

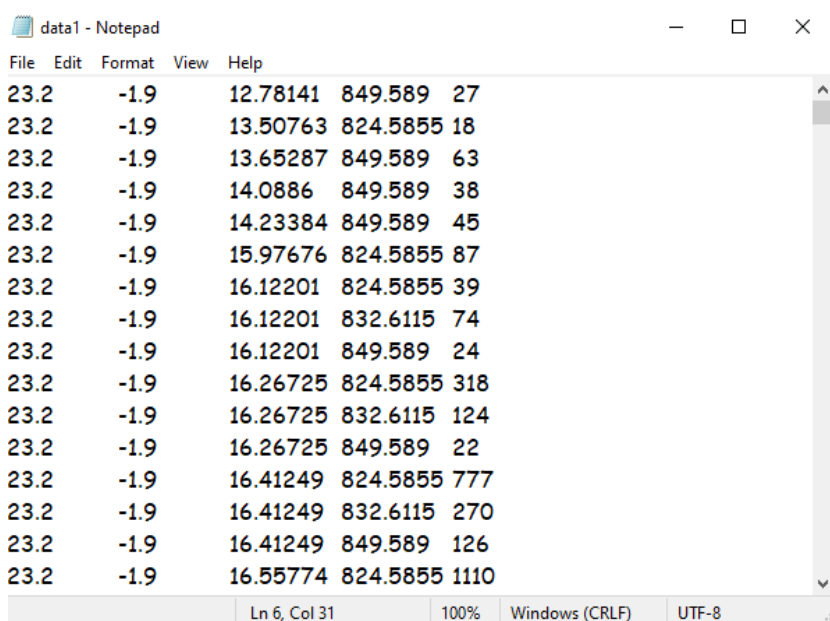
# Report on Data Visualization

Data visualization is the graphical representation of data to help people understand context and significance. Interactive data visualization enables companies to drill down to explore details, identify patterns and outliers, and change which data is processed and/or excluded.

Python provides various libraries that come with different features for visualizing data. All these libraries come with different features and can support various types of graphs. Major libraries used in this analysis is:

- Matplotlib
- Seaborn
- Plotly

Given Dataset: data1.asc



23.2	-1.9	12.78141	849.589	27
23.2	-1.9	13.50763	824.5855	18
23.2	-1.9	13.65287	849.589	63
23.2	-1.9	14.0886	849.589	38
23.2	-1.9	14.23384	849.589	45
23.2	-1.9	15.97676	824.5855	87
23.2	-1.9	16.12201	824.5855	39
23.2	-1.9	16.12201	832.6115	74
23.2	-1.9	16.12201	849.589	24
23.2	-1.9	16.26725	824.5855	318
23.2	-1.9	16.26725	832.6115	124
23.2	-1.9	16.26725	849.589	22
23.2	-1.9	16.41249	824.5855	777
23.2	-1.9	16.41249	832.6115	270
23.2	-1.9	16.41249	849.589	126
23.2	-1.9	16.55774	824.5855	1110

