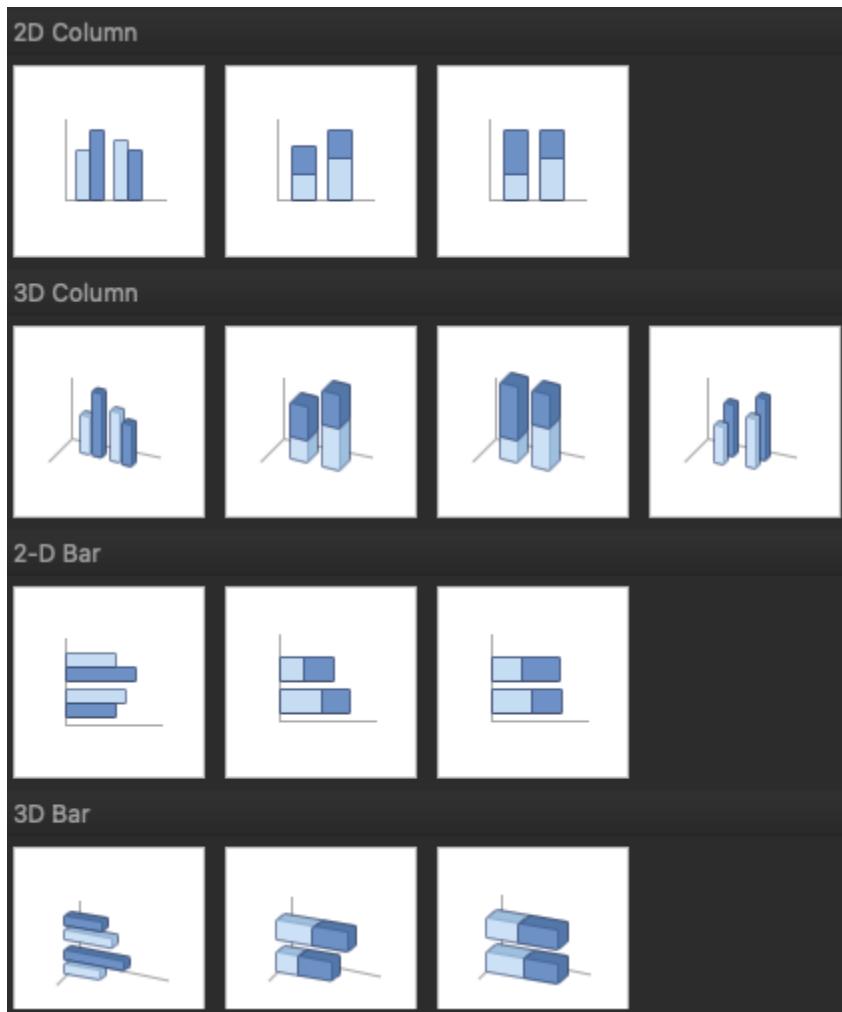
- Open Excel and create a new workbook.
- Copy the dataset above and paste it into cell A1

Format the cells as needed (e.g., adjust column width, apply bold formatting to headers, etc.).

	A	B	C	D
1	Month	Television	Laptop	Mobile Phones
2	1/1/2022	145	335	82
3	2/1/2022	145	362	126
4	3/1/2022	105	311	95
5	4/1/2022	171	259	93
6	5/1/2022	178	277	107
7	6/1/2022	167	292	145
8	7/1/2022	200	385	77
9	8/1/2022	181	388	78
10	9/1/2022	152	291	83
11	10/1/2022	143	345	102
12	11/1/2022	114	399	99
13	12/1/2022	109	250	101

Creating Basic Charts in Excel

Excel has multiple options for choosing a particular chart type. For example, if you want to create a column or bar chart, you are often presented with various visualization options. For example, there are 2D and 3D versions and normal, stacked, and 100% stacked options. Depending on your requirements, you can choose the visualization type that best suits your needs.

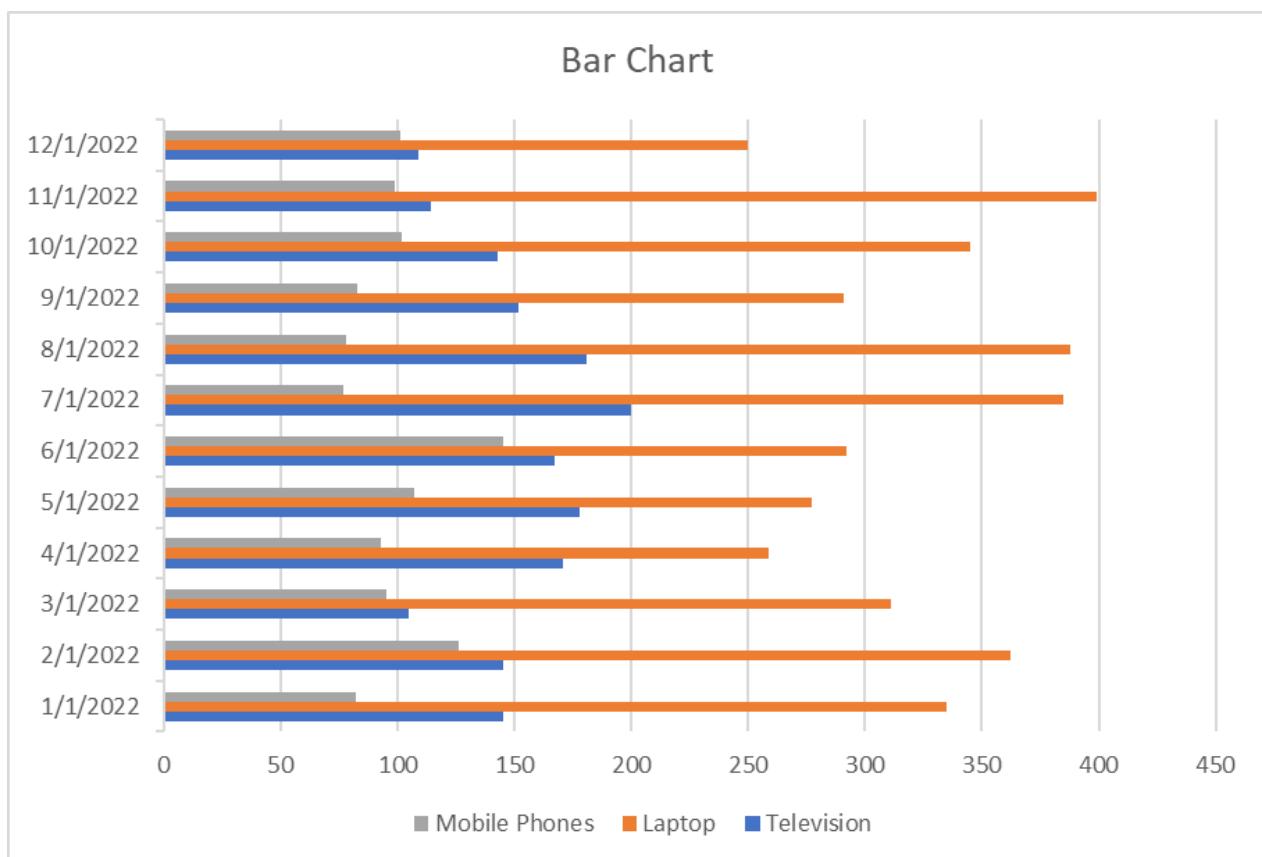


Excel bar charts

Bar charts are one of the easiest charts to interpret, enabling the person viewing the chart an easy way to compare categorical data quickly. On a bar chart, the categorical data is on the y-axis, and the values are on the x-axis.

To create a bar chart:

- Select the data range A1:D13
- Click the "Insert" tab in the Excel ribbon
- Click on the columns icon button dropdown, and under the “2-D Bar” category, choose “Clustered Bar”



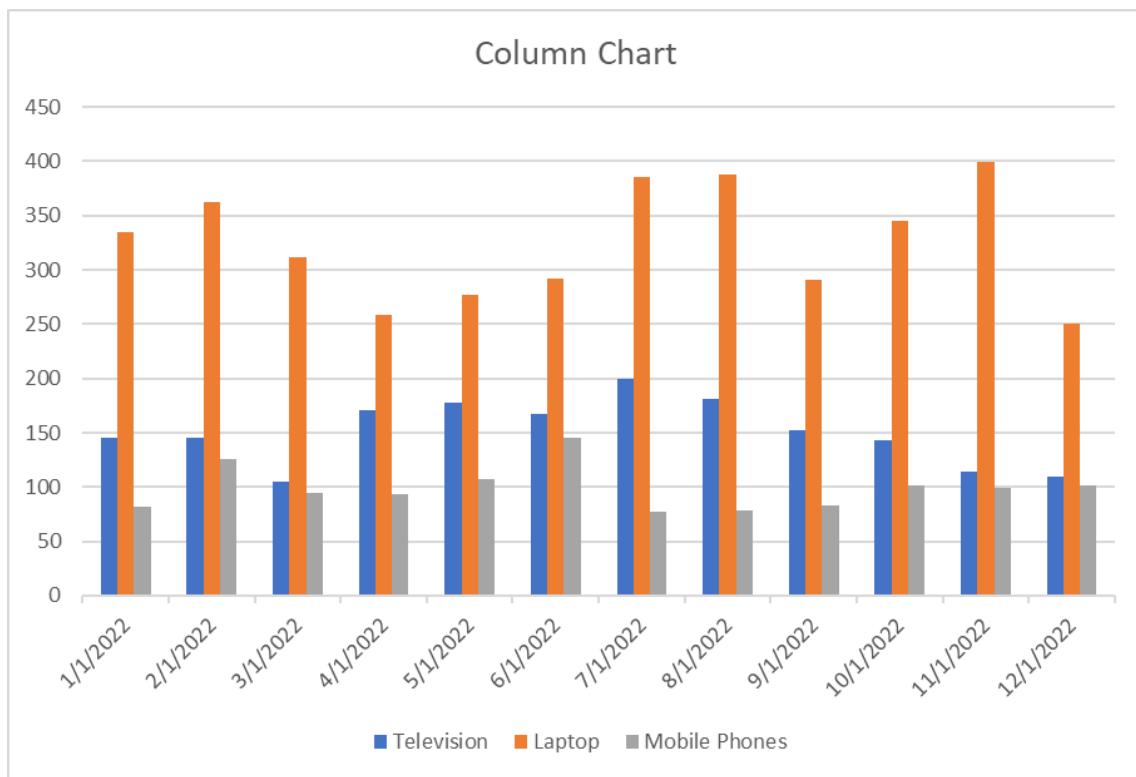
Excel column charts

A column chart, also known as a vertical bar chart, helps visualize data where categories are placed on the x-axis and the values on the y-axis. Similar to bar charts, they help visualize data across categories.

To create a column chart in Excel:

- Select the data range A1:D13
- Click the "Insert" tab in the Excel ribbon
- Click on the columns icon dropdown, and under the “2-D Column” category, choose “Clustered Column”

You can now see a column chart that displays the number of units sold for each product category by the month.



