Machine Learning

A. Supervised Learning (SL) - Regression

A.I Linear Regression

1. Data acquisition

Here row starts from (n-1) *20+2 = 82 and ends with n*20+1 = 101. After retrieving data for training model, data looks like:

Temperature (Celsius)	Net hourly electrical energy output (MW)
16.18	455.14
31.66	431.26
29.14	437.76
18.4	464.63
15.86	462.58
6.31	483.27
23.4	438.51
33.62	431.03
6.89	484
27.41	431.64
26.37	451.78
26.58	439.46
15.25	467.23
21.24	459.81
24.98	447.15
26.63	442
18.87	449.61
5.12	481.28
31.1	437.54
7.91	475.52

2. Data Transformation

We see that data is uniformly distributed across the range. In this case we are using Min-Max normalization to transform our data into standard range from 0 to 1. Formula used for data transformation through Min-Max normalization is:

$$X' = (X - X\_Min)/(X\_Max - X\_Min)$$

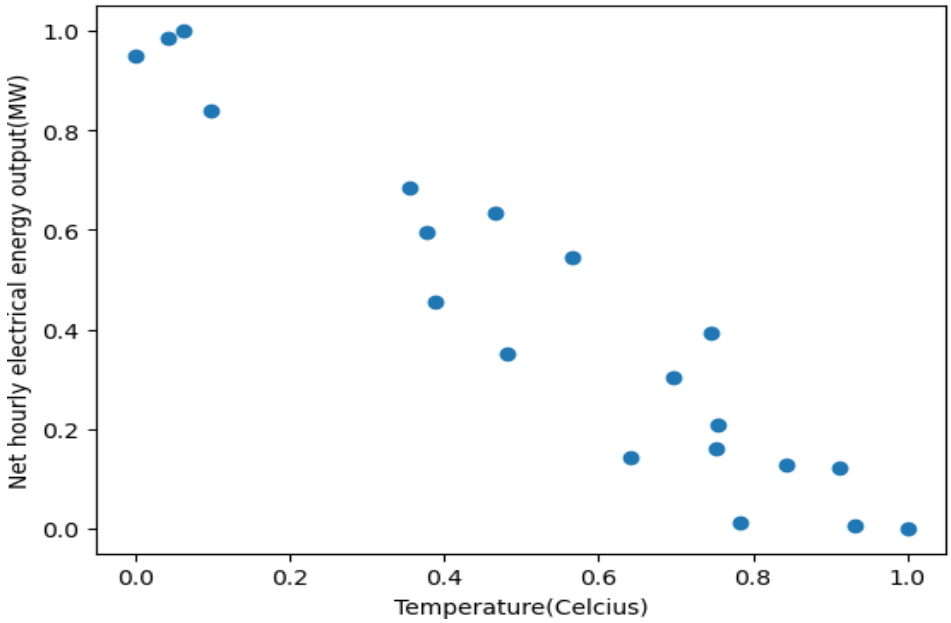
Min-Max normalization can be used for data under following conditions:

1. When we know approximate upper and lower bounds of data with few or no outliers.
2. When data is approximately uniformly distributed across that range.

In our data, we know upper and lower bounds of our data and there are no outliers. After plotting all the data points, we know that data is uniformly distributed across that range. And hence we are using Min-Max normalization to optimize model computation.

Data after transformation:

Temperature (Celsius)	Net hourly electrical energy output (MW)
0.38807018	0.4551633
0.93122807	0.00434208
0.84280702	0.12705305
0.46596491	0.63432131
0.37684211	0.59562016
0.04175439	0.98621861
0.64140351	0.14121201
1.	0.
0.06210526	1.
0.78210526	0.01151595
0.74561404	0.39173117
0.75298246	0.15914669
0.3554386	0.6834057
0.56561404	0.54332641
0.69684211	0.3043232
0.75473684	0.20709836
0.48245614	0.35076458
0.	0.94865018
0.91157895	0.12289975
0.09789474	0.83990938



3. Least Squares:

Least squares method is the method of finding predicted values y for given independent variable x with linear equation y = mx + b.

In this method values of m and b can be found with following formulas which then can be inserted in linear equation to obtain predicted values y_pred

m = ((x - x_mean)*(y-y_mean))/((x-x_mean)^2)

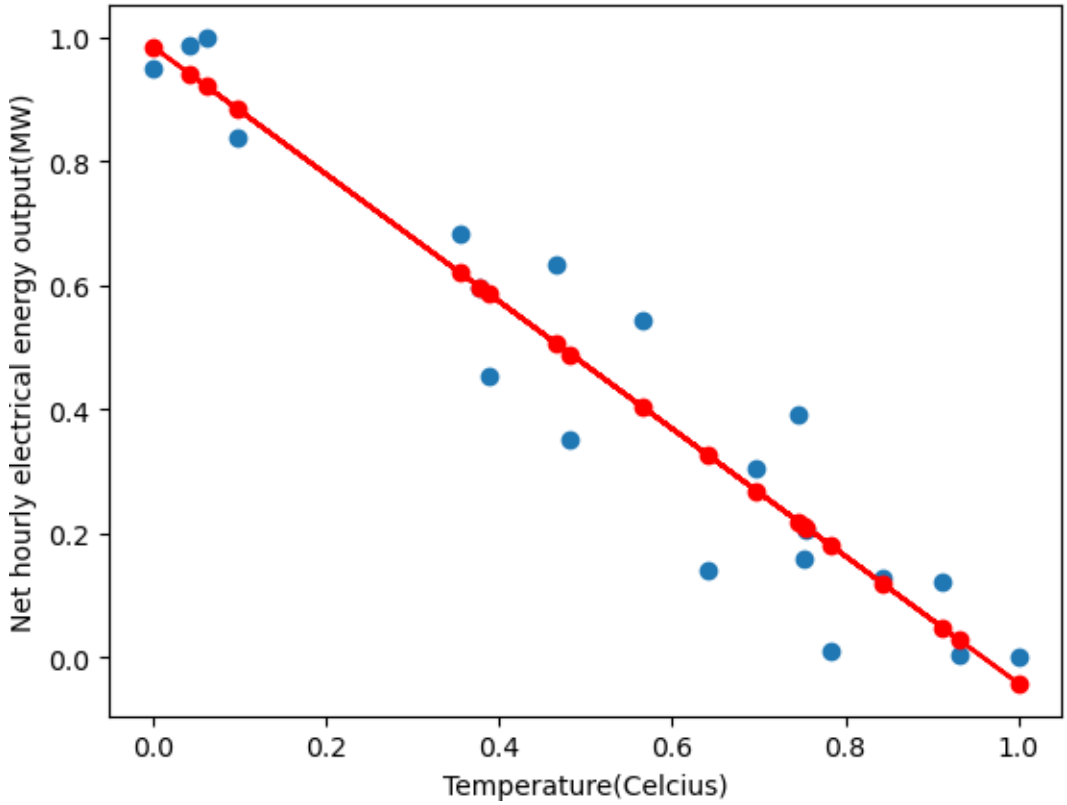
b = (y_mean - m*x_mean)

After applying least squares method on data set, output received is:

m derived from least square method is -1.0273956362438041

b derived from least square method is 0.9850313987868536

After plotting the regression line and data points:



4. Linear Regression with Gradient Descent – Cost Function:

Cost function is nothing but error in predicted value and actual value. In order obtain most accurate values for our regression parameters m and b, we need to minimize this cost. Here in our example, we will cost function as below.

