Data Science / Machine Learning

Capstone Project

CONTENT LIST:

➢ Problem Statement:

- The problem statement in hand is a file containing a variety of attributes of several systems from a datacenter. The goal is to partition this data and find meaningful data clusters that can give us an insight into how the data is structured.

- To yield accurate grouping, data pre-processing will be an important step- leaving no room for discrepancies, and the usage of appropriate models to result in well defined clusters of data will help us better understand the relationships and patterns within the data.

➢ Planning:

- After going through the dataset - which has 15 columns and 500 values, I came to the conclusion that the dataset had some places where corrections were in order, to make its structure more presentable and proper so that fitting it into the model won't be a problem.

- After that comes the data analysis, where we can gain knowledge of the relationships between the different attributes, using graphs. This will not only help me gain deeper insight into the way the data is structured, but can also help me in making critical decisions regarding the data that can be valuable for effective clustering.

- There was also the decision of which algorithm I should use for effective clustering. After some research and thought, I decided that I would use the K-Means Algorithm and use validation methods to verify the effectiveness of clustering.

➢ Exploratory Data Analysis:

❖ Analysis of Dataset:

- First, we must analyze the properties of our dataset. We will begin by importing the necessary modules, and creating a dataframe out of our dataset.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn.preprocessing import LabelEncoder
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from kneed import KneeLocator
import matplotlib.pyplot as plt

dataframe = pd.read_csv("DataCenter.csv")

print(dataframe.columns,'\n')
```

- This will give us the columns that are present in the dataset. They are as follows.

```
Index(['IP Address', 'Processor_Name', 'CPU_Speed', 'CPU_Core', 'CPU_MThread',
       'CPU_Utilization', 'RAM_Size_GB', 'RAM_Utilization', 'Storage_Size_GB',
       'Storage_Used_Size_GB', 'Evn_Type', 'OS_Name', 'OS_Version_Name',
       'SubNet', 'Server Type'],
      dtype='object')
```

- We will now analyze the different data types of these columns.

```
print(dataframe.info())
```

- Output:

```
Data columns (total 15 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   IP Address            500 non-null    object
 1   Processor_Name        500 non-null    object
 2   CPU_Speed             500 non-null    object
 3   CPU_Core              500 non-null    object
 4   CPU_MThread           500 non-null    int64
 5   CPU_Utilization       491 non-null    float64
 6   RAM_Size_GB           500 non-null    object
 7   RAM_Utilization       493 non-null    float64
 8   Storage_Size_GB       500 non-null    int64
 9   Storage_Used_Size_GB  494 non-null    float64
 10  Evn_Type              500 non-null    object
 11  OS_Name               500 non-null    object
 12  OS_Version_Name       500 non-null    object
 13  SubNet                500 non-null    object
 14  Server Type           500 non-null    object
```

- This means that not only are most of the columns object data types (strings, usually), it also shows us that there is some missing data that needs to be treated.

- Upon further investigation of the dataframe, it is apparent that metrics like `RAM_Size_GB, CPU_Core, CPU_Speed` are actually valid to be int/float values, if we remove parts of their features. For example, from the `RAM_Size_GB` Column, removing "GB" from the string, allows us to convert, say, a value like "12 GB" to 12, an integer value.

```
dataframe["RAM_Size_GB"]=dataframe["RAM_Size_GB"].str.replace("GB",'')
dataframe["RAM_Size_GB"]=dataframe["RAM_Size_GB"].astype(int)



dataframe["CPU_Speed"]=dataframe["CPU_Speed"].str.replace("GHz",'')
dataframe["CPU_Speed"]=dataframe["CPU_Speed"].astype(float)
# This column for the most parts has obj vals that can be changed to int, but has one value
"Eight Core" that needs to be looked into.
```

```
# We need to change this before we can change the whole column
dataframe.loc[dataframe['CPU_Core'] == 'Eight Core', 'CPU_Core'] = '8'
dataframe["CPU_Core"]=dataframe["CPU_Core"].astype(int)
```

- Now, we have 3 additional numerical columns.
- We will now find the percentage of values that were null in the dataset.