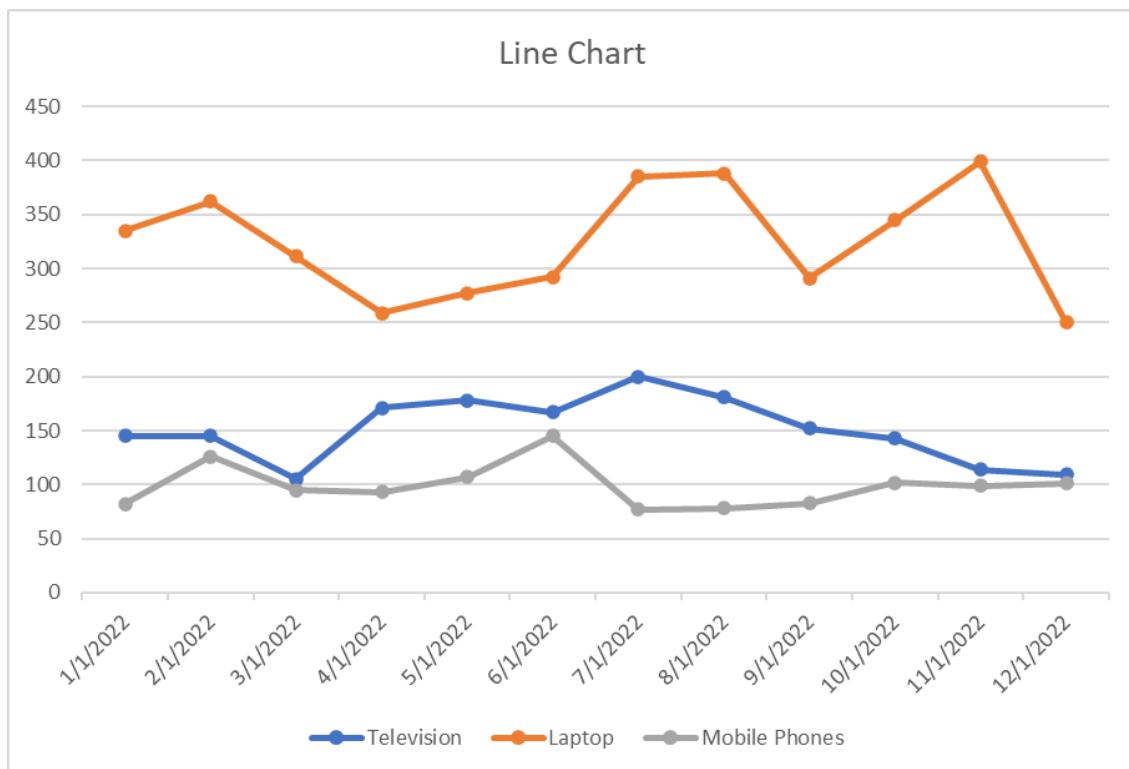
Excel line charts

A line chart is the most useful way to capture how a numerical variable changes over time. This is helpful to identify trends in numeric values.

To create a line chart in Excel:

- Select the data range A1:D13
- Click the "Insert" tab in the Excel ribbon
- Click on the line chart dropdown, and under the “2-D Line” category, choose “Line with Markers”

You can now see a line chart displaying units sold each month split by product category. This enables you to compare each product category's performance over time easily.

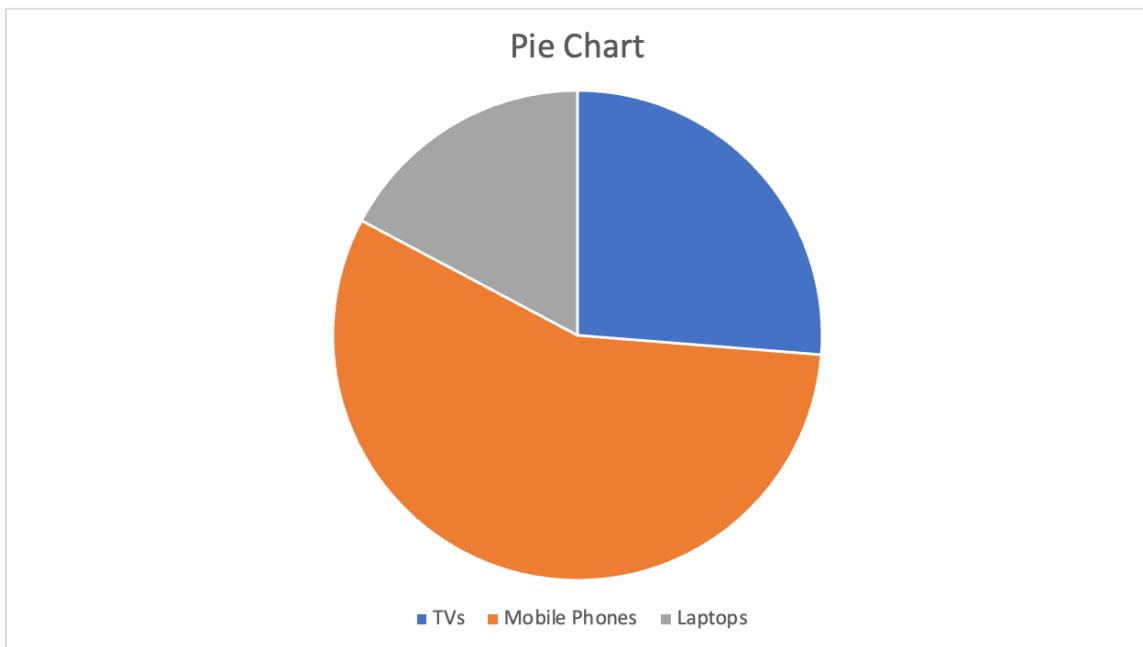


Excel pie charts

A pie chart is most commonly used to show the proportions of a whole. It's like visualizing fractions when you were in high school. With this pie chart, we want to compare the total sales between the three categories.

