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 –

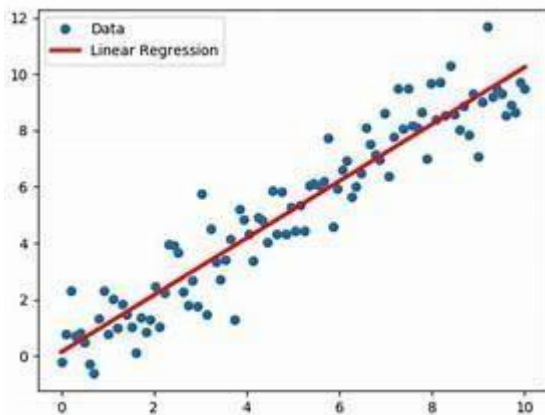A machine with sufficient RAM and processing power for model training (8GB RAM recommended)

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).
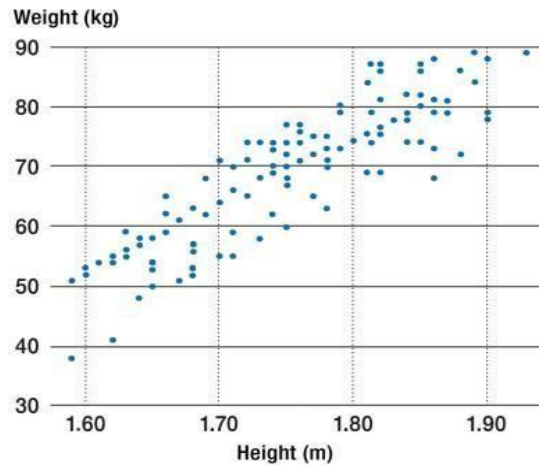
Figure 1
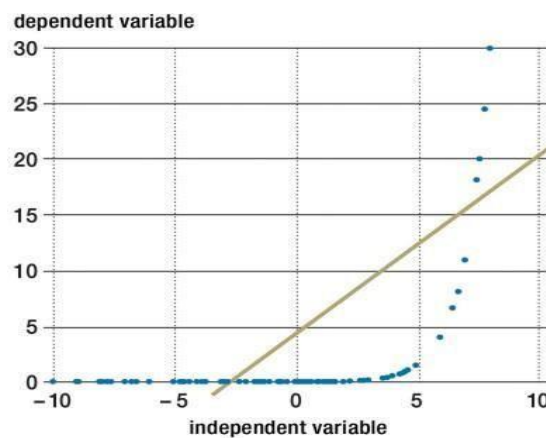
Figure 2

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 <u>by hand</u>, 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
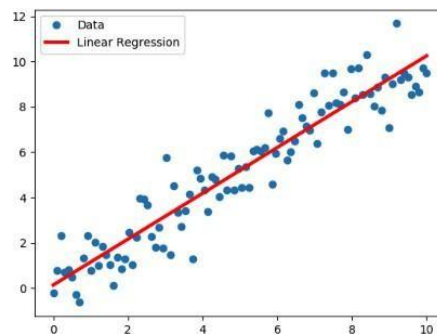
generalizes to unseen data.

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:

```
>>> 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:

```
>>> 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: 5.633333333333329

>>> print(f"slope: {model.coef_}")
slope: [0.54]
```

The code above illustrates how to get b
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 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:

[[ 8.33333333]

[13.73333333]

[19.13333333]
```

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],


>>> 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/huzaifsayed/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.

<div align="center">**ASSIGNMENT No: 02**</div>

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

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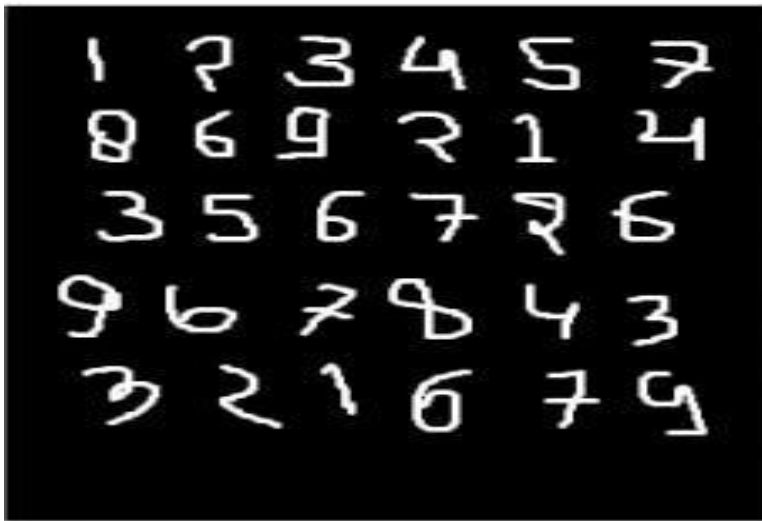
We can get 99.06% accuracy by using CNN(Convolutional Neural Network) with a functional model. The reason for using a functional model is to maintain easiness while connecting the layers.

Firstly, include all necessary libraries

Python3:

```python
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:

```python
(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:

```python
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:

```python
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

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.
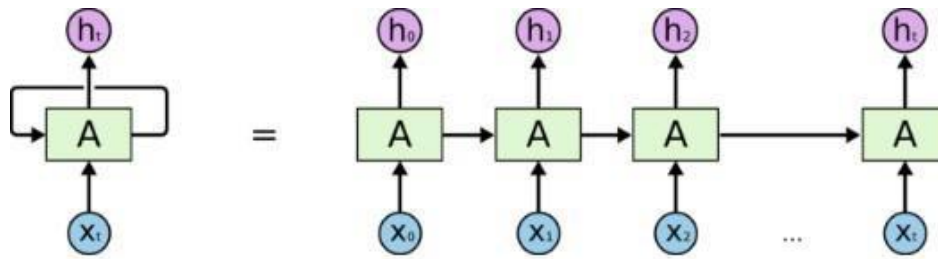
An unrolled recurrent neural network.

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 funtion, that implements a Non-linearity that squashes the activations to the range[-1.1]

$$y_t = W_{hy}h_t$$

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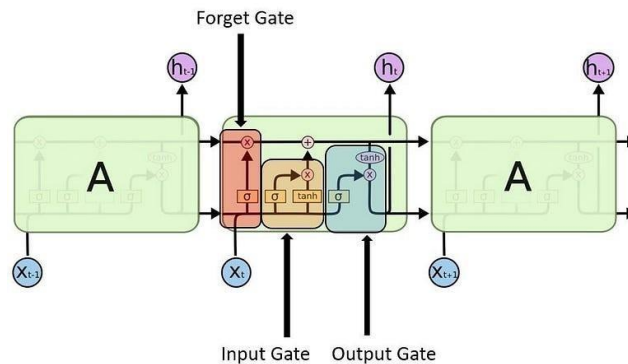
1) <u>Input gate</u> - discover which value from input should be used to modify the memory. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1.

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

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

2) <u>Forget gate</u> - discover what details to be discarded from the block. It is decided by the sigmoid function. it looks at the previous state(ht-1) and the content input(Xt) and outputs a number between 0(omit this)and 1(keep this)for each number in the cell state Ct−1.

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

3) <u>Output gate</u> — 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.

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

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

Algorithm –

1) Load IMDb Movie Reviews dataset (50,000 reviews)
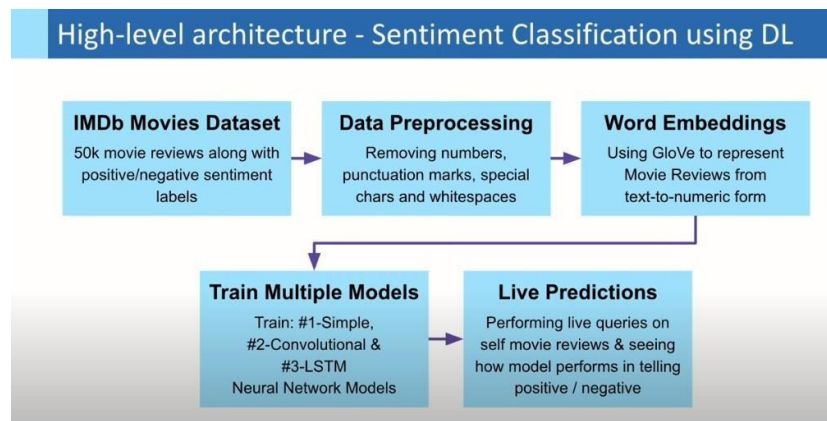
2) Pre-process dataset by removing special characters, numbers, etc. from user reviews + convert sentiment labels positive & negative to numbers 1 & 0, respectively

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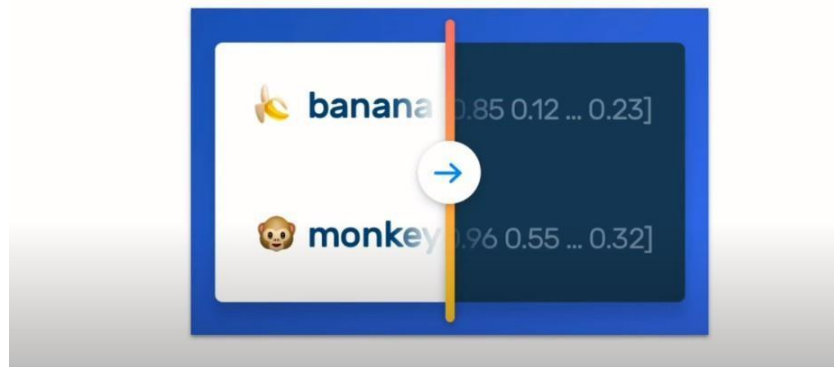
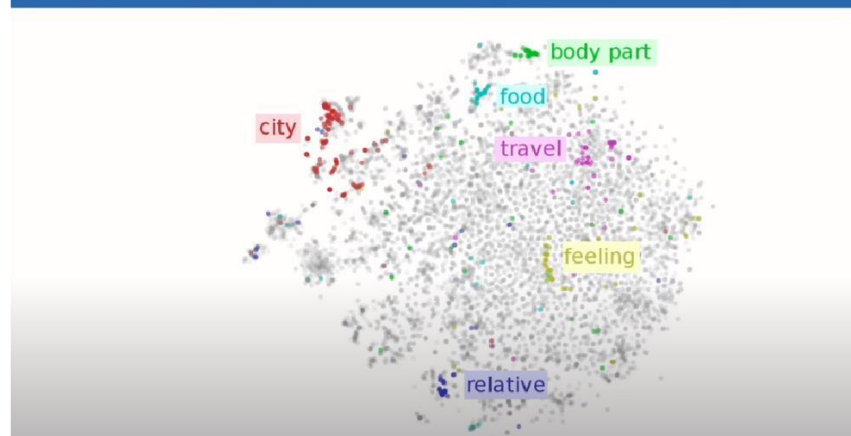3) Import GloVe Word Embedding to build Embedding Dictionary + Use this to build Embedding Matrix for our Corpus

Word Embeddings | Text-to-Numeric Representation

Feline: relating to or behaving like cats

Word Embeddings | Text-to-Numeric Representation

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

Application –

1) <u>Product Review Sentiment Analysis:</u>
Predict sentiment (positive, negative, neutral) from user reviews for product improvement insights.
2) <u>Customer Feedback Analysis:</u>
Analyze sentiments in customer feedback to understand overall satisfaction and identify areas for improvement.
3) <u>Brand Monitoring:</u>
Monitor social media for product mentions and analyze sentiments to assess brand perception.
4) <u>Market Research:</u>
Analyze sentiments in market surveys to gauge consumer opinions about specific products or features.
5) <u>Quality Assurance in E-commerce:</u>
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/oWo9SNcyxlI?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 –

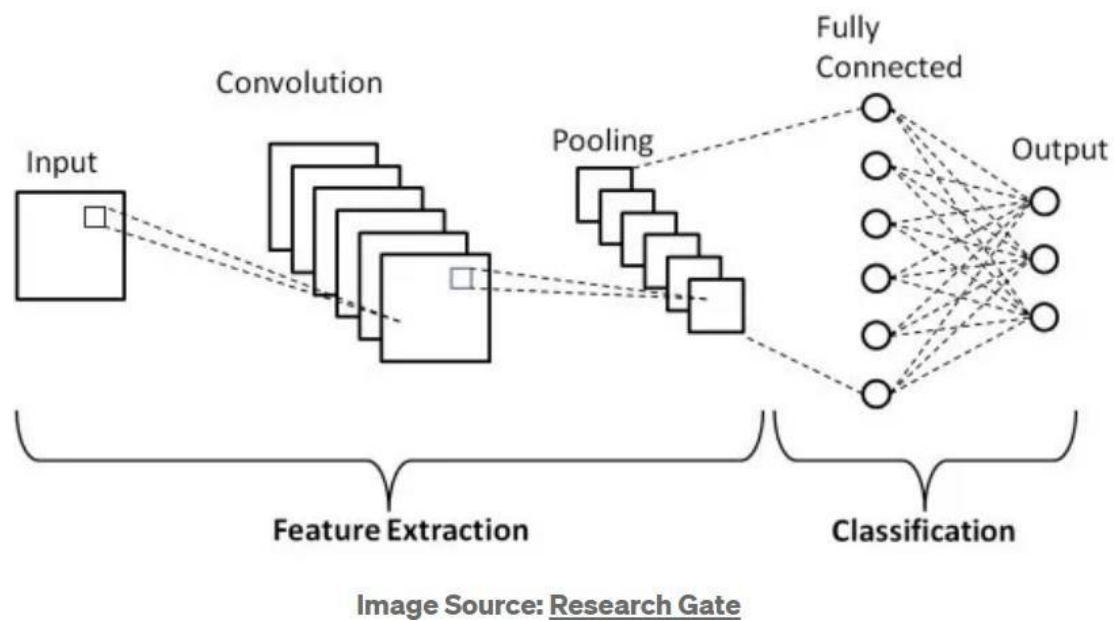## Convolutional Neural Network (CNN)

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).
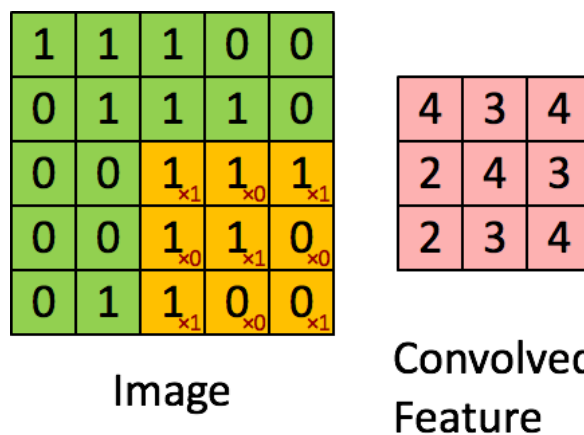
The Convolution Layer:

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.

Kernel

Image

Activation Map

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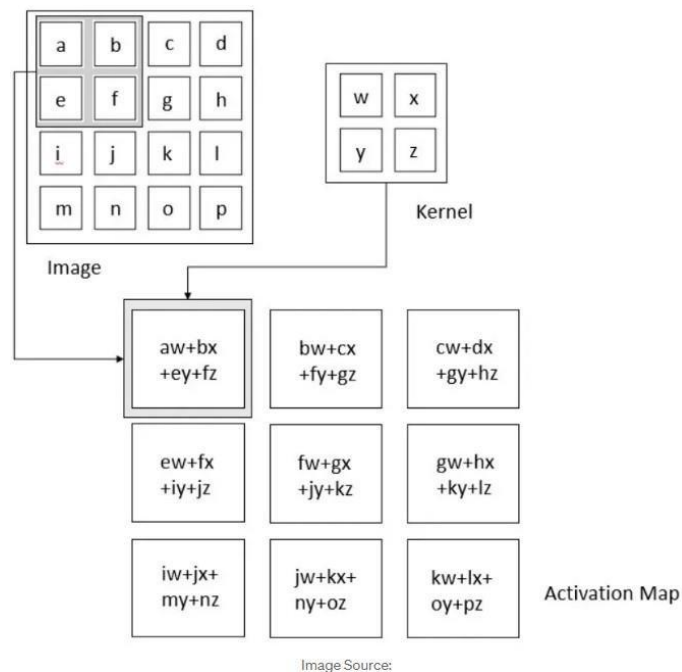
Each element from one tensor (image pixel) is multiplied by the corresponding element (the element in the same position) of the second tensor(kernel value), and then all the values are summed to get the result.

Image

Convolved
Feature

Image Source: NVIDIA

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:

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

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: output_size = 1 + (input_size * kernel_size + (2 * padding)) / stride.

Plugging in the values, we get: output_size = 1 + (6*3 + (2 * 0)) / 1 = 1 + (3 / 1) = 1 + 3 = 4.
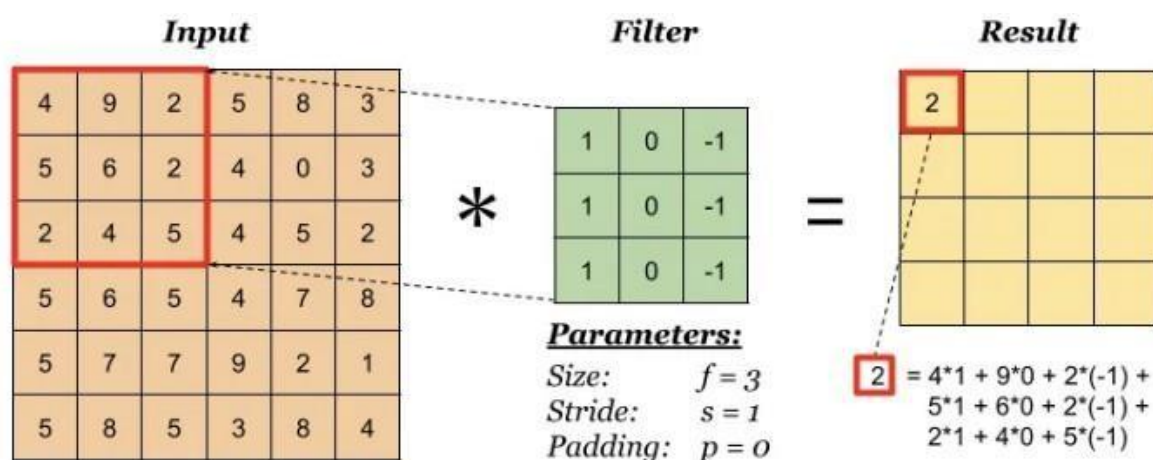
Image By Author: Convolution on 2D Image / Single Channel

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.
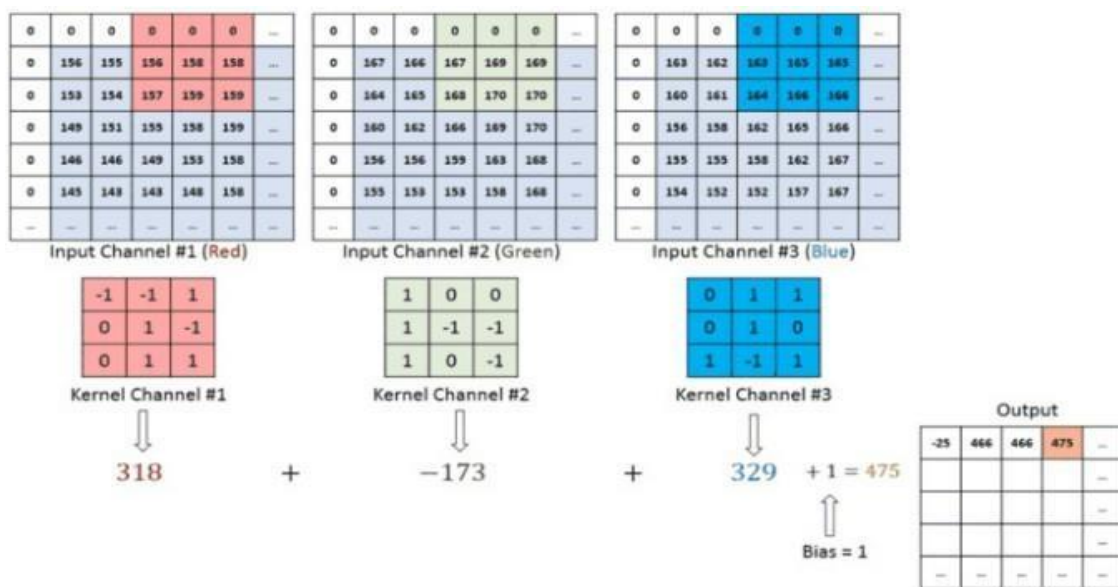
Image By Author: Convolution on RGB Image

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.

Single Depth Slice
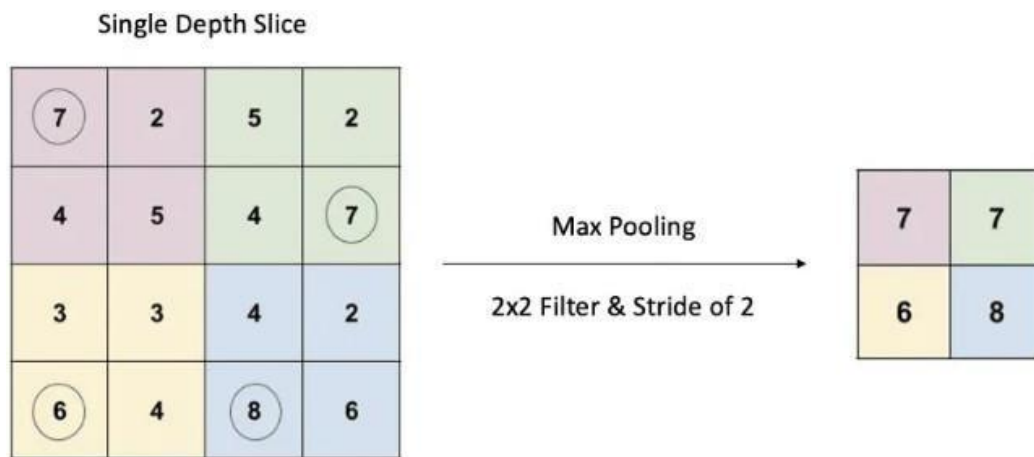
Image By Author: Max Pooling

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.
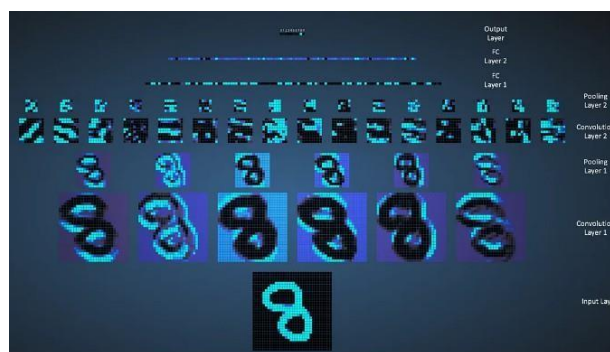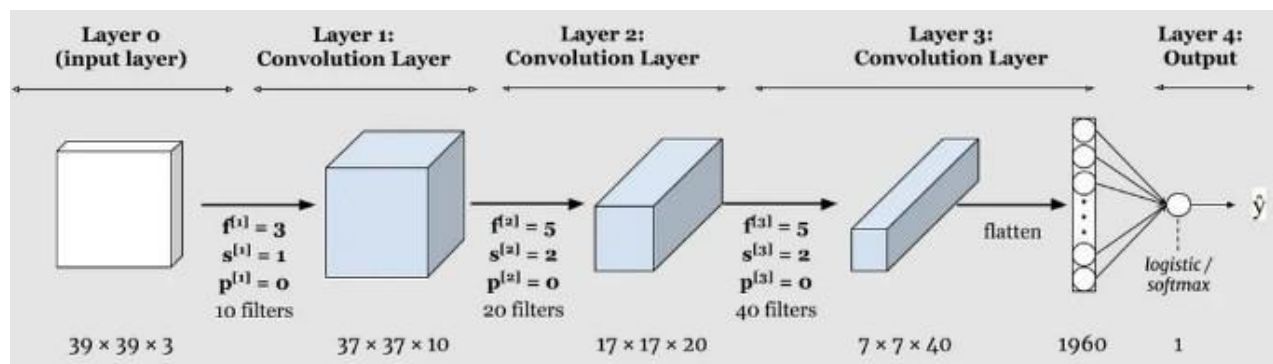

Image Source: Here

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 5x5x2 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
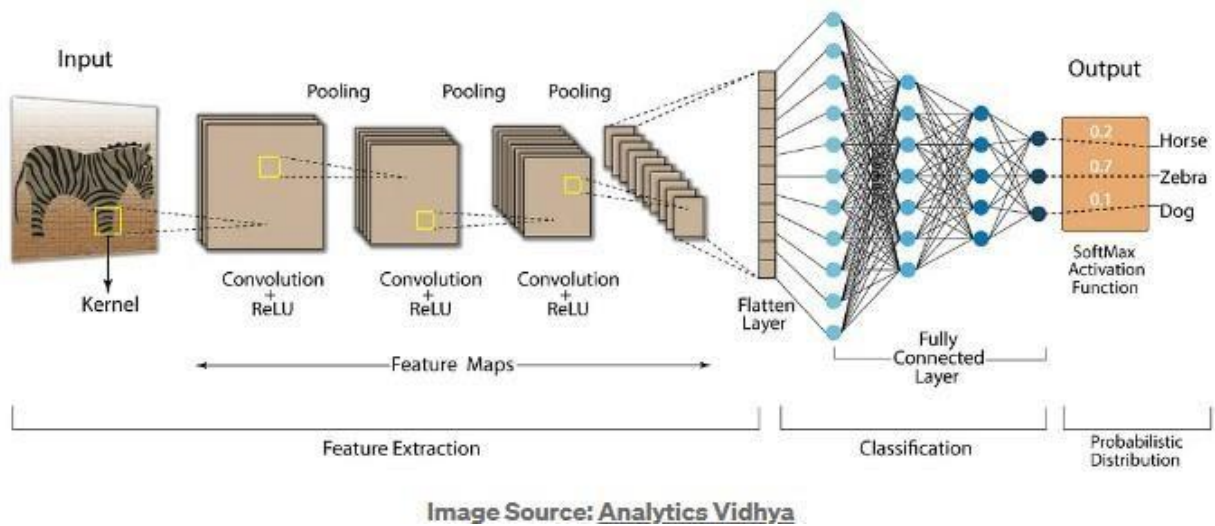
Algorithm –

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)
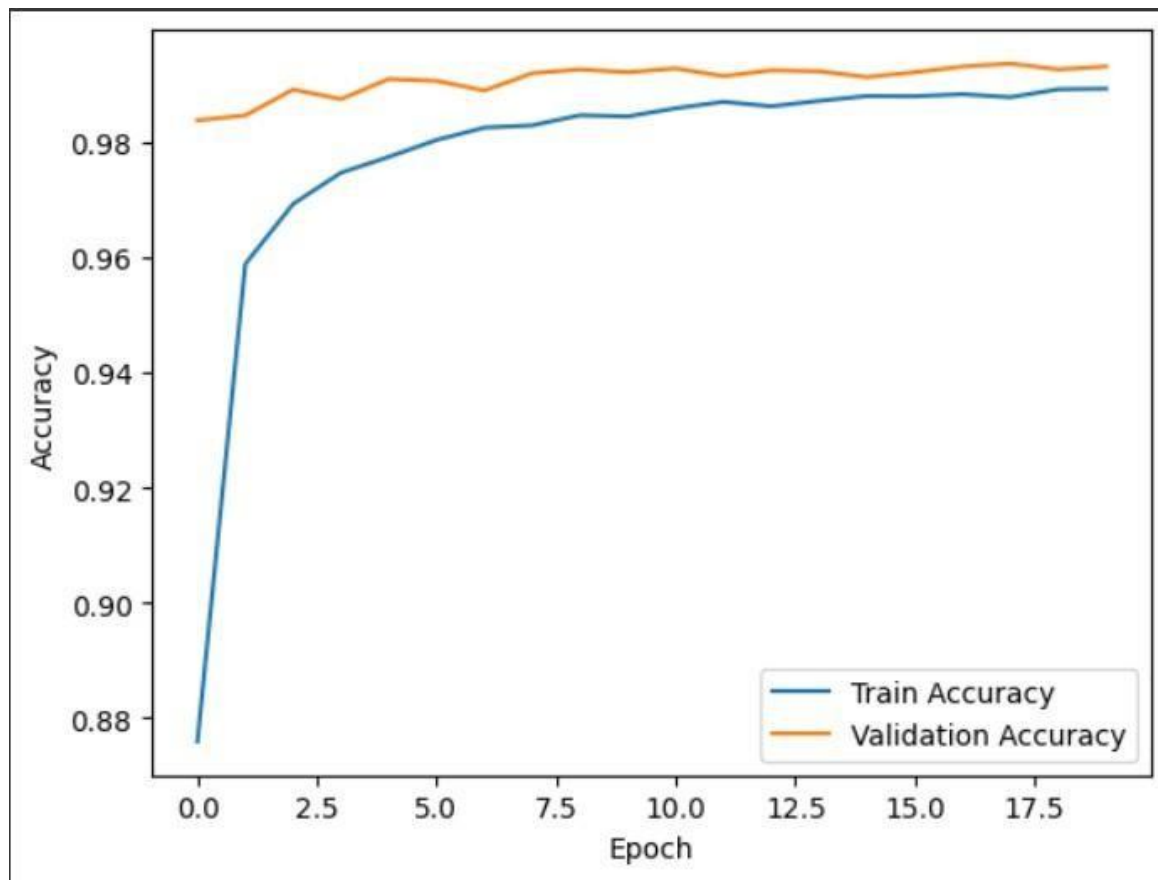print(f"Test Accuracy: {test_acc}")

# 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-aab30dface10

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.
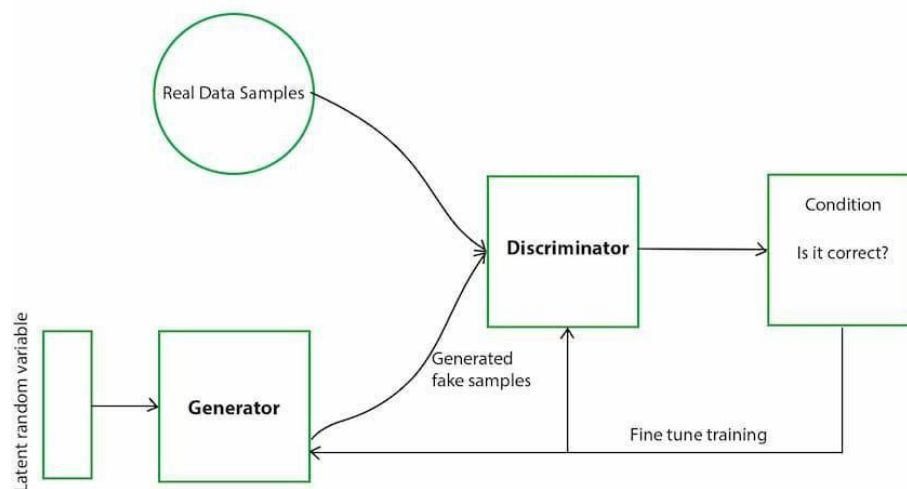
Training Process:

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.
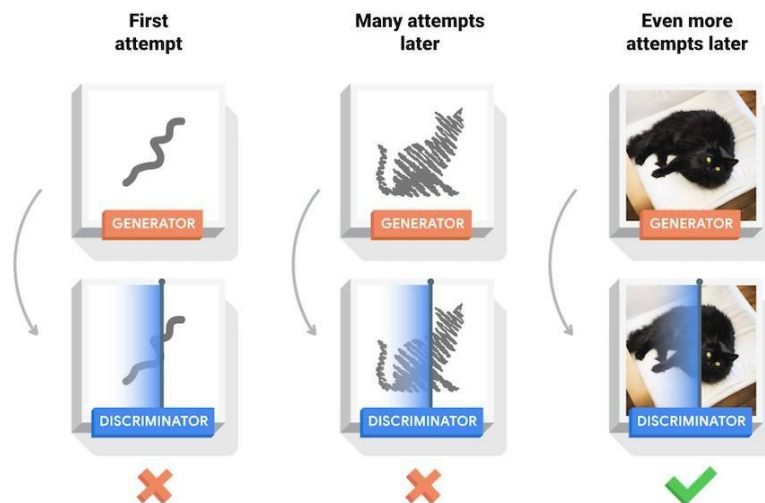
Algorithm –

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]))
```

```python
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

How the noise looks
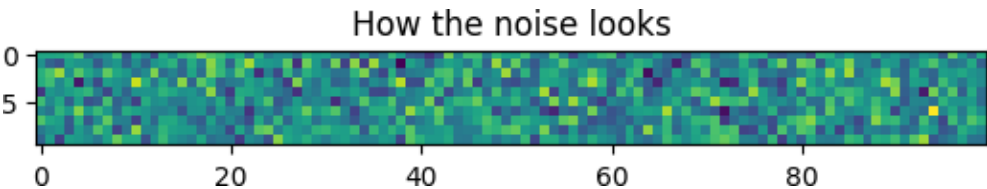
Generated Images from Noise using GANs

References–

https://www.tensorflow.org/tutorials/generative/dcgan
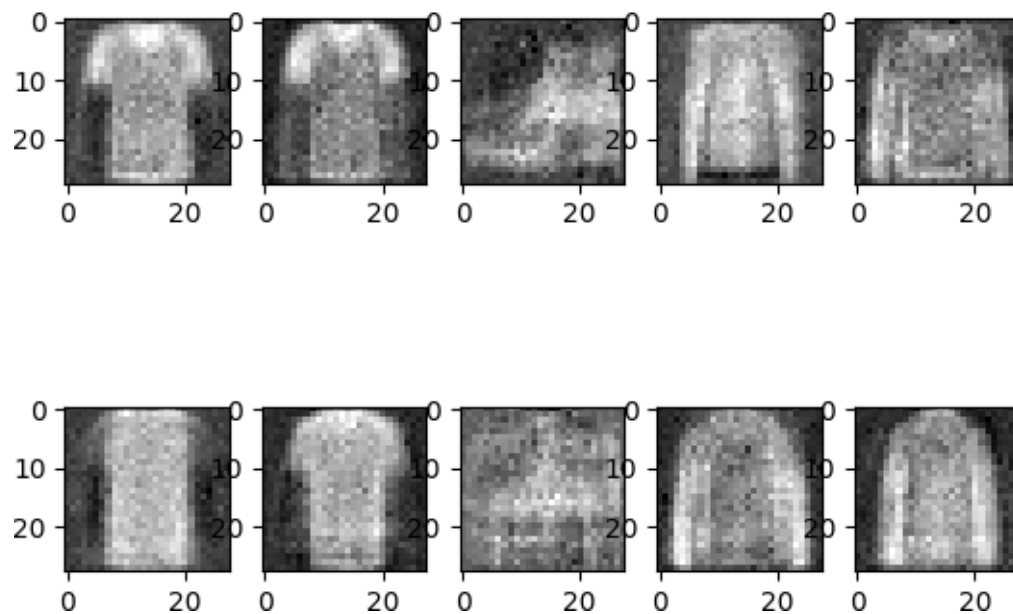
Conclusion: Thus Designed and implemented Deep Convolutional GAN to generate images of faces/digits from a set of given images.

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
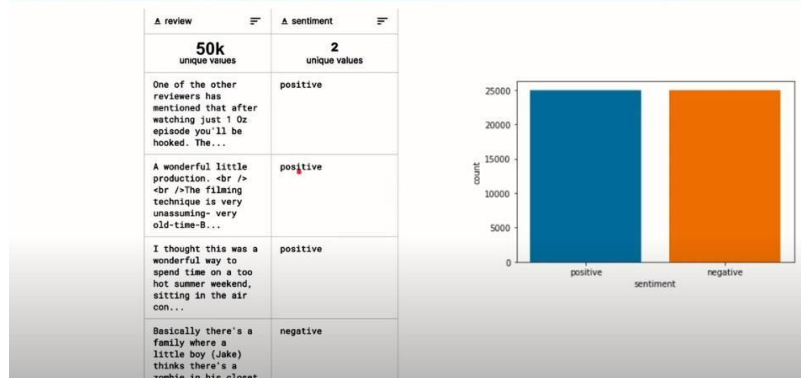
Libraries or Modules Used–

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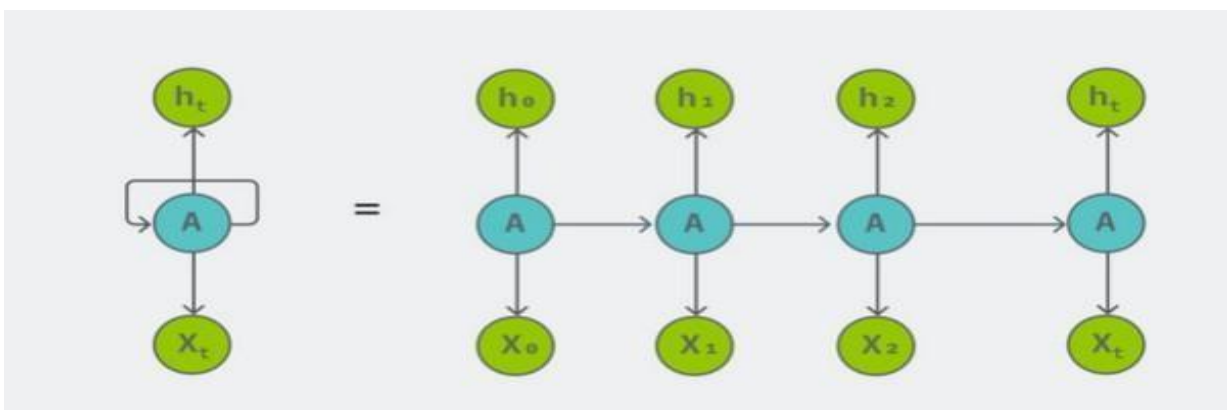
RNNs, uniquely suited for tasks like unsegmented connected handwriting recognition or speech

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 (Whx) and the previous hidden state (Whh), alongside an activation function, typically tanh. This activation function introduces non-linearity, compressing activations within the range of [-1, 1]. The final output state (Yt) is determined by the weight matrix (Why) at the output layer.

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

Applying Activation Function:

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

$$y_t = W_{hy}h_t$$

Yt is the output state. Why 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-   recurrent-neural-network/amp/

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 outof-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"ù 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.

Figure 1. Legacy data sources.
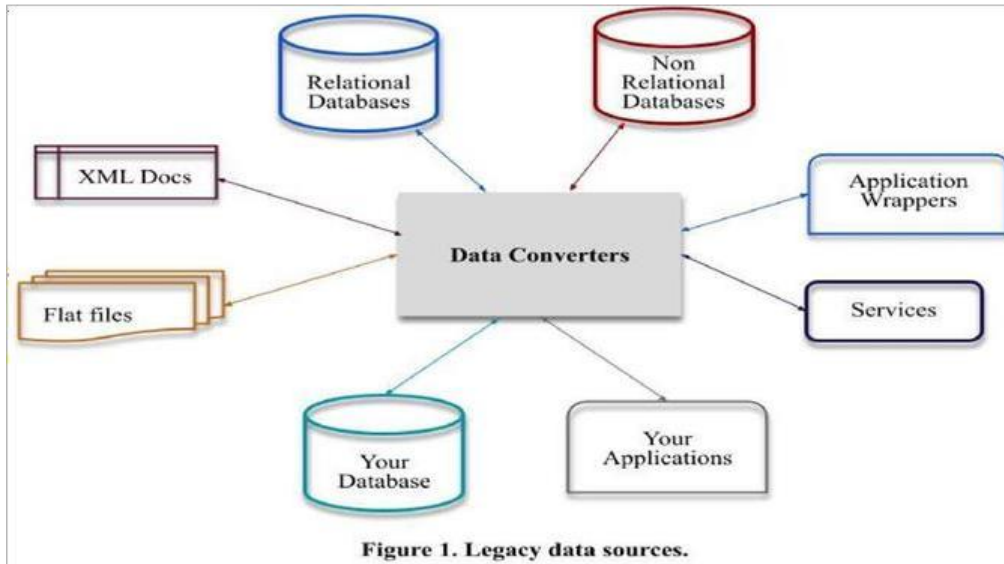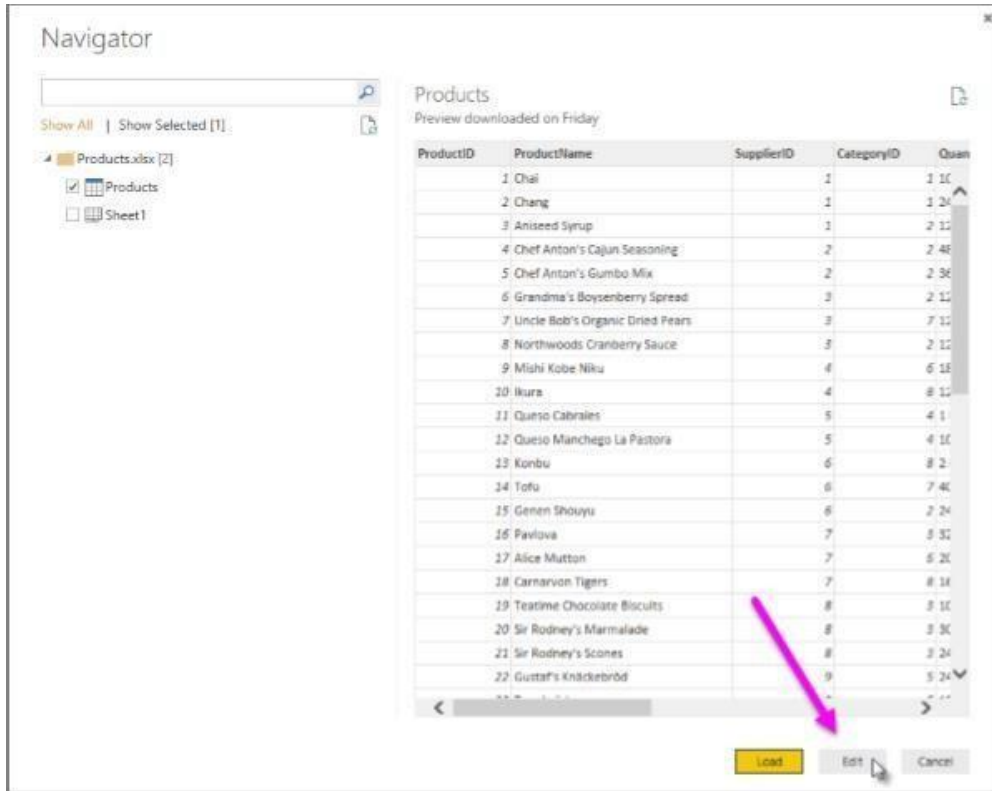
Importing Excel Data

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.
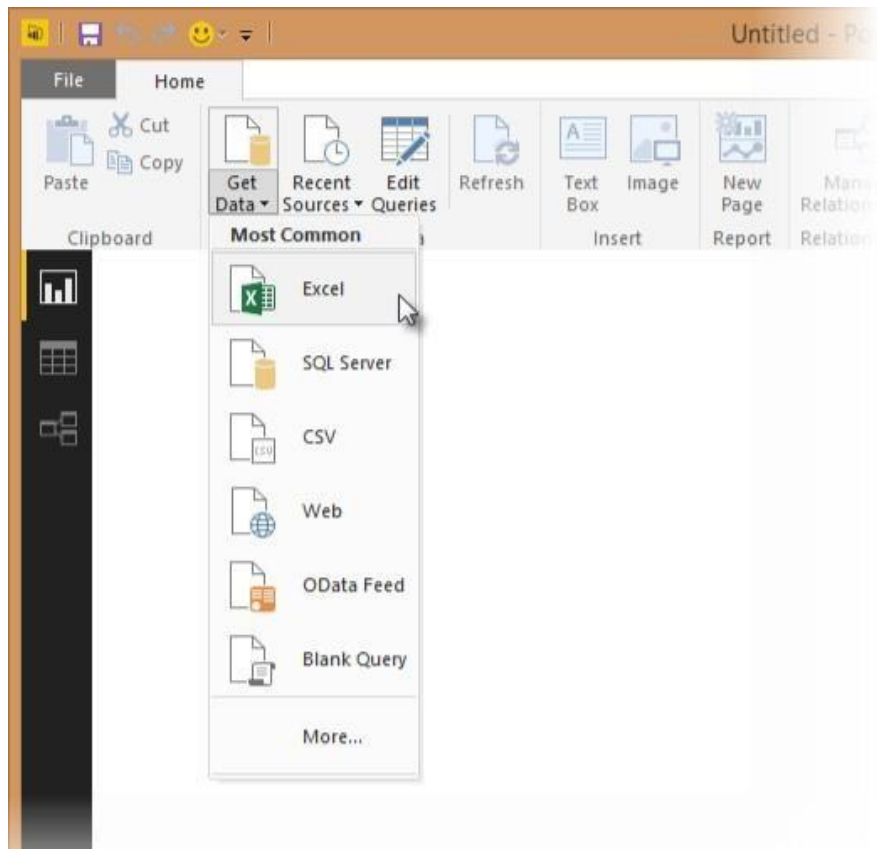
3) If you select the Get Data button directly, you can also select FIle > Excel and select Connect.

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.

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.

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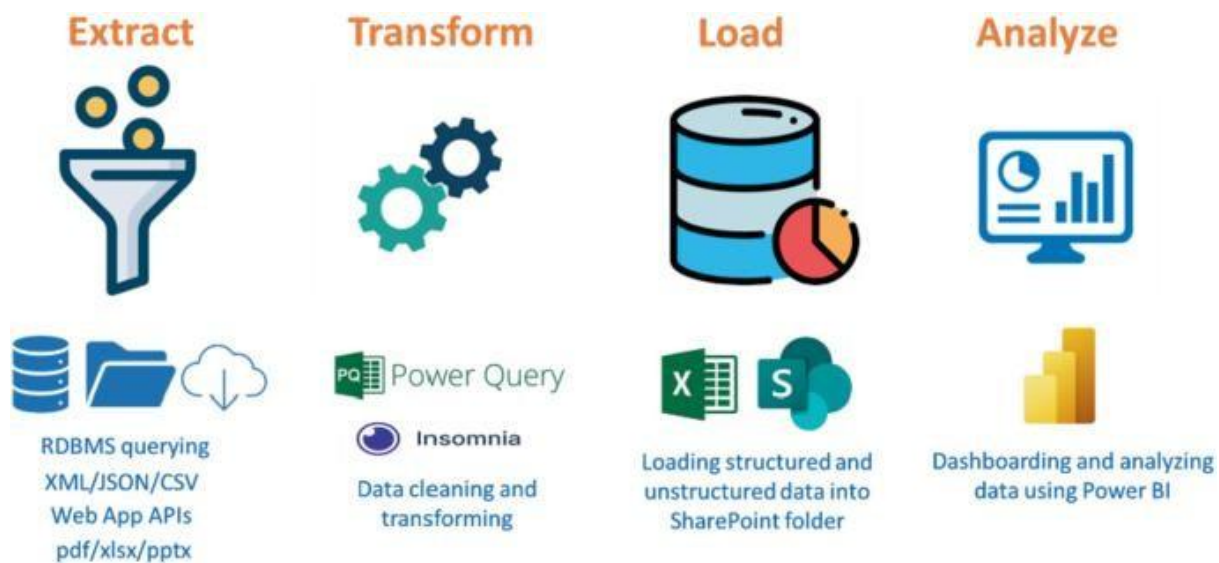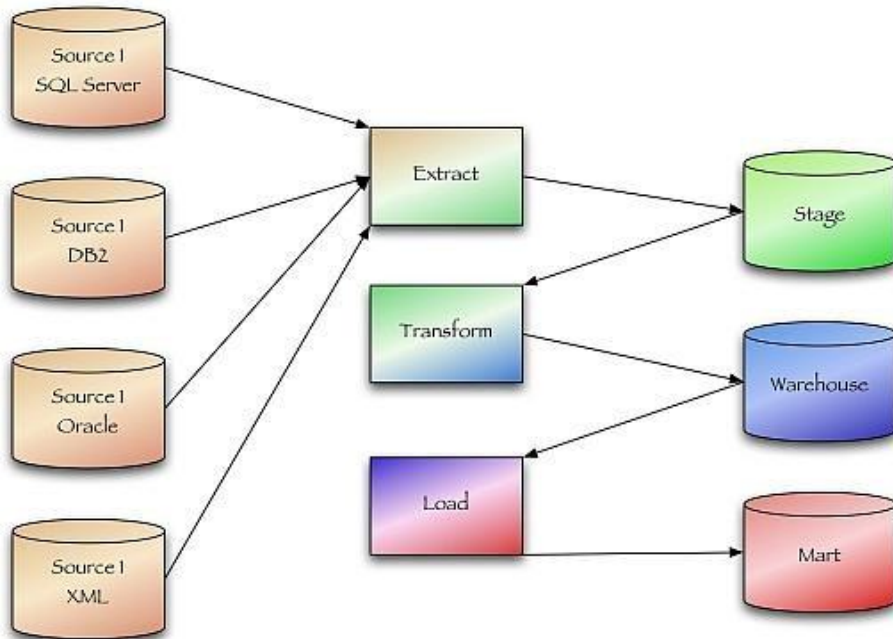
Conclusion:- This way Data Visualization from Extraction Transformation and Loading (ETL) Process is done.

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 DDMMYY format and in target the date is in DDMONYYYY format then this transformation needs to be done at transformation phase.
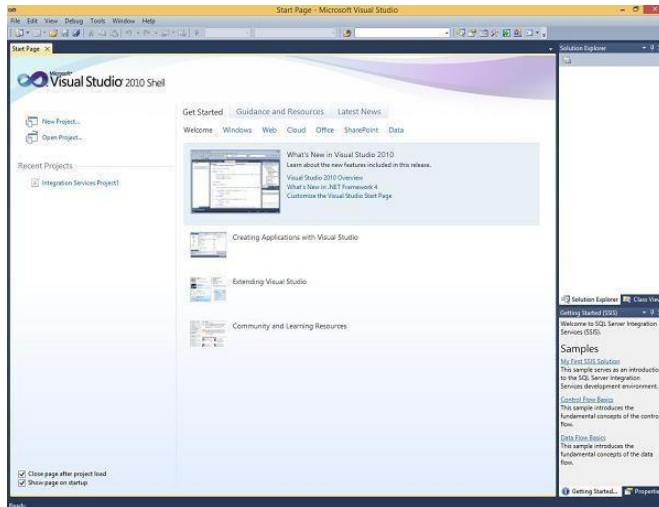
Step 3 : Data Loading

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

ETL process in SQL Server:

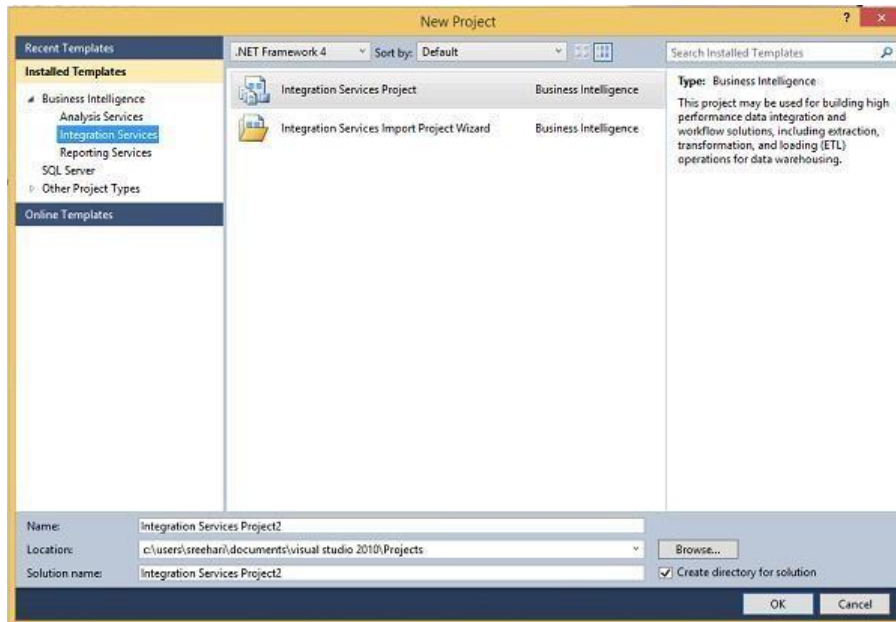Following are the steps to open BIDS\SSDT.

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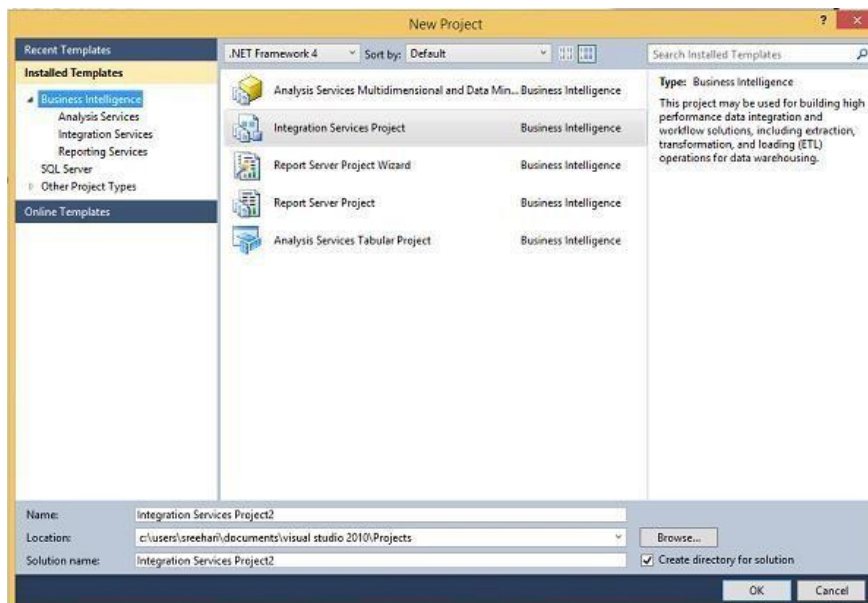
Step 3 − Select Integration Services under Business Intelligence on the top left corner in the above screen to get the following screen.

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.
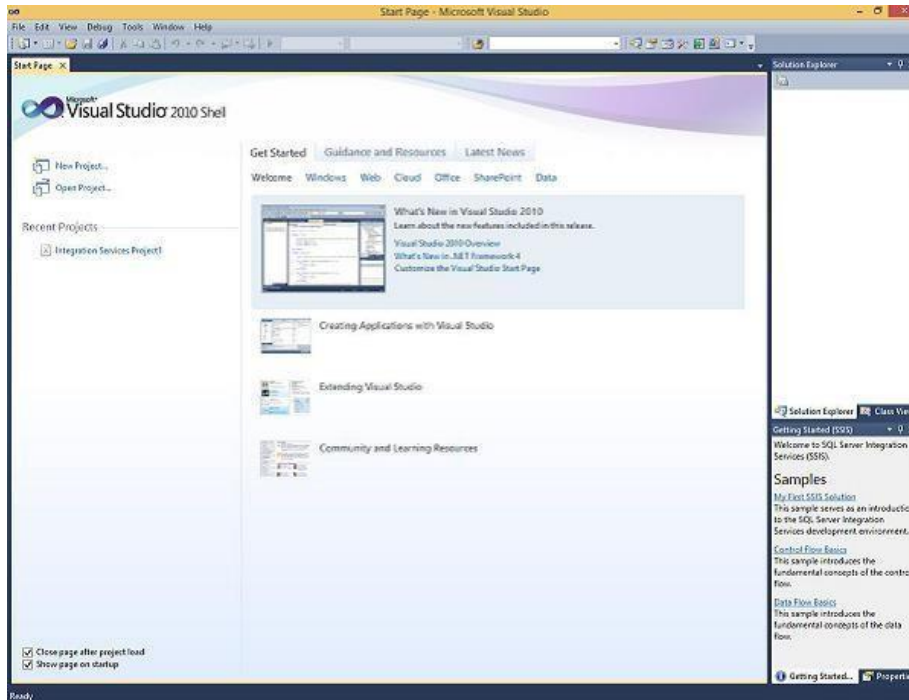
Step 2 − The above screen shows SSDT has opened. Go to file on the top left corner in the above image and click New. Select project and the following screen opens.
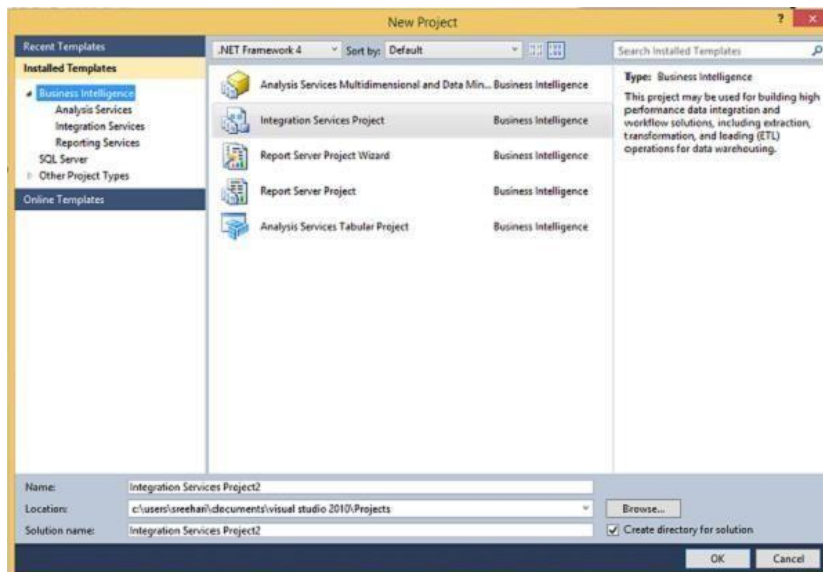
Step 3 − Select Analysis Services in the above screen under Business Intelligence as seen on the top left corner. The following screen pops up.
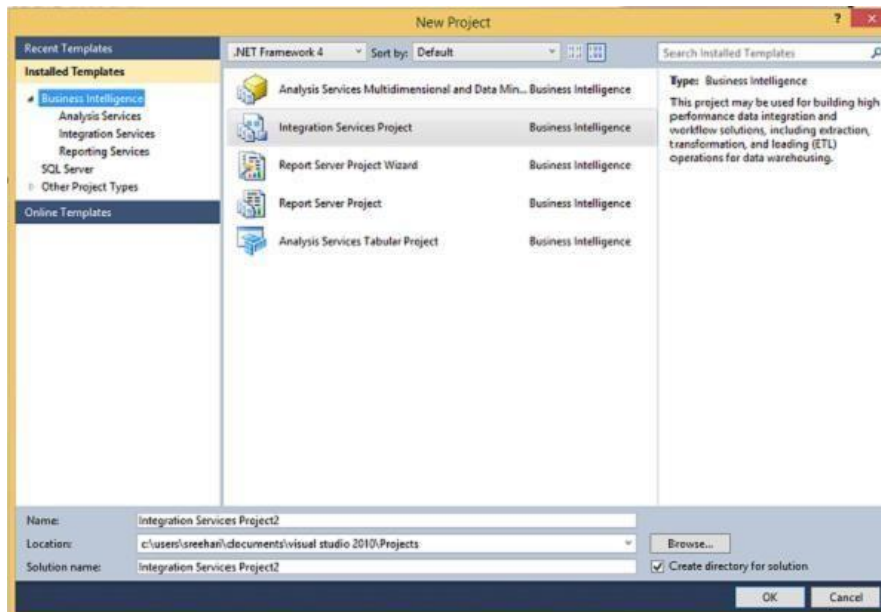
Step 4 − In the above screen, select any one option from the listed five options based on your requirement to work with Analysis services.

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.
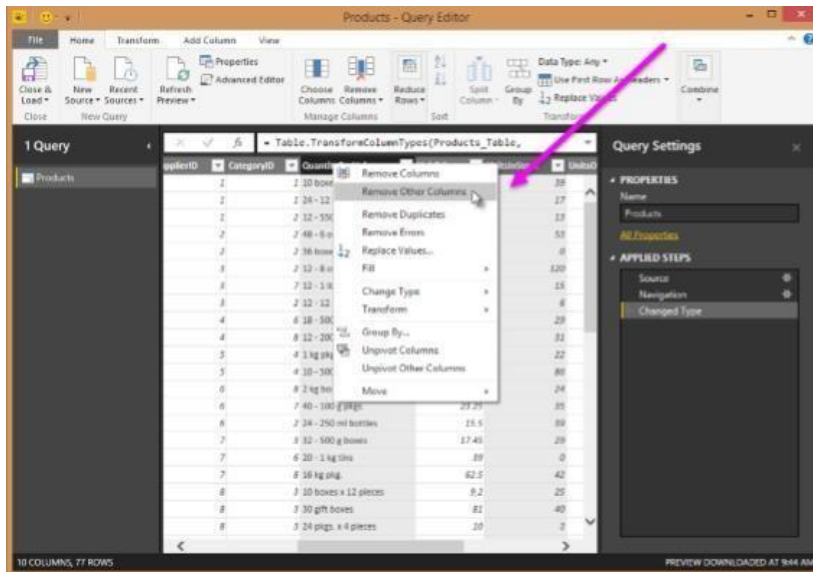
3. Change the data type of the UnitsInStock column

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).
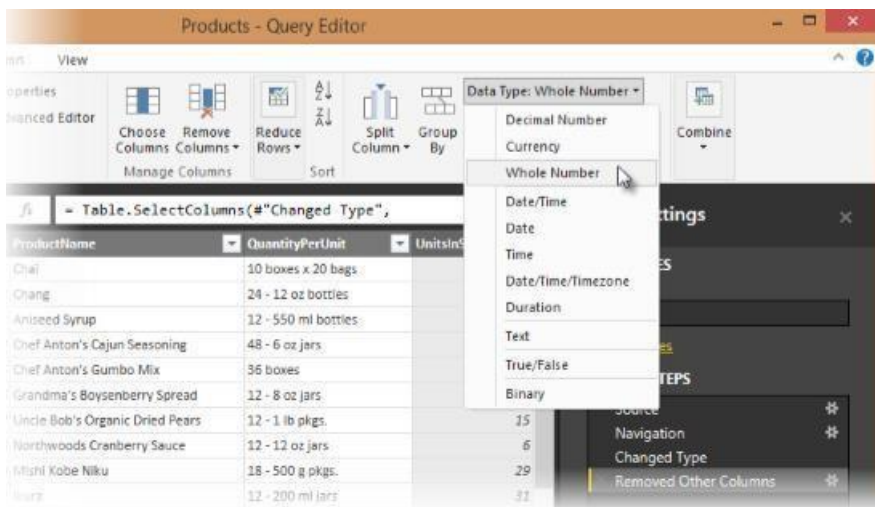

3. Expand the Order_Details table

The Orders table contains a reference to a Details table, which contains the individual productsthat were included in each Order. When you connect to data sources with multiples tables (suchas 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.
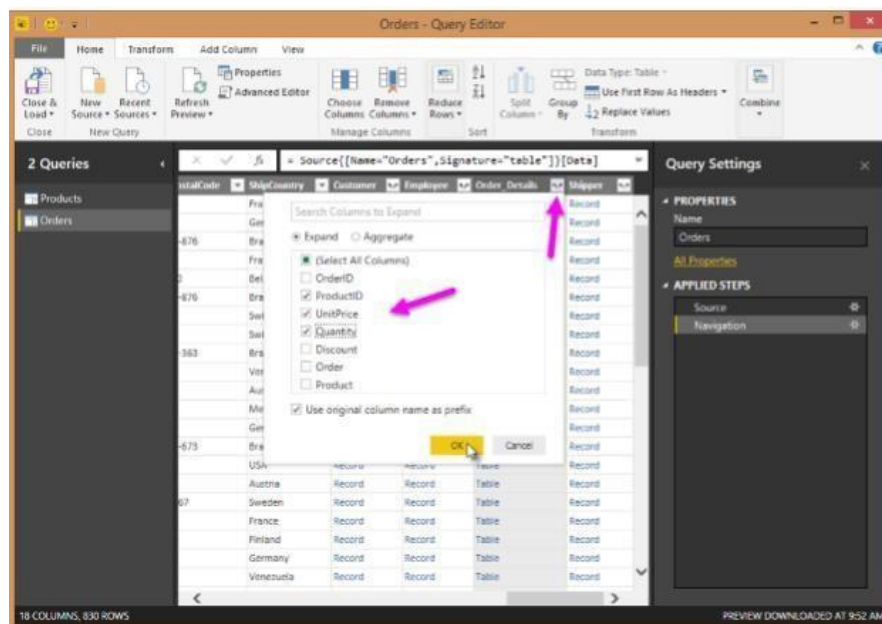
**4. Calculate the line total for each Order_Details row**

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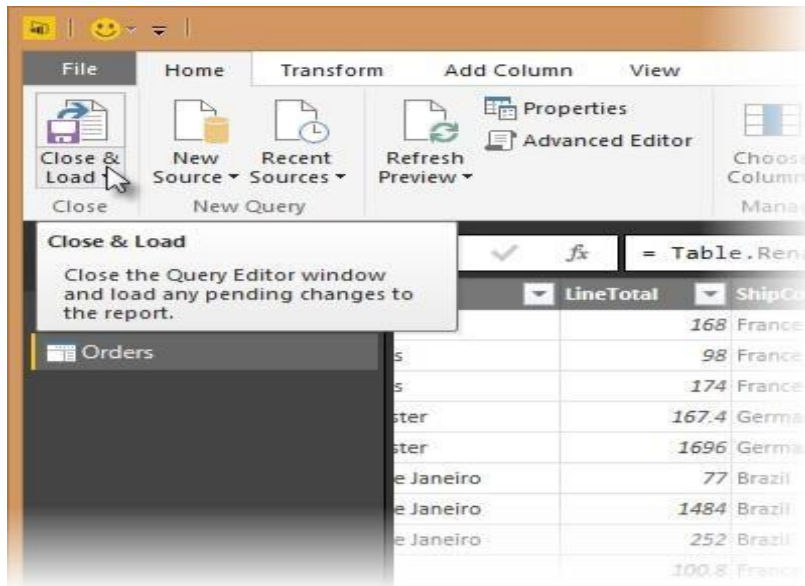
2. In the Add Custom Column dialog box, in the Custom Column Formula textbox, enter [Order_Details.UnitPrice] * [Order_Details.Quantity].
3. In the New column name textbox, enter LineTotal.
4. Click OK.
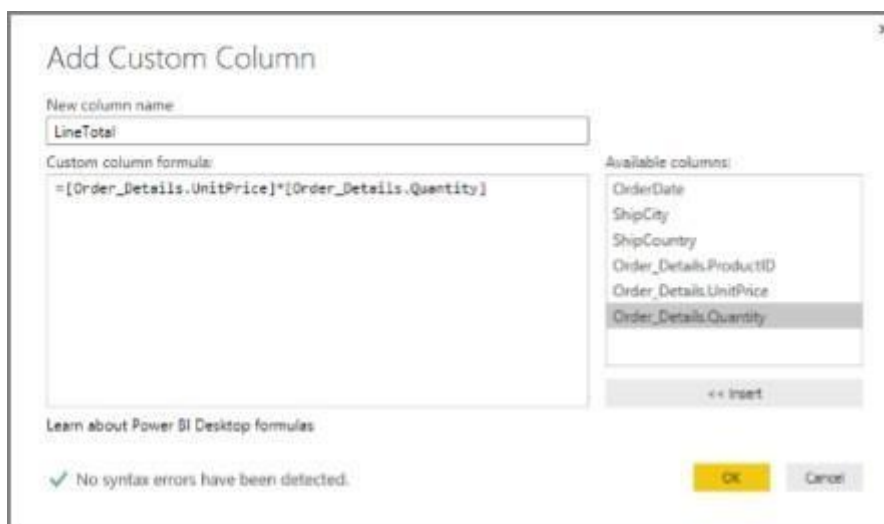
5. Rename and reorder columns in the query

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.

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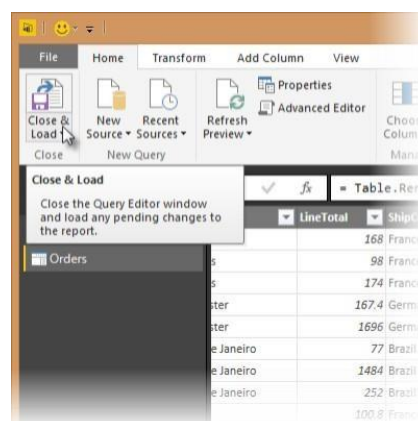
1. Power BI Desktop loads the data from the two queries.

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

3. Select the New… button
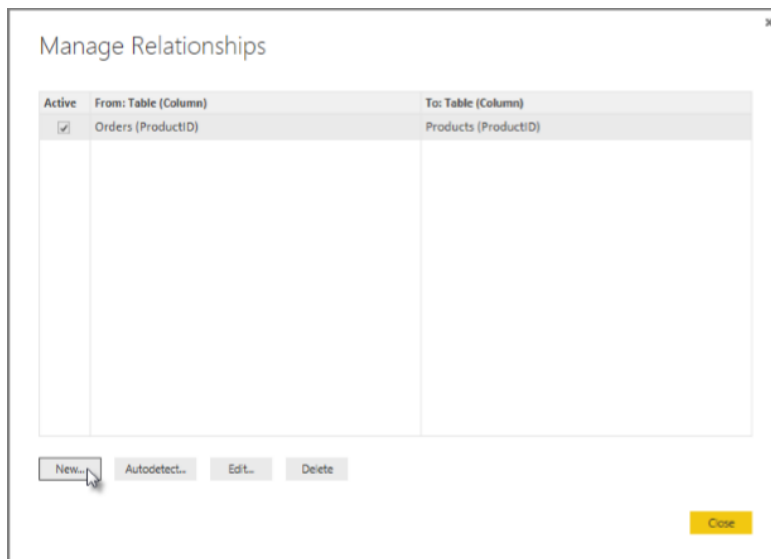
4. When we attempt to create the relationship, we see that one already exists! As shown in theCreate Relationship dialog (by the shaded columns), the ProductsID fields in each query already have an established relationship.
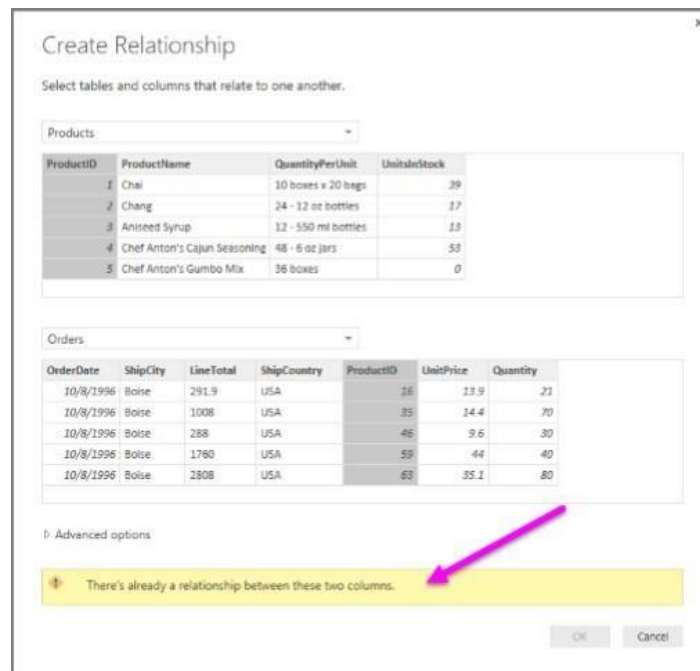
5.   Select Cancel, and then select Relationship view in Power BI Desktop.
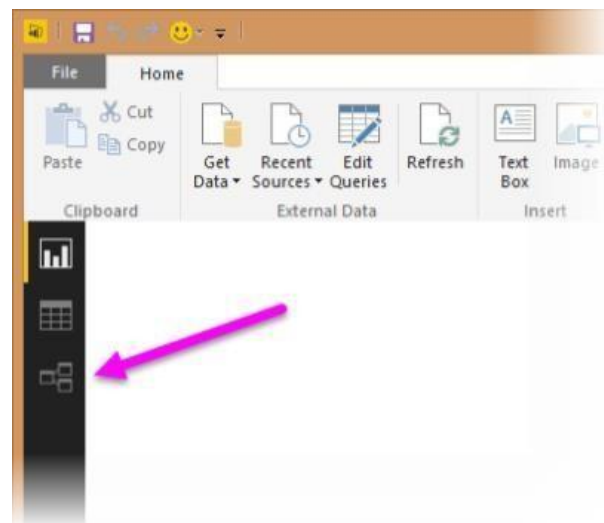
6.   We see the following, which visualizes the relationship between the queries.
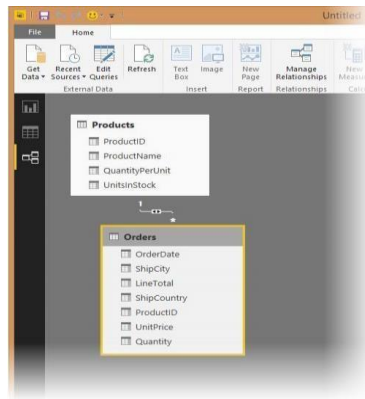
7.   When you double-click the arrow on the line that connects the to queries, an EditRelationship dialog appears.
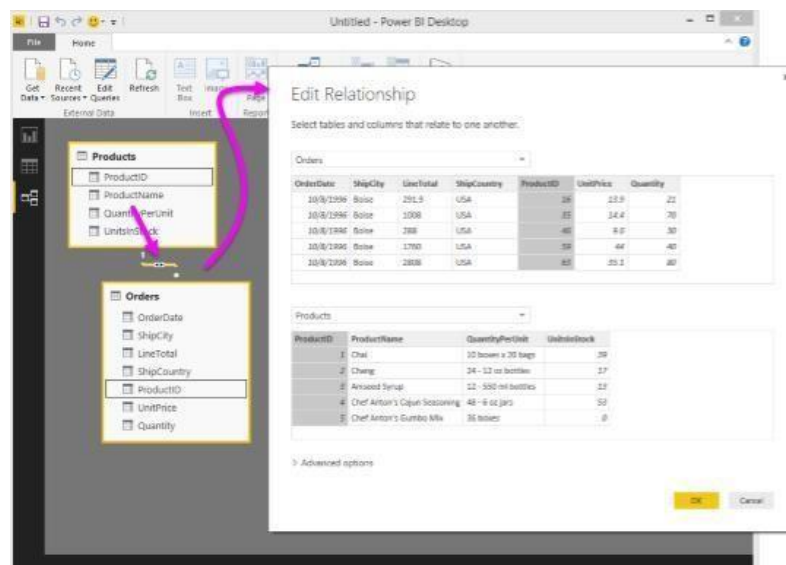
No need to make any changes, so we'll just select Cancel to close the EditRelationship dialog

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.
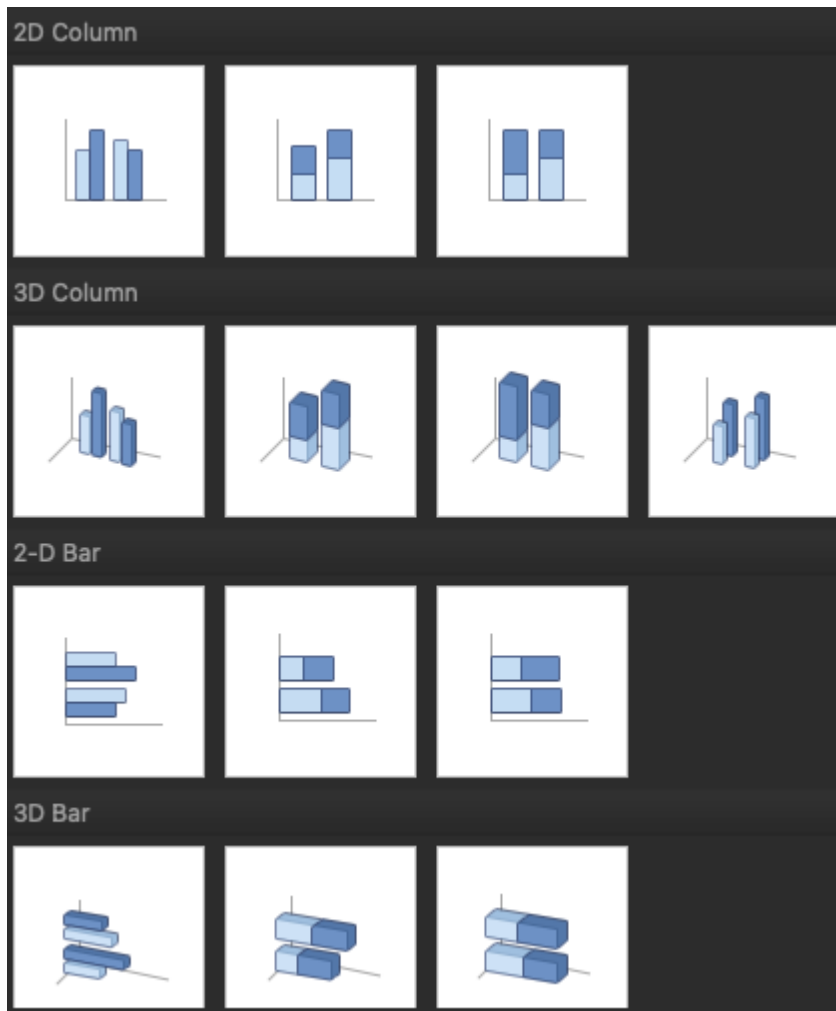
Excel bar charts

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"
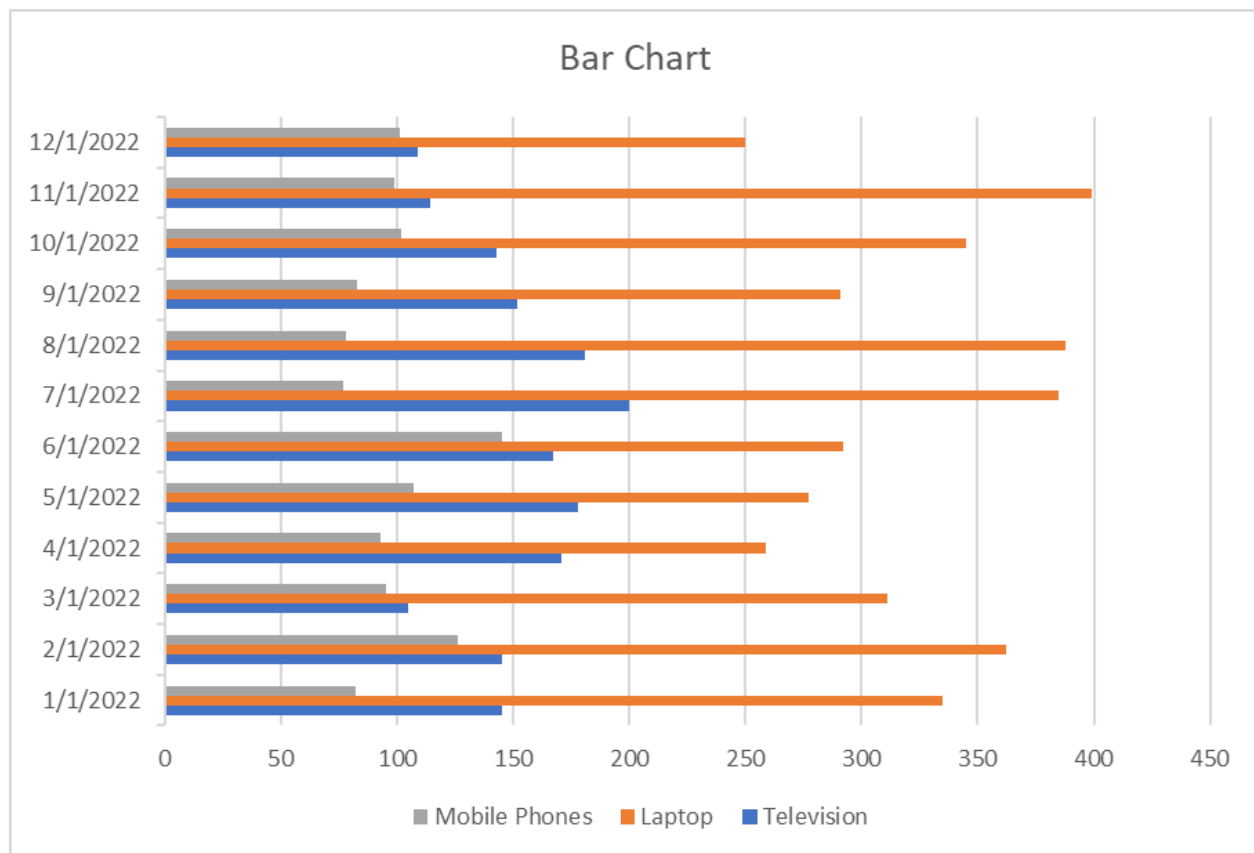
Excel column charts

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.
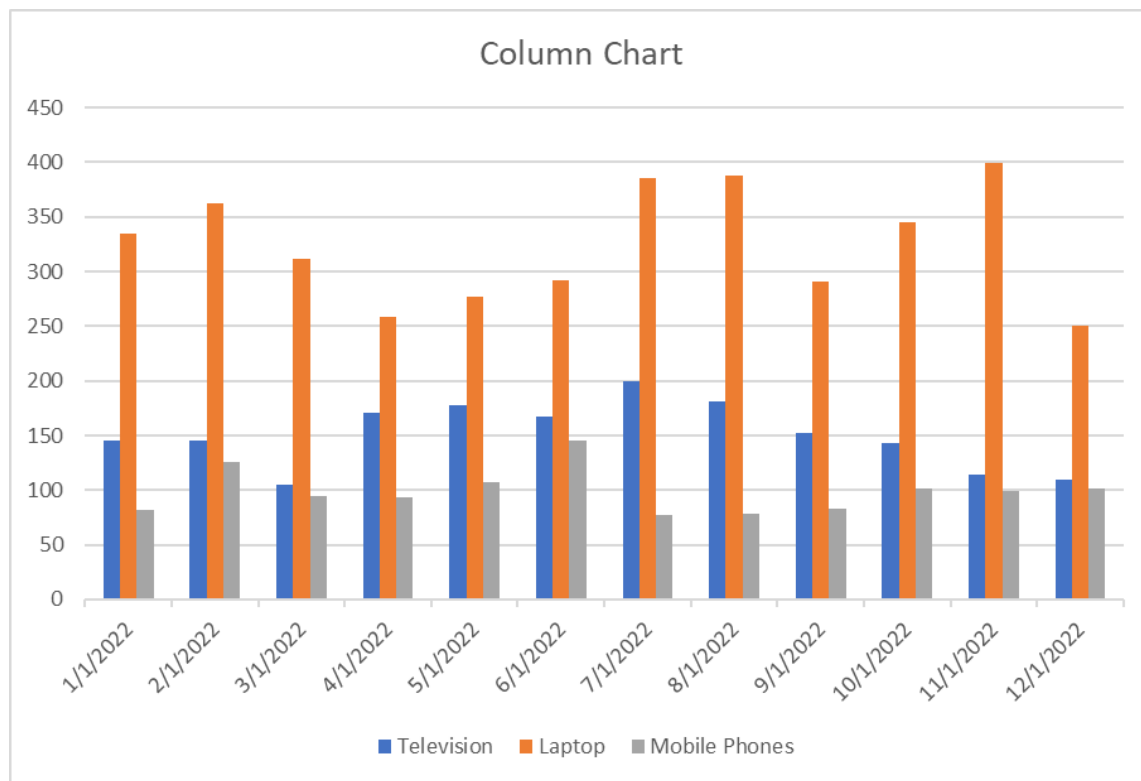
Excel line charts

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.
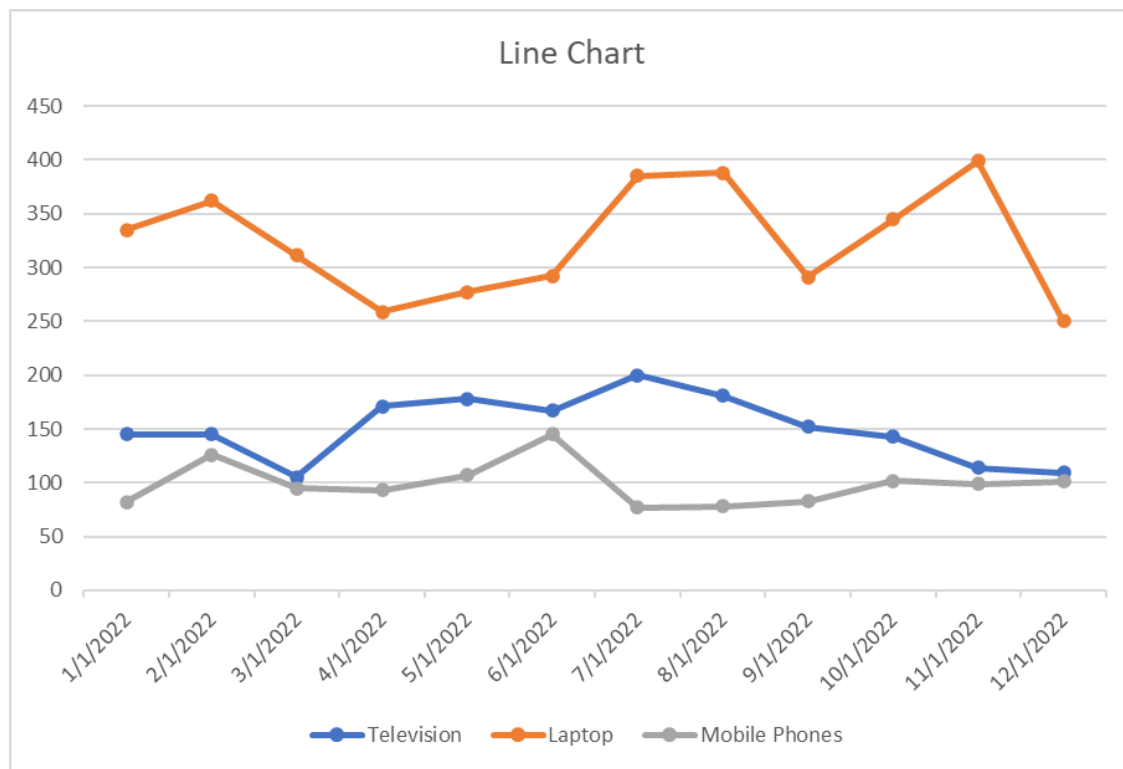
Excel pie charts

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"
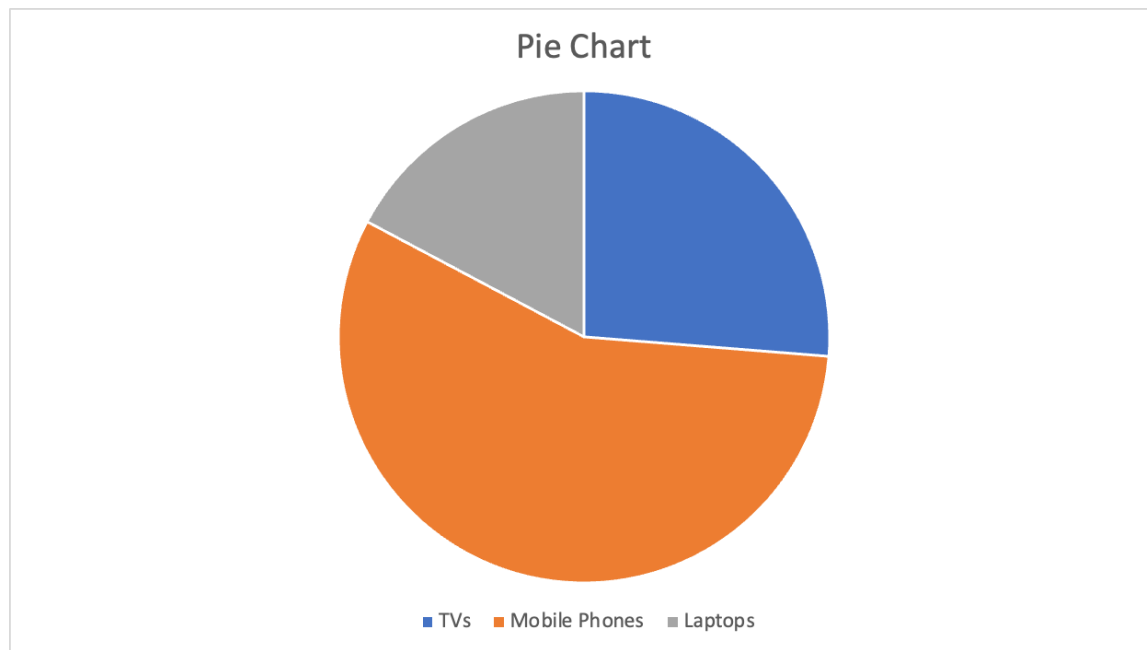
Advanced Excel Visualization Techniques

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"
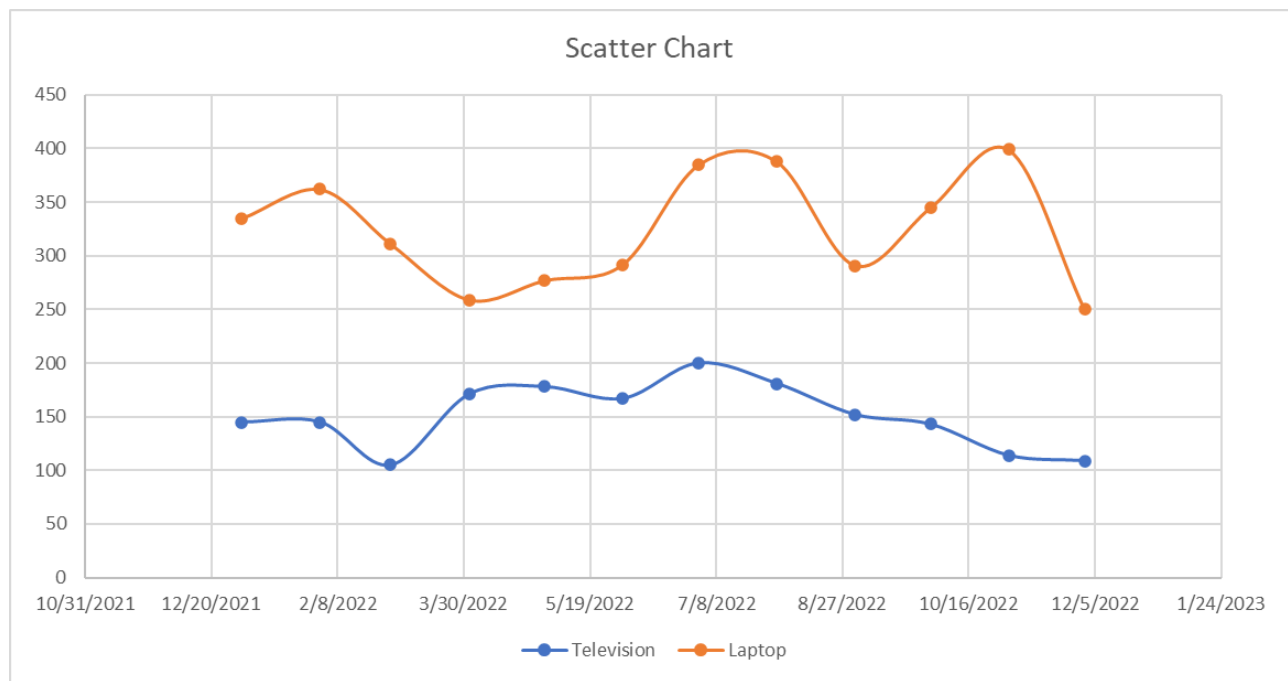
Excel waterfall chart

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.
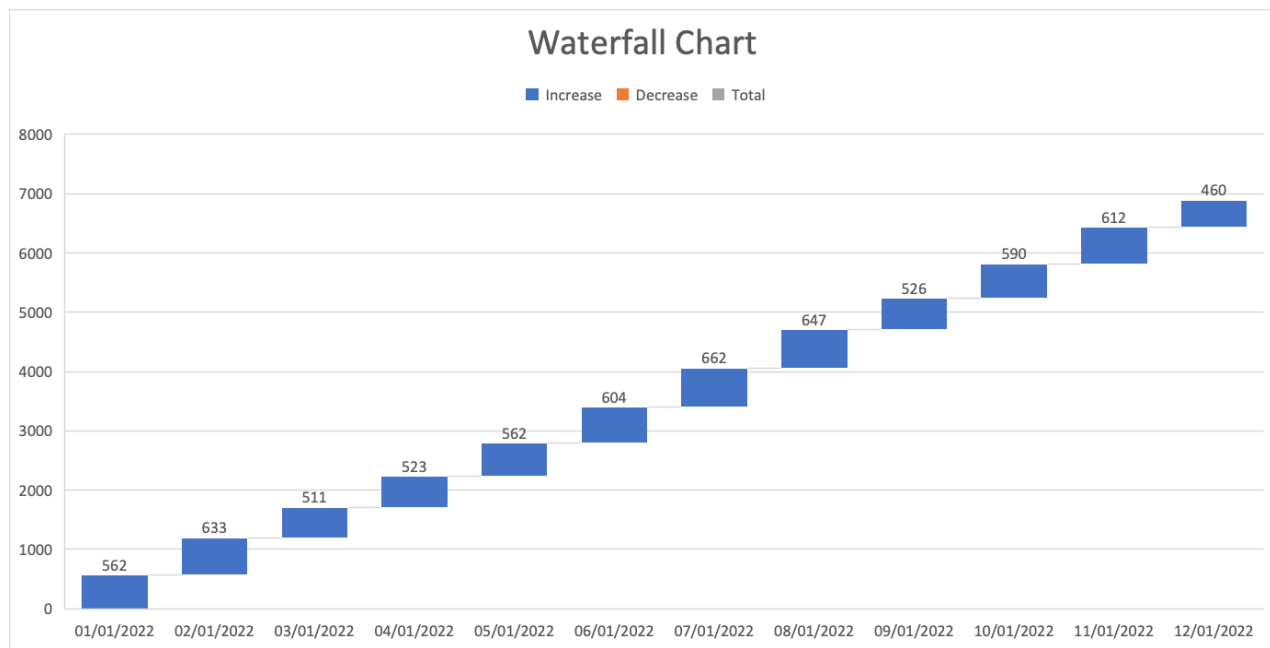
Customizing and Formatting Excel Charts

