|  |  |
| --- | --- |
| Name : Jiyan Mehta | Roll No : A034 |
| SAP ID : 7562100134 | SUBJECT : Machine learning |
| EXPT : ALL EXPTS TILL NOW | Date : 24/02/24 |

EXPT 1: BASICS(NUMPY and PANDAS)

Code and output for numpy:

import numpy

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)

 [1 2 3 4 5]

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

[1 2 3 4 5]

import numpy as np

print(np.\_\_version\_\_)

1.21.6

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))

[1 2 3 4 5]

<class 'numpy.ndarray'>

import numpy as np

arr = np.array((1, 2, 3, 4, 5))

print(arr)

[1 2 3 4 5]

import numpy as np

arr = np.array(42)

print(arr)

42

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

[1 2 3 4 5]

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)

print(arr.ndim)

print(arr.shape)

[[1 2 3]

[4 5 6]]

2

(2, 3)

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:5,3])

[4 9]

import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)

print(arr.ndim)

print(arr.shape)

[[[1 2 3]

[4 5 6]]

[[1 2 3]

[4 5 6]]]

3

(2, 2, 3)

import numpy as np

arr = np.array([[[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]], [[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]])

print(arr)

print(arr.ndim)

print(arr.shape)

[[[[1 2 3]

[4 5 6]]

[[1 2 3]

[4 5 6]]]

[[[1 2 3]

[4 5 6]]

[[1 2 3]

[4 5 6]]]]

4

(2, 2, 2, 3)

import numpy as np

arr = np.array([[[[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]], [[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]],[[[[1,

print(arr)

print(arr.ndim)

print(arr.shape)

[[[[[1 2 3]

[4 5 6]]

[[1 2 3]

[4 5 6]]]

[[[1 2 3]

[4 5 6]]

[[1 2 3]

[4 5 6]]]]

[[[[1 2 3]

[4 5 6]]

[[1 2 3]

[4 5 6]]]

[[[1 2 3]

[4 5 6]]

[[1 2 3]

[4 5 6]]]]]

5

(2, 2, 2, 2, 3)

Code outpur for pandas:

import pandas

mydataset = {

'cars': ["BMW", "Volvo", "Ford"],

'passings': [3, 7, 2]

}

myvar = pandas.DataFrame(mydataset)

print(myvar)

cars passings

0 BMW 3

1 Volvo 7

2 Ford 2

import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)

print(myvar[2])

0 1

1 7

2 2

dtype: int64

2

import pandas as pd

a =[1,2,3]

myvar=pd.Series(a,index=["x","y","z"])

print(myvar)

print(myvar["y"])

x 1

y 2

z 3

dtype: int64

2

import pandas as pd

import numpy as np

# Creating empty series

ser = pd.Series()

print(ser)

# simple array

data = np.array(['p', 'y', 't', 'h', 'o', 'n'])

ser = pd.Series(data)

print(ser)

Series([], dtype: float64)

0 p

1 y

2 t

3 h

4 o

5 n

dtype: object

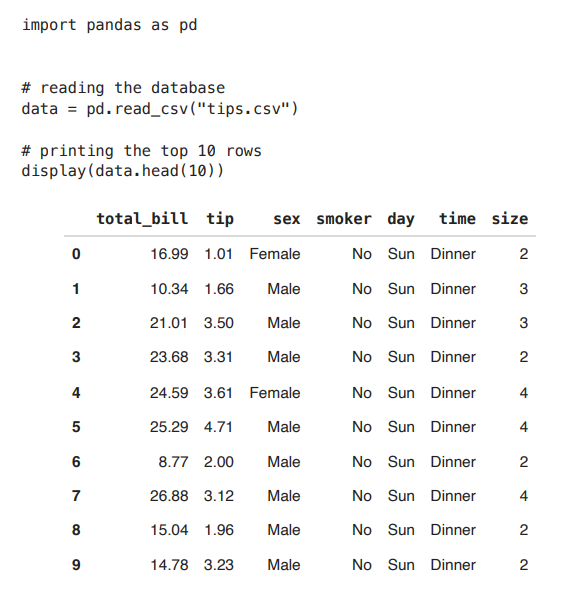
<ipython-input-4-cc45929121df>:6: FutureWarning: The default dtype for empty Series will be 'object' instead of 'float64

ser = pd.Series()

EXPT 2: DATA VISUALIZATION

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc. Data visualization provides a good, organized pictorial representation of the data which makes it easier to understand, observe, analyze. Python provides various libraries that come with different features for visualizing data. All these libraries come with different features and can support various types of graphs. we will be discussing four such libraries. Matplotlib Seaborn Bokeh Plotly.

CODE AND OUTPUT:



A screenshot of a computer program

Description automatically generated

A graph with blue dots

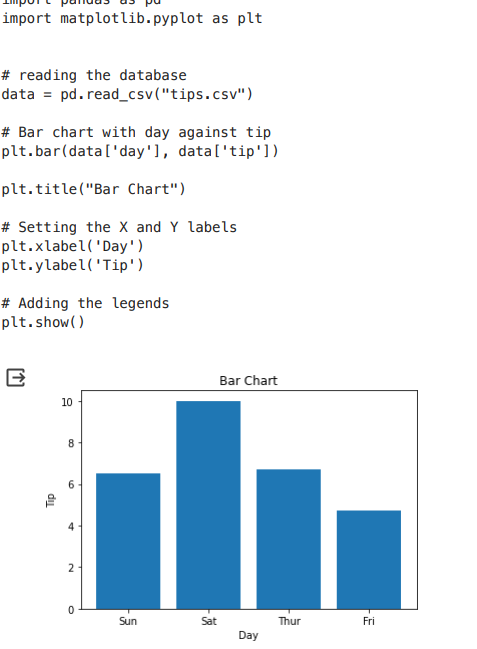
Description automatically generated

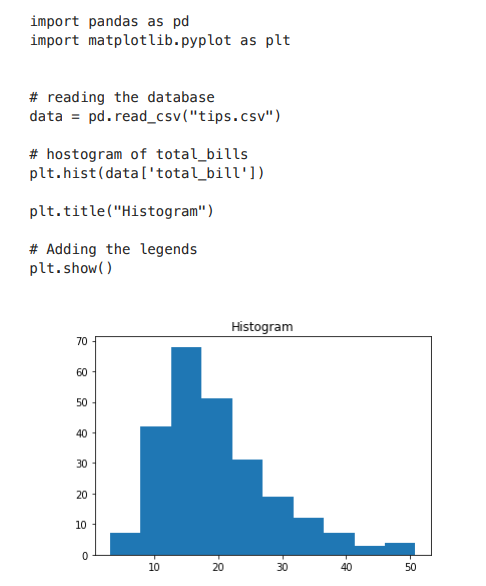
A screen shot of a computer screen

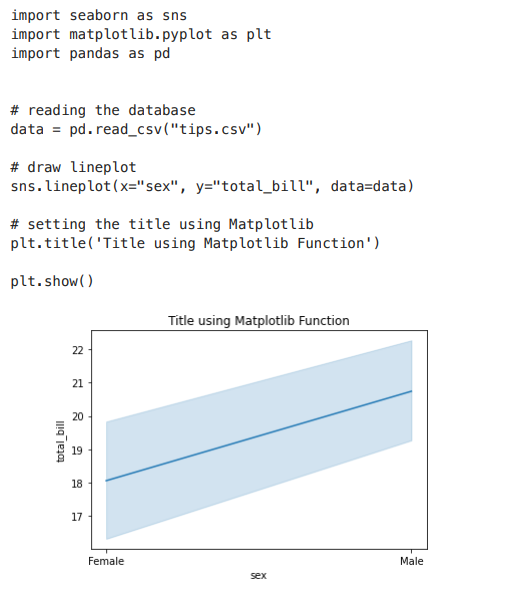
Description automatically generated

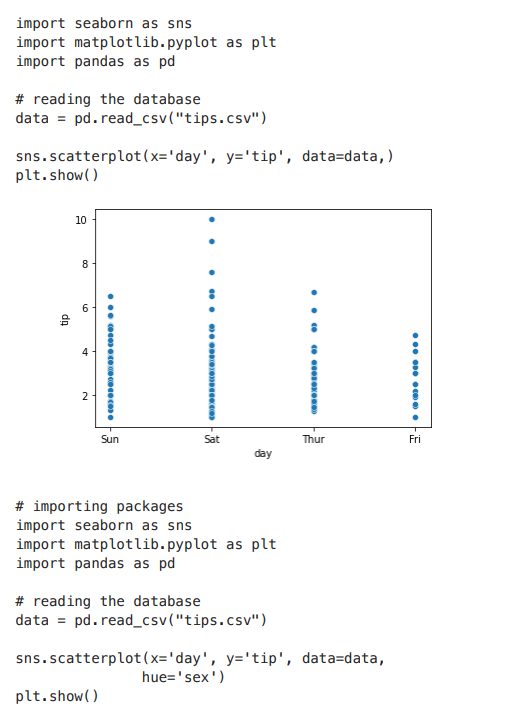
A screenshot of a computer screen

Description automatically generated









A screenshot of a computer screen

Description automatically generated

A screen shot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

Bokeh is mainly famous for its interactive charts visualization. Bokeh renders its plots using HTML and JavaScript that uses modern web