To create a pie chart in Excel:

- First, select the data range B1:D1
- Second, using the command (for Mac) or ctrl (for Windows), select the second date range: B14:D14
- Click the "Insert" tab in the Excel ribbon
- Click on the pie chart dropdown, and under the “2-D Pie” category, choose “Pie”



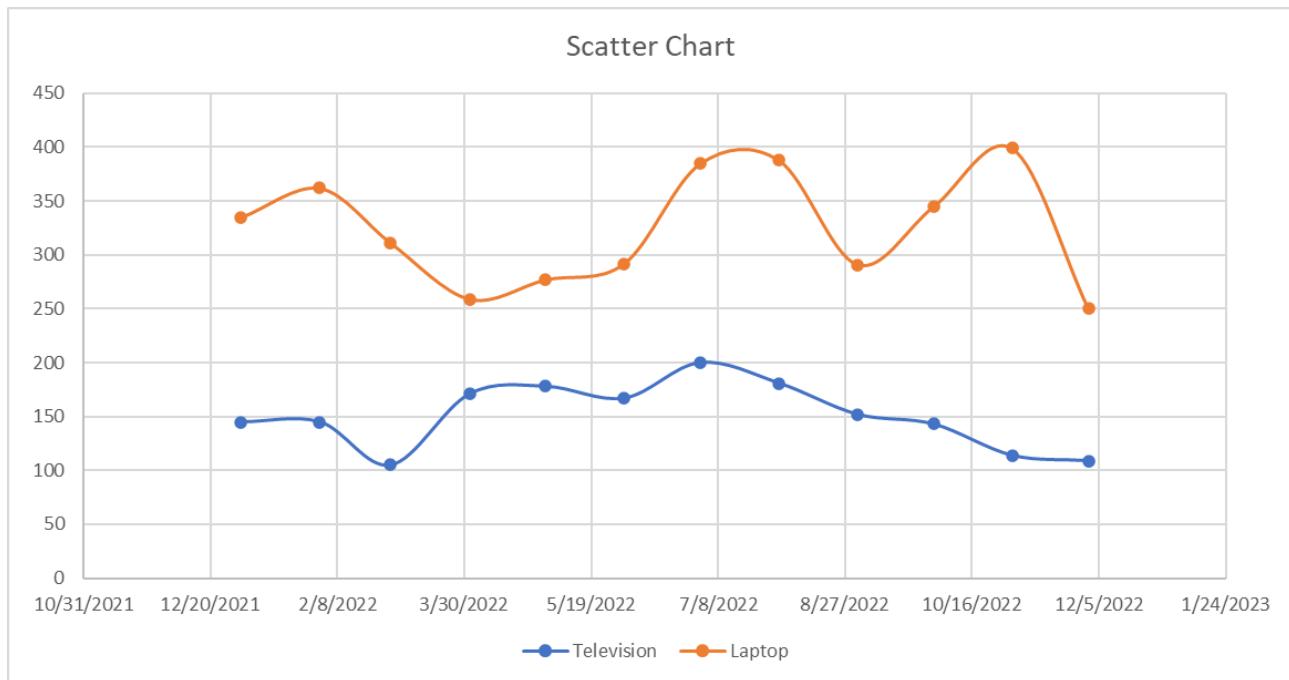
Advanced Excel Visualization Techniques

Excel scatter plots

A scatter plot is commonly used to visualize the relationship between two variables. It can be useful for quickly surfacing potential correlations between data points. We'll create a scatter plot to compare the number of TVs and laptops sold.

To create a scatter plot in Excel:

- Select the data range A1:C13
- Click the "Insert" tab in the Excel ribbon
- Click on the scatter plot dropdown, and under the “Scatter” category, choose “Histogram”
- Click "Scatter or Bubble Chart" and choose "Scatter with Smooth Lines and Markers"