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

Add Chart Element ⌄	**Quick Layout ⌄**	Ch Col	

Axes   >

C   Axis Titles   >

Chart Title   >

1
2   Data Labels   >

3
4   Data Table   >

5   Error Bars   >

6
7   Gridlines   >

8   Legend   >

9
10   Lines   >

11   Trendline   >

12
13   Up/Down Bars   >
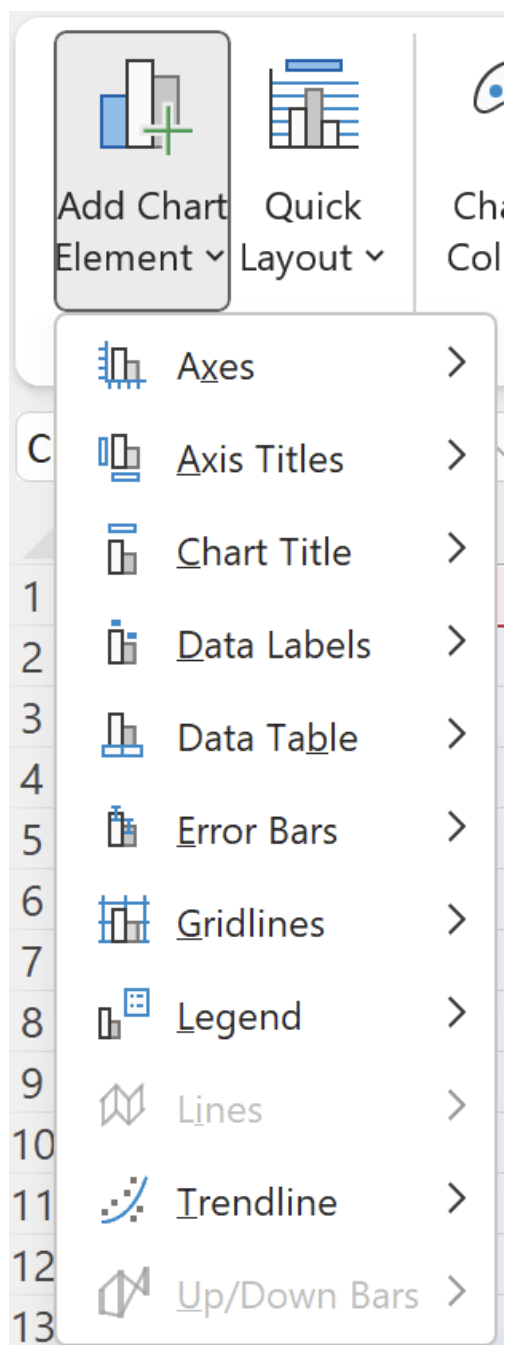
Chart title

- 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:
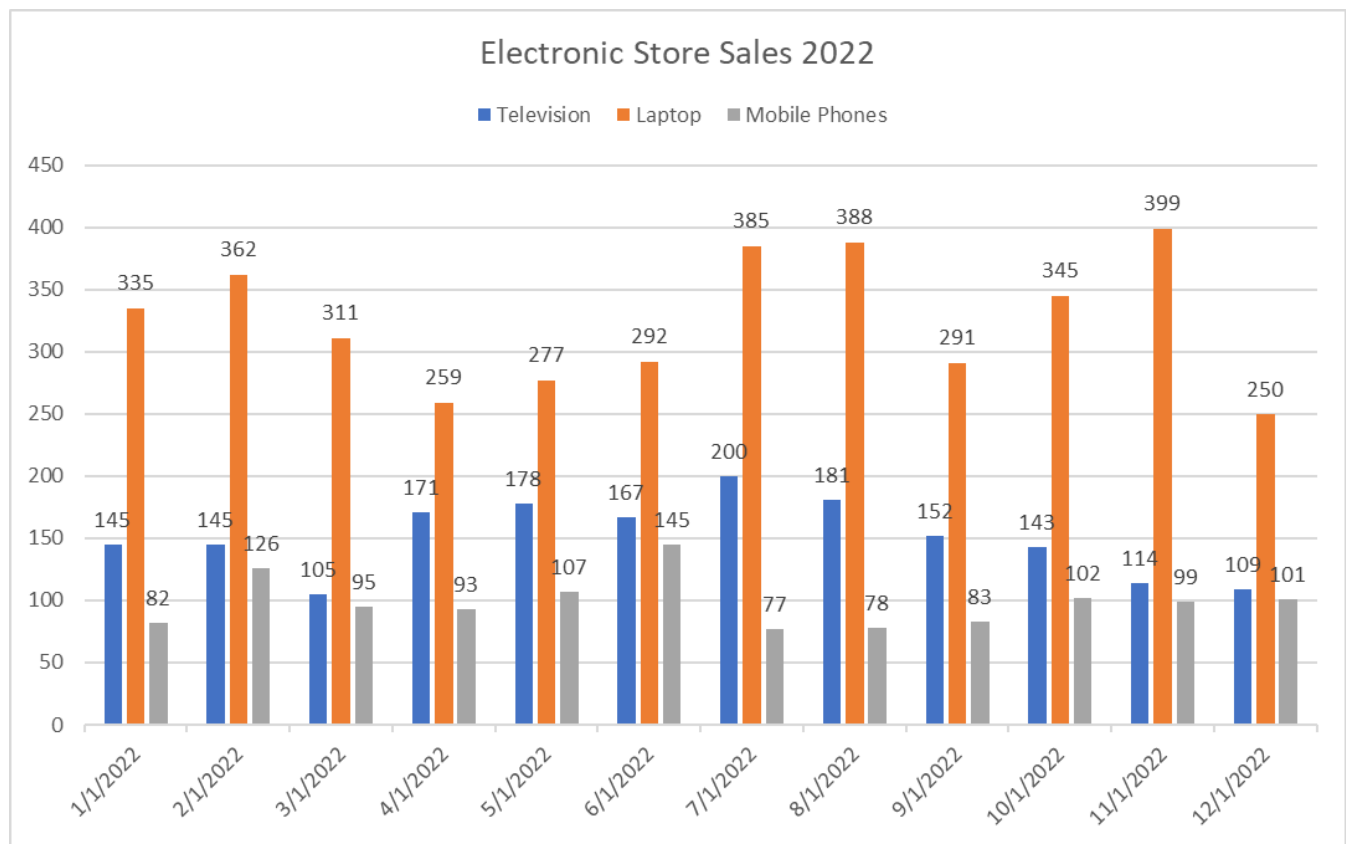
Chart Styles and colors

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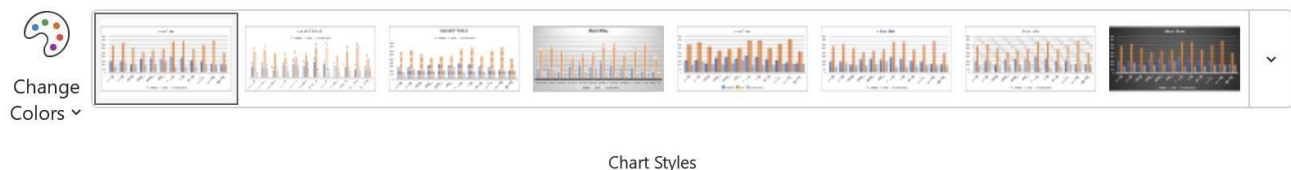
By implementing small visual changes, we can see a big difference in the aesthetic of the whole chart. Here's an example of how our chart has changed by choosing Style 6 and Monochromatic Palette 8.

Electronic Store Sales 2022

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.

Format Axis

Axis Options ∨   Text Options

Axis Options

Bounds

Minimum	0.0	Auto
Maximum	450.0	Auto

Units

Major	50.0	Auto
Minor	10.0	Auto

Horizontal axis crosses

- ⦿ Automatic
- ○ Axis value   0.0
- ○ Maximum axis value

Display units   None ∨

☐ Show display units label on chart

☐ Logarithmic scale   Base   10

☐ Values in reverse order

Tick Marks

Labels

Number

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
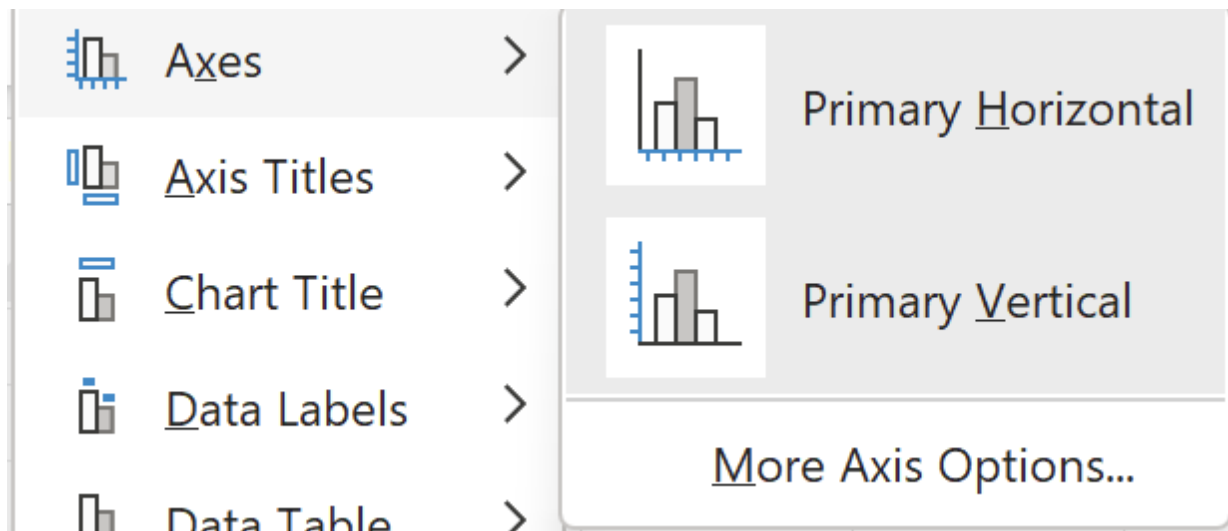
Other Excel formatting options

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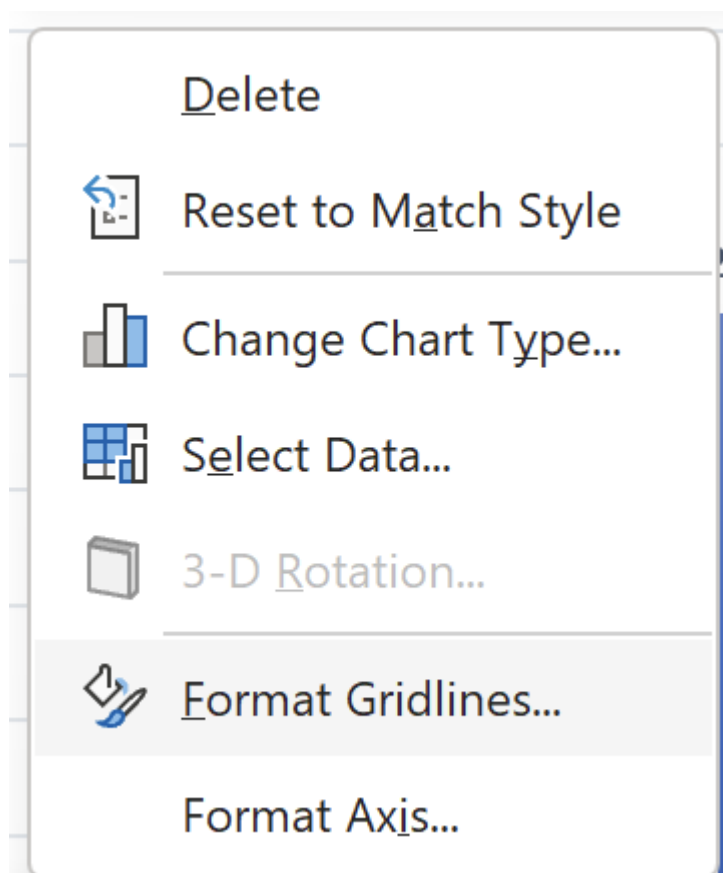
Conclusion:- Thus, this way Data Analysis and Visualization is done using Advanced Excel.

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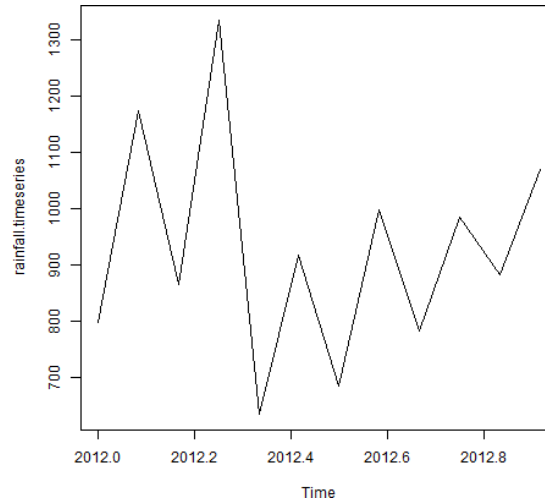
Conclusion:- Thus, this way Performed data classification algorithm using any Classification algorithm

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:
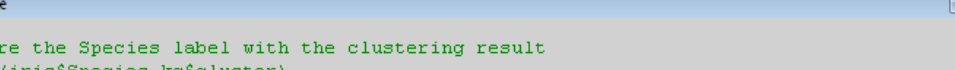
Clustering your data can provide a new way to slice that is based on the properties of the data instead of other labels. For instance, customer data is often sliced by demographic parameters like gender, age, location, etc. This data can be useful in many cases, but what if you could slice your customers 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.

```
R Console

> # Apply K mean to iris and store result
> newiris <- iris
> newiris$Species <- NULL
> (kc <- kmeans(newiris,3))
K-means clustering with 3 clusters of sizes 62, 38, 50

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.901613    2.748387     4.393548    1.433871
2     6.850000    3.073684     5.742105    2.071053
3     5.006000    3.428000     1.462000    0.246000

Clustering vector:
  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [47] 3 3 3 3 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
 [93] 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2 2 1 1 2 2 2 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2
[139] 1 2 2 2 1 2 2 2 1 2 2 1

Within cluster sum of squares by cluster:
[1] 39.82097 23.87947 15.15100
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
>
```

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 R Console                                                    [ - ] [ □ ] [ X ]
> # 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