Number of Fields: 5

Number of Records: 23717

## Importing Libraries:

```
[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
```

## Importing Dataset:

```
[ ] import numpy as np
df = np.loadtxt("/content/data1.asc")
```

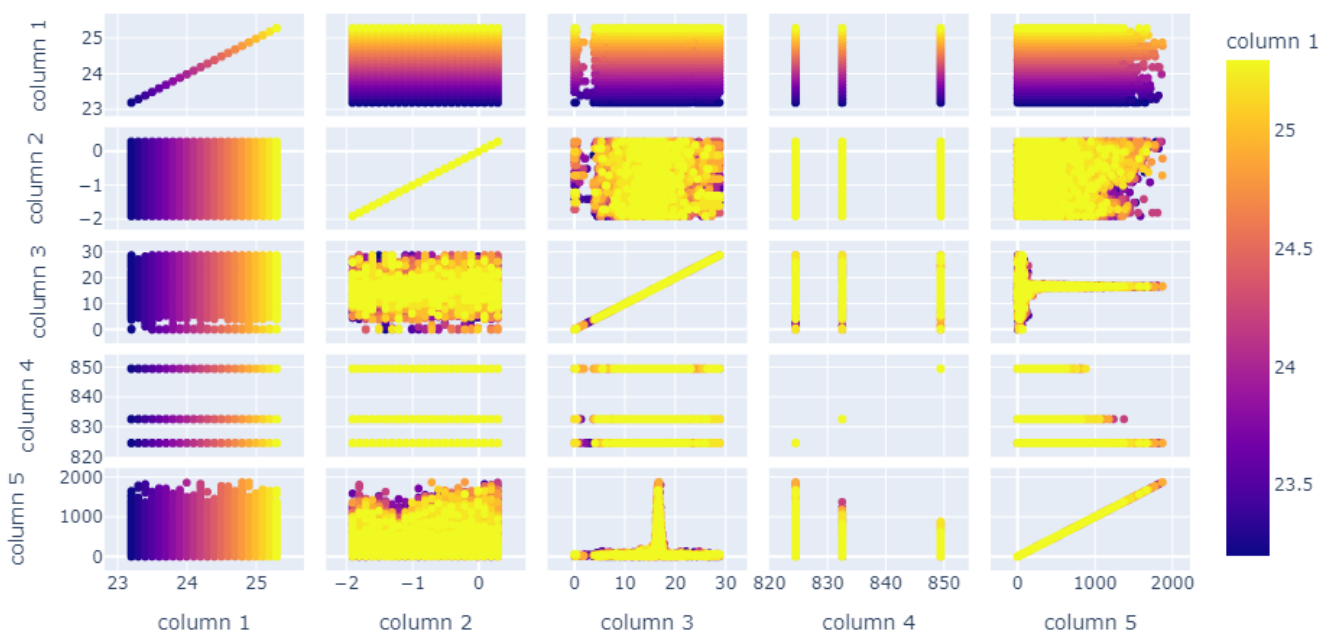
## Plotting a Scatter Matrix:

A scatter plot matrix shows all pair-wise scatter plots for many variables. If the variables tend to increase and decrease together, the association is positive. If one variable tends to increase as the other decreases, the association is negative.

```
[ ] fig=px.scatter_matrix(df, color='column 1')  
fig.show()
```

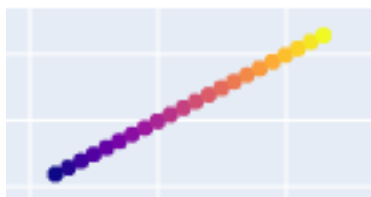
This matrix is plotted according to Column 1 values ranging from 23.2 – 25.3

The Output:



Inferences drawn from this Scatter Matrix:

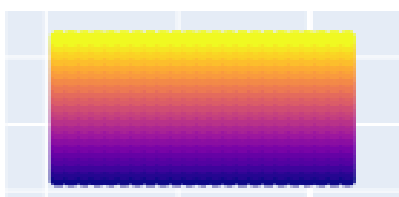
1.



- In this we can infer that the variables are positively correlated with each other
- And we can as well eliminate one of the feature to reduce the dimension

We can observe this pattern only with their own respective columns, which means no two different columns in this data given are highly correlated and none of the columns can be eliminated.

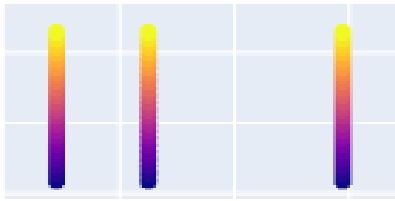
2.



- In this we can infer that when one variable's value stays constant as the other variable's value keeps increasing
- All data values of the respectable variable is present
- We can say that same values are present on every point of both the variables making it look like a rectangle

We can observe this pattern between Column 1 & Column 2 and vice versa

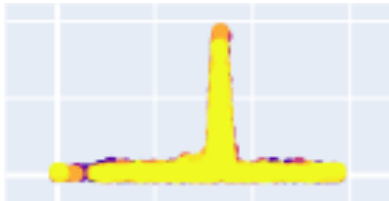
3.



- In this we can infer that only few values of one column keep increasing constantly with the increase in other column

We can observe this pattern in Column 1 & Column 4, Column 2 & Column 4, Column 3 & Column 4 and vice-versa

4.



- In this we can infer that the variables are not correlated to each other
- It resembles big tailed distance, it is skinny, long and symmetric

### Creating a 3D Scatter plot:

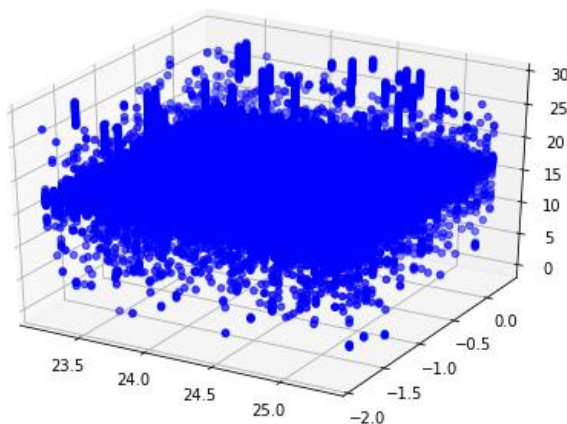
3D scatter plots are used to plot data points on three axes in the attempt to show the relationship between three variables. Each row in the data table is represented by a marker whose position depends on its values in the columns set on the X, Y, and Z axes.

In Matplotlib:

```
fig = plt.figure()
ax = Axes3D(fig)
plot_geeks = ax.scatter(column1, column2, column3, color='blue')
plt.show()
```

3D scatter plot. We first create a figure and then create the plot by using `ax.scatter()` and we give variables as parameters and specify the colour we need the plot to be displayed and the display the plot.

Output:



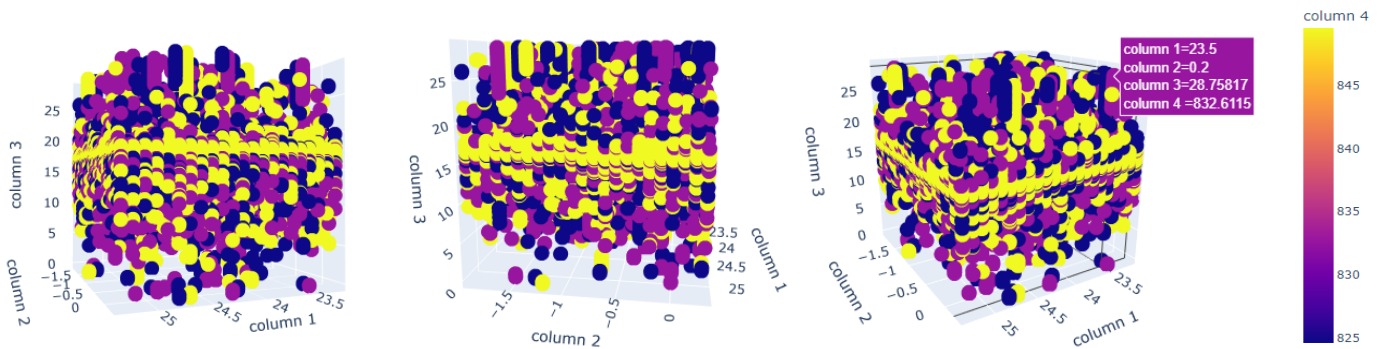
This plot is seen when we plot column1, 2 & 3. It is a static 3D plot

In Plotly:

```
fig = px.scatter_3d(df,x='column 1',y='column 2',z='column 3',color='column 4 ')\nfig.show()
```

In this 3D scatter plot is created by `px.scatter()` the colour gradient changes with the values in the variable initialised to the colour parameter.

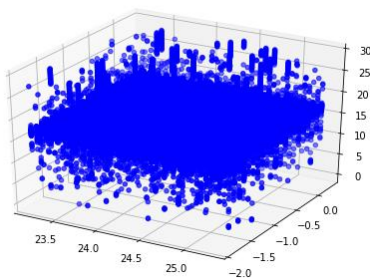
Output:



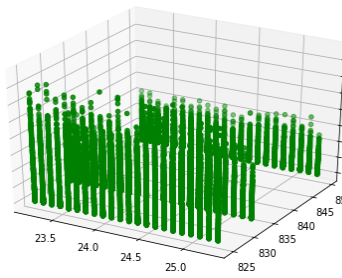
This plot is seen when we plot column1, 2 & 3. This is a dynamically rotatable and interactive plot. The values (i.e. the co-ordinates) of the variables are can be seen when we hover the mouse over the data point.

3D Plot Comparisons:

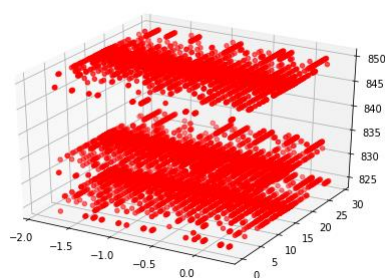
Col.1, Col.2, Col.3



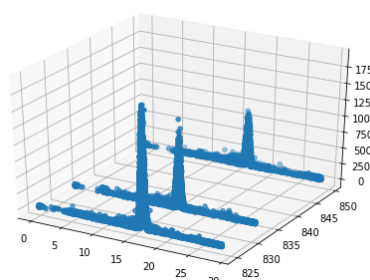
Col.1, Col.4, Col.5



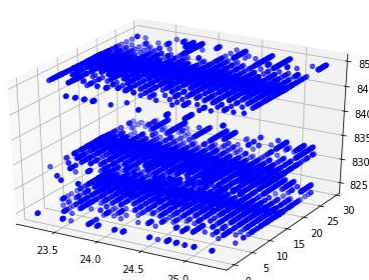
Col.2, Col.3, Col.4



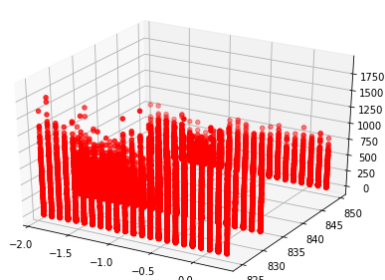
Col.3, Col.4, Col.5



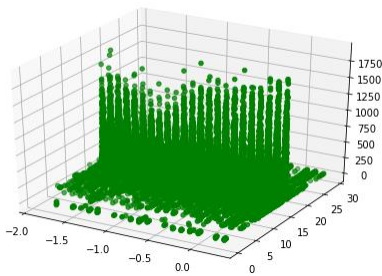
Col.1, Col.3, Col.4



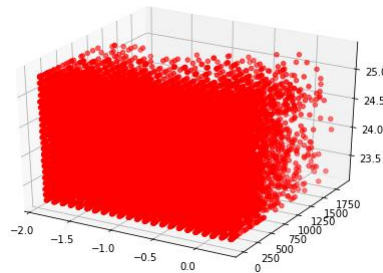
Col.2, Col.4, Col.5



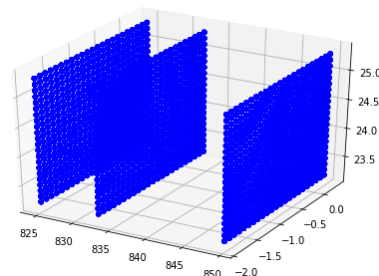
Col.2, Col.3, Col.5



Col.2, Col.5, Col.1



Col.1, Col.2, Col.4

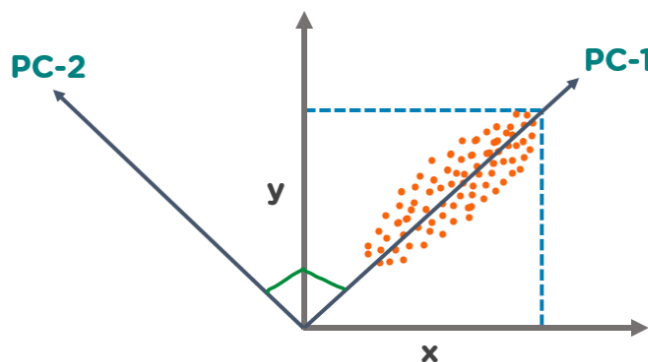


Inferences from the plot:

- High correlation between column1 and column2 can be observed from the above plots
- We can conclude that these plots are not randomly distributed but rather have a pattern correlating the data points

## Principle Component Analysis (PCA):

The Principle Component Analysis is a popular unsupervised learning technique for reducing the dimensionality of data. It increases interpretability yet, at the same time, it minimizes information loss. It helps to find the most significant features in a dataset and makes the data easy for plotting in 2D and 3D. PCA helps in finding a sequence of linear combinations of variables.



### Two principal components:

PC-1 is the primary principal component that explains the maximum variance in the data

PC-2 is another principal component that is orthogonal to PC-1

**Principal Component-** A straight line that captures most of the variance of the data. They have a direction and magnitude. Principal components are orthogonal projects(perpendicular) of data onto lower-dimensional space.

### Steps for PCA algorithm:

#### 1. Getting the dataset

Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set

#### 2. Representing data into a structure

Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.

#### 3. Standardizing the data

In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance. If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.

#### 4. Calculating the Covariance of Z

To calculate the covariance of Z, we will take the matrix Z, and will tranpose it. After transpose, we will multiply it by Z. The output matix will be covariene matrix of Z

#### 5. Calculating the Eigen values and Eigen vectors

Now we need to calculate the Eigen values and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the Eigen values.

#### 6. Sorting the Eigen Vectors

In this step, we will take all the Eigen values and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of Eigen values. The resultant matrix will be named as P\*.

#### 7. Calculating the new features or Principal Components

Here we will calculate the new features. To do this, we will multiply the P\* matrix to the Z. In the resultant matrix Z\*, each observation is the linear combination of original features. Each column of the Z\* matrix is independent of each other.

#### 8. Remove less or unimportant features from the new dataset

The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

## PCA in python:

**sklearn.decomposition.PCA**

```
class sklearn.decomposition.PCA(n_components=None, *, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', n_oversamples=10, power_iteration_normalizer='auto', random_state=None)
```

Principal component analysis (PCA).

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. The input data is centered but not scaled for each feature before applying the SVD.

It uses the LAPACK implementation of the full SVD or a randomized truncated SVD. 2009, depending on the shape of the input data and the number of components to extract.

## 2D PCA Analysis:

Code:

```
✓ [31] from sklearn.preprocessing import StandardScaler  
0s scaler= StandardScaler()  
scaler.fit(df)  
scaled_data=scaler.transform(df)  
from sklearn.decomposition import PCA  
pca=PCA(n_components=2)  
pca.fit(scaled_data)  
x_pca=pca.transform(scaled_data)
```



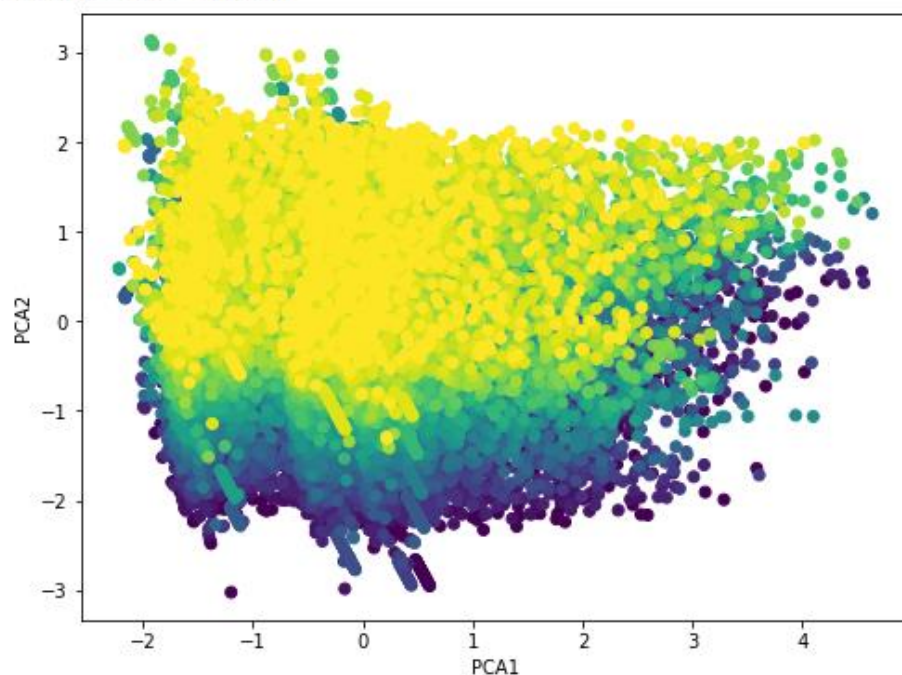
```

✓ [32] plt.figure(figsize=(8,6))
4s    plt.scatter(x_pca[:,0],x_pca[:,1],c=df['column 1'])
    plt.xlabel('PCA1')
    plt.ylabel('PCA2')

```

Output:

Text(0, 0.5, 'PCA2')



### 3D PCA Analysis:

Code:

```

✓ 14s  import plotly.express as px
      from sklearn.decomposition import PCA

      X1 = df[['column 1', 'column 2', 'column 3', 'column 4 ', 'column 5']]

      pca = PCA(n_components=3)
      components = pca.fit_transform(X1)

      total_var = pca.explained_variance_ratio_.sum() * 100

      fig = px.scatter_3d(
          components, x=0, y=1, z=2, color=df['column 1'],
          title=f'Total Explained Variance: {total_var:.2f}%',
          labels={'0': 'PC 1', '1': 'PC 2', '2': 'PC 3'}
      )
      fig.show()

```

Output:

