Data Science and Visualization

Assignment No. 7

Title: Bar Chart Race using D3.js

Aim: To plot interactive visualizations using D3.js library.

Problem Statement: Create a bar chart for the data. Refer this for the bar chart - https://observablehq.com/@d3/bar-chart-race-explained scatterplot

Theory:

D3

D3 allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. For example, you can use D3 to generate an HTML table from an array of numbers. Or, use the same data to create an interactive SVG bar chart with smooth transitions and interaction.

D3 is not a monolithic framework that seeks to provide every conceivable feature. Instead, D3 solves the crux of the problem: efficient manipulation of documents based on data. This avoids proprietary representation and affords extraordinary flexibility, exposing the full capabilities of web standards such as HTML, SVG, and CSS. With minimal overhead, D3 is extremely fast, supporting large datasets and dynamic behaviours for interaction and animation. D3's functional style allows code reuse through a diverse collection of official and community-developed modules.

How to Select Elements in D3

When you're coding in JavaScript and you need to modify elements on a page, you need to select those elements. D3.js works the same way, and provides us with two methods to select DOM elements:

- d3.select()
- d3.selectAll()

Both of this selector methods will take in any CSS selector and return the element that matches the specified selector. If no element matches the selector it will return an empty selection.

The d3.select() method will select the first element that matches in the DOM (from top to bottom).

```
d3.select("#d3_p").style("color", "blue");
```

Example output

hello world 1

Dynamic properties and Data binding

Another main concept of D3 is mapping a set of data to the DOM elements in a dynamic manner. Here we can introduce a datasets and then we can update, append and display the DOM elements using those datasets, realtime.

```
let dataset = [1,2,3,4,5]
d3.selectAll('p')      //Select 'p' element
.data(dataset)         //data()puts data into waiting for processing
.enter()               //take data elements one by one
.append('p')           //for each data item appending <p>
.text('Sample text');  //add sample text to each
```

This will render the text "sample text" 5 times in the front end. This is only a simple example to show that we can use data to manipulate the elements dynamically and real time. There are a lot of things that can be done using this same concept.

Data visualization

Since we are now quite comfortable with the basic concepts of D3 we can go for the data visualization components which consists of various types of graphs, data tables and other visualizations.

Scalable Vector Graphics (SVG) is a way to render graphical elements and images in the DOM. As SVG is vector-based, it's both lightweight and scalable. D3 uses SVG to create all its visuals such as graphs. The great thing about using SVGs is we don't have to worry about distortion in scaling the visuals unlike in other formats. Basically D3 helps us to bridge the gap between the data and the relevant visualizations to give the users meaningful information.

Let's start with creating a simple Bar chart using D3.js. You need two files named index.html, script.js and style.css as below to try this out.

index.html

```
<html><head><link rel=”stylesheet” href=”style.css”><title>My sample D3.js App</title></head><body><h1>Bar Chart using D3.js</h1><svg width=”500" height=”800" class=”bar-chart”></svg><script src=”https://d3js.org/d3.v5.min.js"></script><script src=”script.js”></script></body></html>
```

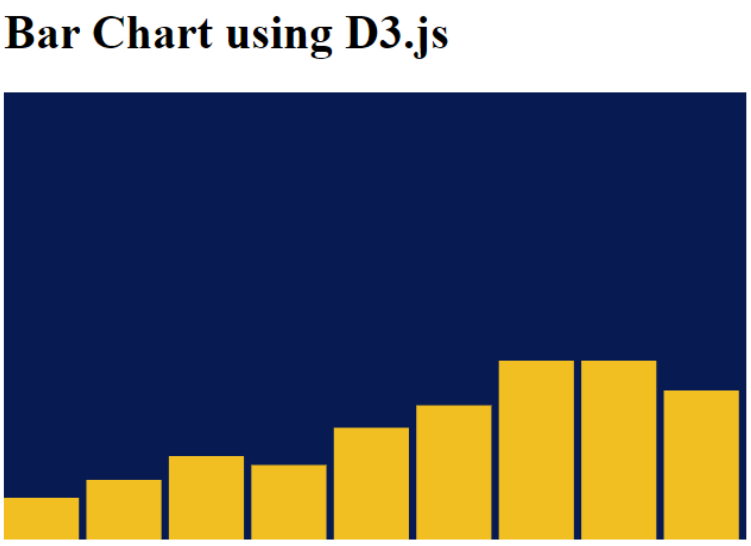
script.js

```
var dataset = [28, 40, 56, 50, 75, 90, 120, 120, 100];var chartWidth = 500, chartHeight = 300, barPadding = 5;var barWidth = (chartWidth / dataset.length);var svg = d3.select(‘svg’).attr(“width”, chartWidth).attr(“height”, chartHeight);var barChart = svg.selectAll(“rect”).data(dataset).enter().append(“rect”).attr(“y”, function(d) {return chartHeight — d}).attr(“height”, function(d) {return d;}).attr(“width”, barWidth — barPadding).attr(“fill”, ‘#F2BF23’).attr(“transform”, function (d, i) {var translate = [barWidth * i, 0];return “translate(“+ translate +”)”;});
```

style.css

```
.bar-chart {background-color: #071B52;}
```

The resulting bar chart would like something like this.



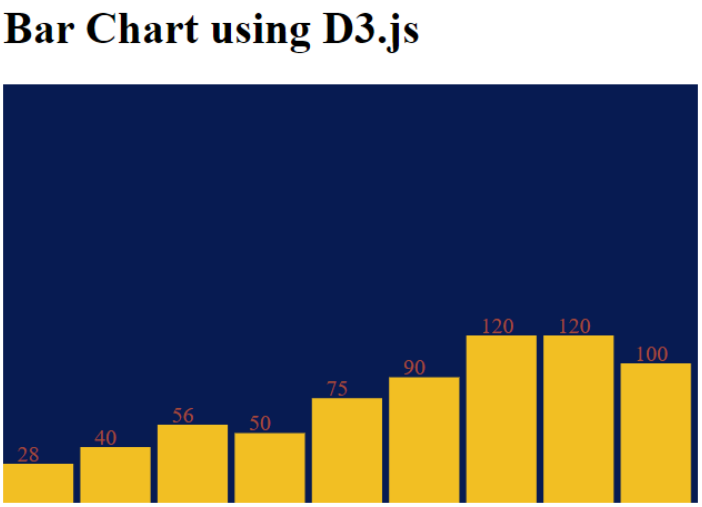
This shows the visualization but there is no precise information other than the trend. So our next task is to add some labels so that the values of each bar will be visible giving more information on visuals.

For that you can add following code snippet add the end of the script.js file.

script.js

```
var text = svg.selectAll("text").data(dataset).enter().append("text").text(function(d) {return d;}).attr("y", function(d, i) {return chartHeight — d — 2;}).attr("x", function(d, i) {return barWidth * i + 10;}).attr("fill", "#A64C38");
```

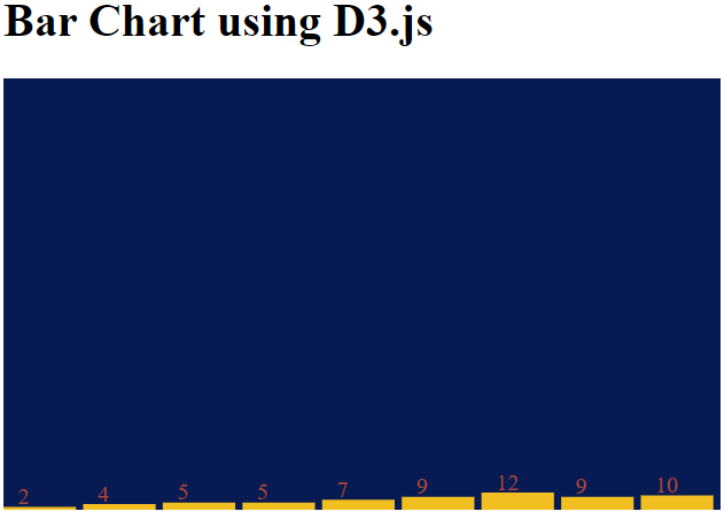
This will result something like below showing the values.



Now what are going to do is add scaling to our chart. You can have various values in your data set; some can be really small and some can be really large. So for better visualizations with consistency it's important to have scaling in your chart.

If we rearrange our dataset as below you can see how the bar chart is rendered.

```
var dataset = [2, 4, 5, 5, 7, 9, 12, 9, 10];
```

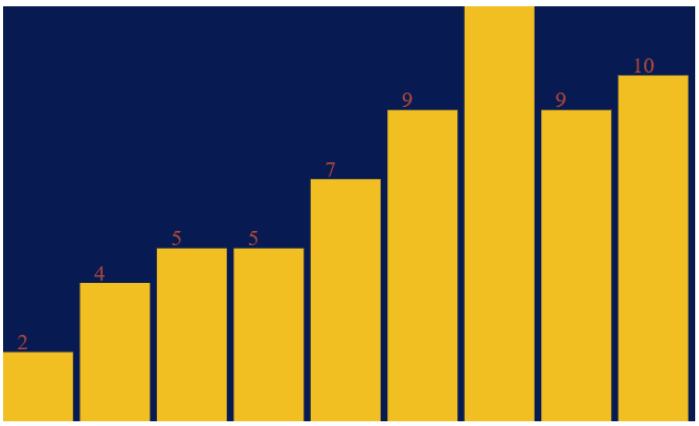


So this way we can barely see the bars in the chart. So we have to scale it up according to the chart height. If you have larger values in your dataset it will be not shown the accurate height within the given chart height. So the solution for this is scaling accordingly. For that we can use a scaler like this and change the bar chart.

script.js

```
var dataset = [2, 4, 5, 5, 7, 9, 12, 9, 10];var chartWidth = 500, chartHeight = 300, barPadding =
5;var barWidth = (chartWidth / dataset.length);var svg = d3.select('svg')
.attr("width", chartWidth)
.attr("height", chartHeight);var yScale = d3.scaleLinear()
.domain([0, d3.max(dataset)])
.range([0, chartHeight])var barChart = svg.selectAll("rect")
.data(dataset)
.enter()
.append("rect")
.attr("y", function(d) {
return chartHeight — yScale(d);})
.attr("height", function(d) {
return yScale(d);})
.attr("width", barWidth — barPadding)
.attr("fill", '#F2BF23')
.attr("transform", function (d, i) {
var translate = [barWidth * i, 0];
return "translate("+ translate +")";
});var text = svg.selectAll("text")
.data(dataset)
.enter()
.append("text")
.text(function(d) {
return d;})
.attr("y", function(d, i) {
return chartHeight — yScale(d) — 2;})
.attr("x", function(d, i) {
return barWidth * i + 10;})
.attr("fill", "#A64C38");
```

Bar Chart using D3.js



So now it's quite obvious that axes are missing out in our chart. So it's really simple to add axes to our graph using D3.js. You can create x-axis and y-axis using x-scale and y-scale in D3.js. You can create a scaled graph with labels using following code snippet.

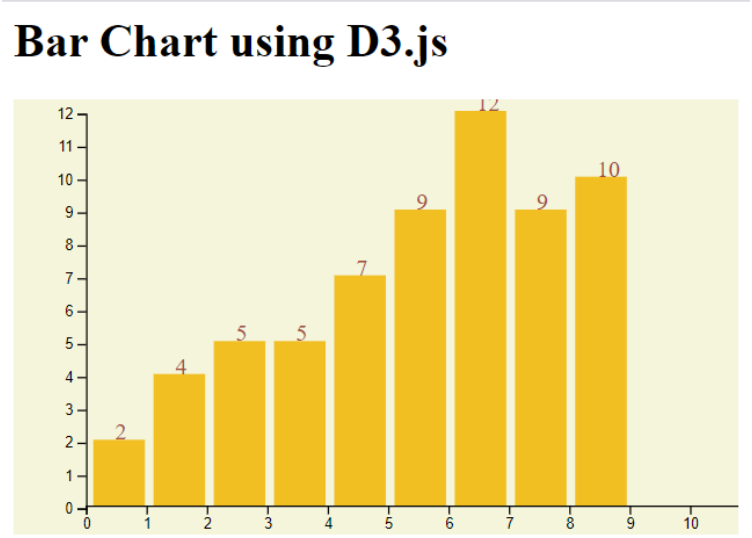
script.js

```
var dataset = [2, 4, 5, 5, 7, 9, 12, 9, 10];var chartWidth = 500, chartHeight = 300, barPadding =
6;var barWidth = (chartWidth / dataset.length -14);var svg = d3.select('svg')
.attr("width", chartWidth)
.attr("height", chartHeight);var xScale = d3.scaleLinear()
.domain([0, d3.max(dataset)])
.range([0, chartWidth]);var yScale = d3.scaleLinear()
.domain([0, d3.max(dataset)])
.range([0,chartHeight — 28]);var yScaleChart = d3.scaleLinear()
.domain([0, d3.max(dataset)])
.range([chartHeight — 28, 0]);var barChart = svg.selectAll("rect")
.data(dataset)
.enter()
.append("rect")
.attr("y", function(d) {
return chartHeight — yScale(d) — 20;})
.attr("height", function(d) {
return yScale(d);})
.attr("width", barWidth — barPadding)
.attr("fill", '#F2BF23')
.attr("transform", function (d, i) {var translate = [barWidth * i +55, 0];
return "translate("+ translate +")";});var text = svg.selectAll("text")
.data(dataset)
.enter()
.append("text")
.text(function(d) {
return d;})
```

```
.attr("y", function(d, i) {
return chartHeight — yScale(d) — 20;})
.attr("x", function(d, i) {
return barWidth * i + 70;})
.attr("fill", "#A64C38");var x_axis = d3.axisBottom().scale(xScale);var y_axis =
d3.axisLeft().scale(yScaleChart);svg.append("g")
.attr("transform", "translate(50, 10)")
.call(y_axis);var xAxisTranslate = chartHeight — 20;svg.append("g")
.attr("transform", "translate(50, " + xAxisTranslate +")")
.call(x_axis);
```

style.css

```
.bar-chart {background-color: beige;}
```



So like above simple bar chart we can create many types of graphs we want. The best thing here is the control we have over what we create. Unlike other ready-made graphs which has are limited-customizable, we can have the freedom of creating our own graphs using SVGs in D3.js.

D3 solves the cause of the problem: efficient manipulation of documents based on data. This avoids proprietary representation and affords extraordinary flexibility, exposing the full capabilities of web standards such as HTML, SVG, and CSS. With minimal overhead, D3 is extremely fast, supporting large datasets and dynamic behaviors for interaction and animation.

Conclusion: D3 allows manipulation of dynamic elements on a webpage. The user can directly make changes to the Document Object Model. It also provides some awesome features for interactions and animations.

References:

[1] https://towardsdatascience.com/data-visualization-with-d3-js-for-beginners-b62921e03b49

[2] https://d3js.org/

Data Science and Visualization

Assignment No. 8

Title: Linear Regression

Aim: Perform Linear Regression on a dataset and predict best fit.

Problem Statement: Create a linear regression model based the positioning of random data generated using numpy and Intercept, and predict a Best Fit.

Theory:

Regression

Regression analysis is a statistical methodology that allows us to determine the strength and relationship of two variables. Regression is not limited to two variables, we could have 2 or more variables showing a relationship. The results from the regression help in predicting an unknown value depending on the relationship with the predicting variables. For example, someone's height and weight usually have a relationship. Generally, taller people tend to weigh more. We could use regression analysis to help predict the weight of an individual, given their height.

When there is a single input variable, the regression is referred to as **Simple Linear Regression**. We use the single variable (independent) to model a linear relationship with the target variable (dependent). We do this by fitting a model to describe the relationship. If there is more than predicting variable, the regression is referred to as **Multiple Linear Regression.**

Ordinary Least Squares

You may have heard of **Ordinary Least Squares Regression**. When we are attempting to find the "best fit line", the regression model can sometimes be referred to as Ordinary Least Squares Regression. This just means that we're using the smallest sum of squared errors. The error is the difference between the predicted y value subtracted from the actual y value. The difference is squared so there is an absolute difference, and summed.

```
error = y_actual - y_predicted
```

Straight Line Formula: $y = mx + c$
- Where {m} is the slope and {c} is the intercept

Straight Line Formula with estimations:

- $\hat{y} = \hat{m}x + \hat{c}$

Or

- $\hat{y} = \hat{\beta}0 + \hat{\beta}1x$

We're just estimating a proper intercept and slope. When we have drawn the "best fit line" we are ready to make some predictions. However, since our prediction is based on the parameter values we estimate, when we predict new y values given x, we will have error or vertical offset (blue lines). This error is denoted as $|\hat{y} - y|$ where $\hat{y}$ is on our regression line and **y** is the actual observed value (green dot).

Regression Coefficients

When performing simple linear regression, the four main components are:

- **Dependent Variable** — Target variable / will be estimated and predicted
- **Independent Variable** — Predictor variable / used to estimate and predict
- **Slope** — Angle of the line / denoted as m or $\beta 1$
- **Intercept** — Where function crosses the y-axis / denoted as c or $\beta 0$

The last two, slope and intercept, are the coefficients/parameters of a linear regression model, so when we calculate the regression model, we're just calculating these two. In the end, we're

trying to find the best-fit line describing the data, out of an infinite number of lines. To find the slope of a line, we can choose a random part of the line, and divide the **change in x by the change in y**.

- **Δy — Change in y**
- **Δx — Change in x**

We need to calculate some statistical measures before calculating the “best fit line”:

- $(\bar{X})$: The mean of the X
- $(\bar{Y})$: The mean of the Y
- (S_X): The standard deviation of the X values
- (S_Y): The standard deviation of the y values
- (ρ - Pearson Correlation): The correlation between X and Y

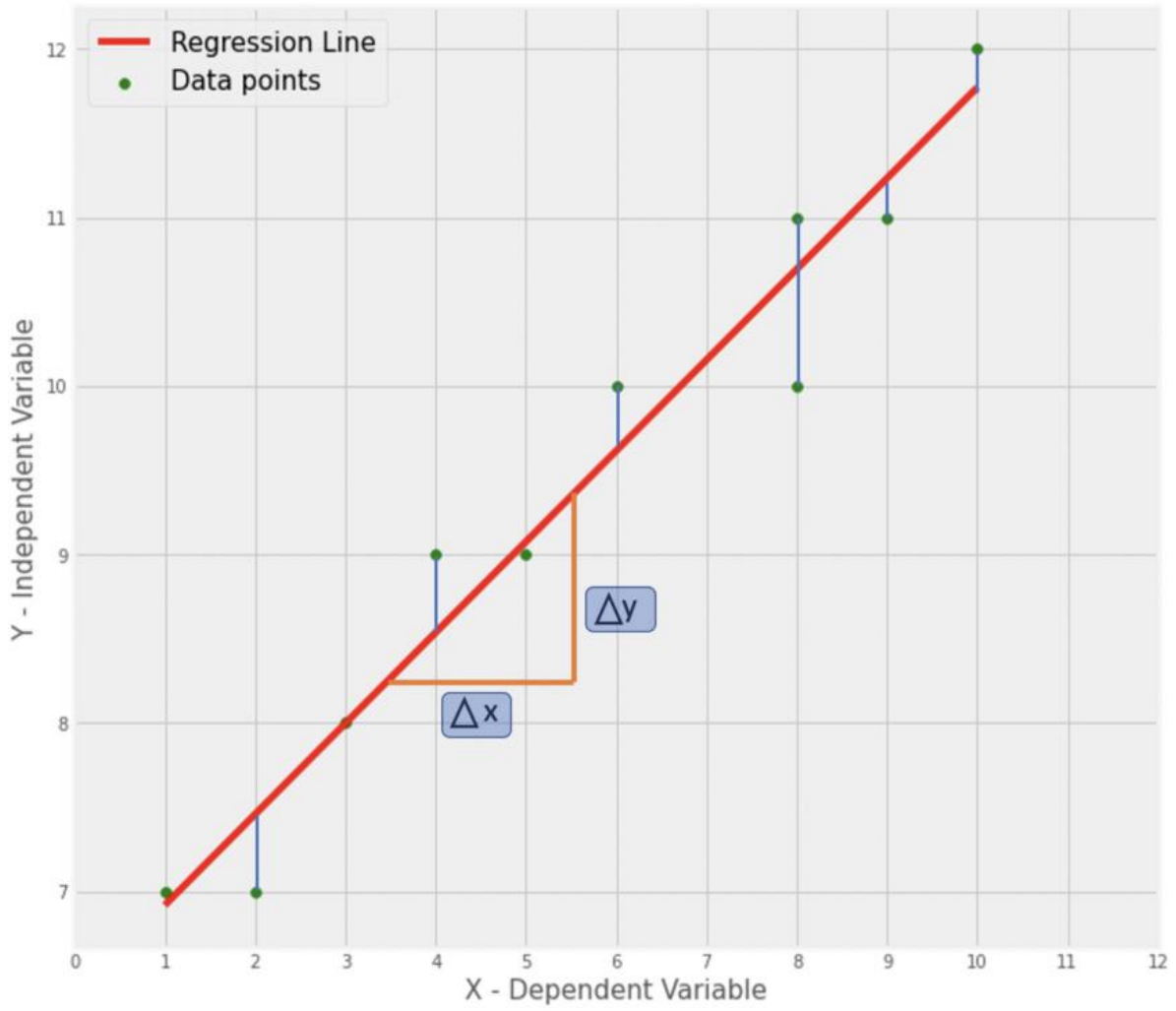
Slope Formula With the Least Squares Method

$$\hat{m} = \rho \frac{S_Y}{S_X}$$

Intercept Formula Using the Slope

$$\hat{c} = \bar{Y} - \hat{m}\bar{X}$$

The formula above is *multiplying* the **slope** by the **mean of x** and *subtracting* that **value** from the **mean of y**.



Linear Regression in sci-kit learn

Examples

```
>>>
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> # y = 1 * x_0 + 2 * x_1 + 3
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

Methods

fit(X, y[, sample_weight])	Fit linear model.
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict using the linear model.
score(X, y[, sample_weight])	Return the coefficient of determination of the prediction.
set_params(**params)	Set the parameters of this estimator.

Example

```
import pandas as pd

penguins = pd.read_csv("../datasets/penguins_regression.csv")
feature_name = "Flipper Length (mm)"
target_name = "Body Mass (g)"
data, target = penguins[[feature_name]], penguins[target_name]
```

Note

If you want a deeper overview regarding this dataset, you can refer to the Appendix - Datasets description section at the end of this MOOC.

```
from sklearn.linear_model import LinearRegression

linear_regression = LinearRegression()
linear_regression.fit(data, target)
```

LinearRegression

```
LinearRegression()
```

The instance `linear_regression` will store the parameter values in the attributes `coef_` and `intercept_`. We can check what the optimal model found is:

```
weight_flipper_length = linear_regression.coef_[0]
weight_flipper_length
```

```
49.68556640610011
```

```
intercept_body_mass = linear_regression.intercept_
intercept_body_mass
```

```
-5780.831358077066
```

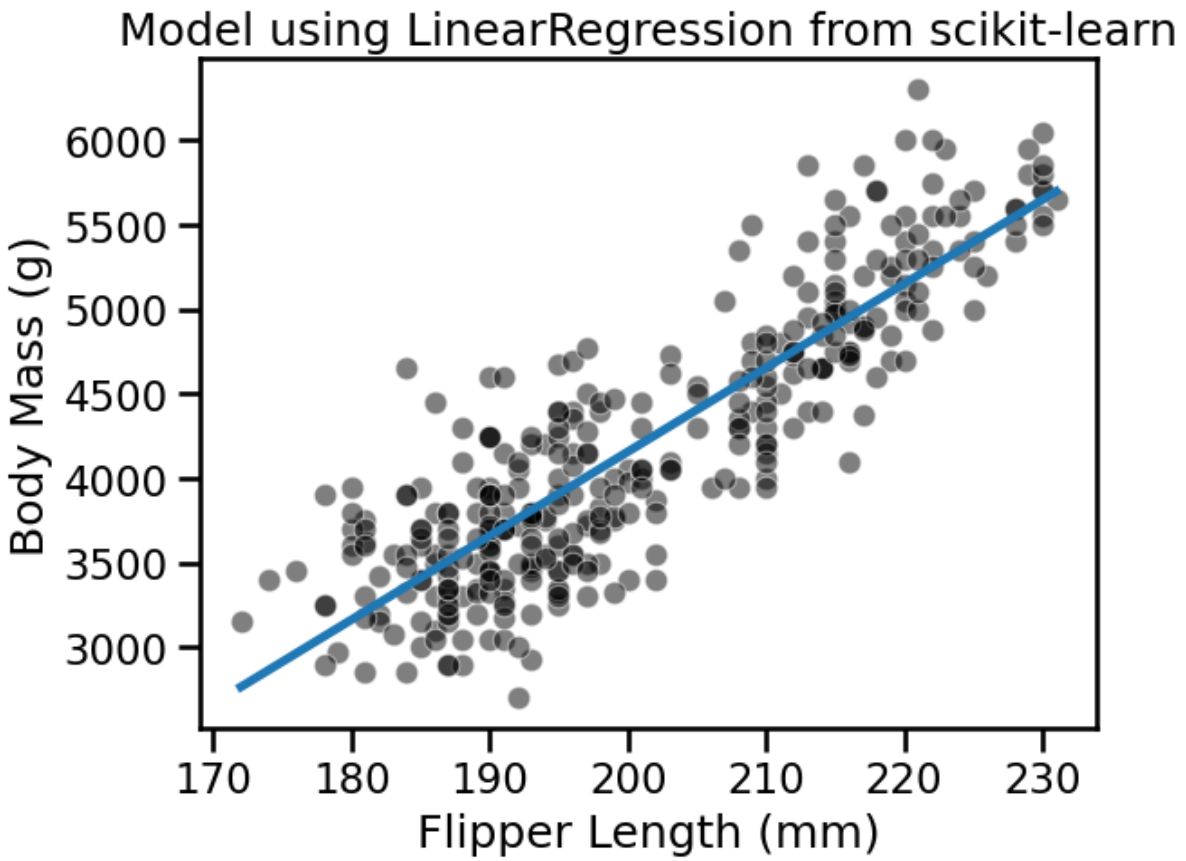
We will use the weight and intercept to plot the model found using the scikit-learn.

```
import numpy as np

flipper_length_range = np.linspace(data.min(), data.max(), num=300)
predicted_body_mass = (
    weight_flipper_length * flipper_length_range + intercept_body_mass)
```

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.scatterplot(x=data[feature_name], y=target, color="black",
alpha=0.5)
plt.plot(flipper_length_range, predicted_body_mass)
_ = plt.title("Model using LinearRegression from scikit-learn")
```



In the solution of the previous exercise, we implemented a function to compute the goodness of fit of a model. Indeed, we mentioned two metrics: (i) the mean squared error and (ii) the mean absolute error. These metrics are implemented in scikit-learn and we do not need to use our own implementation.

We can first compute the mean squared error.

```
from sklearn.metrics import mean_squared_error

inferred_body_mass = linear_regression.predict(data)
model_error = mean_squared_error(target, inferred_body_mass)
print(f"The mean squared error of the optimal model is
{model_error:.2f}")
```

```
The mean squared error of the optimal model is 154546.19
```

A linear regression model minimizes the mean squared error on the training set. This means that the parameters obtained after the fit (i.e. `coef_` and `intercept_`) are the

optimal parameters that minimizes the mean squared error. In other words, any other choice of parameters will yield a model with a higher mean squared error on the training set.

However, the mean squared error is difficult to interpret. The mean absolute error is more intuitive since it provides an error in the same unit as the one of the target.

```
from sklearn.metrics import mean_absolute_error

model_error = mean_absolute_error(target, inferred_body_mass)
print(f"The mean absolute error of the optimal model is
{model_error:.2f} g")
```

```
The mean absolute error of the optimal model is 313.00 g
```

Conclusion: Simple linear regression is a regression model that figures out the relationship between independent variable and one dependent variable using a straight line.

References:

[1] Jason Wong, https://towardsdatascience.com/linear-regression-explained-1b36f97b7572

[2] https://inria.github.io/scikit-learn-mooc/python_scripts/linear_regression_in_sklearn.html

[3] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Data Science and Visualization

Assignment No. 9

Title: Clustering

Aim: Study KMeans Clustering algorithm and perform clustering on a dataset.

Problem Statement: Plot digit data using KMeans algorithms in scikitlearn. Cluster the data points using different colors. Mark the centroids in black color.

Theory:

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm

, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering

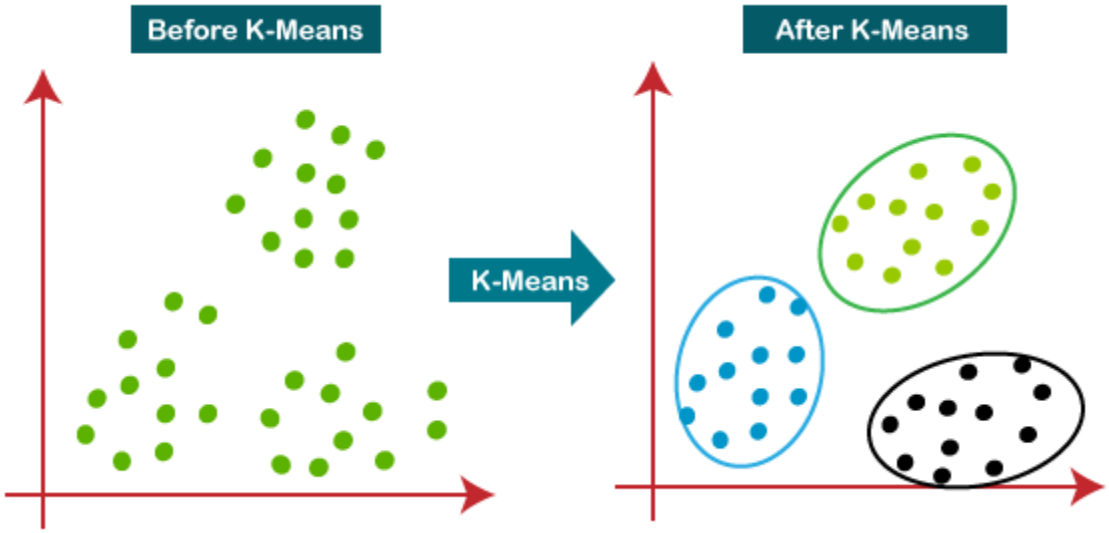
algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.

- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different

ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Kmeans in scikit learn

The k-means problem is solved using either Lloyd's or Elkan's algorithm.

The average complexity is given by O(k n T), where n is the number of samples and T is the number of iteration.

The worst case complexity is given by O(n^(k+2/p)) with n = n_samples, p = n_features. (D. Arthur and S. Vassilvitskii, 'How slow is the k-means method?' SoCG2006)

In practice, the k-means algorithm is very fast (one of the fastest clustering algorithms available), but it falls in local minima. That's why it can be useful to restart it several times.

Examples

```
>>>
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

Methods

fit(X[, y, sample_weight])	Compute k-means clustering.
fit_predict(X[, y, sample_weight])	Compute cluster centers and predict cluster index for each sample.
fit_transform(X[, y, sample_weight])	Compute clustering and transform X to cluster-distance space.
get_feature_names_out([input_features])	Get output feature names for transformation.
get_params([deep])	Get parameters for this estimator.
predict(X[, sample_weight])	Predict the closest cluster each sample in X belongs to.
score(X[, y, sample_weight])	Opposite of the value of X on the K-means objective.
set_params(**params)	Set the parameters of this estimator.

`transform`(X)	Transform X to a cluster-distance space.

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS= \sum_{Pi \text{ in Cluster1}} distance(P_i\ C_1)^2 + \sum_{Pi \text{ in Cluster2}} distance(P_i\ C_2)^2 + \sum_{Pi \text{ in CLuster3}} distance(P_i\ C_3)^2$$

In the above formula of WCSS,

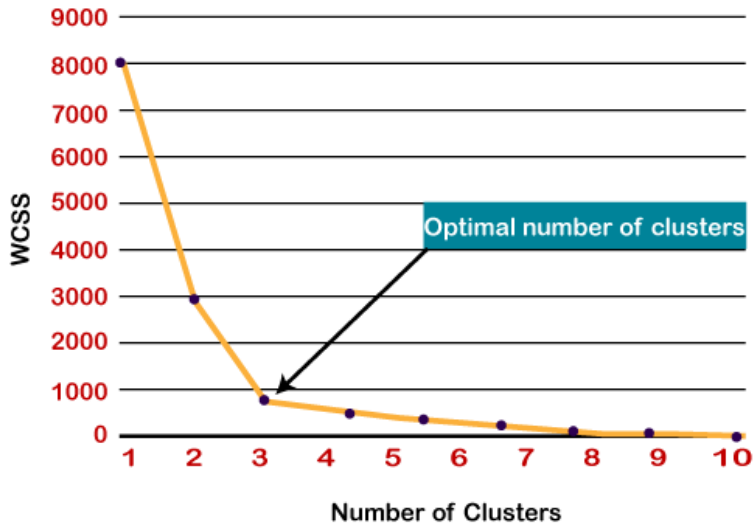
$\sum_{Pi \text{ in Cluster1}} distance(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Conclusion: K-means clustering is the unsupervised machine learning algorithm that is part of a much deep pool of data techniques and operations in the realm of Data Science. It is the fastest and most efficient algorithm to categorize data points into groups even when very little information is available about data.

References:

[1] https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning

[2] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html