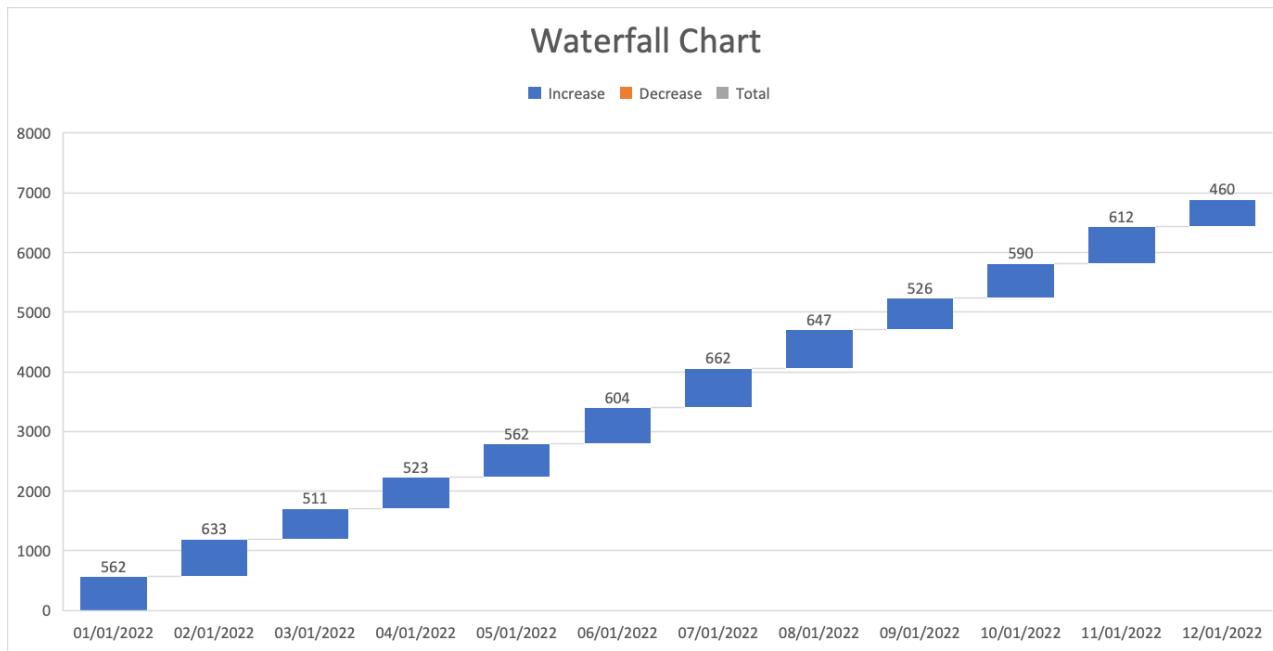
Excel waterfall chart

A waterfall chart is a special chart that helps illustrate how positive and negative values can contribute to a total. They can be great for visualizing changes over time. In our example, we'll compare the total sales for each month regardless of the categories.

To create a waterfall chart in Excel:

- First, select the data range A2:A13
- Second, using the command (for Mac) or ctrl (for Windows), select the second data range E2:E13
- Click on the waterfall chart dropdown, and under the “Waterfall” category, choose “Waterfall”

We'll only see positive values in our example because the Total column only contains positive values, but this chart can be great for comparing financial data and changes over time.



Customizing and Formatting Excel Charts

Aside from creating charts in Excel, there are many options to customize and format a chart. From here, we'll discuss some common formatting options that'll help you enhance your visuals.

First, create a column chart based on our previous guidance that displays the number of units sold for each product category by the month.

Chart elements

Chart elements include titles, legends, data labels, gridlines, and axes. You can add, remove, or modify these elements as needed.

Click on the chart to select it.

- In the Excel ribbon, click the "Chart Design" tab
- Click the "Add Chart Element" dropdown to access the available chart elements

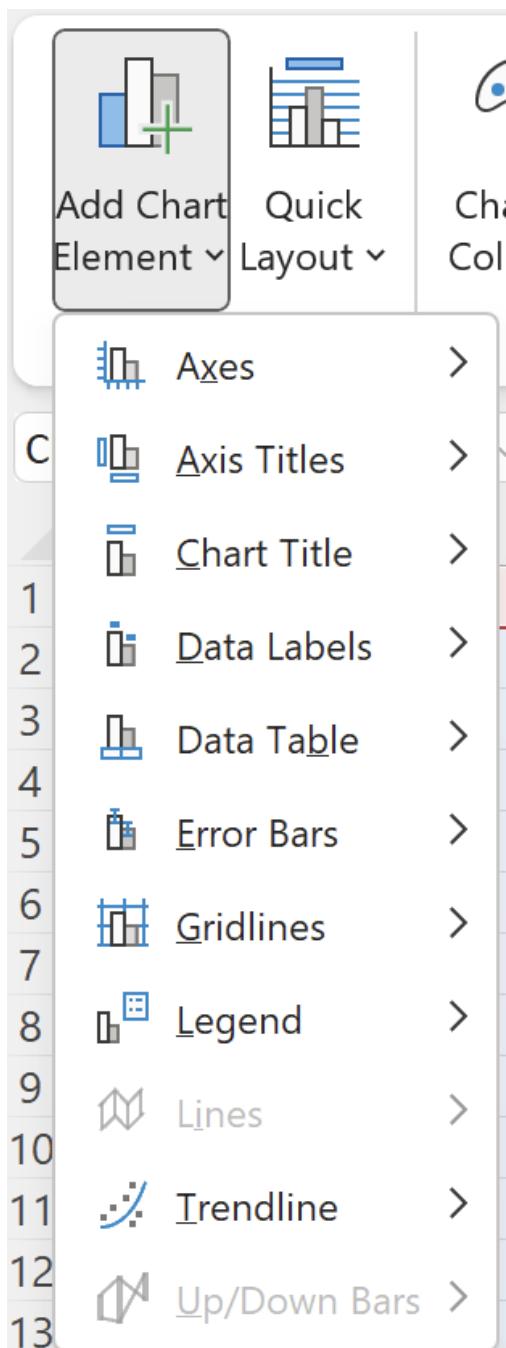


Chart title

- To add a chart title, click "Add Chart Element" > "Chart Title" and choose one of the available options (e.g., "Above Chart" or "Centered Overlay"). You can then click the title placeholder and type your desired title
- To remove a chart title, right-click on the title and select "Delete"

For this tutorial, rename the chart “Electronic Store Sales 2022”

Legend

- To modify the chart legend, click "Add Chart Element" > "Legend" and choose a position for the legend (e.g., "Right" or "Top")
- To remove the legend, click "Add Chart Element" > "Legend" > "None"

For this tutorial, move the legend to the top of the chart.

Data labels

- To add data labels, click "Add Chart Element" > "Data Labels" and choose one of the available options (e.g., "Center" or "Above"). This will display the data values directly on the chart
- To remove data labels, click "Add Chart Element" > "Data Labels" > "None"
- To remove data labels from individual categories, right-click on the data label of the chosen category; this will highlight all relevant data labels. You can then click "Delete"

For this tutorial, add data labels for all categories.

Your chart should now look like this:



Chart Styles and colors

On top of customizing individual chart elements, you can change your chart's overall look and feel by applying different styles and color schemes.

To customize a chart style, select the visual you want to update. Then take the following steps:

- In the Excel ribbon, click the "Chart Design" tab
- Under the “Chart Styles” section, there is an option to change styles and change colors
- For chart styles: Browse the available styles and click one to apply to your chart
- For color changes: Click the “Change Colors” dropdown and choose a color scheme

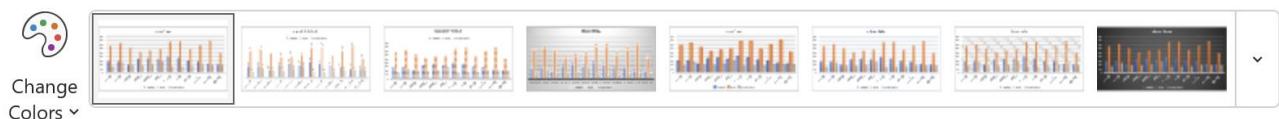


Chart Styles

By implementing small visual changes, we can see a big difference in the aesthetic of the whole chart. Here's an example of how our chart has changed by choosing Style 6 and Monochromatic Palette 8.



Formatting Excel chart axes

To improve the readability of our chart, we can format our axis in several ways, including axis titles, scale, or even visibility.

Axis titles

- To add an axis title, click "Add Chart Element" > "Axis Titles" and choose "Primary Horizontal" or "Primary Vertical." You can then click the title placeholder and type your desired title
- To remove an axis title, right-click on the axis title and click "Delete"

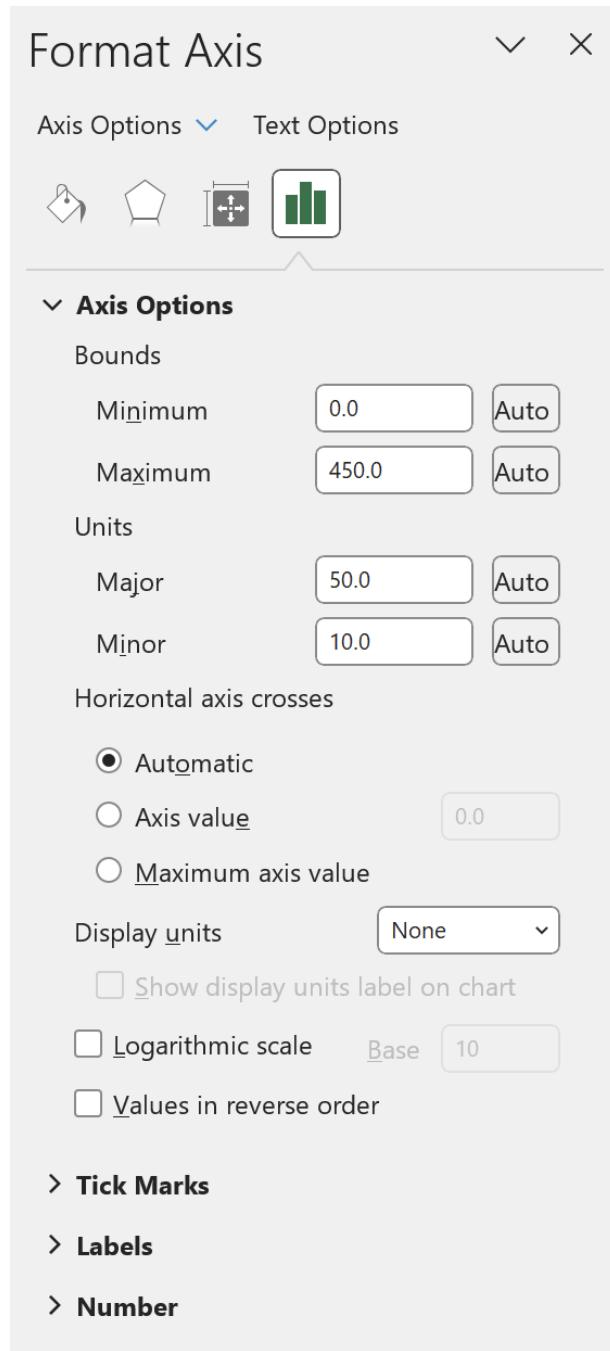
For this tutorial, add an x-axis title “Month” and a y-axis title “Products Sold.”

Axis scale and number format

To adjust the axis scale or number format:

- Right-click the axis you want to modify and choose "Format Axis"

In the "Format Axis" pane, you can change the minimum and maximum values, major and minor units, or number format.

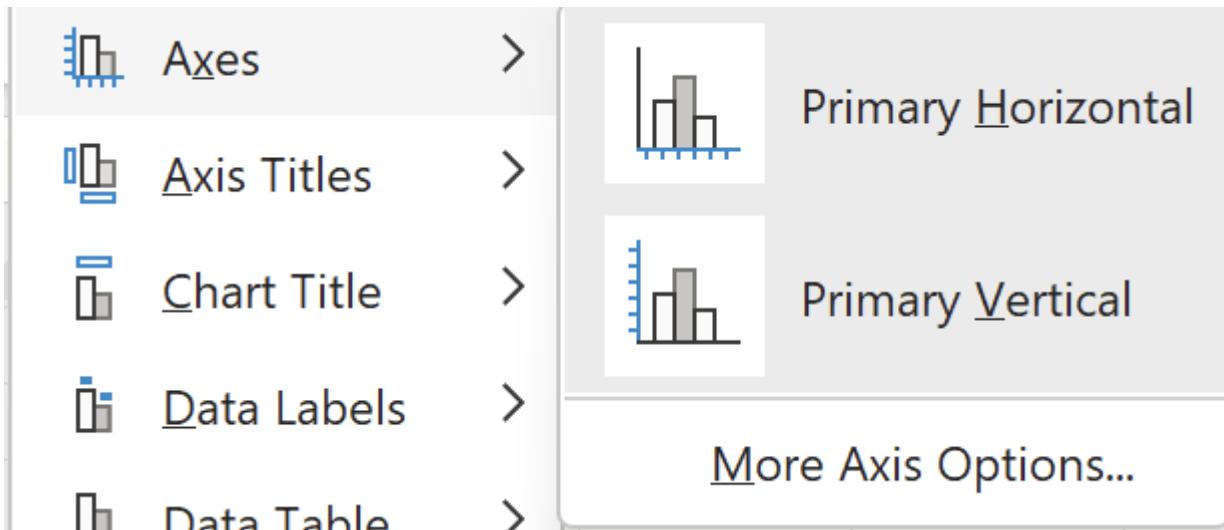


For this tutorial, we intend to keep these options the same but feel free to play around and test what each does.