```
# CPU_Utilization has 13 NaN Values, Storage_Used_Size_GB has 6 NaN Values, RAM_Utilization
has 8 NaN Values
print("\nNull Values:")
print(f'{dataframe.CPU_Utilization.isnull().sum()*100/dataframe.CPU_Utilization.shape[0]}% of
CPU_Utilization values are Null.')
print(f'{dataframe.Storage_Used_Size_GB.isnull().sum()*100/dataframe.Storage_Used_Size_GB.shap
e[0]}% of Storage_Used_Size_GB values are Null.')
print(f'{dataframe.RAM_Utilization.isnull().sum()*100/dataframe.RAM_Utilization.shape[0]}% of
RAM_Utilization values are Null.\n')
```

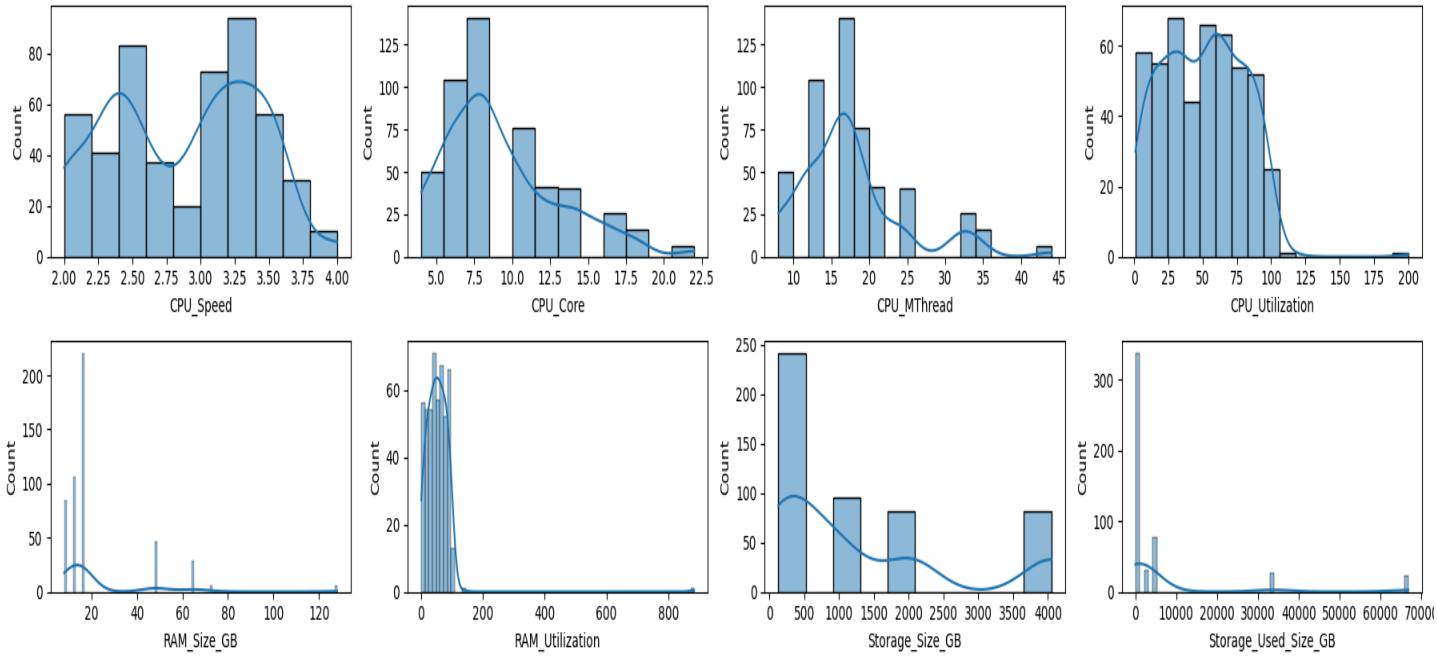
- Output:

```
Null Values:
2.6% of CPU_Utilization values are Null.
1.2% of Storage_Used_Size_GB values are Null.
1.6% of RAM_Utilization values are Null.
```

- In tech features like ram size, utilization, etc, 0 is not a possible value. Hence, we will address this as well, by converting all the zeros to nulls, and then filling the null values.

```
def replace_with_nan(dataframe):
    dataframe.replace({'CPU_MThread': 0, 'CPU_Utilization': 0, 'RAM_Utilization': 0,
                       'CPU_Speed':0, 'CPU_Core': 0, 'Storage_Size_GB': 0,
                       'Storage_Used_Size_GB': 0, 'RAM_Size_GB':0
                    }, np.nan,
                    inplace = True)
    print("0 replaced with null !!\n")
    return dataframe
#Replace Zeros with NaN to identify ALL Null values
replace_with_nan(dataframe)
```

- Then, we plot histograms for these numerical columns.



Histograms

- We will be using median() to fill the null values, since the data is skewed.

```
# Fills NaN Values With median
def fill_nan(dataframe, columns):
    dataframe[columns] = dataframe[columns].fillna(dataframe[columns].median())
    print("Null Values Fixed! \n")
    return dataframe
dataframe = fill_nan(dataframe, ["CPU_Utilization","RAM_Utilization","Storage_Used_Size_GB"])
print(f'Null Values: \n{dataframe.isnull().sum()}')
```

```
Null Values:
IP Address              0
Processor_Name          0
CPU_Speed               0
CPU_Core                0
CPU_MThread             0
CPU_Utilization         0
```

```
RAM_Size_GB              0
RAM_Utilization          0
Storage_Size_GB          0
Storage_Used_Size_GB     0
Evn_Type                 0
OS_Name                  0
OS_Version_Name          0
SubNet                   0
Server Type              0
dtype: int64
```

- We will now plot the histogram after filling the null values

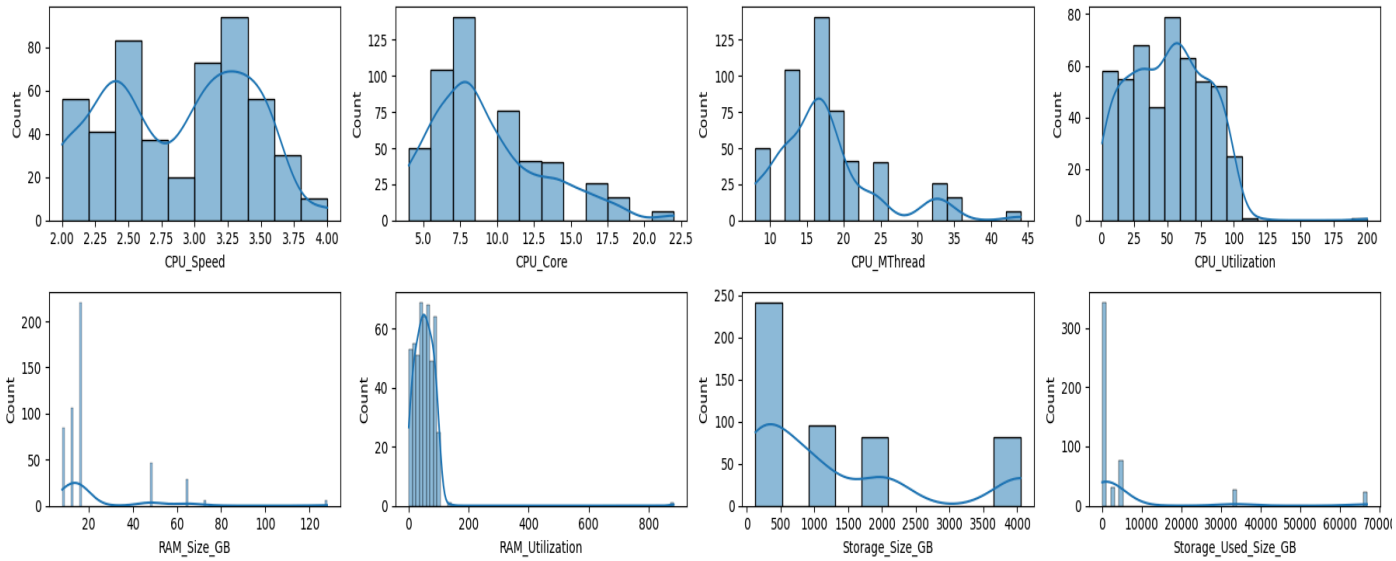


Figure 4.

- No significant change is noted, although some really minor changes are present in the histogram.

- So far, we have corrected the data type of some features, removed zeros, replacing them with NaN, and subsequently filled those NaN values with the median of the respective column.

- Now, we move on to the next part - Outlier detection and handling.

❖ Outlier Detection Using Boxplot:

- Outliers are extreme values that stand out greatly from the overall pattern of values in a dataset or graph. But sometimes, depending on the context, outliers need not be removed, rather can just be treated.

- We will visualize the outliers using a boxplot and make our decision.

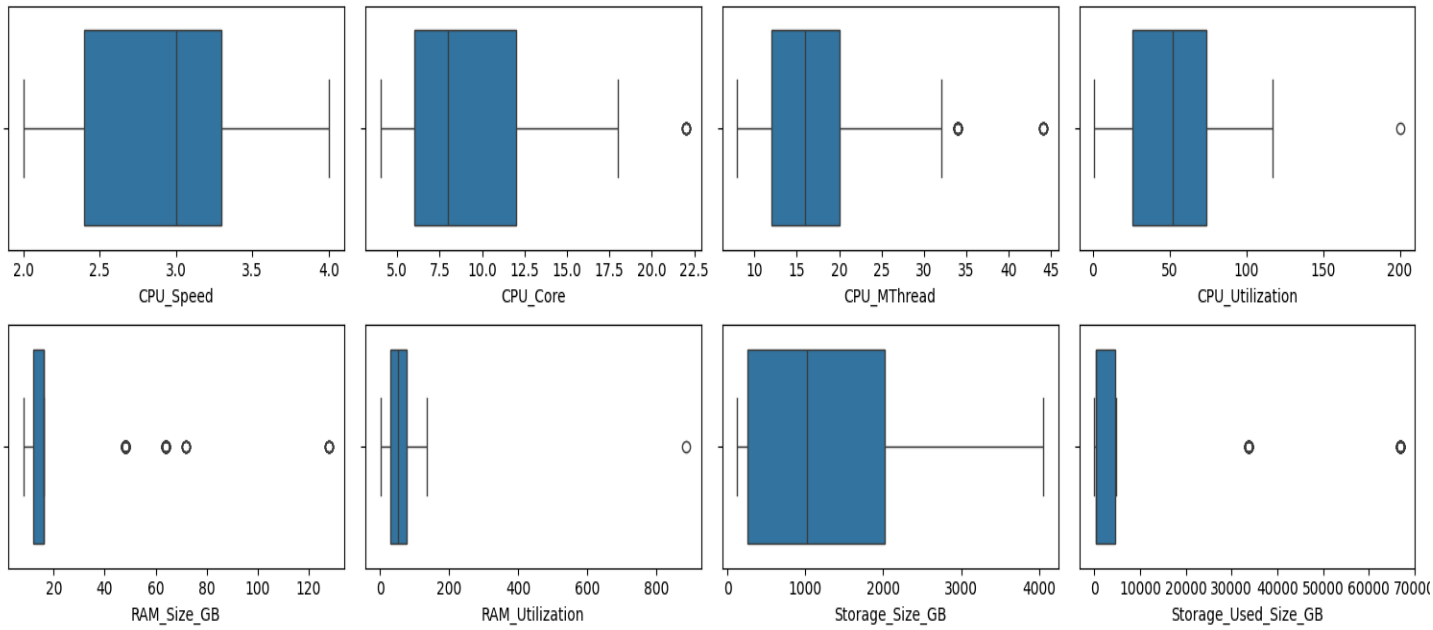


Figure 5.

- From the boxplot, we can see that there are outliers present in the data. We will remove them by calculating the difference between the 75th and 25th percentile, defining outlier boundaries (lower and upper), and removing data outside these bounds.