Mean Squared Error = (1/n)*sum((yi - yi_pred)^2)

Notations:

Mean Squared Error = Cost Function

yi = Actual output values for y

yi_pred = Predicted output values for y

Here error can be represented as (yi – yi_pred). We are taking square of error to handle negative values and then taking mean of it to get the cost for all data points. We know that yi_pred is nothing but (mxi + b) from above equation. Hence putting value for yi_pred our equation becomes:

Cost = (1/n) *sum((yi – (m*xi + b))^2)

Where m*xi + b = y_pred

Partial Derivative with respect to regression parameters m and b.

i) Partial Derivative with respect to regression parameter m

$D_m = (1/n)*sum(d(y_i - (mx_i + b))^2/dm)$

$= (1/n)*sum(2(y_i - (mx_i + b))* d(y_i – (mx_i + b))/dm$

$=(1/n)*sum(2(y_i - (mx_i + b))*(0 – x_i – 0)$

$D_m = (-2/n)* sum((x_i)*(y_i - (mx_i + b)))$

$$\boldsymbol{D_m = (-2/n)* sum((x_i)*(y_i - (mx_i + b)))}$$

ii) Partial Derivative with respect to regression parameter b

$D_b = (1/n)*sum(d(y_i - (mx_i + b))^2/db)$

$= (1/n)*sum(2(y_i - (mx_i + b))* d(y_i – (mx_i + b))/db$

$=(1/n)*sum(2(y_i - (mx_i + b))*(0 – 0 – 1)$

$D_b = (-2/n)* sum(y_i - (mx_i + b))$

$$\boldsymbol{D_b = (-2/n)* sum(y_i - (mx_i + b))}$$

Update rules for current values of m and b can be given as:

$$\boldsymbol{m = m – Learning\_Rate * D_m}$$

$$\boldsymbol{b = b – Learning\_Rate * D_b}$$

Learning rate can be a very small value initially to get the good accuracy.

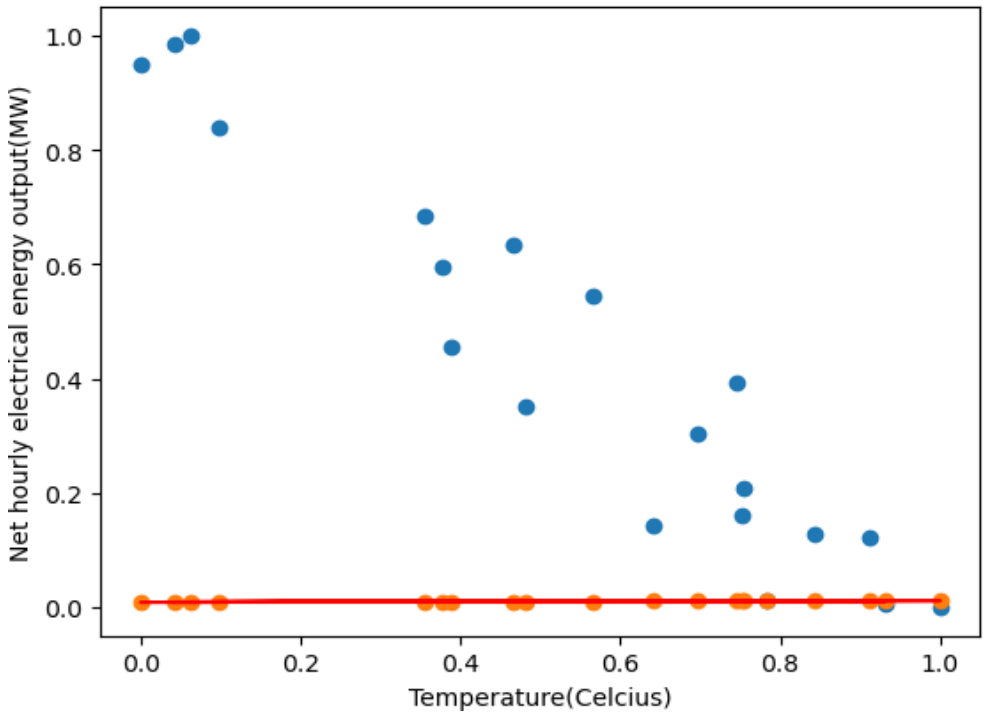
5. Linear Regression with Gradient Descent – First iteration:

In the first iteration of our gradient descent algorithm, we choose initial values of m and b as m = 0 and b = 0

We have chosen learning rate as 0.01. After applying first iteration we get the values of m and b as m = 0.002704641753524838 and b = 0.008506701906739666

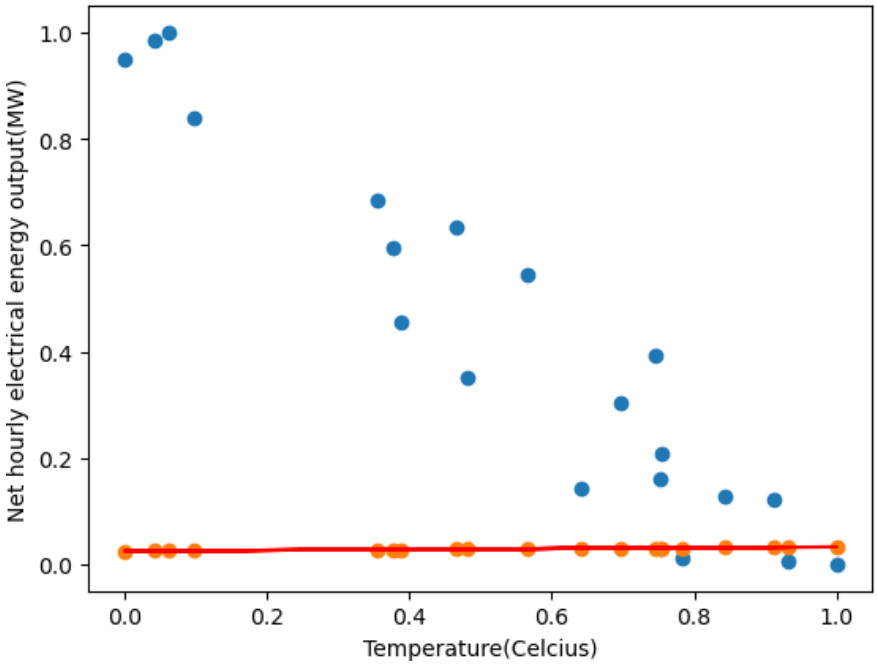
```
After first iteration m 0.002704641753524838, b 0.008506701906739666, cost 0.
28966089032981396
```

```
After first iteration graph of regression line looks like:
```



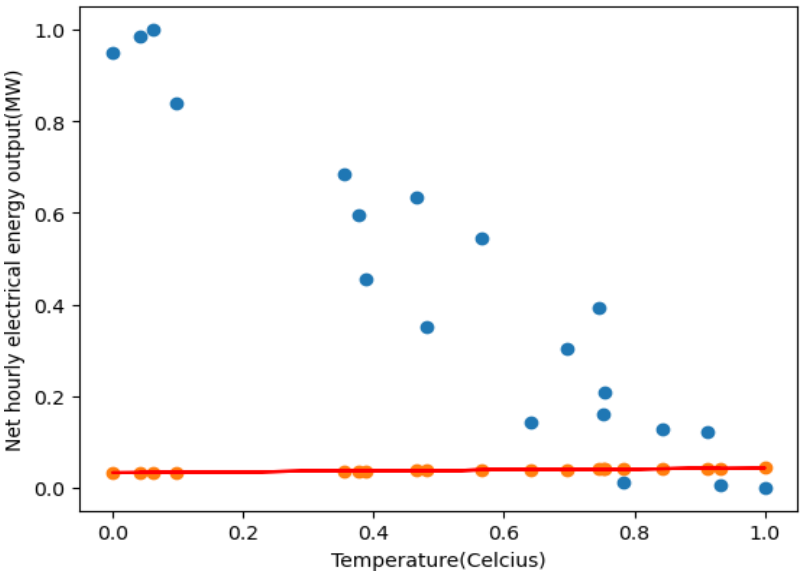
6. **Linear Regression with Gradient Descent – Second iteration**
   After second iteration of gradient descent values of m and b obtained are m = 0.00529546614142996 and b = 0.016813801517193503

```
After second iteration m 0.00777553724705514, b 0.02492653096740555, cost 0.2
7431632575166026
After second iteration graph of regression line looks like:
```



7. **Linear Regression with Gradient Descent – Third iteration**
   After third iteration values of m and b becomes m = 0.00777553724705514
   And b = 0.02492653096740555

```
After third iteration m 0.010147838205655957, b 0.03284998436616708, cost 0.2
672097537207561
After third iteration graph of regression line looks like:
```

