LAB 2 REPORT

1. Chosen Data Set: Flights*.csv*
2. **Task 1: Data Clustering and decimation**
   a. **Client-server architecture using python-Flask** for data processing in the back-end at server side and visualization in front-end using d3.
   b. **Data Cleanup**: Removed columns which did not have any data or had nulls or NA or 0. Also removed columns with text because it requires coding of text or using TF-IDF for convert text to number.
   c. **Sampling**:
      i. **Random Sampling:** Created random samples of size = samplesize in code. In the code samplesize is taken as 200 to show visualizations.

```
def random_sampling():
    # Random samples
    global data
    global data_csv
    global random_samples
    global samplesize
    features = data_csv[ftrs]
    data = np.array(features)
    rnd = random.sample(range(len(data_csv)),
samplesize)
    for j in rnd:
        random_samples.append(data[j])
```

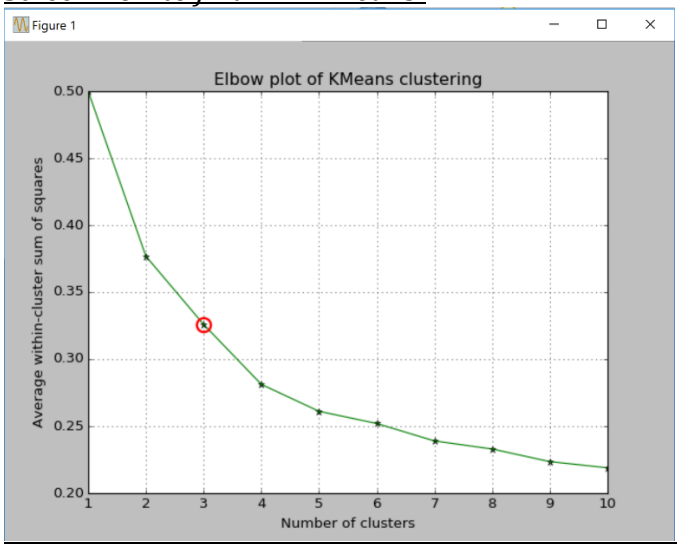
      ii. **Stratified/Adaptive Sampling**: In adaptive sampling, the total population is divided into groups in our case clusters. Then probability sample is taken from each group. Created adaptive samples of size = samplesize. In the code adaptive samples is taken as 200 for visualization. We see points in same cluster are together.

```
def adaptive_sampling():
    # Adaptive samples
    global data_csv
    global adaptive samples

    kcluster0 = data_csv[data_csv['kcluster'] == 0]
    kcluster1 = data_csv[data_csv['kcluster'] == 1]
    kcluster2 = data_csv[data_csv['kcluster'] == 2]

    size_kcluster0 = len(kcluster0) * samplesize / len(data_csv)
    size_kcluster1 = len(kcluster1) * samplesize / len(data_csv)
    size_kcluster2 = len(kcluster2) * samplesize / len(data_csv)

    sample_cluster0 = kcluster0.ix[random.sample(kcluster0.index, int(size_kcluster0))]
    sample_cluster1 = kcluster1.ix[random.sample(kcluster1.index, int(size_kcluster1))]
    sample_cluster2 = kcluster2.ix[random.sample(kcluster2.index, int(size_kcluster2))]

    adaptive_samples = pandas.concat([sample_cluster0, sample_cluster1,
```

iii. **Clustering:** For clustering K-means algorithm is used. K is calculated using elbow. In the code K is checked from 1 to 11 and a scree plot is done to identify the elbow. As we see from the plot, we got k=3.

```
k = range(1, 11)

clusters = [KMeans(n_clusters=c, init='k-means++').fit(features) for c in k]
centr_lst = [cc.cluster_centers_ for cc in clusters]
```

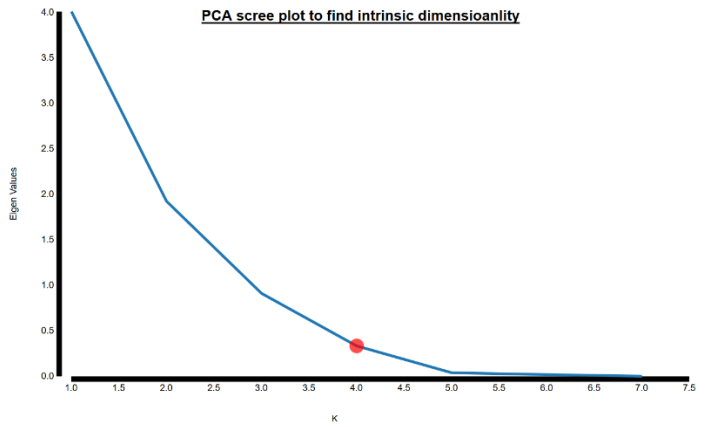
Scree PLOT to find k in KMeans:



3. **Task 2: Dimension Reduction using decimated data:**
   a. **Find intrinsic dimensionality of data using PCA:**
   Eigen values are calculated from covariance matrix using linear algebra library in Python. These eigen values are sorted in descending order. To find the intrinsic dimensionality, plot eigen values in scree plot.

```
def generate_eigenValues(data):
    mean_vec = np.mean(data, axis=0)
    cov_mat = np.cov(data.T)
    eig_values, eig_vectors = np.linalg.eig(cov_mat)

    #Sorting Eigen Values
    idx = eig_values.argsort()[::-1]
    eig_values = eig_values[idx]
    eig_vectors = eig_vectors[:, idx]
    return eig_values, eig_vectors
```

   b. **Produce scree plot visualization and mark the intrinsic dimensionality**
   Plot scree plot across all eigen values and identify the elbow to find optimal number of components for our dataset. Here for our dataset, the number of components from elbow plot is 4. So, the dataset can best be represented using four principal components.



c. **Obtain the three attributes with highest PCA loadings:**

- To find the three attributes with highest PCA loading, we need to calculate the squared loadings. Each element of eigen vector represents the contribution of a given variable to a component. Loadings are these eigen vector elements or the correlation between variables and principal components.
- Then sort these squared loadings and take the top n components. In our case, it is 3 from PCA scree plot to do further data analysis.

```
def plot_intrinsic_dimensionality_pca(data, k):
    [eigenValues, eigenVectors] = generate_eigenValues(data)
    print eigenValues
    squaredLoadings = []
    ftrCount = len(eigenVectors)
    for ftrId in range(0,ftrCount):
        loadings = 0
        for compId in range(0, k):
            loadings = loadings + eigenVectors[compId][ftrId] * eigenVectors[compId][ftrId]
        squaredLoadings.append(loadings)

return squaredLoadings
```

4. **Task 3: Visualization using dimension reduced data:**

a. **Visualize data projected into the top two PCA vectors via 2D scatterplot**

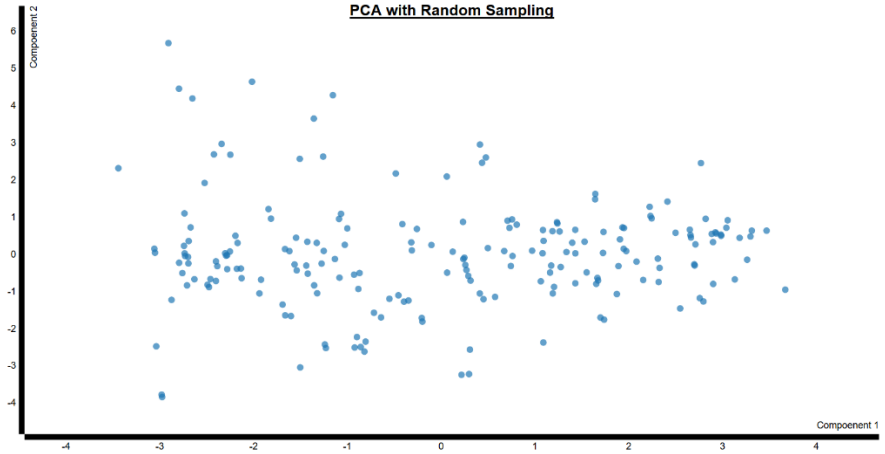
Top two PCA vectors are those that correspond to highest eigen values. Here scikit library is used to calculate PCA components with number of components = 2. Visualization of top 2 PCA vectors are done for randomly sampled data and adaptive sampled data. Code for PCA random sampling is shown below. Similarly, for adaptive samples pass adaptive samples to PCA transformation.

OBSERVATION: We see that in adaptive the cluster points formed correctly. The green cluster was like the outliers of PCA components.

```
pca_data = PCA(n_components=2)
X = random_samples
pca_data.fit(X)
X = pca_data.transform(X)
data_columns = pandas.DataFrame(X)
```

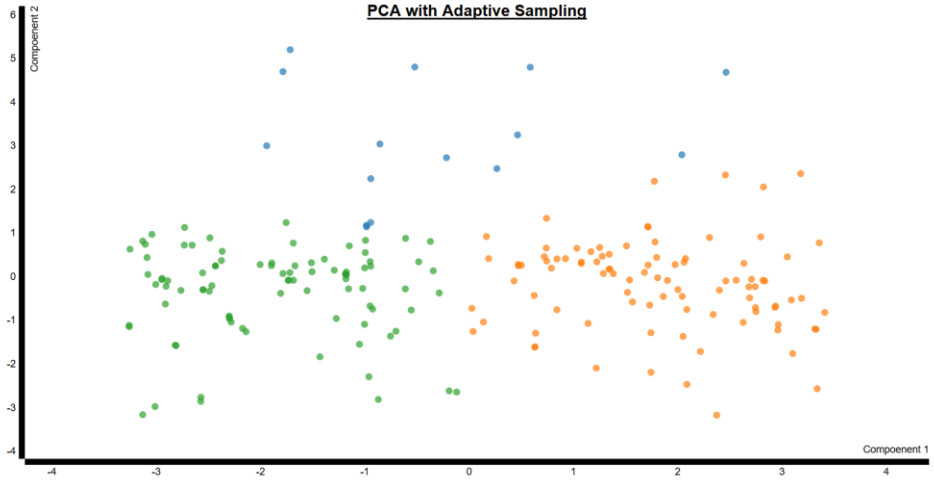
PCA Random Sampling Visualization:

We see all the points are somewhat parallel to x-axis. This shows that PCA compoenent1 has more weight compared to PCA compoenent2.



PCA Adaptive Sampling Visualization:

Also we see that in PCA adaptive the third cluster looks like the outliers which are not part of clustered points parallel to x-axis.



b. **Visualize data via MDS (Euclidian & correlation distance) in 2D scatterplots**

MDS (Multidimensional Scaling) maps the distance between observations from N-D into low-D (day 2D). It attempts to ensure that the differences between pair of points in this reduced space match as closely as possible. To find the distance between pair of observations/sample points we can use:

- Euclidean distance:
  - MDS with Euclidean distance and Random Sampling

```
mds_data = manifold.MDS(n_components=2, dissimilarity='precomputed')
similarity = pairwise_distances(random_samples, metric='euclidean')
X = mds_data.fit_transform(similarity)
data_columns = pandas.DataFrame(X)
```

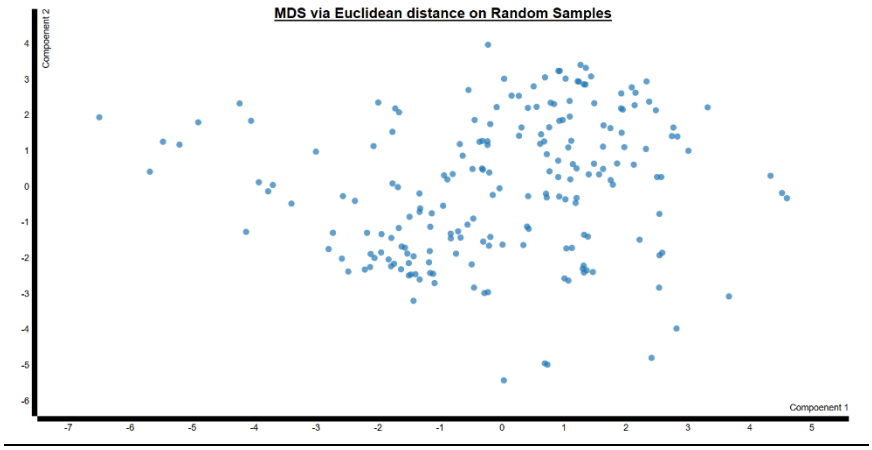
- Correlation distance:
  - MDS with Correlation distance and Random Sampling

```
mds_data = manifold.MDS(n_components=2, dissimilarity='precomputed')
similarity = pairwise_distances(random_samples, metric='correlation')
X = mds_data.fit_transform(similarity)
data_columns = pandas.DataFrame(X)
```

Just replace random samples with adaptive samples while doing the MDS with adaptive sampling.

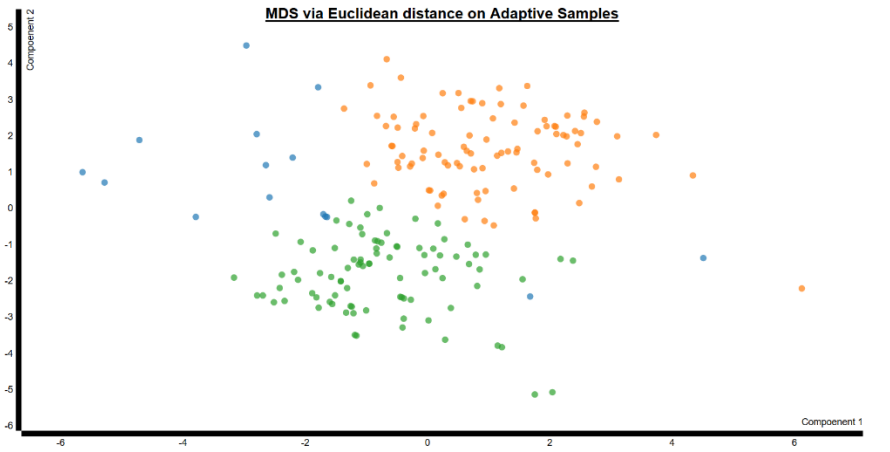
MDS with Euclidean distance and Random Sampling:

MDS with Euclidean distance gives points which seem equidistant from both the axis. This shows that both components contribute to MDS.