Axis visibility

Finally, if you would like to remove the axis labels from showing, you can take the following steps:

- In the Excel ribbon, click the "Chart Design" tab
- Click the "Add Chart Element" dropdown and navigate to "Axes"
- By default, both options will have a darker gray box surrounding them, to remove an Axes, simply de-select the axes you'd like to remove

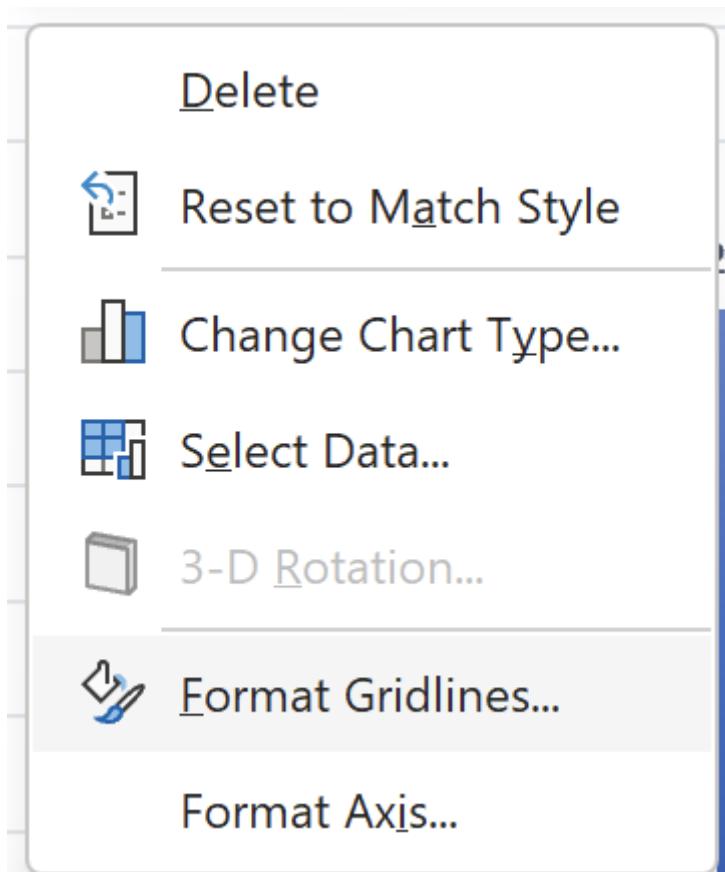


Other Excel formatting options

There are other formatting options available, including modifying data series colors, adjusting chart and plot area backgrounds, and customizing gridlines.

To access these options:

- Right-click the chart element you want to modify (e.g., data series, plot area, or gridlines)
- Select "Format [element]"



Conclusion:- Thus, this way Data Analysis and Visualization is done using Advanced Excel.

ASSIGNMENT No: 11

Title: Perform the data classification algorithm using any Classification algorithm

Problem Statement: Perform the data classification algorithm using any Classification algorithm

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to Perform the data classification algorithm using any Classification algorithm

Outcomes:

After completion of this assignment students are able to understand Perform the data classification algorithm using any Classification algorithm

Theory:

What is Classification?

We use the training dataset to get better boundary conditions which could be used to determine each target class. Once the boundary conditions are determined, the next task is to predict the target class. The whole process is known as classification.

Target class examples:

- Analysis of the customer data to predict whether he will buy computer accessories (Target class: Yes or No)
- Classifying fruits from features like color, taste, size, weight (Target classes: Apple, Orange, Cherry, Banana)
- Gender classification from hair length (Target classes: Male or Female)

Let's understand the concept of classification algorithms with gender classification using hair length (by no means am I trying to stereotype by gender, this is only an example). To classify gender (target class) using hair length as feature parameter we could train a model using any classification algorithms to come up with some set of boundary conditions which can be used to differentiate the male and female genders using hair length as the training feature. In gender classification case the boundary condition could be the proper hair length value. Suppose the differentiated boundary hair length value is 15.0 cm then we can say that if hair length is less than 15.0 cm then gender could be male or else female.

Classification Algorithms vs Clustering Algorithms

In clustering, the idea is not to predict the target class as in classification, it's more about trying to group the similar kind of things by considering the most satisfied condition, all the items in the same group should be similar and no two different group items should not be similar.

Group items Examples:

- While grouping similar language type documents (Same language documents are one group.)
- While categorizing the news articles (Same news category(Sport) articles are one group)

Let's understand the concept with clustering genders based on hair length example. To determine gender, different similarity measure could be used to categorize male and female genders. This could be done by finding the similarity between two hair lengths and keep them in the same group if the similarity is less (Difference of hair length is less). The same process could continue until all the hair length properly grouped into two categories.

Basic Terminology in Classification Algorithms

- Classifier: An algorithm that maps the input data to a specific category.
- Classification model: A classification model tries to draw some conclusion from the input values given for training. It will predict the class labels/categories for the new data.
- Feature: A feature is an individual measurable property of a phenomenon being observed.
- Binary Classification: Classification task with two possible outcomes. Eg: Gender classification (Male / Female)
- Multi-class classification: Classification with more than two classes. In multi-class classification, each sample is assigned to one and only one target label. Eg: An animal can be a cat or dog but not both at the same time.
- Multi-label classification: Classification task where each sample is mapped to a set of target labels (more than one class). Eg: A news article can be about sports, a person, and location at the same time.

Applications of Classification Algorithms

- Email spam classification
- Bank customers loan pay willingness prediction.
- Cancer tumor cells identification.
- Sentiment analysis
- Drugs classification
- Facial key points detection
- Pedestrians detection in an automotive car driving.

Types of Classification Algorithms

Classification Algorithms could be broadly classified as the following:

- Linear Classifiers
 - Logistic regression
 - Naive Bayes classifier
 - Fisher's linear discriminant
- Support vector machines
 - Least squares support vector machines
- Quadratic classifiers
- Kernel estimation
 - k-nearest neighbor
- Decision trees
 - Random forests
- Neural networks
- Learning vector quantization

Consider the annual rainfall details at a place starting from January 2012. We create an R time series object for a period of 12 months and plot it.

```
# Get the data points in form of a R vector.rainfall <-
c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)

# Convert it to a time series object.
rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)

# Print the timeseries data.
print(rainfall.timeseries)

# Give the chart file a name.png(file =
"rainfall.png")

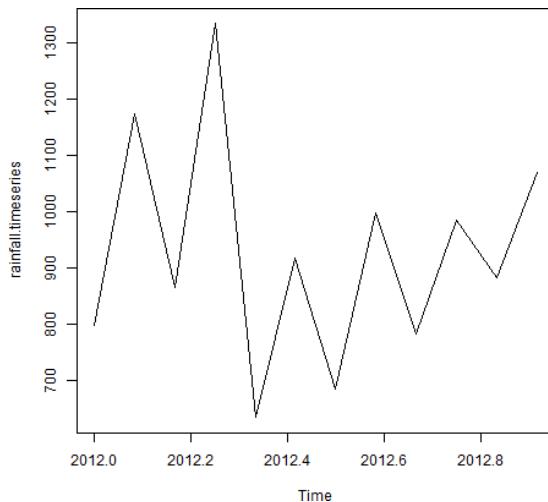
# Plot a graph of the time series.
plot(rainfall.timeseries)
```

```
# Save the file.  
dev.off()
```

Output:

When we execute the above code, it produces the following result and chart –

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	
2012	799.0	1174.8	865.1	1334.6	635.4	918.5	685.5	998.6	784.2
		Oct	Nov	Dec					
2012	985.0	882.8	1071.0						



Conclusion:- Thus, this way Performed data classification algorithm using any Classification algorithm

ASSIGNMENT No: 12

Title: Perform the data clustering algorithm using any Clustering algorithm

Problem Statement: Implement Page Rank Algorithm. (Use python or beautiful soup for implementation).

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn to Perform the data clustering algorithm using any Clustering algorithm

Outcomes:

After completion of this assignment students are able to understand how to Perform the data clustering algorithm using any Clustering algorithm

Theory:

Clustering your data can provide a new way to slice that is based on the properties of the data instead of other labels. For instance, customer data is often sliced by demographic parameters like gender, age, location, etc. This data can be useful in many cases, but what if you could slice your customers by their behaviour? What they buy, how often, how much they spend, etc. This information can help with advertising because you are now looking at past behaviour that can correlate better with future actions than demographics.

k-Mean Clustering

From the results of my testing, I believe the algorithm responsible for clustering in Power BI is the k-means algorithm. I did not find any confirmation on this, but it seems reasonable given the results found below. Knowing this can help you understand how Power BI finds clusters and how it will work in the situation you are using.

The goal of k-means is to minimize the distance between the points of each cluster. Each cluster has a centre. Data points are labeled as part of a cluster depending on which centre they are closest to.

As a result, certain types of clusters are easy to find, and in others, the algorithm will fail. Below, you will see examples of both cases.

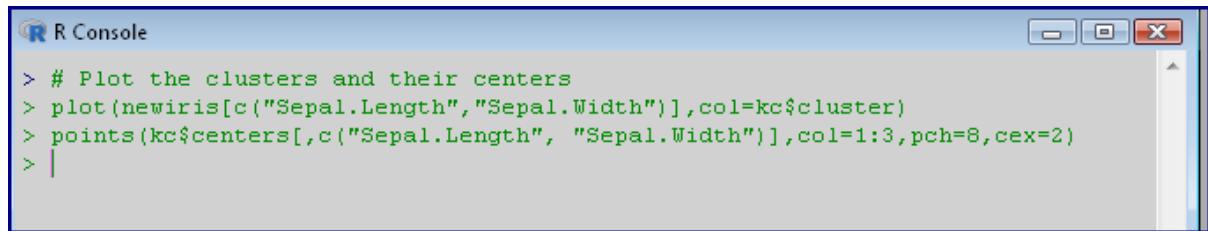
Compare the Species label with the clustering result

```
R Console
```

```
> #Compare the Species label with the clustering result
> table (iris$Species,kc$cluster)

      1 2 3
setosa 0 0 50
versicolor 48 2 0
virginica 14 36 0
> |
```

Plot the clusters and their centre



R Console

```
> # Plot the clusters and their centers
> plot(newiris[c("Sepal.Length", "Sepal.Width")], col=kc$cluster)
> points(kc$centers[,c("Sepal.Length", "Sepal.Width")], col=1:3, pch=8, cex=2)
> |
```

Conclusion:- Thus, this way Performed data clustering algorithm using any Clustering algorithm