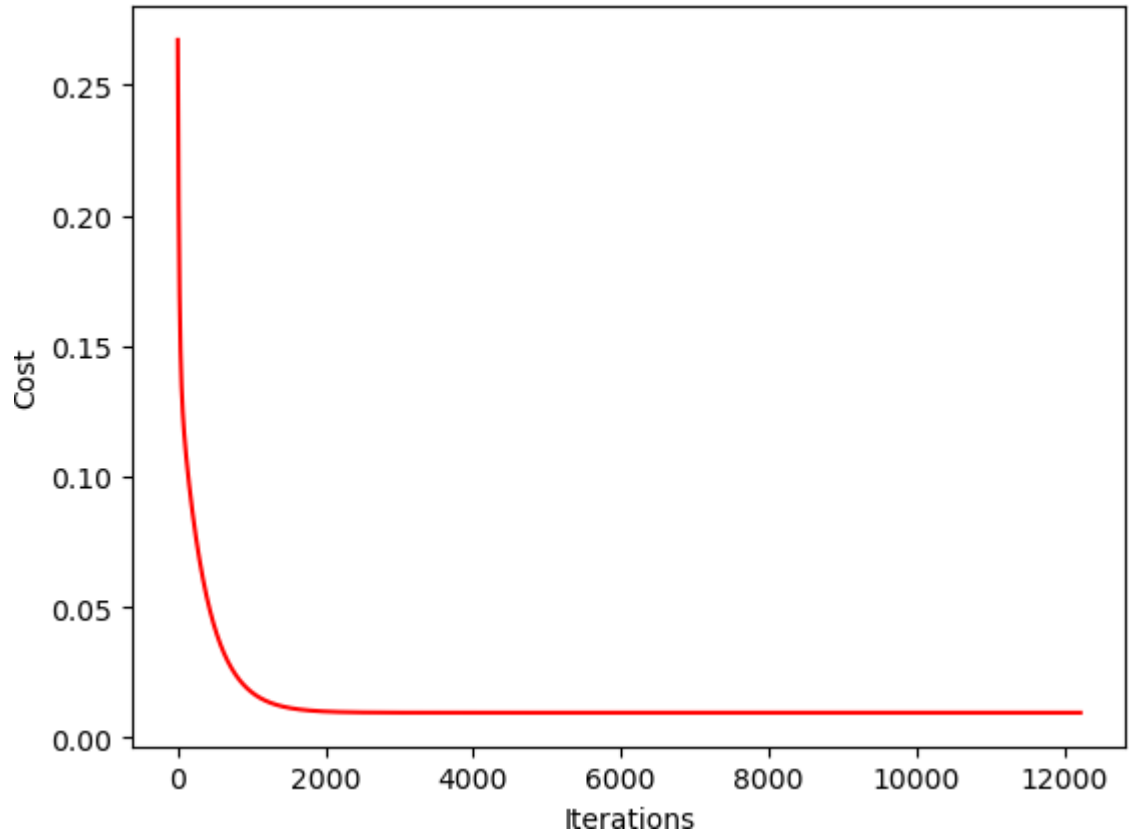


8. Linear Regression with Gradient Descent – Last iteration

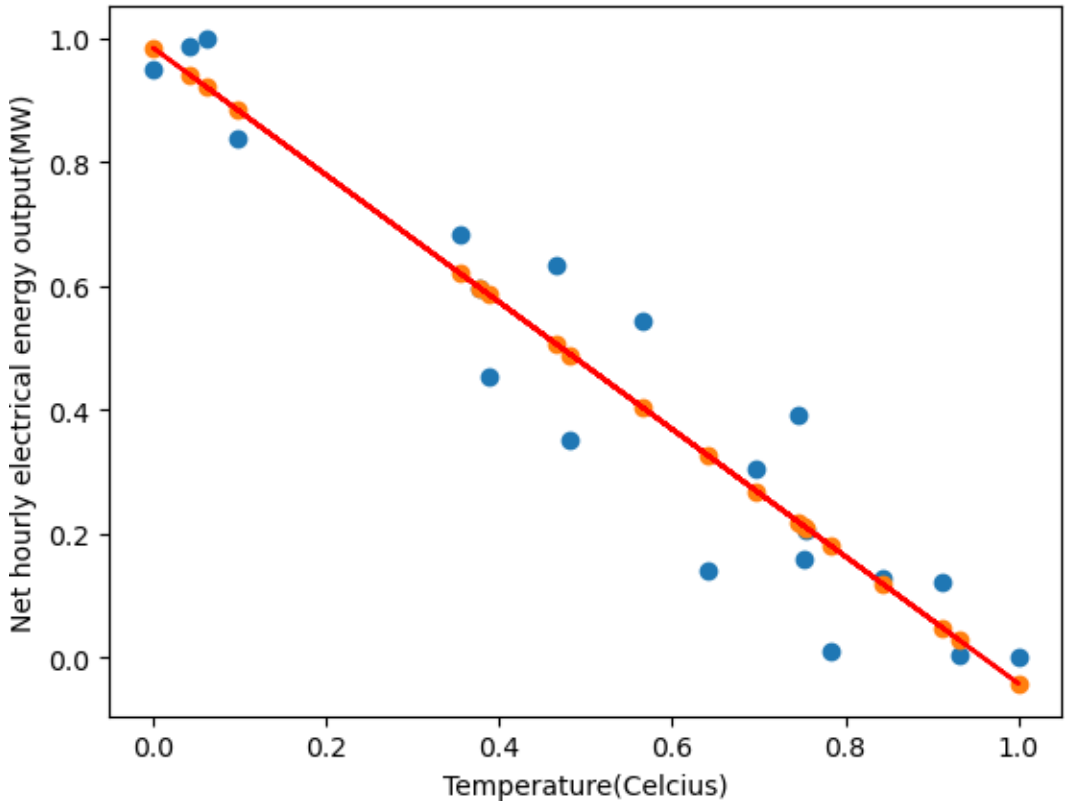
We continue to perform gradient descent for our data till the cost becomes almost constant. I n our code we have performed 12220 iterations. At this point we observe that our cost is min imum and has become constant. After our last iteration we get values of m and b as m = -1.0 273956034266931 and b = 0.9850313795392164. Output of last few iterations is as follows:

m -1.0273956029084692, b 0.985031379235272, cost 0.009629323175596102, iterations 12208
m -1.0273956029559168, b 0.9850313792631005, cost 0.009629323175596099, iterations 12209
m -1.0273956030032967, b 0.9850313792908894, cost 0.009629323175596099, iterations 12210
m -1.0273956030506093, b 0.9850313793186387, cost 0.009629323175596095, iterations 12211
m -1.0273956030978544, b 0.9850313793463485, cost 0.009629323175596095, iterations 12212
m -1.0273956031450324, b 0.985031379374019, cost 0.009629323175596097, iterations 12213
m -1.0273956031921432, b 0.98503137940165, cost 0.009629323175596095, iterations 12214
m -1.0273956032391869, b 0.9850313794292417, cost 0.009629323175596095, iterations 12215
m -1.0273956032861637, b 0.9850313794567941, cost 0.009629323175596095, iterations 12216
m -1.0273956033330736, b 0.9850313794843073, cost 0.009629323175596095, iterations 12217
m -1.0273956033799168, b 0.9850313795117814, cost 0.009629323175596097, iterations 12218
m -1.0273956034266931, b 0.9850313795392164, cost 0.009629323175596095, iterations 12219

Cost at each iteration:



After last iteration regression line looks like:



9. Discussion:

After comparing performances of both Least Squares and Gradient descent solutions we can say that both methods work towards single of finding best fitted line for dataset.
Least square method is a closed-form solution and it will always find best fitted solution when it exists. It directly obtains the regression parameters and is computationally more efficient when feature dimension is small.
Whereas, gradient descent on the other hand is iterative method and it takes small steps to reach optimum solution for best fitted line. However it is useful for wide range of solutions where datasets are very large(feature dimension is large). Setting the learning rate and number of iterations certainly affects it's overall performance.
Let's look at some similarities and differences between them.
Similarities:
Both methods can be used to find best fitted line for dataset i.e. both can be used for linear regression. Both methods work towards one goal that is finding optimum solution by minimizing a cost function.

Differences:

Least Squares	**Linear regression optimized with Gradient descent**
It's a closed-form solution.	It's an iterative method to find optimum solution.
It is more suitable for simple linear regression with small feature dimension.	It is suitable when dataset is large and with large feature dimension.

Computationally efficient when data is linearly distributed.	It is more versatile to find solutions for no linear data distribution as well.
This method directly finds solution when linear equation exists.	It needs initializing of learning rate and number of iterations to reach optimum solution.

A.II. Polynomial regression

1. **Cost function:**

J(theta) = (1/4n)* sum((y$_i$ – h$_{theta}$(x$_i$)^4)

i = iterates over all data points n
x_i = input of a data point i
y_i = true output of data point i
Regression model: $h_{theta}(x) = theta_2*x^2 + theta_1*x + theta_0$
Where $theta_2$, $theta_1$, and $theta_0$ are model parameters.

Partial derivative of cost function with respect to $theta_0$:

$d(J(theta))/dtheta_0 = (1/4n)* sum(d((y_i – h_{theta}(x_i))^4)/ dtheta_0)$

$= (4/4n)* sum((y_i – h_{theta}(x_i))^3)*(d((y_i – (theta_2*x^2 + theta_1*x + theta_0)))/ dtheta_0))$

$= (1/n)*sum((y_i – h_{theta}(x_i))^3)*(0-0-0-1)$

d(J(theta))/dtheta$_0$ = (-1/n)*sum((y$_i$ – h$_{theta}$(x$_i$)^3)

Partial derivative of cost function with respect to $theta_1$:

$d(J(theta))/dtheta_1 = (1/4n)* sum(d((y_i – h_{theta}(x_i)^4)/ dtheta_1)$

$= (4/4n)* sum((y_i – h_{theta}(x_i))^3)*(d((y_i – (theta_2*x_i^2 + theta_1*x_i + theta_0)))/ dtheta_1))$

$= (1/n)*sum((y_i – h_{theta}(x_i))^3)*(0-0-x_i-0)$

$= (-1/n)*sum((x_i)*(y_i – h_{theta}(x_i))^3)$

d(J(theta))/dtheta$_1$ = (-1/n)*sum((x$_i$)*(y$_i$ – h$_{theta}$(x$_i$))^3)

Partial derivative of cost function with respect to $theta_2$:

$d(J(theta))/dtheta_2 = (1/4n)* sum(d((y_i – h_{theta}(x_i)^4)/ dtheta_2)$

$= (4/4n)* sum((y_i – h_{theta}(x_i))^3)*(d((y_i – (theta_2*x_i^2 + theta_1*x_i + theta_0)))/ dtheta_2))$

$= (1/n)*sum((y_i – h_{theta}(x_i))^3)*(0-x_i^2-0-0)$

$= (-1/n)*sum((x_i^2)*(y_i – h_{theta}(x_i))^3)$

d(J(theta))/dtheta$_2$ = (-1/n)*sum((x$_i$^2)*(y$_i$ – h$_{theta}$(x$_i$))^3)

2. **Update rules for each parameters:**

   $theta_0 = theta_0 – learning\_rate * d(J(theta))/dtheta_0$

   $theta_1 = theta_1 – learning\_rate * d(J(theta))/dtheta_1$

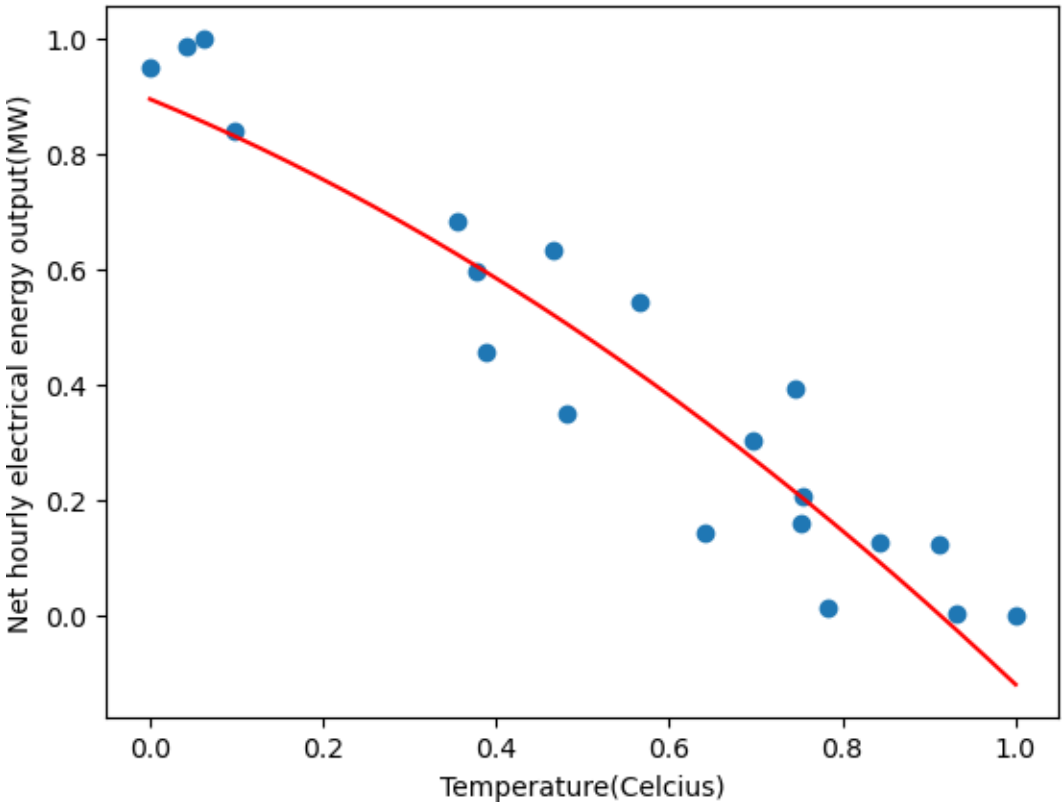
   $theta_2 = theta_2 – learning\_rate * d(J(theta))/dtheta_2$

3. **Initial values of model parameters and learning rate is**

   $theta_0 = 0.3$
   $theta_1 = theta_2 = 1$

   Learning rate = 0.07

After 30000 iterations polynomial plot looks like:



4. Similarities between linear regression and polynomial regression.

- Both linear regression and polynomial regression can be used for supervised learning where their goal is to predict dependent variable based on independent variable.
- They can be estimated using least squares method.
- Both models work on estimating parameters for their equations. In case of linear regression parameters are estimated for linear equation and in case of polynomial regression, parameters are estimated for polynomial regression.

Differences:

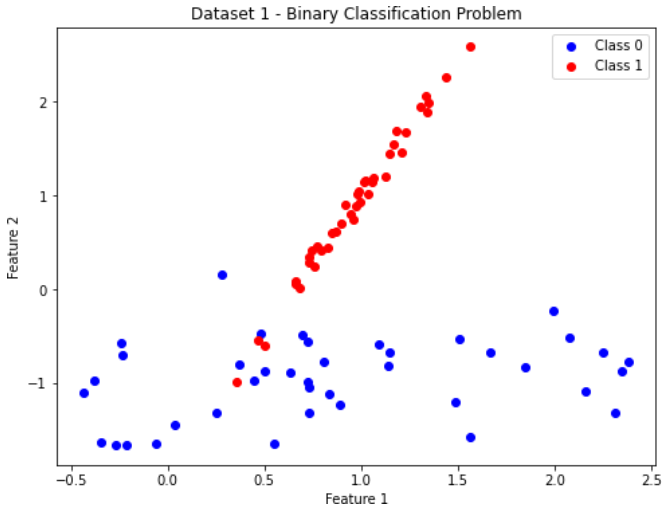
Linear regression	Polynomial regression
Most suitable for linear relationships	More suitable for non-linear relationships
It involves fitting a straight line	It is more complex and involves fitting a polynomial function
It computes faster	Requires more computational power
It is less affected by outliers	It is more sensitive to outliers.
Suitable for few data points	Suitable for many data points

B. Supervised Learning (SL) - Classification

#1.

graph of the data points obtained from make_classification function. Obtained the points and then graphed. the graph of the data points shown. We set random_state equal to 42 so we don't now what the distribution is be we are able to replicate it for consistency. We equally distribute

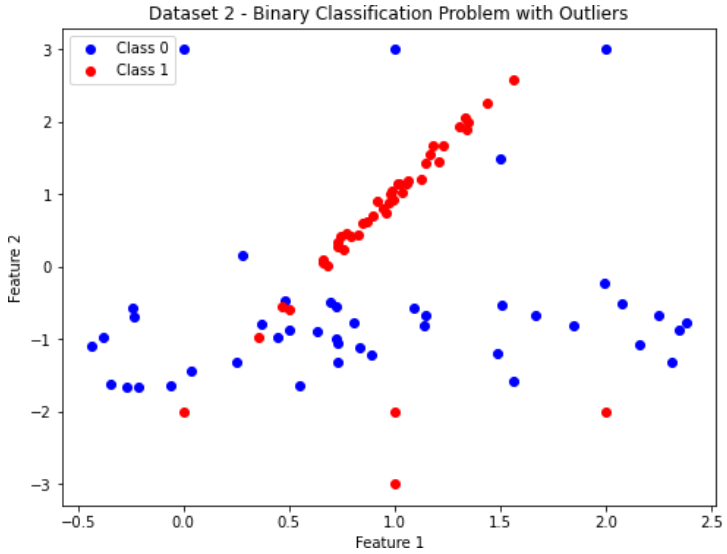
the data points so there are 40 datapoints for one class and 40 datapoints belonging to the other class. Looking at the datapoints printed out on a graph, we see class 0 being spread out more so then class 1, and class 1 having its points clustered like a line.



Graph 1. Dataset 1 graphed for 2 classes.

#2.

for my approach I want to add some outlier points that go over the clusters to areas where they skew the clusters and would allow for some readjustment in the data. Dataset 2 was created by adding four outliers to dataset 1 for the class 0 and class 1, detailed below for the graph. To ensure some difference in the methods used the outliers were added away from their main class clusters from looking at them. This is to ensure see the use of the methods for varying degrees of methods.



Graph 2: dataset 2 graphed for 2 classes.

3.

data is split along the points of training and testing. we want it to be 80%-20% training and testing data. as training data requires more space than testing data. we randomly sample the testing and training data. We randomly sample individually for the dataset 1 and also individually for dataset 2 determining which the testing and training data is.

#4.

Dataset 1

	KNN	Naive Baye	Decision tree classifier	random forest
Classification accuracy on Training data	0.96875	0.95	1.0	0.96875
Classification accuracy on Testing data	0.9375	0.94	0.875	0.9375
Confusion matrix	[[9 0] [1 6]]	[[9 0] [1 6]]	[[8 1] [1 6]]	[[9 0] [1 6]]

Dataset 2

	KNN	Naive Baye	Decision tree classifier	random forest
Classification accuracy on Training data	0.9402985074626866	0.90	0.9701492537313433	0.9402985074626866
Classification accuracy on Testing data	1.0	0.94	0.9411764705882353	1.0
Confusion matrix	[[8 0] [0 9]]	[[7 1] [0 9]]	[[7 1] [0 9]]	[[9 0] [1 6]]

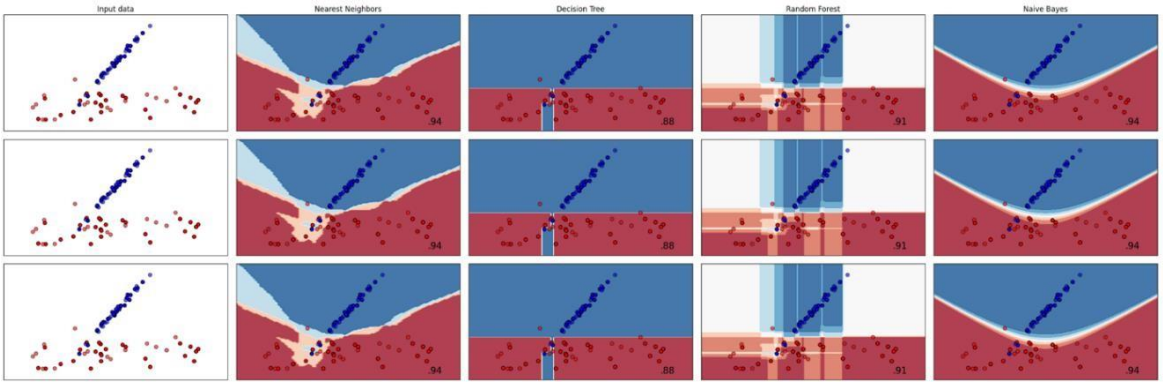
Looking at these results. Classification accuracy relates to the percentage of correct classification for the training dataset, indicating how well the model is trained on. Higher classification is better indicating good pattern recognition. The confusion matrix, describes the performance of the classification model on the test data set. This helps us evaluate the performance of the classification model, showing the number correct and incorrect predictions for the true or false, positive and negative.

Looking at datasets 1 and 2. They all have good accuracy for the training and testing. Knn, has high accuracy for training and testing with some miscalculation in the confusion matrix but

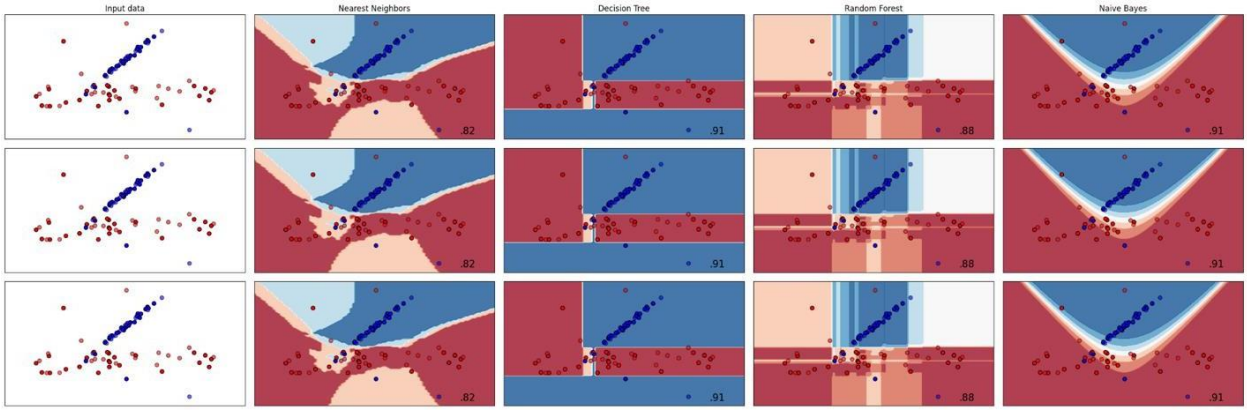
overall good. Naïve Bayes has good performance for testing and training. With good confusion matrix accuracy. Decision tree classifier has perfect accuracy for training possible overfitting to the data. With the confusion matrix having higher then the other miscalculation but not by much. Random forest has high accuracy for both. With confusion matrix suggesting miscalculation. All these models have higher classification accuracy for training then testing. Indicating overfitting on the training data.

For dataset 2. Main difference with the data is the inclusion of the outliers, potentially decreasing the accuracy of the model. Generally, has less classification accuracy for the training then dataset 1, but higher classification accuracy for the testing data. In comparison to dataset 1, dataset 2 has higher classification accuracy for the testing then the training. This indicates that the model is not fully being able to training itself on the training dataset.

#5. inserted screen shots



Unmodified dataset, Dataset 1.



Modified dataset with outliers, dataset 2.

Both decision boundaries of the various classification models used

6. similarities and dissimilarities, comparing the decision boundaries nearest neighbor and naive bayes were very similar when it comes to boundary curves. Being more like multiple curved lines for the different boundaries of datasets.

decision tree and random forest were similar as well when it came to boundaries with them being straight lines as it defines itself by boundaries.

K nearest neighbor KNN: the simplest algorithm in terms of complexity, does not need much for it to make assumptions about the data. it is very sensitive to the local structure of the data, that is why it was unsure of the outliers in the plots. sensitive to the k-nearest neighbors.

Naive Bayes: estimating the joint probability is not practically as it overfits, it is fast to train and fit. It is not sensitive to outliers as much as Knn. works wells with simple data yes or no. disadvantage is that it assumes independent predictors not working well with complex assumptions. it has layers of uncertainty with points meaning it can be unsure when points clusters intersect as well as more complex models.

Decision tree: A simpler version of the random forest this. basic as its able to. it makes its decision boundaries out of straight lines, for areas of points. it may lead to overfitting; it is prone to small variating points like the outliers. like knn having simply boundaries for its boundary lines dissimilarly to random forest and naive bayes it is not very complex in comparison to them.

Random forest: are a compilation of decision trees making it more complex than a decision tree, giving less overfitting, being able to handle more features for decision boundaries. This allows it to be more accurate with its decision bound but this means that it can make decision bounds that are hard to read and more overley complex.

C. Unsupervised Learning (UL) - free choice study

Mall Customer Segmentation Dataset using Clustering Algorithms

About The Dataset

This dataset consists of the customer segmentation concepts known as market basket analysis. I will demonstrate this by using an unsupervised ML technique (KMeans Clustering and DBSCAN Algorithm) in the simplest form.

Link to the Dataset (https://www.kaggle.com/datasets/kandij/mall-customers/data).

Content

Imagine owning a supermarket mall and through membership cards, you have some basic data about your customers like Customer ID, age, gender, annual income, and spending score. Spending Score is something you assign to the customer based on your defined parameters like customer behavior and purchasing data.

Problem Statement
You own the mall and want to understand the customers who can easily converge [Target Customers] so that the sense can be given to the marketing team and plan the strategy accordingly.

CLUSTERING:
Clustering is a type of unsupervised machine learning in which the algorithm processes our data and divides them into “clusters”. Clustering algorithms try to find natural clusters in data, and the various aspects of how the algorithms cluster data can be tuned and modified. Clustering is based on the principle that items within the same cluster must be similar to each other. The data is grouped in such a way that related elements are close to each other.

K-Means Clustering
K-Means clustering is an unsupervised machine learning algorithm that divides the given data into a given number of clusters. Here, the “K” is the given number of predefined clusters that need to be created. It is a centroid-based algorithm in which each cluster is associated with a centroid. The main idea is to reduce the distance between the data points and their respective cluster centroids. The algorithm takes raw unlabeled data as an input and divides the dataset into clusters and the process is repeated until the best clusters are found.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
DBSCAN is a density-based clustering algorithm that groups data points based on their density. It defines clusters as dense regions separated by sparser areas. DBSCAN can discover clusters of arbitrary shapes, handle noisy data, and does not require specifying the number of clusters in advance. It classifies points as core, border, or noise based on density and connectivity.

Operations performed on the dataset
We first load the data into my data frame called **df**. By using **df.head()** check the first five rows.

```
df.rename(columns = {'Genre' : 'Gender',
                     'Spending Score (1-100)':'Spending_Score',
                     'Annual Income (k$)':'Annual_Income'}, inplace=True)
```

```
df.head()
```

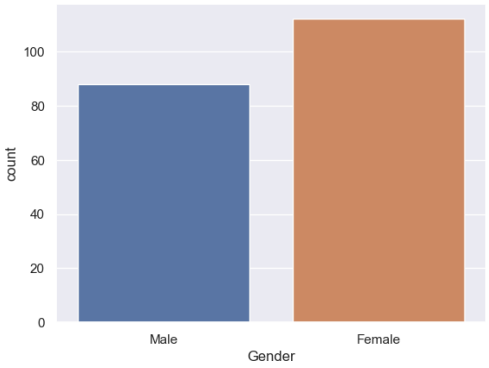
	CustomerID	Gender	Age	Annual_Income	Spending_Score
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
df.describe()
```

	CustomerID	Age	Annual_Income	Spending_Score
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

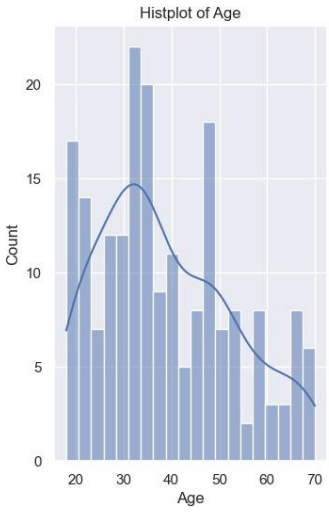
After which we check for null values. There are no null values present in the dataset and we drop the CustomerID column since it is not useful for our Clustering task.

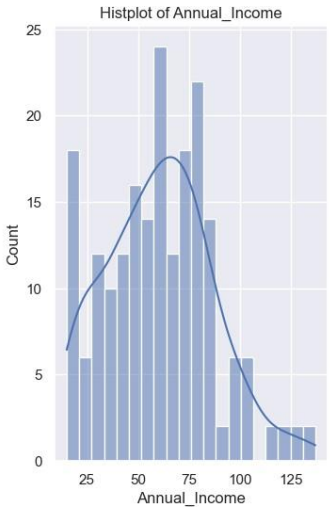
There are a total of 200 customers. We get to know that the number of female customers is higher than male customers and females spend more than males.

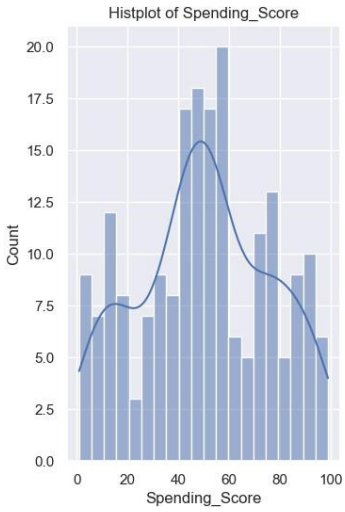


Gender	Spending_Score
Female	51.526786
Male	48.511364

From the graph given below, we can understand that most of the customers are between the age 25-35 years and have an annual income between 60(K$)-90(K$), and the majority of the spending score is between 40-60.







We have divided Age, Annual Income, and Spending Score each into five categories as given below:

```
age_18_25 = df.Age[(df.Age >= 18) & (df.Age <= 25)]
age_26_35 = df.Age[(df.Age >= 26) & (df.Age <= 35)]
age_36_45 = df.Age[(df.Age >= 36) & (df.Age <= 45)]
age_46_55 = df.Age[(df.Age >= 46) & (df.Age <= 55)]
age_55above = df.Age[df.Age >= 56]
```

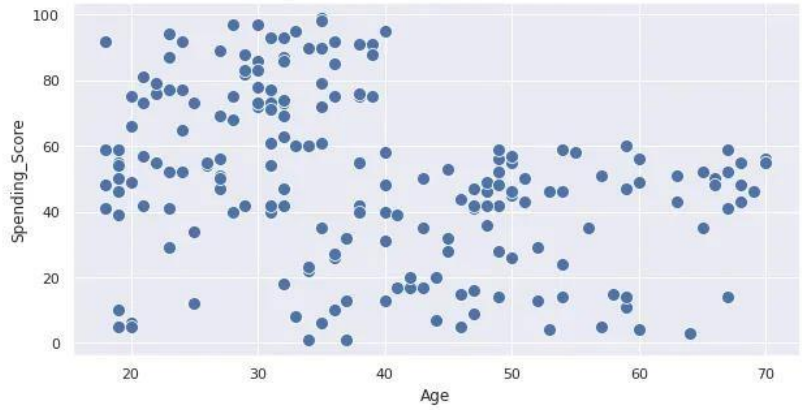
```
#Spending Scores
ss_1_20 = df["Spending_Score"][(df["Spending_Score"] >= 1) & (df["Spending_Score"] <= 20)]
ss_21_40 = df["Spending_Score"][(df["Spending_Score"] >= 21) & (df["Spending_Score"] <= 40)]
ss_41_60 = df["Spending_Score"][(df["Spending_Score"] >= 41) & (df["Spending_Score"] <= 60)]
ss_61_80 = df["Spending_Score"][(df["Spending_Score"] >= 61) & (df["Spending_Score"] <= 80)]
ss_81_100 = df["Spending_Score"][(df["Spending_Score"] >= 81) & (df["Spending_Score"] <= 100)]
```

```
#Annual Income
ai_0_30 = df["Annual_Income"][(df["Annual_Income"] >= 0) & (df["Annual_Income"] <= 30)]
ai_31_60 = df["Annual_Income"][(df["Annual_Income"] >= 31) & (df["Annual_Income"] <= 60)]
ai_61_90 = df["Annual_Income"][(df["Annual_Income"] >= 61) & (df["Annual_Income"] <= 90)]
ai_91_120 = df["Annual_Income"][(df["Annual_Income"] >= 91) & (df["Annual_Income"] <= 120)]
ai_121_150 = df["Annual_Income"][(df["Annual_Income"] >= 121) & (df["Annual_Income"] <= 150)]
```

From the scatter plot given below, we can tell that we can conclude that age depends on the spending of the customer. Younger customers tend to spend the most compared to the rest of the customers.

```
plt.figure(figsize=(10,5))
sns.scatterplot(x=df.Age,y=df.Spending_Score,s=100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f40724837d0>
```



Feature Selection

From the exploratory data analysis above, it is obvious that all the variables have some sort of relationship with spending score. We will be using all the variables to build the clustering models.

Drop unnecessary column.

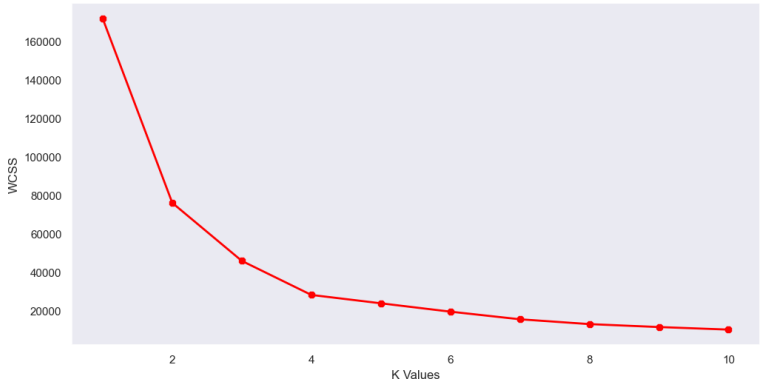
```
df.drop(["CustomerID"],axis=1,inplace=True)
```

KMeans Clustering

We first use **the Sum of Squares** method to find the optimal number of clusters.

The Sum of Squares (WCSS)
The Sum of Squares (WCSS) method in K-means clustering calculates the total sum of squared distances of data points to their respective cluster centroids. It helps evaluate the compactness of clusters and find the optimal number of clusters by minimizing this value.

Now we will find the optimal number of clusters using **The Sum of Squares method** for **Age** and **Spending Score**. From the graph given below, we can take the number of clusters to be as **4** since we can see the elbow point is at **4** ("elbow" is the point where the rate of decrease in WCSS sharply changes).



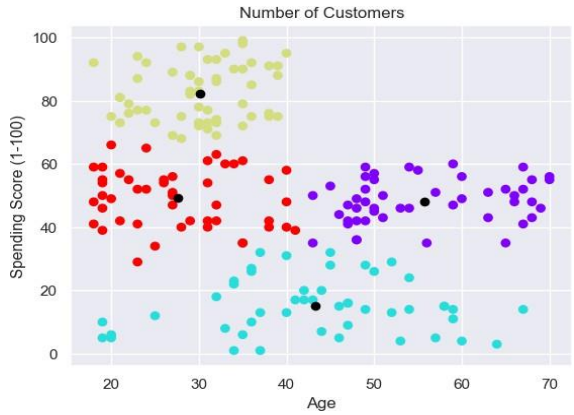
After performing K-means clustering on the data **X1** with **4** clusters. We print the cluster labels for each data point, indicating which cluster they are assigned to. Following this we print the centroids of each cluster.

```
[3 2 1 2 3 2 1 2 1 2 1 2 1 2 1 2 3 3 1 2 3 2 1 2 1 2 1 3 1 2 1 2 1 2 1
 2 1 2 0 2 0 3 1 3 0 3 3 3 0 3 3 0 0 0 0 0 3 0 0 3 0 0 0 3 0 0 3 0 0 0 0
 0 3 0 3 3 0 0 3 0 0 3 0 0 3 3 0 0 3 0 3 3 3 0 3 0 3 3 0 0 3 0 3 0 0 0 0
 3 3 3 3 3 0 0 0 0 3 3 3 2 3 2 0 2 1 2 1 2 3 2 1 2 1 2 1 2 1 2 3 2 1 2 0 2
 1 2 1 2 1 2 1 2 1 2 1 2 0 2 1 2 1 2 1 2 1 3 1 2 1 2 1 2 1 2 1 2 1 2 1 2 3
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2]
```