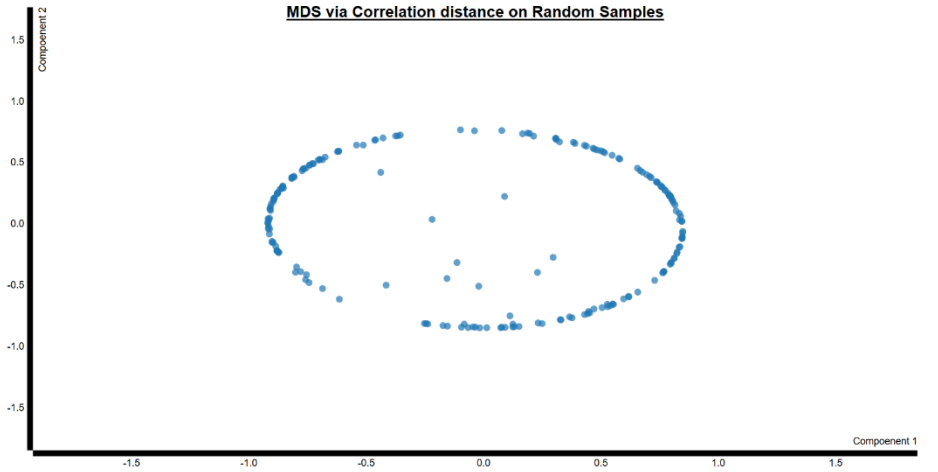


MDS with Euclidean distance and Adaptive Sampling:

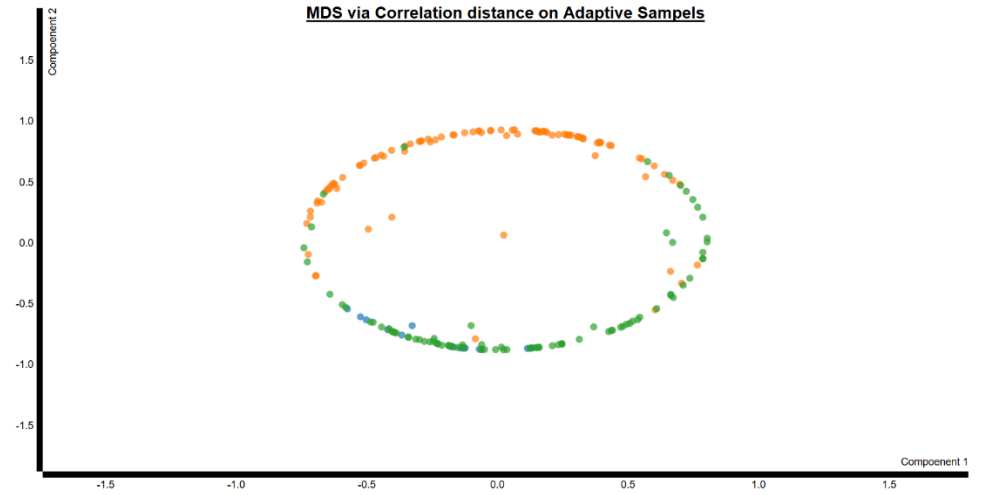
Like before a third cluster seems like an outlier which have less values for component1.



MDS with Correlation distance and Random Sampling:
We see that both the components are correlated as it forms a circle of points.



MDS with Correlation distance and Adaptive Sampling:
Points from same cluster are together in the correlation distance with MDS on adaptive sampling.



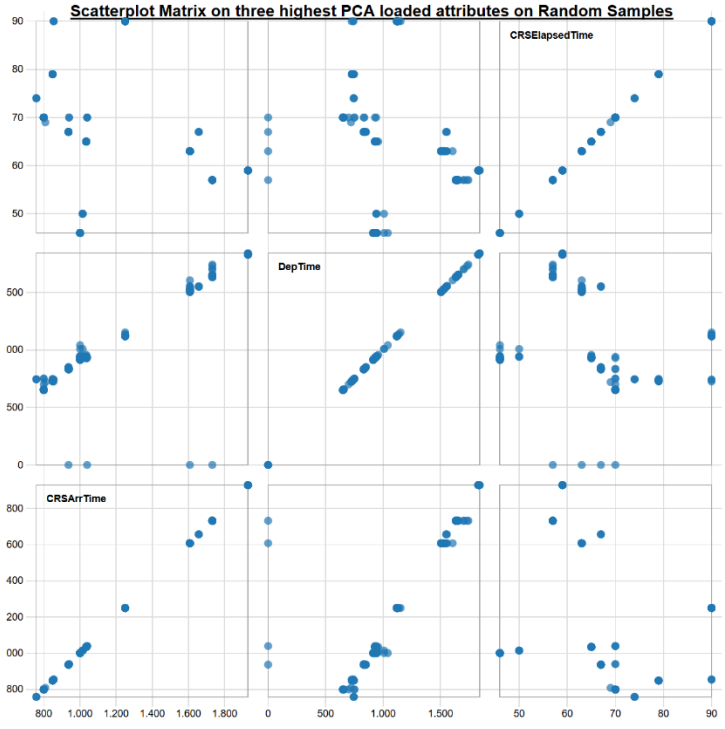
c. **Visualize scatterplot matrix of the three highest PCA loaded attributes**

From the data, the highest loaded PCA attributes calculated from intrinsic dimensionality are CRSElapsedTime, Departure Time, CRSArrTime.

OBSERVATION: Yes, it is correctly symmetric about the right diagonal. Also in adaptive, as the clusters are together. And both random scatter plot and adaptive scatter plot are similar. Right diagonal forms a line showing that DeptTime with DeptTime contribute equally which is obvious.

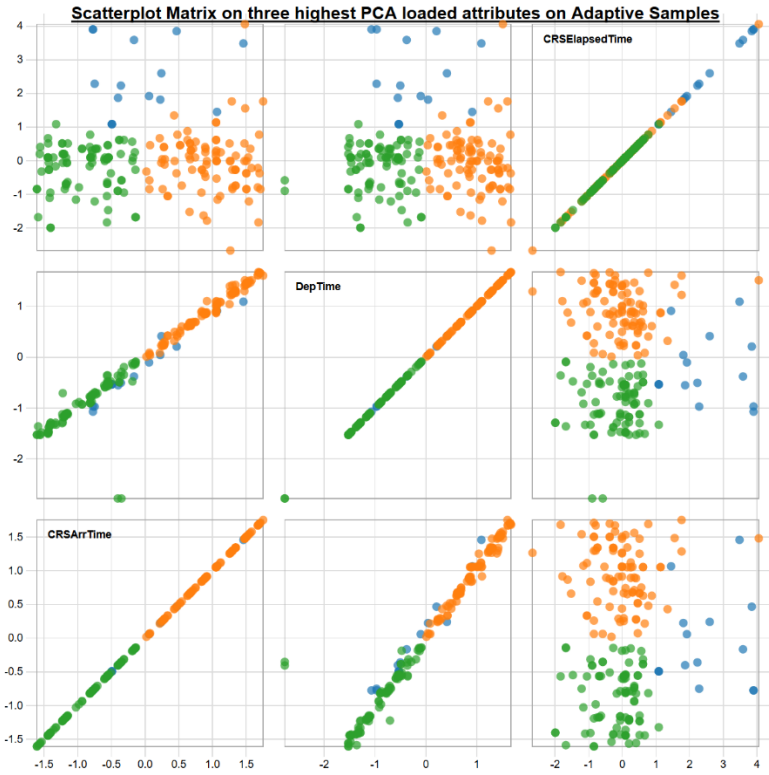
Scatterplot matrix with random sampling

Many points coincide probably because of similar relationship..



Scatterplot matrix with adaptive sampling:

The third cluster separates out beautifully like a cluster of outlier points.



CONCLUSION:

From PCA, we see that first component from PCA weighs more than second component. MDS with correlation distance showed up beautifully with a ellipse , saying the components were correlated and boarder axis of ellipse is parallel to x-axis. Saying component 1 weighs more than component 2. MDS with Euclidean distance was not a good metric but still the observation that component 1 weighs more than component 2 still holds as all point are seemingly having greater value for compoenent1. Both PCA and MDS says that compoenent1 weighs more than compoenent2. Now what were the important features which contributed to these components? By calculating squared loadings, we found these features as CRSElapsedTime, Departure Time, CRSArrTime. This makes sense, departure and arrival time are related to elapsed time. Hence the correlation from MDS with correlation distance. Elapsed time is the most weighted feature which looks like a genuine observation as we can write elapsed time=departure time-arrival time.