```
def irqmethod(dataframe, column):
    q1,q3 = dataframe[column].quantile(.25), dataframe[column].quantile(.75)
    irq = q3 - q1
    dataframe = dataframe[(dataframe[column] > q1 - 1.5*irq) & (dataframe[column] <q3+1.5*irq)]
    return dataframe
dataframe = irqmethod(dataframe,'CPU_Utilization')
dataframe = irqmethod(dataframe,'RAM_Utilization')
dataframe = irqmethod(dataframe, 'RAM_Size_GB')
dataframe = irqmethod(dataframe, 'Storage_Used_Size_GB')
dataframe = irqmethod(dataframe, "CPU_MThread")
```

- The Outliers have been successfully removed. Based on the silhouette scores and the cluster plots, a decision can be made later on whether the outliers can stay. But for now, we will work with no outliers.

- Correlation of data allows us to analyze the importance of features and their impact on the clustering model. The next step is to use heatmap() and pairplot() to figure out the correlations.

- **<u>Heatmap:</u>**

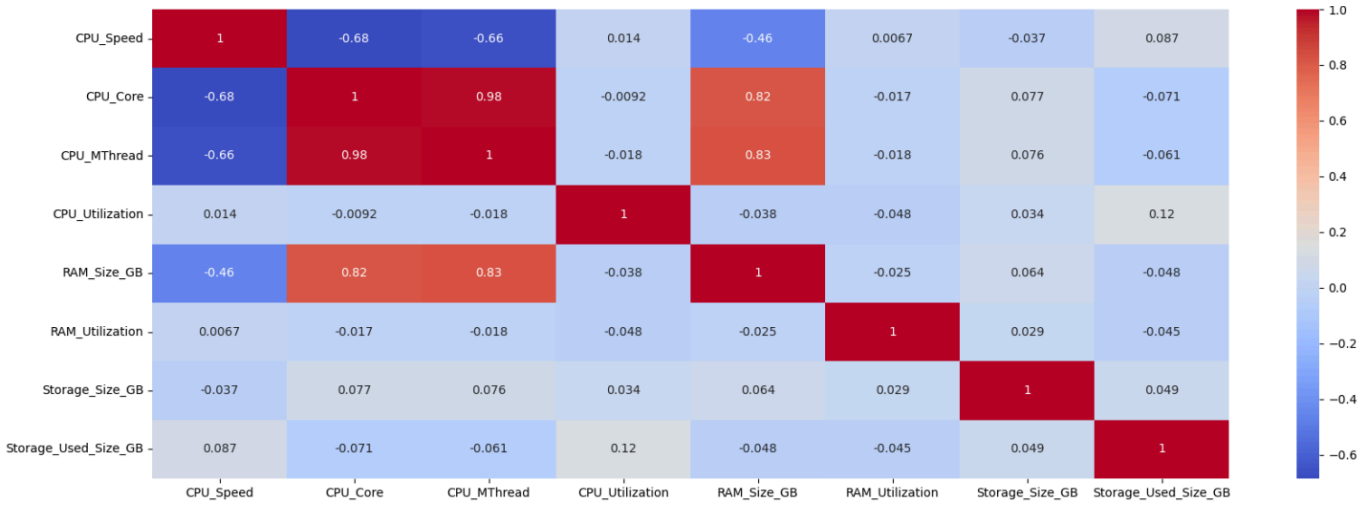


Figure 6.

- From the heatmap, it is clear that there are some features that have high correlation, and for the most part, correlation is low.
- Highly correlated features have a lot of data that are “common”, as in, they tell us the same thing. For clustering models, low correlation is preferred as it can lead to well separated clusters. Here are some highly correlated features that we can see from the heatmap:

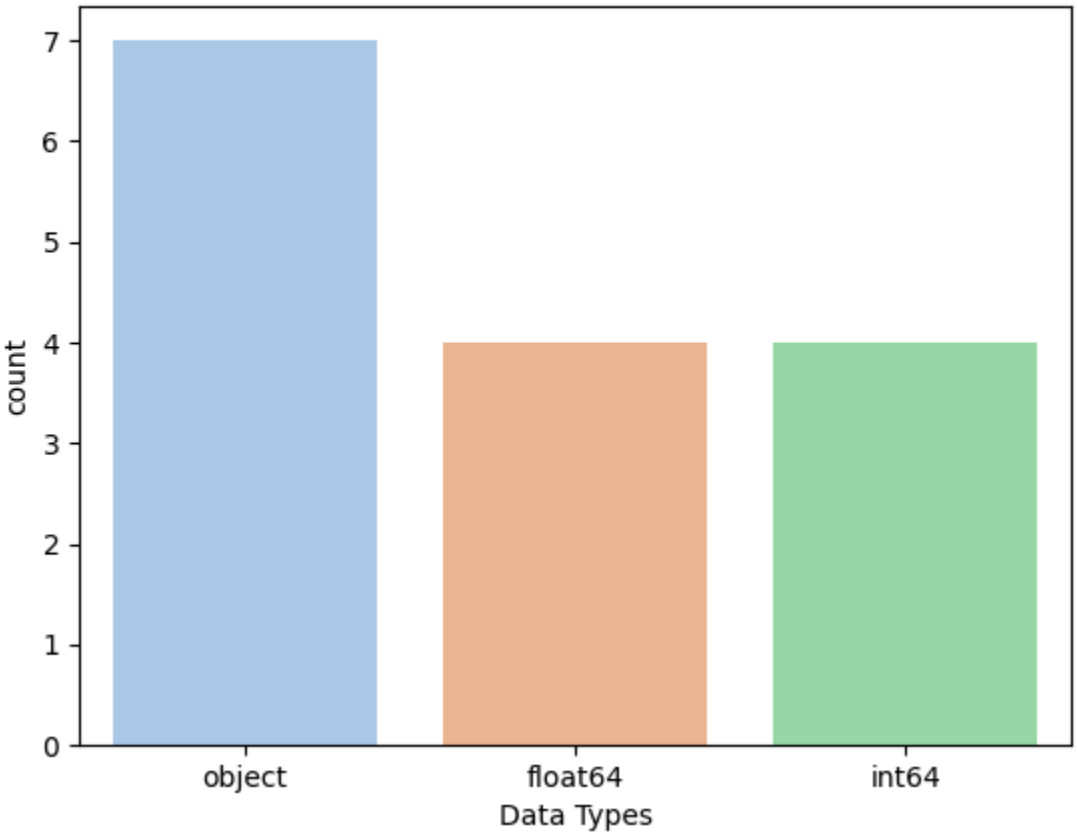
1. CPU_Core and CPU_MThread - 0.98
2. RAM_Size_GB and CPU_Core - 0.82
3. RAM_Size_GB and CPU_Mthread - 0.83

- Clustering using these features will provide us a cluster that is poorly defined. Out of all approaches one can use to prevent poorly clustered data, one approach that is simple would be to scale the features before clustering them. We will now use pairplot().

Pairplot to find out the relationship between features.

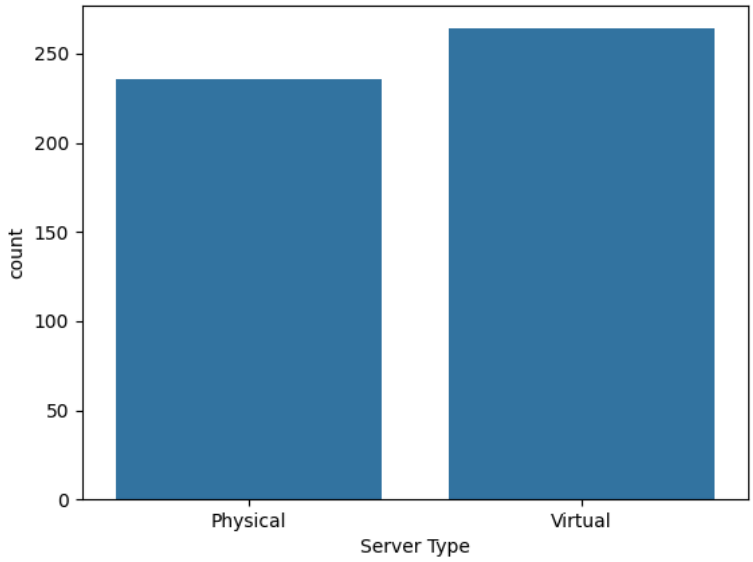
- The same type of relationship in terms of correlation can be visualized here as well.

- Now, we will plot basic graphs to understand our data further.



Datatype Analysis.

- From this bar graph, it is understood that there are a total of 7 object data type columns, 4 integer columns and 4 float value columns, with a total of 15 columns.



- From this count plot, we can see that the total number of Server types (Physical server types and Virtual server types) have been distributed almost equally among the dataset, without any sort of inequality.

- We will now move on with our model selection and implementation.

❖ **Model Building**:

- The algorithm that we are going to be using for clustering is the k - means algorithm.
- In brief, the k-means algorithm, an unsupervised learning algorithm, is a clustering technique that aims to split a dataset into, say, k numbers of clusters through by iteration:
- The thing about k means, or any clustering algorithm that makes it a useful tool is that given a dataset, it allows the user to differentiate the data into different groups based on the clusters, without the need of training.

➢ This algorithm follows a few steps:

1. Initialization: Randomly select k initial centroids from the dataset.
2. Assignment: Assign each data point to the nearest centroid based on the Euclidean distance, forming k clusters.
3. Update: Recalculate the centroids as the mean of all points assigned to each cluster.

- These steps are repeated until the centroids no longer change significantly or a predefined number of iterations is reached. It uses several distance metrics such as Euclidean, Cosine, Manhattan, Etc, for this. The final output is k distinct clusters.
- Note that in these snippets, I have included functions that I wrote to implement the k-means algorithm. These functions are too big to show, hence i will only be adding snippets of code to convey the basic ideas.

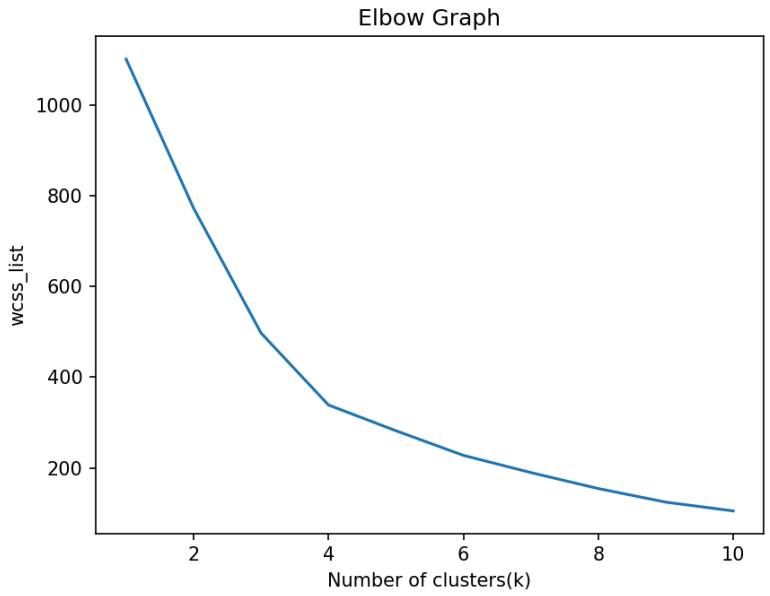
```
def elbowpoint(wss):
    k_values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    knee_locator = KneeLocator(k_values, wss, curve='convex', direction='decreasing')
    elbow_point = knee_locator.elbow
    return elbow_point
```

- ‘wss’ here is the total distance of data points from their respective cluster centroids. We iterate through a limit, and gather the wss score for our clusters. This score will be useful in predicting the elbow point

```
wcss_list= []  #Snippet from a function that i used to fit data in kmeans model
    for i in range(1, 11):
        kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
        kmeans.fit(x)
        wcss_list.append(kmeans.inertia_)
```

- Elbow point in k-means is obtained by plotting the values of wss obtained over iteration.

```
plt.plot(range(1, 11), wcss_list)
plt.title('Elbow Graph')
plt.xlabel('Number of clusters(k)')
plt.ylabel('wcss_list')
plt.show()
```



Example of how a graph is plotted using wcc scores to find the elbow point (k).

- In this example graph, the elbow point is 4, since that's where the plot begins to show an insignificant change or a steady decline.

- Now that we know how to obtain the elbow value, we will see how to fit the data in a k means model. It is recommended to scale data before fitting for best results.

```
scaler = StandardScaler()
x = scaler.fit_transform(x)
kmeans = KMeans(n_clusters= elbowpoint(wcss_list), init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
```

- Then, to validate the clusters formed, we can use a method (here, I am using the silhouette score).

- A silhouette score ranges from -1 to 1. By rule of thumb, a score in the range of roughly -1 to around 0.5 means that the clustering is poor, and a score around roughly 0.5 is a moderately defined cluster, and anything higher than that is well defined. However, this is purely subjective and depends on the context of data.

```
# Yet another snippet from a function that I made.
labels = kmeans.labels_
score = silhouette_score(x, labels, metric='cosine')
print(f'Silhouette Score of this cluster: {score}')
```

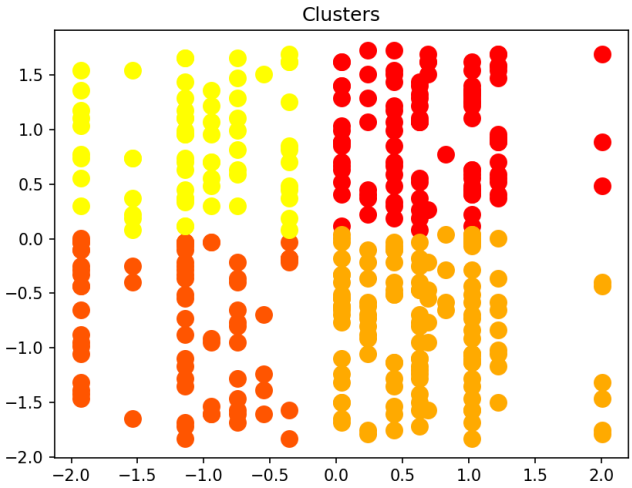
- And we are all set! All that's left is to plot the clusters, and compare the clusters between different data.

```
list_of_columns = dataframe.columns
cluster_features, scores_list, features_list = [[2,3],[2,4],[2,5],[2,6],[2,7],[2,8],[2,9],
                                                [3,5],[3,7],[3,8],[3,9],
                                                [4,5],[4,7],[4,8],[4,9],
                                                [5,6],[5,7],[5,8],[5,9],
                                                [6,7],[6,8],[6,9],
                                                [7,8],[7,9],
                                                [8,9]],[],[]
for i, feature in enumerate(cluster_features):
    versus = f'{list_of_columns[feature[0]]} vs {list_of_columns[feature[1]]}'
    print(f"Cluster {i+1}: {list_of_columns[feature[0]]} vs {list_of_columns[feature[1]]}")
    silscore = kmeans_master_function(dataframe, feature) # elbow graph,k value,cluster plot.
       scores_list.append(silscore)
```

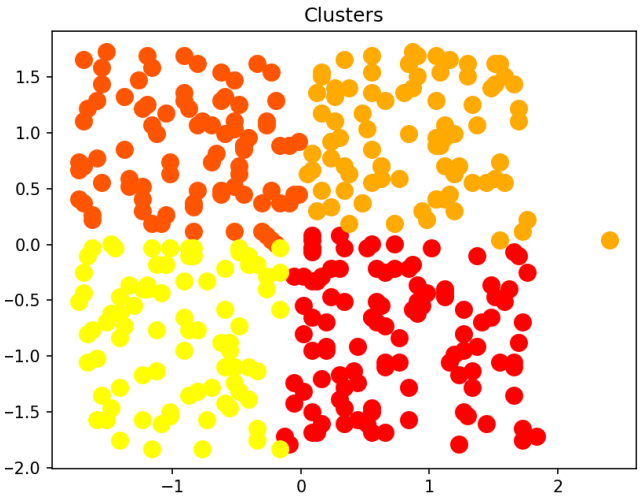
```
        features_list.append(versus)
scores_dataframe = pd.DataFrame({"Features": features_list, "Score": scores_list})
print(scores_dataframe)
```

- Since there are too many clusters to show here, I will show some notable clusters, to get some understanding of how this algorithm can be visualized.

(i) CPU_Speed vs CPU_Utilization



(ii) CPU_Speed vs RAM_Utilization



(iii) CPU_Utilization vs RAM_Utilization

(iv) CPU_Utilization vs Storage_Used_Size_GB.

➢ **Result Analysis**

- The Features that were used in the respective clusters, and their silhouette scores have been stored in a dataset. Taking a look at that dataset will give more insight into these graphs. Small note

	Features	Score
0	CPU_Speed vs CPU_Core	0.734627
1	CPU_Speed vs CPU_MThread	0.797057
2	CPU_Speed vs CPU_Utilization	0.716080
3	CPU_Speed vs RAM_Size_GB	0.756650
4	CPU_Speed vs RAM_Utilization	0.714431
5	CPU_Speed vs Storage_Size_GB	0.613826
6	CPU_Speed vs Storage_Used_Size_GB	0.745812
7	CPU_Core vs CPU_Utilization	0.592948
8	CPU_Core vs RAM_Utilization	0.574016
9	CPU_Core vs Storage_Size_GB	0.527206
10	CPU_Core vs Storage_Used_Size_GB	0.682914
11	CPU_MThread vs CPU_Utilization	0.760003
12	CPU_MThread vs RAM_Utilization	0.764864
13	CPU_MThread vs Storage_Size_GB	0.632841
14	CPU_MThread vs Storage_Used_Size_GB	0.792530
15	CPU_Utilization vs RAM_Size_GB	0.792987
16	CPU_Utilization vs RAM_Utilization	0.682103
17	CPU_Utilization vs Storage_Size_GB	0.617027
18	CPU_Utilization vs Storage_Used_Size_GB	0.737956
19	RAM_Size_GB vs RAM_Utilization	0.761874
20	RAM_Size_GB vs Storage_Size_GB	0.690212
21	RAM_Size_GB vs Storage_Used_Size_GB	0.820763
22	RAM_Utilization vs Storage_Size_GB	0.587483
23	RAM_Utilization vs Storage_Used_Size_GB	0.718318
24	Storage_Size_GB vs Storage_Used_Size_GB	0.468273

- An observation can be made - the silhouette scores of all these clusters are consistently high - indicating a well defined cluster. The average score of all clusters seems to be 0.691, and the highest score belonging to the clustering of `RAM_Size_GB vs Storage_Used_Size_GB` - 0.82.

- As we discussed earlier, let us see how the scores change when we remove the outliers from the dataset - and compare both average scores to see when the model performs better.

- Scores after keeping the outliers:

	Features	Score
0	CPU_Speed vs CPU_Core	0.576131
1	CPU_Speed vs CPU_MThread	0.663910
2	CPU_Speed vs CPU_Utilization	0.734778
3	CPU_Speed vs RAM_Size_GB	0.914777
4	CPU_Speed vs RAM_Utilization	0.452510
5	CPU_Speed vs Storage_Size_GB	0.659405
6	CPU_Speed vs Storage_Used_Size_GB	0.932585
7	CPU_Core vs CPU_Utilization	0.610161
8	CPU_Core vs RAM_Utilization	0.139848
9	CPU_Core vs Storage_Size_GB	0.457111
10	CPU_Core vs Storage_Used_Size_GB	0.612325
11	CPU_MThread vs CPU_Utilization	0.554818
12	CPU_MThread vs RAM_Utilization	0.229196
13	CPU_MThread vs Storage_Size_GB	0.400328
14	CPU_MThread vs Storage_Used_Size_GB	0.545198
15	CPU_Utilization vs RAM_Size_GB	0.838197
16	CPU_Utilization vs RAM_Utilization	0.344342
17	CPU_Utilization vs Storage_Size_GB	0.608462
18	CPU_Utilization vs Storage_Used_Size_GB	0.848289
19	RAM_Size_GB vs RAM_Utilization	0.242024
20	RAM_Size_GB vs Storage_Size_GB	0.596568
21	RAM_Size_GB vs Storage_Used_Size_GB	0.955441
22	RAM_Utilization vs Storage_Size_GB	0.275187
23	RAM_Utilization vs Storage_Used_Size_GB	0.548311
24	Storage_Size_GB vs Storage_Used_Size_GB	0.535944

- This is interesting - although we can see that some scores are lower - going as low as 0.13, the magnitude of scores that are above 0.8 are higher than the first set of scores. The average score of this becomes 0.57, which is much lower than when we removed outliers.

	Outlier Handling	Average Score
0	Removing all outliers	0.6913
1	Keeping all outliers	0.5710

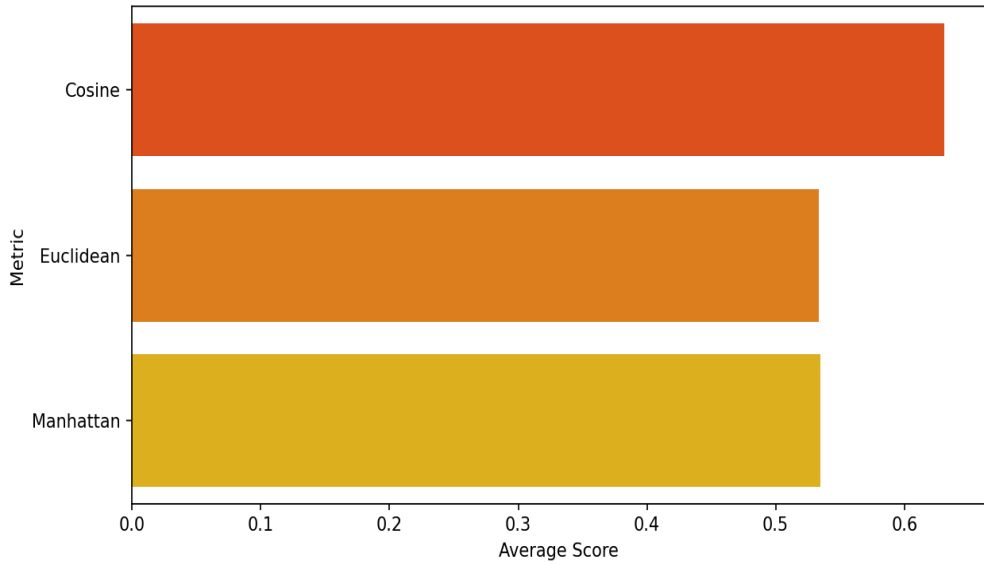
➢ Extra Analysis on different metrics:

- This is something I thought would be useful. Note that for the model I used, I used the "cosine" metric, which measures the angle between two vectors. To see if any other metric would provide a higher result, I changed the metric to other metrics such as manhattan and euclidean, and these are the results I got:

	Outlier Handling	Metric	Average Score
0	Removing all outliers	Cosine	0.6913
1	Keeping all outliers	Cosine	0.5710
2	Removing all outliers	Euclidean	0.5138
3	Keeping all outliers	Euclidean	0.5529
4	Removing all outliers	Manhattan	0.5206
5	Keeping all outliers	Manhattan	0.5489

- We can see that the cosine metric has come up with the best scores on both situations of Outlier Handling, whereas the other two metrics have similar performance.

	Metric	Average Score
0	Cosine	0.63115
1	Euclidean	0.53335
2	Manhattan	0.53475



(ii) Cosine has the best performance out of all three metrics, whereas the other two are similar.

❖ Conclusion:

➢ Dataset:

- The Dataset was the information of a number of systems in a datacenter. My objective was to use the k means clustering model to come up with insightful information about the relationships between different data in the dataset.

➢ Dataset Description:

- The dataset had 15 columns with 500 values. All 15 columns were either of integer, float or object type values and provided a variety of information about the systems in the datacenter. It had a small amount of zero values and null values in it.

➢ Exploratory Data Analysis:

- The dataset had both zero values as well as null values, and it was dealt with by finding the percentage of null values (after converting the zeros to nulls) and filling them with the median values after looking at their histograms.

- The dataset had outliers - and some of them were extreme. Although these may be just really extreme but true data entries, I wanted to see how the model performed with and without outliers present in the dataset. So I first removed the outliers to check the model and then bought them back in the data to compare. Finally, I decided to remove them since the model performed better without the outliers. The outliers were hence removed, using the IRQ method.

- Correlation - For the most part, features had low correlation with each other, but 3 sets of features stood out: CPU_Core and CPU_MThread - 0.98, RAM_Size_GB and CPU_Core - 0.82, and RAM_Size_GB and CPU_Mthread - 0.83. Since clustering produces better results with features that are less correlated, these sets were not involved in the model. Hence they were not used.

- Overall, the clustering was moderate to good, with several features having well defined clusters and it helped in providing a deeper understanding of the relationship between the different features in the dataset.

➢ **Model Building:**

- The k-means algorithm was selected for the purpose of clustering the data in hand.

- Elbow Search: Using iteration, the best value for ‘k’ - The number of clusters, was found by fitting each model with ‘i’ number in clusters in the range of 10. Then, the wss score of each model in the loop was appended to a list, and from that, the best value for k was chosen, by plotting a graph. I made this into a function so that it would be easier for me to calculate the k value for any two features, as typing it again and again would be redundant and a waste of time.

- After choosing the best value for k for two given features, another function fits the model with the data, k value and clusters it using the “cosine” metric. This function returns the wss score, an array of the predicted clusters, and the fitted k means model.

- Finally, a master function executes all these functions, and on top of that, uses the k-means model to calculate the silhouette score.

- So in total, one function to predict the elbow score, another function to fit the model using the elbow score, and a third function to predict it, and finally, a master function that executes all these functions.

- Scaling was used to scale the values of the features for better clustering, with the help of the StandardScaler() method.

- Then, after visualizing the clusters, the scores are stored in a dataframe for analysis.

- Validation - Different metrics were used in place of cosine, and their scores were calculated and compared. Cosine had the highest average score (>0.6) out of all metrics used (Euclidean and Manhattan (>0.5 and < 0.6)).

➢ Final Conclusion:

- Tackling the challenges that I encountered such as data type discrepancies, outliers, missing data, etc, I was able to create a basic clustering model for this dataset successfully. This helped me deepen my understanding of how clustering worked in machine learning. Below are the highlights of the model:

- The clustering model performed well in data that had no outliers, with a consistent average score of above 0.5 among all different metrics used, which indicates well defined clusters.

- The Cosine metric performed best out of the three metrics, with an average score of >0.65, significantly higher than the other two metrics that were used (Euclidean - ~0.5, Manhattan, ~0.5).

- The clusters produced enable us to draw our own ideas of the relationship between features. For example - in page 16 fig (iii), we can see how the data points are clustered based on low CPU and RAM utilization, low CPU but high RAM utilization, high CPU but low ram utilization, and high CPU as well as high RAM utilization.

Ultimately, clustering tools are very helpful in allowing the user to segregate data based on different features, without the need of training the data. Clustering is a reliable resource in fields like pattern recognition, customer analysis, image analysis, bio - informatics, etc and is overall, an important feature in unsupervised machine learning.

Thank you for time and consideration,
Shri Sai Aravind. R