```
print(kmeans.cluster_centers_)
```

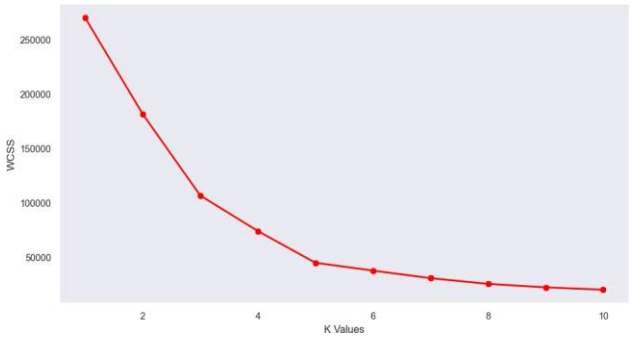
```
[[55.70833333 48.22916667]
 [43.29166667 15.02083333]
 [30.1754386  82.35087719]
 [27.61702128 49.14893617]]
```

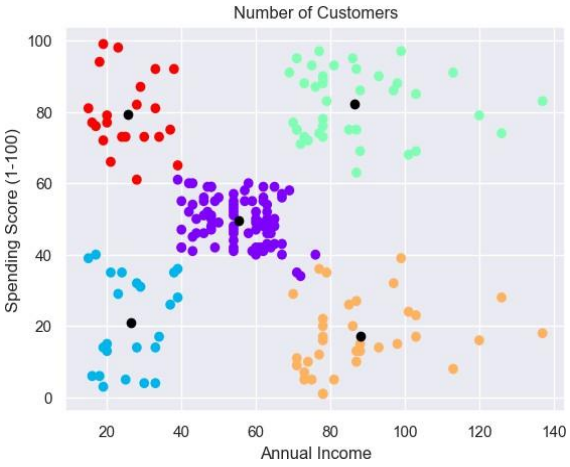
From the scatter plot given below, we can see that the customers have been grouped into **4** clusters based on their similarities.

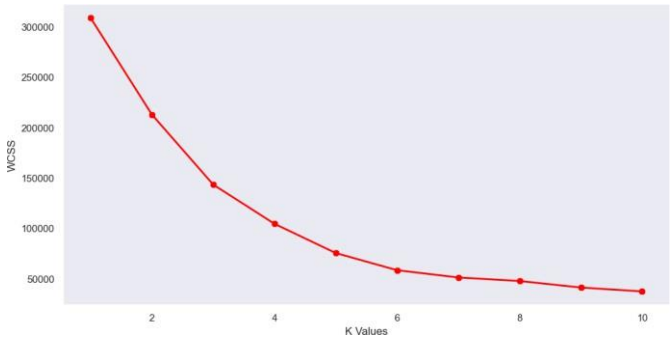


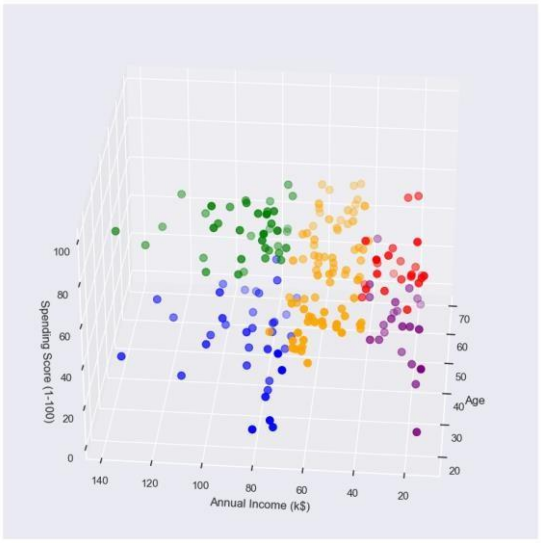
The same method is repeated to find the number of clusters for the **Annual Income** and **Spending Score**. From the graph given below, we can take the number of clusters to be **5** since we can see the elbow point is at **5**. After which we print its labels and centroids. The same method is repeated to show the relation between **Age, Annual Income, and Spending Score**

simultaneously in the form of a 3-dimensional graph where its number of clusters is also **5** which is obtained through **The Sum of Squares method.**







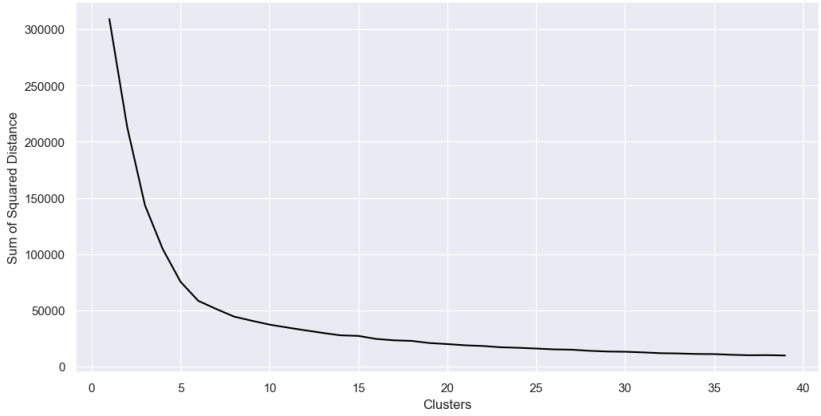


Feature Transformation

Since the Gender column datatype is categorical, then we must convert it into a numerical datatype using one hot encoding(pandas.get_dummies). **Male = True (1)** and **Female = False (0)**.

	Age	Annual_Income	Spending_Score	label	Gender_Male
0	19	15	39	4	True
1	21	15	81	1	True
2	20	16	6	4	False
3	23	16	77	1	False
4	31	17	40	4	False

Now let's initialize the number of clusters to be **5** and then find the number of clusters through the elbow method where we get **clusters = 5**.



Now let's plot the graph simultaneously for **Age vs. Spending Score, Annual Income vs. Spending Score, and Gender vs. Spending Score** where **Male = True (1)** and **Female = False (0)**.

From the plots above, it is obvious that Age is the most important factor in determining Spending Score. Irrespective of their Annual income, younger people tend to spend more.

DBSCAN Clustering Model
To find clusters using DBSCAN:

- Define your data points **X**.
- Create a DBSCAN model with **eps** and **min_samples** parameters.
- Fit the model to your data with **model.fit(X)**.
- Get cluster labels using **model.labels_**.

Initially, we set the number of clusters to 5 and then calculate the number of clusters using the algorithm.

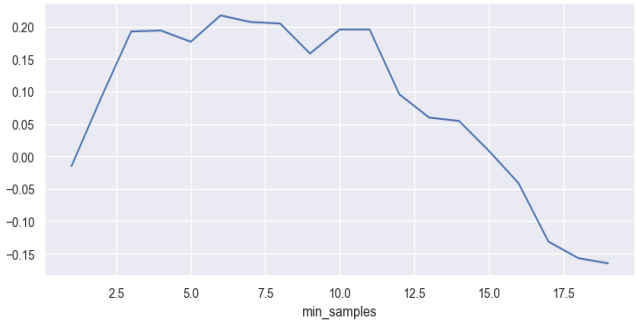
```
dbscan = DBSCAN(eps=5, min_samples=5)
db_labels = dbscan.fit_predict(df)
```

```
no_clusters = len(set(db_labels)) - (1 if -1 in db_labels else 0)
print('There are {} clusters'.format(no_clusters))
```

```
There are 4 clusters
```

To find the best values for **eps** and **min_samples** we will have to experiment with different values, but we can also use “for loop” to find **min_samples** as shown in the code below which will generate the following graph.

```
from sklearn import metrics
sh_score = []
for num in range(1,20):
    dbscan=DBSCAN(eps=12, min_samples=num)
    model=dbscan.fit(df)
    d_labels=model.labels_
    score = metrics.silhouette_score(df,d_labels)
    sh_score.append(score)
plt.figure(figsize=(10,4))
plt.plot(list(range(1,20)), sh_score)
plt.xlabel('min_samples')
plt.show()
```



Now let's plot the graph simultaneously for **Age vs. Spending Score, Annual Income vs. Spending Score, and Gender vs. Spending Score** where **Male = True (1)** and **Female = False (0)**.

Similar to KMeans, DBSCAN also shows that Age is the most important factor to consider.

Models Performance and Evaluation Using Silhouette Coefficient

The Silhouette Coefficient is a metric used to evaluate the quality of clusters created by clustering algorithms, such as K-means or DBSCAN. It measures how similar an object is to its cluster compared to other clusters. The Silhouette Coefficient ranges from -1 to 1, where:

- Values close to 1 indicate that the object is well-matched to its cluster and poorly matched to neighboring clusters.
- Values close to 0 indicate that the object is on or very close to the decision boundary between two neighboring clusters.
- Values close to -1 indicate that the object is poorly matched to its cluster and well matched to a neighboring cluster.

For KMeans

```
coef = metrics.silhouette_score (df,k_labels)

print('The sihoutte score is {}'.format(coef))
```

```
The sihoutte score is 0.44474910239085313
```

For DBSCAN

```
coef = metrics.silhouette_score (df,d_labels)

print('The sihoutte score is {}'.format(coef))
```

```
The sihoutte score is -0.16497526497583068
```

Hence, KMeans Clustering performs better than DBSCAN Clustering since the silhouette score is greater for KMeans compared to DBSCAN.

Conclusion and Impact of Clustering Analysis on Business Decision-Making

- Age plays an important factor in clustering the data.
- From the analysis, we saw that younger people aged between 20 to 40 patronize the product(s)/service(s) more than older people.
- The business should target Ads on this population, as they will get a higher turnover and conversion rate.
- It is also seen that the number of female customers is higher than the number of male customers, and their spending score is comparatively higher even if their annual income is less than 50(k$).

LIMITATIONS

- One of the biggest issues with customer segmentation is data quality. Inaccurate data in source systems will usually result in poor grouping. For example, for customers who are individuals, attributes like age, gender, and marital status are frequently used.
- Customers can belong to multiple segments which can make clustering difficult.
- Segmentation only works when segments are clearly defined and are distinct from one another.
- K-means clustering is useful, but it has its limitations. It can be sensitive to the initial guess, outliers can impact the results, it assumes round clusters, we need to know the number of clusters in advance, and it may face challenges with large datasets.
- DBSCAN It is sensitive to the choice of parameters (eps and min_samples). It may not perform well with clusters of varying densities or with high-dimensional data. The algorithm's performance may degrade as the size of the dataset increases.

FUTURE IMPROVEMENTS

- By enabling companies to target specific groups of customers, a customer segmentation model allows for the effective allocation of marketing resources and the maximization of cross- and up-selling opportunities.
- When a group of customers is sent personalized messages as part of a marketing mix that is designed around their needs, it's easier for companies to send those customers special offers meant to encourage them to buy more products. Customer segmentation can also improve customer service and increase customer loyalty and retention.

- As a by-product of its personalized nature, marketing materials sent out using customer segmentation tend to be more valued and appreciated by the customer who receives them as opposed to impersonal brand messaging that doesn't acknowledge purchase history or any kind of customer relationship.
- Other benefits of customer segmentation include staying a step ahead of competitors in specific sections of the market and identifying new products that existing or potential customers could be interested in or improving products to meet customer expectations.

REFERENCES

- **Link to the dataset https://www.kaggle.com/datasets/kandij/mall-customers/data**
- **T. Sajana, C. M. Sheela Rani and K. V. Narayana "A Survey on Clustering Techniques for Big Data Mining", Indian Journal of Science and Technology, Volume 9, Issue 3, Jan 2016.**
- **https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1**
- **https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/#:~:text=DBSCAN%20is%20a%20density%2Dbased,points%20into%20a%20single%20cluster.**