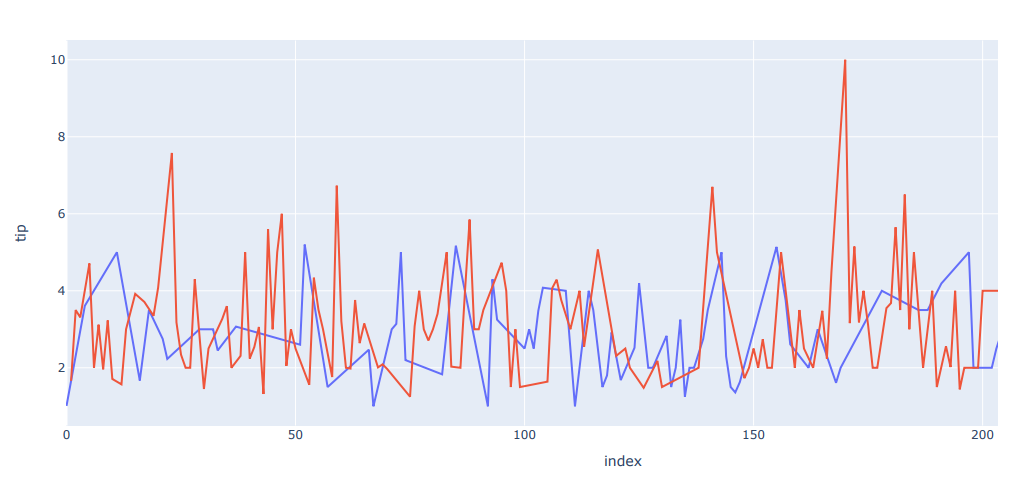
browsers for presenting elegant, concise construction of novel graphics with high-level interactivity.

Plotly has hover tool capabilities that allow us to detect any outliers or anomalies in numerous data points. It allows more customization. It makes the graph visually more attractive.



A screenshot of a computer code

Description automatically generated



A screenshot of a graph

Description automatically generated

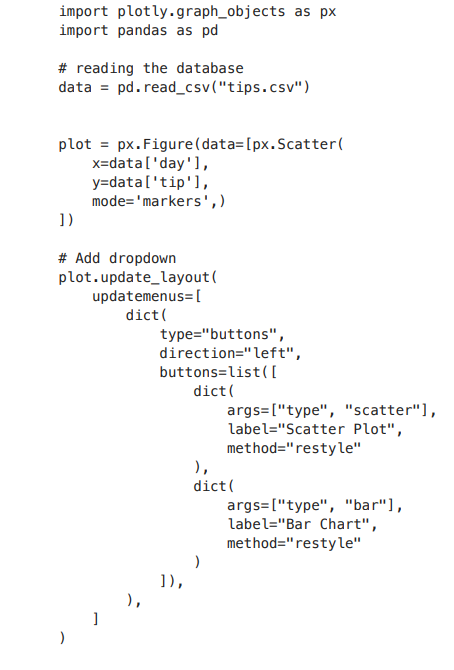
A graph with red and blue squares

Description automatically generated

Creating Dropdown Menu: A drop-down menu is a part of the menu-button which is displayed on a screen all the time. Every menu button is

associated with a Menu widget that can display the choices for that menu button when clicked on it. In plotly, there are 4 possible methods to modify the charts by using updatemenu method.

restyle: modify data or data attributes relayout: modify layout attributes update: modify data and layout attributes animate: start or pause an animation.



EXPT 3 : SIMPLE LINEAR REGRESSION

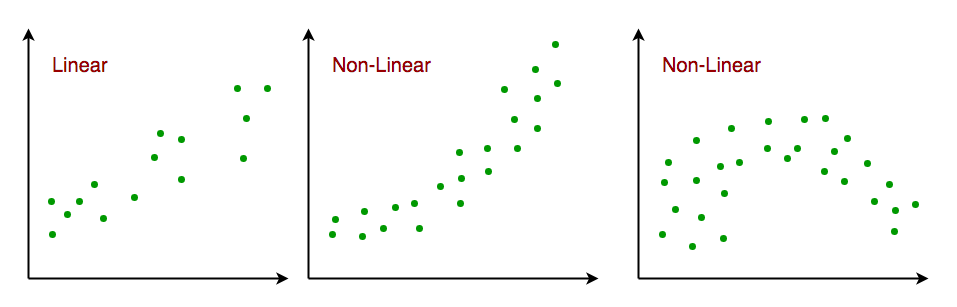
**What is Linear Regression?**

[Linear regression](https://www.geeksforgeeks.org/ml-linear-regression/) is a statistical method that is used to predict a continuous dependent variable(target variable) based on one or more independent variables(predictor variables). This technique assumes a linear relationship between the dependent and independent variables, which implies that the dependent variable changes proportionally with changes in the independent variables. In other words, linear regression is used to determine the extent to which one or more variables can predict the value of the dependent variable.

**Assumptions We Make in a Linear Regression Model:**

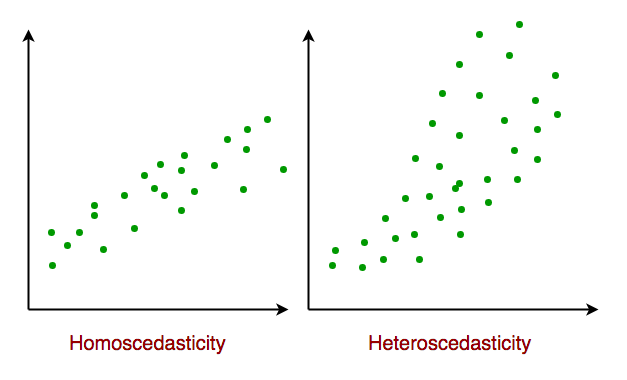
Given below are the basic assumptions that a linear regression model makes regarding a dataset on which it is applied:

* **Linear relationship**: The relationship between response and feature variables should be linear. The linearity assumption can be tested using scatter plots. As shown below, 1st figure represents linearly related variables whereas variables in the 2nd and 3rd figures are most likely non-linear. So, 1st figure will give better predictions using linear regression.



*Linear relationship i the feature space*

* **Little or no multi-collinearity**: It is assumed that there is little or no multicollinearity in the data. Multicollinearity occurs when the features (or independent variables) are not independent of each other.
* **Little or no autocorrelation**: Another assumption is that there is little or no autocorrelation in the data. Autocorrelation occurs when the residual errors are not independent of each other. You can refer here for more insight into this topic.
* **No outliers:** We assume that there are no outliers in the data. Outliers are data points that are far away from the rest of the data. Outliers can affect the results of the analysis.
* **Homoscedasticity**: Homoscedasticity describes a situation in which the error term (that is, the “noise” or random disturbance in the relationship between the independent variables and the dependent variable) is the same across all values of the independent variables. As shown below, figure 1 has homoscedasticity while Figure 2 has heteroscedasticity.



*Homoscedasticity in Linear Regression*

As we reach the end of this article, we discuss some applications of linear regression below.

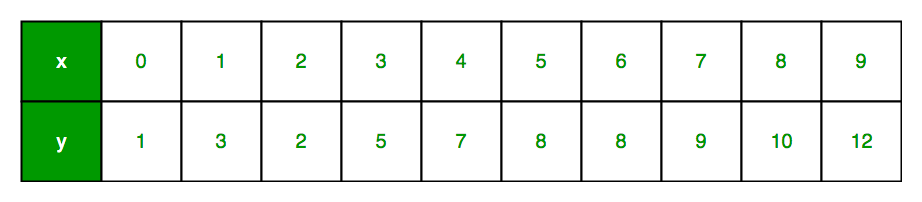
**Types of Linear Regression**

There are two main types of linear regression:

* **Simple linear regression:** This involves predicting a dependent variable based on a single independent variable.
* **Multiple linear regression:** This involves predicting a dependent variable based on multiple independent variables.

**Simple Linear Regression**

Simple [linear regression](https://www.geeksforgeeks.org/ml-linear-regression/) is an approach for predicting a **response** using a **single feature**. It is one of the most basic [machine learning](https://www.geeksforgeeks.org/machine-learning/) models that a machine learning enthusiast gets to know about. In linear regression, we assume that the two variables i.e. dependent and independent variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x). Let us consider a dataset where we have a value of response y for every feature x:

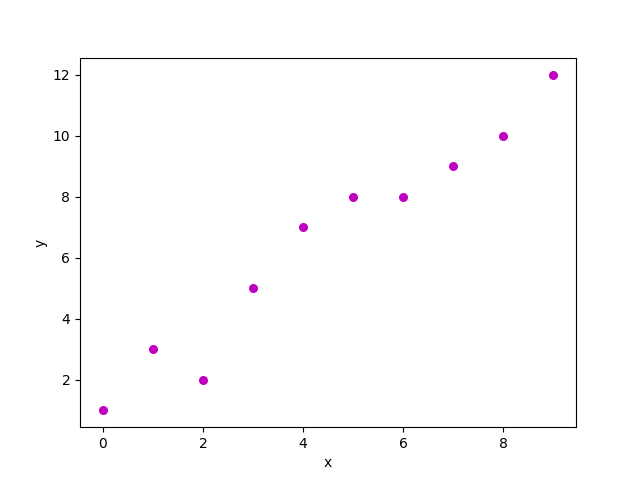


For generality, we define:

*x as****feature vector****, i.e x = [x\_1, x\_2, …., x\_n],*

*y as****response vector****, i.e y = [y\_1, y\_2, …., y\_n]*

for **n** observations (in the above example, n=10). A scatter plot of the above dataset looks like this:-



*Scatter plot for the randomly generated data*

Now, the task is to find a **line that fits best** in the above scatter plot so that we can predict the response for any new feature values. (i.e a value of x not present in a dataset) This line is called a [**regression line**](https://www.geeksforgeeks.org/scatter-plot-with-regression-line-using-altair-in-python/). The equation of the regression line is represented as:

Here,

* h(x\_i) represents the **predicted response value** for ith observation.
* b\_0 and b\_1 are regression coefficients and represent the **y-intercept** and **slope** of the regression line respectively.

To create our model, we must “learn” or estimate the values of regression coefficients b\_0 and b\_1. And once we’ve estimated these coefficients, we can use the model to predict responses!  
In this article, we are going to use the principle of [**Least Squares**](https://www.geeksforgeeks.org/ordinary-least-squares-ols-using-statsmodels/).

 Now consider:

Here, e\_i is a **residual error** in ith observation. So, our aim is to minimize the total residual error. We define the squared error or cost function, J as:

And our task is to find the value of b0 and b1 for which J(b0, b1) is minimum! Without going into the mathematical details, we present the result here:

Where SSxy is the sum of cross-deviations of y and x:

And SSxx is the sum of squared deviations of x:

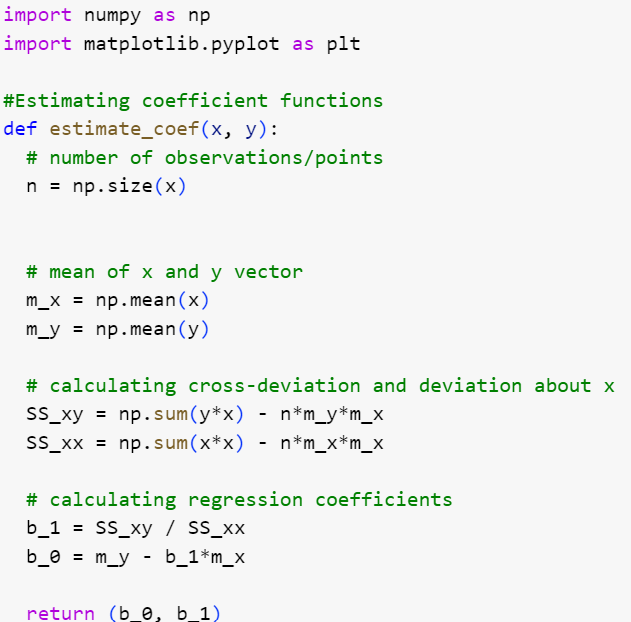
**Python Implementation of Simple Linear Regression**

We can use the [Python](https://www.geeksforgeeks.org/python-programming-language/) language to learn the coefficient of linear regression models. For plotting the input data and best-fitted line we will use the [matplotlib](https://www.geeksforgeeks.org/python-introduction-matplotlib/) library. It is one of the most used Python libraries for plotting graphs. Here is the example of simpe Linear regression using Python.

**Estimating Coefficients Function**

This function, estimate\_coef(), takes the input data x (independent variable) and y (dependent variable) and estimates the coefficients of the linear regression line using the least squares method.

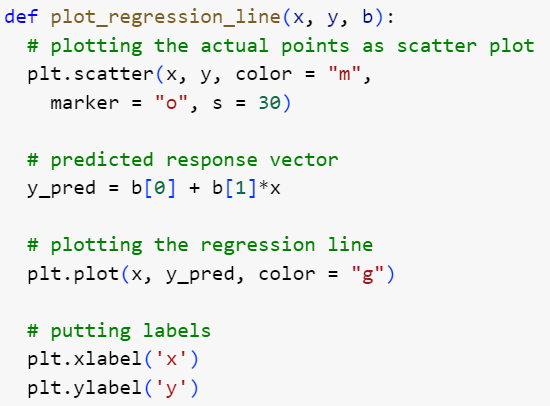
* **Calculating Number of Observations:** n = np.size(x) determines the number of data points.
* **Calculating Means:** m\_x = np.mean(x) and m\_y = np.mean(y) compute the mean values of x and y, respectively.
* **Calculating Cross-Deviation and Deviation about x:** SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x and SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x calculate the sum of squared deviations between x and y and the sum of squared deviations of x about its mean, respectively.
* **Calculating Regression Coefficients:** b\_1 = SS\_xy / SS\_xx and b\_0 = m\_y - b\_1\*m\_x determine the slope (b\_1) and intercept (b\_0) of the regression line using the least squares method.
* **Returning Coefficients:** The function returns the estimated coefficients as a tuple (b\_0, b\_1).



**Plotting Regression Line Function**

This function, plot\_regression\_line(), takes the input data x (independent variable), y (dependent variable), and the estimated coefficients b to plot the regression line and the data points.

1. **Plotting Scatter Plot:** plt.scatter(x, y, color = "m", marker = "o", s = 30) plots the original data points as a scatter plot with red markers.
2. **Calculating Predicted Response Vector:** y\_pred = b[0] + b[1]\*x calculates the predicted values for y based on the estimated coefficients b.
3. **Plotting Regression Line:** plt.plot(x, y\_pred, color = "g") plots the regression line using the predicted values and the independent variable x.
4. **Adding Labels:** plt.xlabel('x') and plt.ylabel('y') label the x-axis as 'x' and the y-axis as 'y', respectively.



Output:

A green line with purple dots

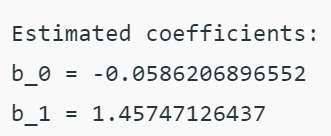
Description automatically generated

**Main Function**

The provided code implements simple linear regression analysis by defining a function main() that performs the following steps:

1. **Data Definition:** Defines the independent variable (x) and dependent variable (y) as NumPy arrays.
2. **Coefficient Estimation:** Calls the estimate\_coef() function to determine the coefficients of the linear regression line using the provided data.
3. **Printing Coefficients:** Prints the estimated intercept (b\_0) and slope (b\_1) of the regression line.
4. **def** main():
5. # observations / data
6. x **=** np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
7. y **=** np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
9. # estimating coefficients
10. b **=** estimate\_coef(x, y)
11. **print**("Estimated coefficients:\nb\_0 **=** {} \
12. \nb\_1 **=** {}".format(b

Output:



EXPT 4: MULTIPLE LINEAR REGRESSION

Multiple linear regression attempts to model the relationship between **two or more features** and a response by fitting a linear equation to the observed data.  
Clearly, it is nothing but an extension of simple linear regression. Consider a dataset with **p** features(or independent variables) and one response(or dependent variable).   
Also, the dataset contains **n** rows/observations.

We define:

X (**feature matrix**) = a matrix of size **n X p** where xij denotes the values of the jth feature for ith observation.

 So,

and

y (**response vector**) = a vector of size **n** where y\_{i} denotes the value of response for ith observation.

The **regression line** for **p** features is represented as:

where h(x\_i) is **predicted response value** for ith observation and b\_0, b\_1, …, b\_p are the **regression coefficients**. Also, we can write:

where e\_i represents a **residual error** in ith observation. We can generalize our linear model a little bit more by representing feature matrix **X** as:

So now, the linear model can be expressed in terms of matrices as:

where,

and

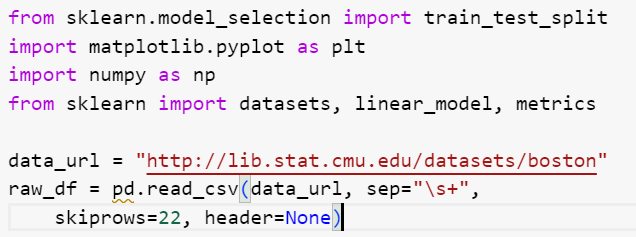
Now, we determine an **estimate of b**, i.e. b’ using the **Least Squares method**. As already explained, the Least Squares method tends to determine b’ for which total residual error is minimized.  
We present the result directly here:

where ‘ represents the transpose of the matrix while -1 represents the [matrix inverse](https://www.geeksforgeeks.org/adjoint-inverse-matrix/). Knowing the least square estimates, b’, the multiple linear regression model can now be estimated as:

where y’ is the **estimated response vector**.

**Python Implementation of Multiple Linear Regression**

For [multiple linear regression](https://www.geeksforgeeks.org/ml-multiple-linear-regression-using-python/) using Python, we will use the Boston house pricing dataset.



**Preprocessing Data**

This extracts the input variables (X) and target variable (y) from the DataFrame. The input variables are selected from every other row to match the target variable, which is available every other row.

A computer code with text

Description automatically generated with medium confidence

**Splitting Data into Training and Testing Sets**

Here it divides the data into training and testing sets using the train\_test\_split() function from scikit-learn. The test\_size parameter specifies that 40% of the data should be used for testing.

A math equation with numbers and symbols

Description automatically generated with medium confidence

**Creating and Training the Linear Regression Model**

This initializes a LinearRegression object (reg) and trains the model using the training data (X\_train, y\_train)

A close up of a text

Description automatically generated

**Evaluating Model Performance**

Evaluates the model’s performance by printing the regression coefficients and calculating the variance score, which measures the proportion of explained variance. A score of 1 indicates perfect prediction.

A screen shot of a computer code

Description automatically generated

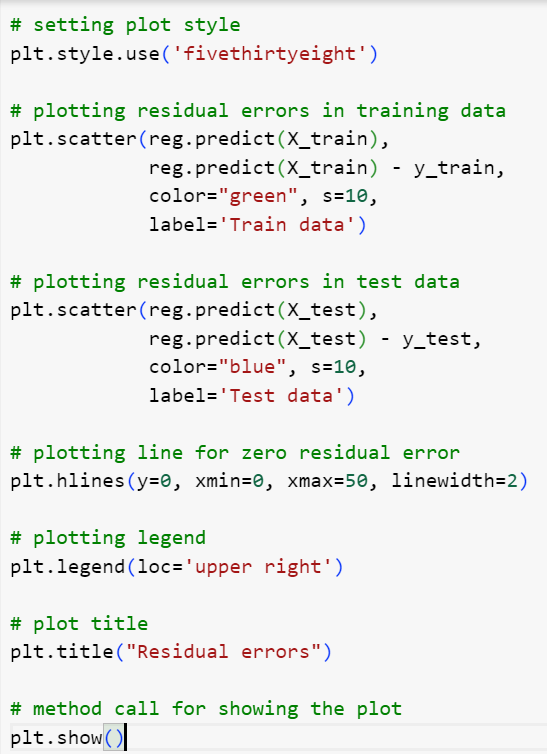
Output:

A white background with numbers and symbols

Description automatically generated

**Plotting Residual Errors**

Plotting and analyzing the residual errors, which represent the difference between the predicted values and the actual values.



Output:

A graph with blue and green dots

Description automatically generated

In the above example, we determine the accuracy score using **Explained Variance Score**. We define:

explained\_variance\_score = 1 – Var{y – y’}/Var{y}

where y’ is the estimated target output, y is the corresponding (correct) target output, and Var is Variance, the square of the standard deviation. The best possible score is 1.0, lower values are worse.

EXPT 5: POLYNOMIAL LINEAR REGRESSION

**Polynomial Linear Regression**

**Polynomial Regression**is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modeled as an *nth-degree* polynomial. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y | x).

**Choosing a Degree for Polynomial Regression**

The choice of degree for polynomial regression is a trade-off between bias and variance. Bias is the tendency of a model to consistently predict the same value, regardless of the true value of the dependent variable. Variance is the tendency of a model to make different predictions for the same data point, depending on the specific training data used.

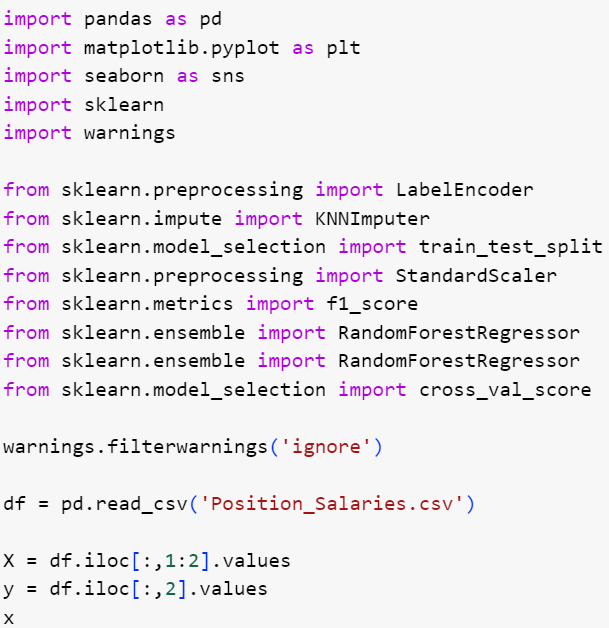
A higher-degree polynomial can reduce bias but can also increase variance, leading to overfitting. Conversely, a lower-degree polynomial can reduce variance but can also increase bias.

There are a number of methods for choosing a degree for polynomial regression, such as cross-validation and using information criteria such as Akaike information criterion (AIC) or Bayesian information criterion (BIC).

**Implementation of Polynomial Regression using Python**

Implementing the Polynomial regression using Python:

Here we will import all the necessary libraries for data analysis and machine learning tasks and then loads the ‘Position\_Salaries.csv’ dataset using Pandas. It then prepares the data for modeling by handling missing values and encoding categorical data. Finally, it splits the data into training and testing sets and standardizes the numerical features using StandardScaler.



Output:

A screenshot of a computer code

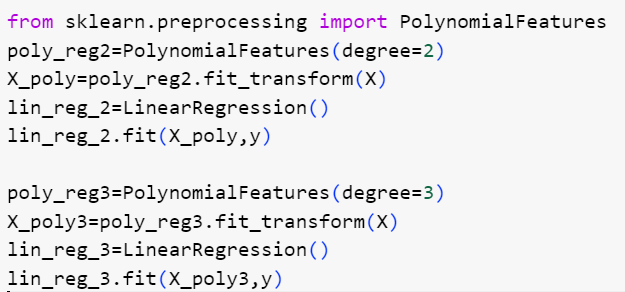
Description automatically generated

The code creates a linear regression model and fits it to the provided data, establishing a linear relationship between the independent and dependent variables.

A close up of words

Description automatically generated

The code performs quadratic and cubic regression by generating polynomial features from the original data and fitting linear regression models to these features. This enables modeling nonlinear relationships between the independent and dependent variables.



The code creates a scatter plot of the data point, It effectively visualizes the linear relationship between position level and salary.

A close-up of a computer code

Description automatically generated

Output:

A graph with a green line and red dots

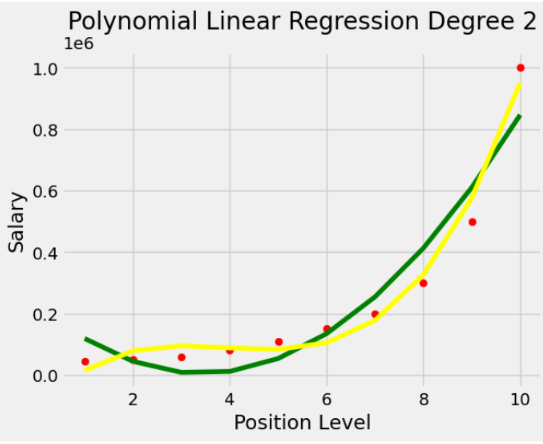
Description automatically generated

The code creates a scatter plot of the data points, overlays the predicted quadratic and cubic regression lines. It effectively visualizes the nonlinear relationship between position level and salary and compares the fits of quadratic and cubic regression models.

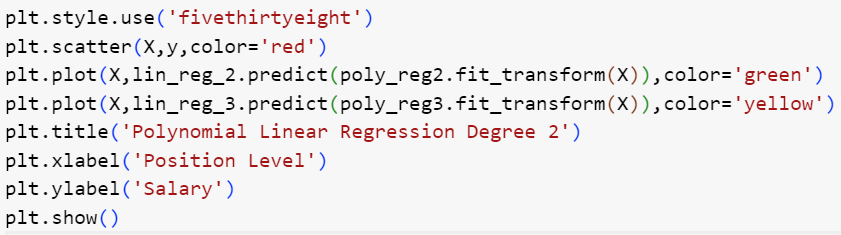
A computer code with red and blue text

Description automatically generated

Output:



The code effectively visualizes the relationship between position level and salary using cubic regression and generates a continuous prediction line for a broader range of position levels.



Output:

A graph with a green line

Description automatically generated

**Applications of Linear Regression**

* **Trend lines:** A trend line represents the variation in quantitative data with the passage of time (like GDP, oil prices, etc.). These trends usually follow a linear relationship. Hence, linear regression can be applied to predict future values. However, this method suffers from a lack of scientific validity in cases where other potential changes can affect the data.
* **Economics:** Linear regression is the predominant empirical tool in economics. For example, it is used to predict consumer spending, fixed investment spending, inventory investment, purchases of a country’s exports, spending on imports, the demand to hold liquid assets, labor demand, and labor supply.
* **Finance:** The capital price asset model uses linear regression to analyze and quantify the systematic risks of an investment.
* **Biology:** Linear regression is used to model causal relationships between parameters in biological systems.

**Advantages of Linear Regression**

* **Easy to interpret:** The coefficients of a linear regression model represent the change in the dependent variable for a one-unit change in the independent variable, making it simple to comprehend the relationship between the variables.
* **Robust to outliers:** Linear regression is relatively robust to outliers meaning it is less affected by extreme values of the independent variable compared to other statistical methods.
* **Can handle both linear and nonlinear relationships:** Linear regression can be used to model both linear and nonlinear relationships between variables. This is because the independent variable can be transformed before it is used in the model.
* **No need for feature scaling or transformation:** Unlike some machine learning algorithms, linear regression does not require feature scaling or transformation. This can be a significant advantage, especially when dealing with large datasets.

**Disadvantages of Linear Regression**

* **Assumes linearity:** Linear regression assumes that the relationship between the independent variable and the dependent variable is linear. This assumption may not be valid for all data sets. In cases where the relationship is nonlinear, linear regression may not be a good choice.
* **Sensitive to multicollinearity:** Linear regression is sensitive to multicollinearity. This occurs when there is a high correlation between the independent variables. Multicollinearity can make it difficult to interpret the coefficients of the model and can lead to overfitting.
* **May not be suitable for highly complex relationships:** Linear regression may not be suitable for modeling highly complex relationships between variables. For example, it may not be able to model relationships that include interactions between the independent variables.
* **Not suitable for classification tasks:** Linear regression is a regression algorithm and is not suitable for classification tasks, which involve predicting a categorical variable rather than a continuous variable.

EXP 6: GRADIENT DESCENT

**Gradient Descent in Machine Learning**

**What is Gradient?**

A gradient is nothing but a derivative that defines the effects on outputs of the function with a little bit of variation in inputs.

**What is Gradient Descent?**

Gradient Descent stands as a cornerstone orchestrating the intricate dance of model optimization. At its core, it is a numerical optimization algorithm that aims to find the optimal parameters—weights and biases—of a neural network by minimizing a defined cost function.

Gradient Descent (GD) is a widely used optimization algorithm in machine learning and deep learning that minimises the cost function of a neural network model during training. It works by iteratively adjusting the weights or parameters of the model in the direction of the negative gradient of the cost function until the minimum of the cost function is reached.

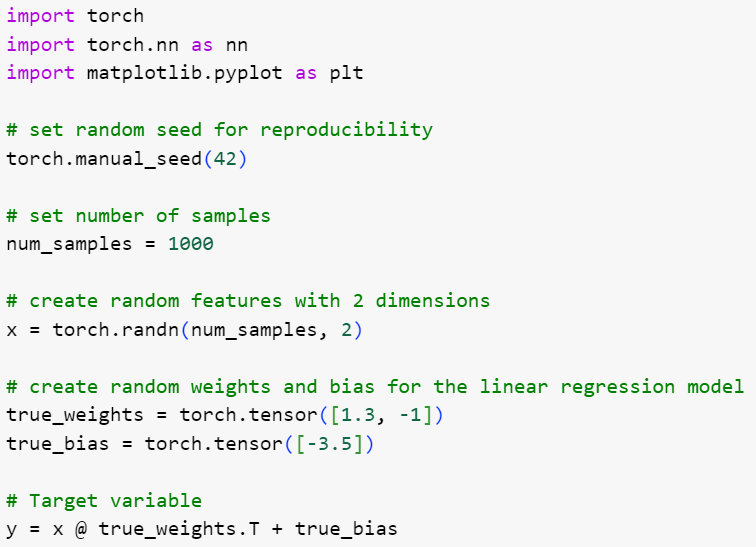
The learning happens during the [**backpropagation**](https://www.geeksforgeeks.org/backpropagation-in-data-mining/) while training the neural network-based model. There is a term known as [**Gradient Descent**](https://www.geeksforgeeks.org/gradient-descent-in-linear-regression/), whichis used to optimize the weight and biases based on the cost function. The cost function evaluates the difference between the actual and predicted outputs.

Gradient Descent is a fundamental optimization algorithm in [machine learning](https://www.geeksforgeeks.org/machine-learning/) used to minimize the cost or loss function during model training.

* It iteratively adjusts model parameters by moving in the direction of the steepest decrease in the cost function.
* The algorithm calculates gradients, representing the partial derivatives of the cost function concerning each parameter.

These gradients guide the updates, ensuring convergence towards the optimal parameter values that yield the lowest possible cost.

Gradient Descent is versatile and applicable to various machine learning models, including linear regression and neural networks. Its efficiency lies in navigating the parameter space efficiently, enabling models to learn patterns and make accurate predictions. Adjusting the **learning rate**is crucial to balance convergence speed and avoiding overshooting the optimal solution



A screenshot of a computer program

Description automatically generated

Output:

A pair of blue dots

Description automatically generated

**Let’s first try with a linear model:**

A black text on a white background

Description automatically generated

Prediction:

A black text with green and blue text

Description automatically generated

Output:

A black squares with white text

Description automatically generated with medium confidence

**Define the loss function**

Here we are calculating the Mean Squared Error by taking the square of the difference between the actual and the predicted value and then dividing it by its length (i.e n = the Total number of output or target values) which is the mean of squared errors.

A computer screen shot of text

Description automatically generated



As we can see from the above right now the Mean Squared Error is 30559.4473. All the steps which are done till now are known as forward propagation.

 Now our task is to find the optimal value of weight w and bias b which can fit our model well by giving very less or minimum error as possible.

A close up of a sign

Description automatically generated

Now to update the weight and bias value and find the optimal value of weight and bias we will do backpropagation. Here the Gradient Descent comes into the role to find the optimal value weight and bias.

**How the Gradient Descent Algorithm Works**

For the sake of complexity, we can write our loss function for the single row as below

A math equations with numbers and symbols

Description automatically generated with medium confidence

In the above function x and y are our input data i.e constant. To find the optimal value of weight w and bias b. we partially differentiate with respect to w and b. This is also said that we will find the gradient of loss function J(w,b) with respect to w and b to find the optimal value of w and b.

**Gradient of J(w,b) with respect to w**

A screenshot of a math problem

Description automatically generated

A screenshot of a math problem

Description automatically generated

Here we have considered the linear regression. So that here the parameters are weight and bias only. But in a fully connected neural network model there can be multiple layers and multiple parameters.  but the concept will be the same everywhere. And the below-mentioned formula will work everywhere.

A black text on a white background

Description automatically generated

Here,

* = Learning rate
* J = Loss function
* = Gradient symbol denotes the derivative of loss function J
* Param = weight and bias     There can be multiple weight and bias values depending upon the complexity of the model and features in the dataset

In our case:

A group of symbols on a white background

Description automatically generated

In the current problem, two input features, So, the weight will be two.

**Implementations of the Gradient Descent algorithm for the above model**

Steps:

1. Find the gradient using loss.backward()
2. Get the parameter using model.linear.weight and model.linear.bias
3. Update the parameter using the above-defined equation.
4. Again assign the model parameter to our model

# Find the gradient using

loss.backward()

# Learning Rate

learning\_rate = 0.001

# Model Parameter

w = model.linear.weight

b = model.linear.bias

# Matually Update the model parameter

w = w - learning\_rate \* w.grad

b = b - learning\_rate \* b.grad

# assign the weight & bias parameter to the linear layer

model.linear.weight = nn.Parameter(w)

model.linear.bias = nn.Parameter(b)

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Output:

A screenshot of a graph

Description automatically generated

From the above graph and data, we can observe the Losses are decreasing as per the weight and bias variations.

Now we have found the optimal weight and bias values. Print the optimal weight and bias and

A white background with black text

Description automatically generated



Prediction:

A close-up of a computer

Description automatically generated

Output:

A black text on a white background

Description automatically generated

**Gradient Descent Learning Rate**

The [learning rate](https://www.geeksforgeeks.org/impact-of-learning-rate-on-a-model/) is a critical hyperparameter in the context of gradient descent, influencing the size of steps taken during the optimization process to update the model parameters. Choosing an appropriate learning rate is crucial for efficient and effective model training.

When the learning rate is **too small**, the optimization process progresses very slowly. The model makes tiny updates to its parameters in each iteration, leading to sluggish convergence and potentially getting stuck in local minima.

On the other hand, an **excessively large learning rate** can cause the optimization algorithm to overshoot the optimal parameter values, leading to divergence or oscillations that hinder convergence.

Achieving the right balance is essential. A small learning rate might result in vanishing gradients and slow convergence, while a large learning rate may lead to overshooting and instability.

**Advantages & Disadvantages of gradient descent**

**Advantages of Gradient Descent**

1. **Widely used:** Gradient descent and its variants are widely used in machine learning and optimization problems because they are effective and easy to implement.
2. **Convergence**: Gradient descent and its variants can converge to a global minimum or a good local minimum of the cost function, depending on the problem and the variant used.
3. **Scalability**: Many variants of gradient descent can be parallelized and are scalable to large datasets and high-dimensional models.
4. **Flexibility**: Different variants of gradient descent offer a range of trade-offs between accuracy and speed, and can be adjusted to optimize the performance of a specific problem.

**Disadvantages of gradient descent:**

1. **Choice of learning rate:** The choice of learning rate is crucial for the convergence of gradient descent and its variants. Choosing a learning rate that is too large can lead to oscillations or overshooting while choosing a learning rate that is too small can lead to slow convergence or getting stuck in local minima.
2. **Sensitivity to initialization:**Gradient descent and its variants can be sensitive to the initialization of the model’s parameters, which can affect the convergence and the quality of the solution.
3. **Time-consuming:**Gradient descent and its variants can be time-consuming, especially when dealing with large datasets and high-dimensional models. The convergence speed can also vary depending on the variant used and the specific problem.
4. **Local optima:**Gradient descent and its variants can converge to a local minimum instead of the global minimum of the cost function, especially in non-convex problems. This can affect the quality of the solution, and techniques like random initialization and multiple restarts may be used to mitigate this issue.

EXPT 7: LOGISTIC REGRESSION

A basic machine learning approach that is frequently used for binary classification tasks is called logistic regression. Though its name suggests otherwise, it uses the sigmoid function to simulate the likelihood of an instance falling into a specific class, producing values between 0 and 1. Logistic regression, with its emphasis on interpretability, simplicity, and efficient computation, is widely applied in a variety of fields, such as marketing, finance, and healthcare, and it offers insightful forecasts and useful information for decision-making.

**Logistic Regression**

A statistical model for binary classification is called [logistic regression](https://www.geeksforgeeks.org/ml-linear-regression-vs-logistic-regression/). Using the sigmoid function, it forecasts the likelihood that an instance will belong to a particular class, guaranteeing results between 0 and 1. To minimize the log loss, the model computes a linear combination of input characteristics, transforms it using the sigmoid, and then optimizes its coefficients using methods like gradient descent. These coefficients establish the decision boundary that divides the classes. Because of its ease of use, interpretability, and versatility across multiple domains, Logistic Regression is widely used in machine learning for problems that involve binary outcomes. Overfitting can be avoided by implementing regularization.

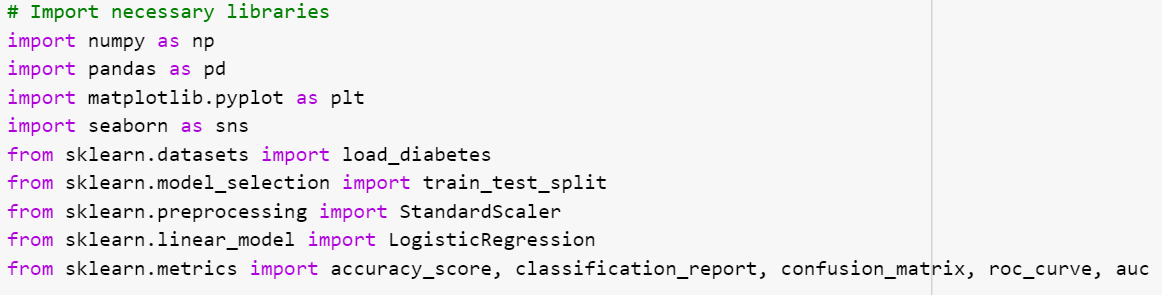
**How the Logistic Regression Algorithm Works**

[Logistic Regression](https://www.geeksforgeeks.org/understanding-logistic-regression/) models the likelihood that an instance will belong to a particular class. It uses a linear equation to combine the input information and the sigmoid function to restrict predictions between 0 and 1. Gradient descent and other techniques are used to optimize the model’s coefficients to minimize the [log loss](https://www.geeksforgeeks.org/ml-log-loss-and-mean-squared-error/). These coefficients produce the resulting decision boundary, which divides instances into two classes. When it comes to binary classification, logistic regression is the best choice because it is easy to understand, straightforward, and useful in a variety of settings. Generalization can be improved by using regularization.

**Key Concepts of Logistic Regression**

Important key concepts in logistic regression include:

* **Sigmoid Function:** The main function that ensures outputs are between 0 and 1 by converting a linear combination of input data into probabilities.  
  The [sigmoid function](https://www.geeksforgeeks.org/derivative-of-the-sigmoid-function/) is denoted as, and is defined as:  
    
  Where, z is linear combination of input features and coefficients.
* **Hypothesis Function:** uses the sigmoid function and weights (coefficients) to combine input features to estimate the likelihood of falling into a particular class.  
  In logistic regression, the [hypothesis function](https://www.geeksforgeeks.org/ml-understanding-hypothesis/) is provided by:  
    
  Where,is the predicted probability that y = 1,is the vector of coefficients, and x is the vector of input features.
* **Log Loss:** The optimization [cost function](https://www.geeksforgeeks.org/what-is-cost-function/)is a measure of the discrepancy between actual class labels and projected probability.  
  The definition of the log loss for a single instance is:
* **Decision Boundary:**The surface or line used to divide instances into several classes according to the determined probability.
* **Probability Threshold:** a number (usually 0.5) that is used to calculate the class assignment using the probabilities that are anticipated.
* **Odds Ratio:** The likelihood that an event will occur as opposed to not, which sheds light on how characteristics and the target variable are related.



A computer screen shot of words

Description automatically generated

This code loads the diabetes dataset using the load\_diabetes function from scikit-learn, passing in feature data X and target values y. Then, it converts the binary representation of the continuous target variable y. A patient’s diabetes measure is classified as 1 (indicating diabetes) if it is higher than the median value, and as 0 (showing no diabetes).

**Splitting The Dataset: Train and Test dataset**

Splitting the dataset to train and test. 80% of data is used for training the model and 20% of it is used to test the performance of our model.

A black and green text

Description automatically generated

This code divides the diabetes dataset into training and testing sets using the [train\_test\_split](https://www.geeksforgeeks.org/how-to-split-the-dataset-with-scikit-learns-train_test_split-function/) function from scikit-learn: The binary target variable is called y\_binary, and the characteristics are contained in X. The data is divided into testing (X\_test, y\_test) and training (X\_train, y\_train) sets. Twenty percent of the data will be used for testing, according to the setting test\_size=0.2. By employing a fixed seed for randomization throughout the split, random\_state=42 guarantees reproducibility.

**Feature Scaling**

A close-up of a computer code

Description automatically generated

This code uses [StandardScaler](https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/) from scikit-learn to achieve feature standardization:

The StandardScaler instance is created; this will be used to standardize the features. It uses the scaler’s fit\_transform method to normalize the training data (X\_train) and determine its mean and standard deviation. Then, itstandardizes the testing data (X\_test) using the calculated mean and standard deviation from the training set. Model training and evaluation are made easier by standardization, which guarantees that the features have a mean of 0 and a standard deviation of 1.

**Train The Model**

A close up of a text

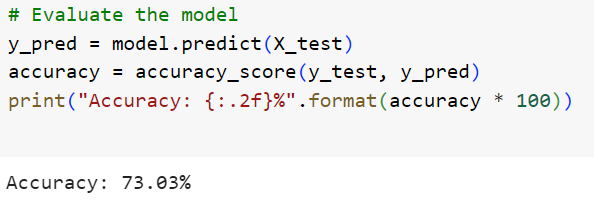
Description automatically generated

Using scikit-learn’s [LogisticRegression](https://www.geeksforgeeks.org/advantages-and-disadvantages-of-logistic-regression/), this code trains a logistic regression model:

It establishes a logistic regression model instance.Then, itemploys the fit approach to train the model using the binary target values (y\_train) and standardized training data (X\_train). Following execution, the model object may now be used to forecast new data using the patterns it has learnt from the training set.

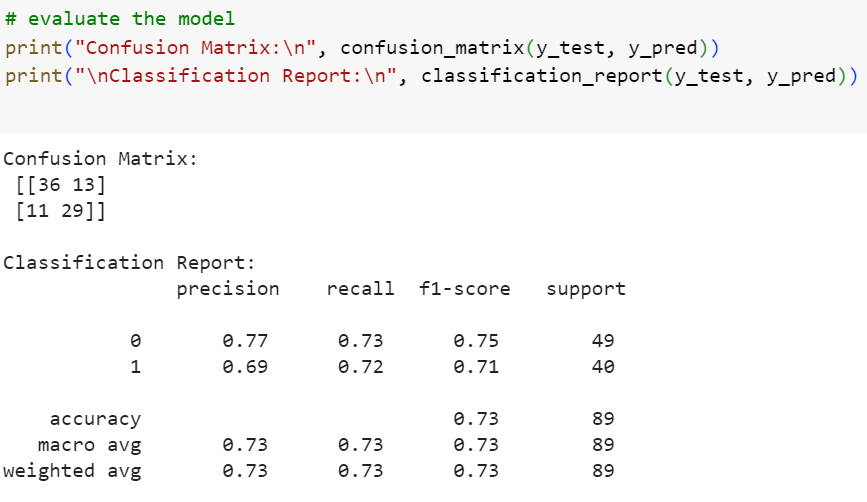
**Evaluation Metrics**

Metrics are used to check the model performance on predicted values and actual values.



This code predicts the target variable and computes its accuracy in order to assess the logistic regression model on the test set. The accuracy\_score function is then used to compare the predicted values in the y\_pred array with the actual target values (y\_test).

**Confusion Matrix and Classification Report**



**Visualizing the performance of our model.**

A screenshot of a computer code

Description automatically generated

A screen shot of a graph

Description automatically generated

To see a logistic regression model’s decision border, this code creates a scatter plot. An individual from the test set is represented by each point on the plot, which has age on the Y-axis and BMI on the X-axis. The points are color-coded according to the actual status of diabetes, making it easier to evaluate how well the model differentiates between those with and without the disease. An instant visual context for the model’s performance on the test data is provided by the plot’s title, which includes the accuracy information. The inscription located in the upper right corner denotes the colors that represent diabetes (1) and no diabetes (0).

**Plotting ROC Curve**

A computer screen shot of a program

Description automatically generated

A graph of a curve

Description automatically generated with medium confidence

For the logistic regression model, this code creates and presents the Receiver Operating Characteristic (ROC) curve. The true positive rate (sensitivity) and false positive rate at different threshold values are determined using the probability estimates for positive outcomes (y\_prob), which are obtained using the predict\_proba method. Use of the roc\_auc\_score yields the area under the [ROC curve](https://www.geeksforgeeks.org/auc-roc-curve/) (AUC). An illustration of the resulting curve is provided, and the legend shows the AUC value. The ROC curve for a random classifier is shown by the dotted line.

EXPT 8: KNN ALGORITHM

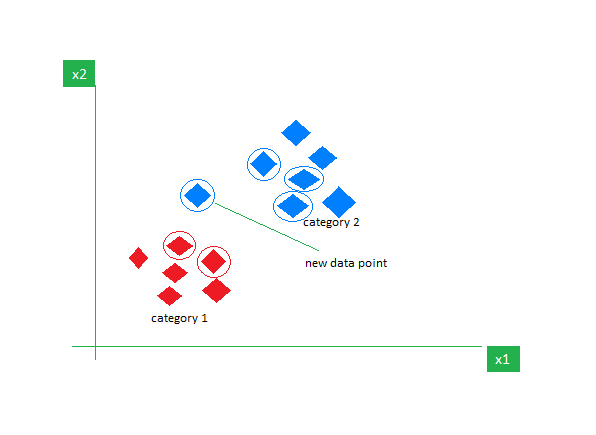
The **K-Nearest Neighbors (KNN) algorithm** is a supervised machine learning method employed to tackle classification and regression problems. Evelyn Fix and Joseph Hodges developed this algorithm in 1951, which was subsequently expanded by Thomas Cover. The article explores the fundamentals, workings, and implementation of the KNN algorithm.

**What is the K-Nearest Neighbors Algorithm?**

KNN is one of the most basic yet essential classification algorithms in machine learning. It belongs to the [supervised learning](https://www.geeksforgeeks.org/supervised-unsupervised-learning/) domain and finds intense application in pattern recognition, [data mining](https://www.geeksforgeeks.org/data-mining/), and intrusion detection.

It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a [Gaussian distribution](https://www.geeksforgeeks.org/mathematics-probability-distributions-set-3-normal-distribution/) of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

As an example, consider the following table of data points containing two features:



*KNN Algorithm working visualization*

Now, given another set of data points (also called testing data), allocate these points to a group by analyzing the training set. Note that the unclassified points are marked as ‘White’.

**Intuition Behind KNN Algorithm**

If we plot these points on a graph, we may be able to locate some clusters or groups. Now, given an unclassified point, we can assign it to a group by observing what group its nearest neighbors belong to. This means a point close to a cluster of points classified as ‘Red’ has a higher probability of getting classified as ‘Red’.

Intuitively, we can see that the first point (2.5, 7) should be classified as ‘Green’, and the second point (5.5, 4.5) should be classified as ‘Red’.

**Why do we need a KNN algorithm?**

(K-NN) algorithm is a versatile and widely used machine learning algorithm that is primarily used for its simplicity and ease of implementation. It does not require any assumptions about the underlying data distribution. It can also handle both numerical and categorical data, making it a flexible choice for various types of datasets in classification and regression tasks. It is a non-parametric method that makes predictions based on the similarity of data points in a given dataset. K-NN is less sensitive to outliers compared to other algorithms.

The K-NN algorithm works by finding the K nearest neighbors to a given data point based on a distance metric, such as Euclidean distance. The class or value of the data point is then determined by the majority vote or average of the K neighbors. This approach allows the algorithm to adapt to different patterns and make predictions based on the local structure of the data.

**Distance Metrics Used in KNN Algorithm**

As we know that the KNN algorithm helps us identify the nearest points or the groups for a query point. But to determine the closest groups or the nearest points for a query point we need some metric. For this purpose, we use below distance metrics:

**Euclidean Distance**

This is nothing but the cartesian distance between the two points which are in the plane/hyperplane. [Euclidean distance](https://www.geeksforgeeks.org/calculate-the-euclidean-distance-using-numpy/) can also be visualized as the length of the straight line that joins the two points which are into consideration. This metric helps us calculate the net displacement done between the two states of an object.

A black square root and a black triangle

Description automatically generated with medium confidence

**Manhattan Distance**

[Manhattan Distance](https://www.geeksforgeeks.org/how-to-calculate-manhattan-distance-in-r/) metric is generally used when we are interested in the total distance traveled by the object instead of the displacement. This metric is calculated by summing the absolute difference between the coordinates of the points in n-dimensions.



**Minkowski Distance**

We can say that the Euclidean, as well as the Manhattan distance, are special cases of the [Minkowski distance](https://www.geeksforgeeks.org/minkowski-distance-python/).



From the formula above we can say that when p = 2 then it is the same as the formula for the Euclidean distance and when p = 1 then we obtain the formula for the Manhattan distance.

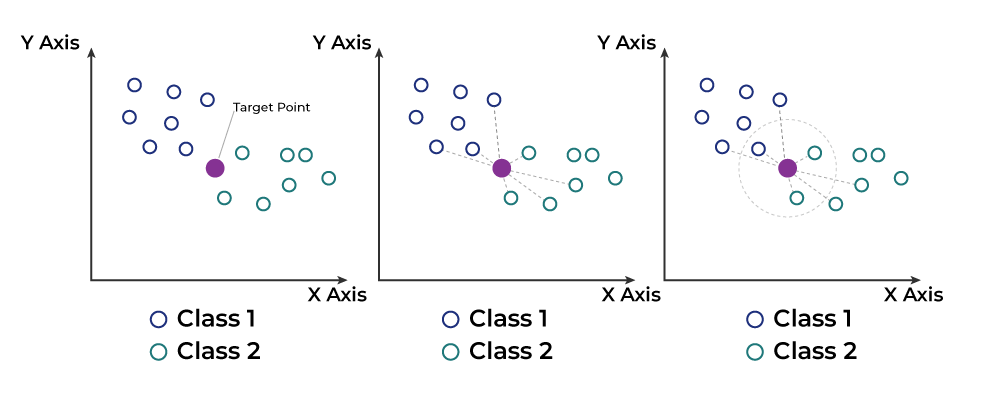
The above-discussed metrics are most common while dealing with a [Machine Learning](https://www.geeksforgeeks.org/machine-learning/) problem but there are other distance metrics as well like [Hamming Distance](https://www.geeksforgeeks.org/hamming-distance-two-strings/) which come in handy while dealing with problems that require overlapping comparisons between two vectors whose contents can be Boolean as well as string values.

**How to choose the value of k for KNN Algorithm?**

The value of k is very crucial in the KNN algorithm to define the number of neighbors in the algorithm. The value of k in the k-nearest neighbors (k-NN) algorithm should be chosen based on the input data. If the input data has more outliers or noise, a higher value of k would be better. It is recommended to choose an odd value for k to avoid ties in classification. [Cross-validation](https://www.geeksforgeeks.org/cross-validation-machine-learning/) methods can help in selecting the best k value for the given dataset.

**Workings of KNN algorithm**

Thе K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity, where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.



Step-by-Step explanation of how KNN works is discussed below:

**Step 1: Selecting the optimal value of K**

* K represents the number of nearest neighbors that needs to be considered while making prediction.

**Step 2: Calculating distance**

* To measure the similarity between target and training data points, Euclidean distance is used. Distance is calculated between each of the data points in the dataset and target point.

**Step 3: Finding Nearest Neighbors**

* The k data points with the smallest distances to the target point are the nearest neighbors.

**Step 4: Voting for Classification or Taking Average for Regression**

* In the classification problem, the class labels of are determined by performing majority voting. The class with the most occurrences among the neighbors becomes the predicted class for the target data point.
* In the regression problem, the class label is calculated by taking average of the target values of K nearest neighbors. The calculated average value becomes the predicted output for the target data point.

Let X be the training dataset with n data points, where each data point is represented by a d-dimensional feature vector  and Y be the corresponding labels or values for each data point in X. Given a new data point x, the algorithm calculates the distance between x and each data point  in X using a distance metric, such as Euclidean distance:

The algorithm selects the K data points from X that have the shortest distances to x. For classification tasks, the algorithm assigns the label y that is most frequent among the K nearest neighbors to x. For regression tasks, the algorithm calculates the average or weighted average of the values y of the K nearest neighbors and assigns it as the predicted value for x.

**Advantages of the KNN Algorithm**

* **Easy to implement** as the complexity of the algorithm is not that high.
* **Adapts Easily** – As per the working of the KNN algorithm it stores all the data in memory storage and hence whenever a new example or data point is added then the algorithm adjusts itself as per that new example and has its contribution to the future predictions as well.
* **Few Hyperparameters** – The only parameters which are required in the training of a KNN algorithm are the value of k and the choice of the distance metric which we would like to choose from our evaluation metric.

**Disadvantages of the KNN Algorithm**

* **Does not scale** – As we have heard about this that the KNN algorithm is also considered a Lazy Algorithm. The main significance of this term is that this takes lots of computing power as well as data storage. This makes this algorithm both time-consuming and resource exhausting.
* **Curse of Dimensionality** – There is a term known as the peaking phenomenon according to this the KNN algorithm is affected by the [curse of dimensionality](https://www.geeksforgeeks.org/videos/curse-of-dimensionality-in-machine-learning/) which implies the algorithm faces a hard time classifying the data points properly when the dimensionality is too high.
* **Prone to Overfitting** – As the algorithm is affected due to the curse of dimensionality it is prone to the problem of overfitting as well. Hence generally [feature selection](https://www.geeksforgeeks.org/feature-selection-techniques-in-machine-learning/) as well as [dimensionality reduction](https://www.geeksforgeeks.org/dimensionality-reduction/) techniques are applied to deal with this problem.

**Code:**

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Output:

