# CSE 564
# Visualization and Visual Analytics

Mini Project #2

Pratik Mukund Velhal (112675099)

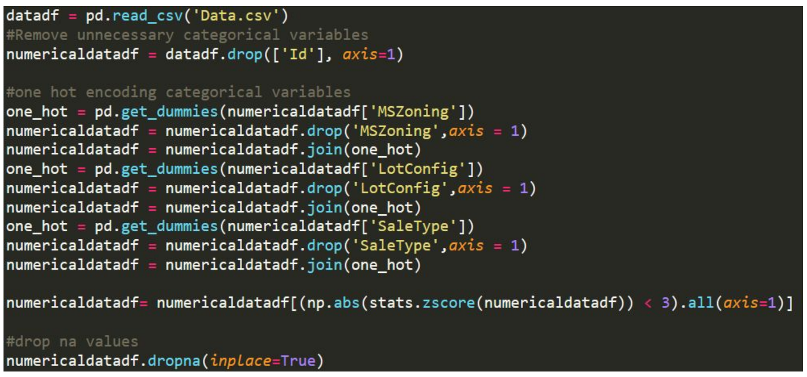
Project Overview:
This project incorporates the three basic tasks of visual data analytics:

1. Data clustering and decimation (Sampling and k-means clustering)
2. Dimension reduction (Scree Plot, PCA Scatterplot)
3. Visualization of original and reduced data (MDS, Scatterplot matrix)

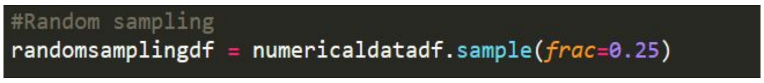
Demo: CSE 564:Visualization Project 2- Pratik Mukund Velhal

Data:
I have used the Ames Housing Dataset (**https://tinyurl.com/yzjolrx8**) available on Kaggle in this project. I had used this dataset previously for another data science project, and this dataset was chosen due to familiarity with all the variables. The data has around 31 different columns consisting of both categorical and numerical variables where each variable distinctly describes a unique characteristic of housing in Iowa. The dataset has 1460 rows of data items.

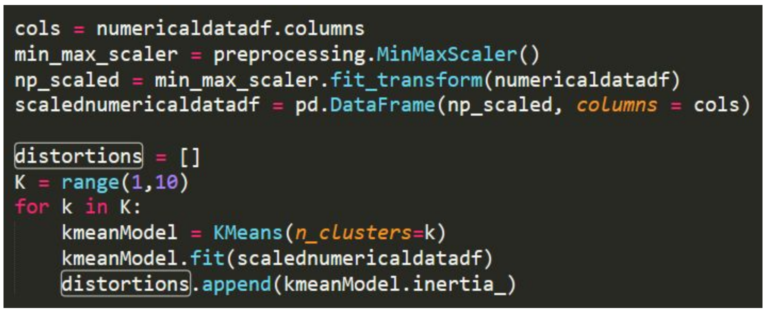
Implementation:
We have a client-server architecture where we are using python flask for the server, and d3.js is used for visualizing data on the front-end.

1. Reading and pre-processing data: Our input is in the form of a csv file. We are loading data from this file and storing it in pandas dataframe using pd.read_csv(). Data is loaded from the input file initially on server start, and thereafter is re-used to improve efficiency. We have used one hot encoding on nominal categorical data. Also, we have removed outliers with z-score of greater than 3. All N.A values from the data have also been removed.

```
datadf = pd.read_csv('Data.csv')
#Remove unnecessary categorical variables
numericaldatadf = datadf.drop(['Id'], axis=1)

#one hot encoding categorical variables
one_hot = pd.get_dummies(numericaldatadf['MSZoning'])
numericaldatadf = numericaldatadf.drop('MSZoning',axis = 1)
numericaldatadf = numericaldatadf.join(one_hot)
one_hot = pd.get_dummies(numericaldatadf['LotConfig'])
numericaldatadf = numericaldatadf.drop('LotConfig',axis = 1)
numericaldatadf = numericaldatadf.join(one_hot)
one_hot = pd.get_dummies(numericaldatadf['SaleType'])
numericaldatadf = numericaldatadf.drop('SaleType',axis = 1)
numericaldatadf = numericaldatadf.join(one_hot)

numericaldatadf= numericaldatadf[(np.abs(stats.zscore(numericaldatadf)) < 3).all(axis=1)]

#drop na values
numericaldatadf.dropna(inplace=True)
```
2. Data clustering and decimation: We have performed both random and stratified sampling on the data while removing 75% of the data rows. For stratified sampling, we

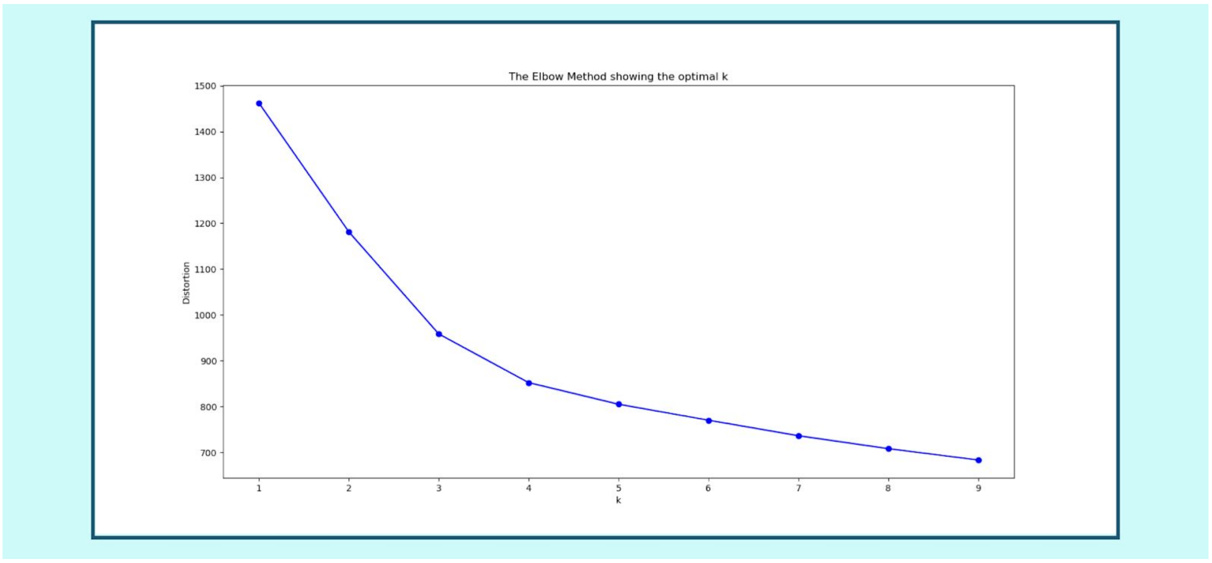
have first scaled the data before applying k-means clustering. We are checking distortions for k in the range of 1 to 9. Using the elbow method, we can see that our optimal value of k is 4. We are then clustering our data using k-means clustering with k='4'. Finally, for each cluster around 25% of data rows are taken and stored in the stratified sampled dataframe.



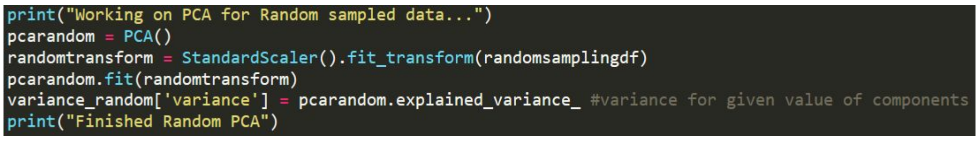
```
#Random sampling
randomsamplingdf = numericaldatadf.sample(frac=0.25)
```



```
cols = numericaldatadf.columns
min_max_scaler = preprocessing.MinMaxScaler()
np_scaled = min_max_scaler.fit_transform(numericaldatadf)
scalednumericaldatadf = pd.DataFrame(np_scaled, columns = cols)

distortions = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(scalednumericaldatadf)
    distortions.append(kmeanModel.inertia_)
```

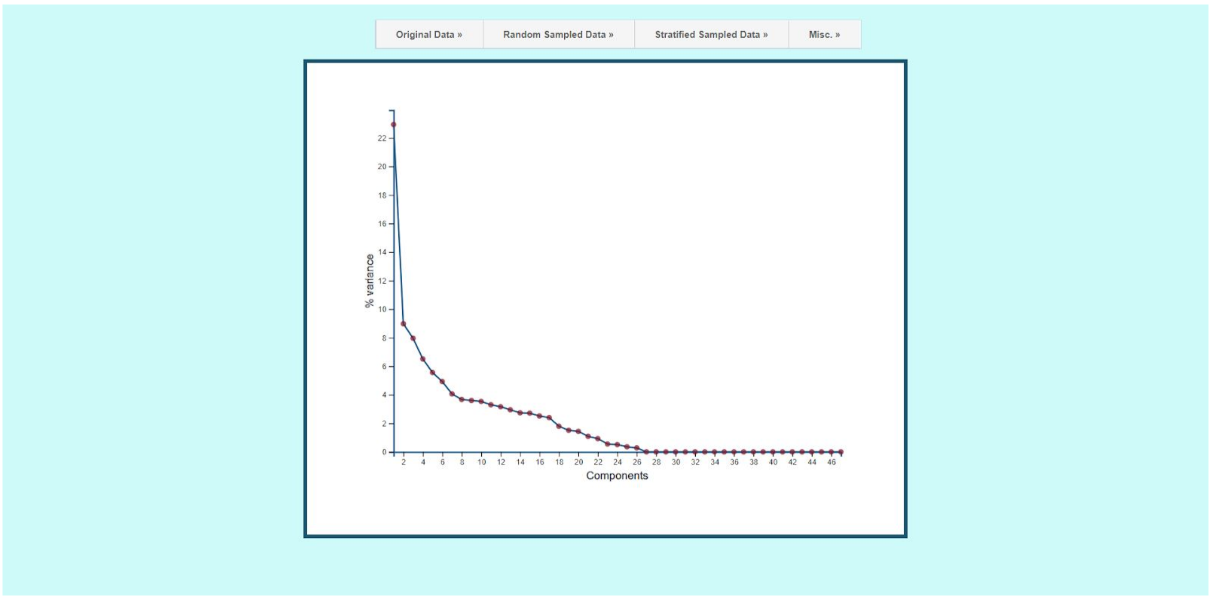


3. <u>Scree plot visualization and intrinsic dimensionality using PCA</u>: For each type of data(original, random sampled and stratified sampled), we first compute variance for each principal component, and store it in a dataframe. We then compute percent variance for each principal component and plot a line chart to display the scree plot.



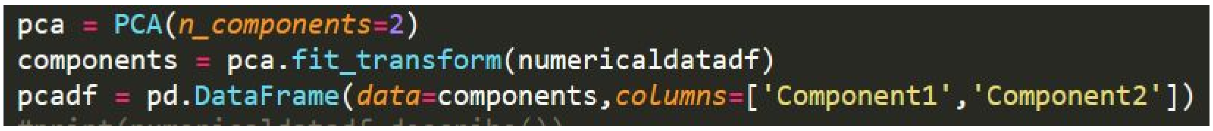
```
print("Working on PCA for Random sampled data...")
pcarandom = PCA()
randomtransform = StandardScaler().fit_transform(randomsamplingdf)
pcarandom.fit(randomtransform)
variance_random['variance'] = pcarandom.explained_variance_ #variance for given value of components
print("Finished Random PCA")
```

From the scree plots, we can see that the intrinsic dimensionality is 8 using the elbow method, and given cumulative percent variance reaches around 75%. We can also see that a bias gets introduced in the scree plots for both random and stratified sampled data.

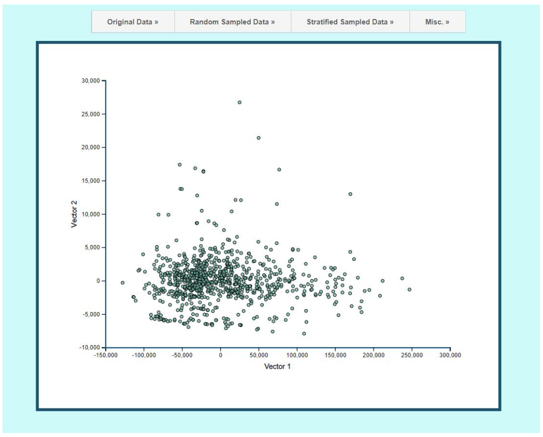


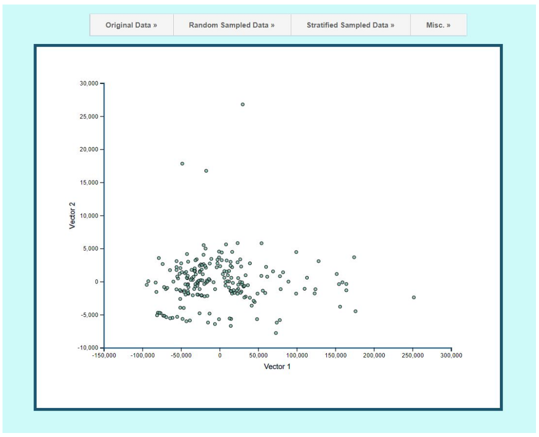
To find three attributes with highest PCA loadings, we compare the sum of squared loadings for all the attributes and choose the top three attributes with highest respective values. On computing, we can see that the three attributes with highest PCA loadings are “BsmtUnfSF”, “GrLivArea” and “2ndFlrSF”.

4. Visualizing top two PCA vectors: We create a 2D scatter plot of data projected into top two PCA vectors for the original data as well as sampled data.

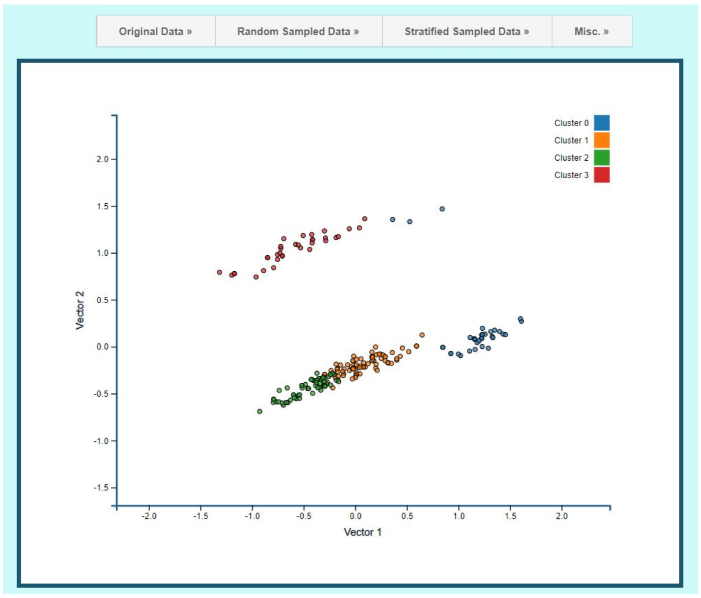


```
pca = PCA(n_components=2)
components = pca.fit_transform(numericaldatadf)
pcadf = pd.DataFrame(data=components,columns=['Component1','Component2'])
```



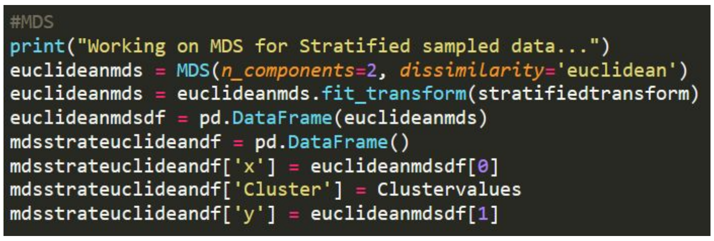


Above are 2d scatterplots of original data and randomly sampled data. From the visualizations, we can clearly see that the number of points on the scatter plot have been reduced for randomly sampled data. We can say that the points in the second graph are a subset of the first graph.



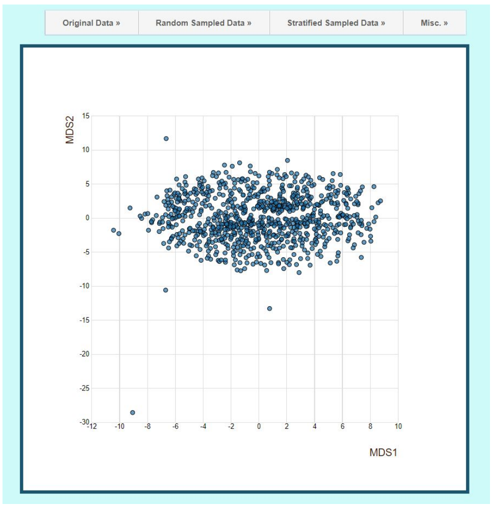
In the 2D scatter plot for stratified sampled data, we can clearly see elements of a single cluster together on the plot, ie. clusters are maintained in the PCA scatter plot too.

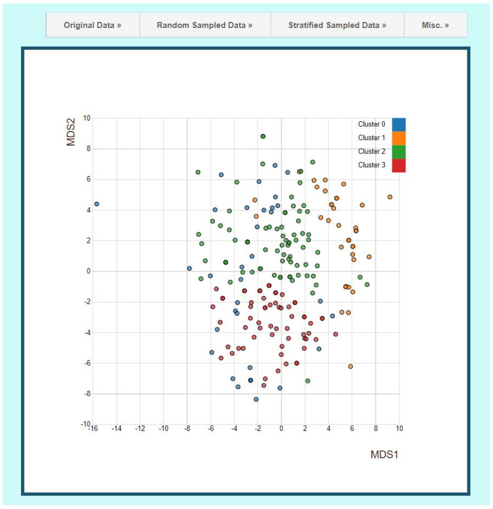
5. Visualizing data using MDS: We are using multidimensional scaling to visualize the data into 2D scatterplots. We have used both euclidean and correlation distance metrics for MDS.



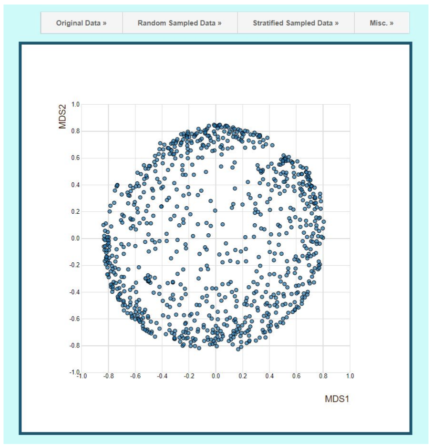
```
#MDS
print("Working on MDS for Stratified sampled data...")
euclideanmds = MDS(n_components=2, dissimilarity='euclidean')
euclideanmds = euclideanmds.fit_transform(stratifiedtransform)
euclideanmdsdf = pd.DataFrame(euclideanmds)
mdsstrateuclideandf = pd.DataFrame()
mdsstrateuclideandf['x'] = euclideanmdsdf[0]
mdsstrateuclideandf['Cluster'] = Clustervalues
mdsstrateuclideandf['y'] = euclideanmdsdf[1]
```

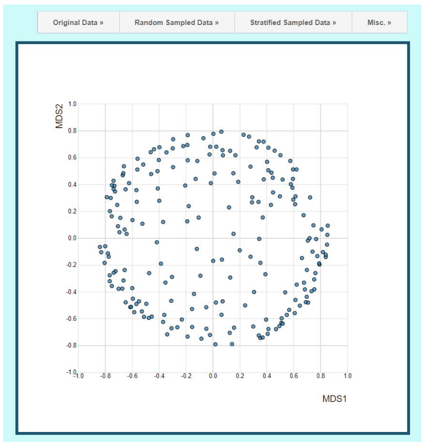
Euclidean MDS looks like the following due to the presence of outliers (see first figure). This has been fixed well in MDS for stratified sampled data (see second figure).

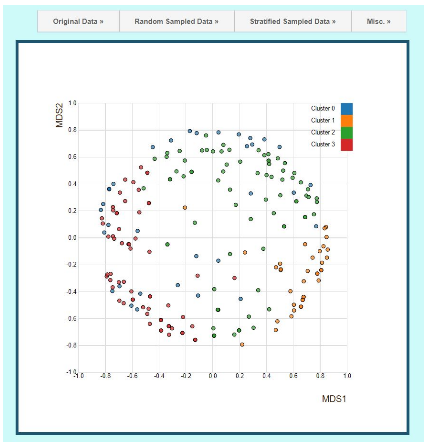




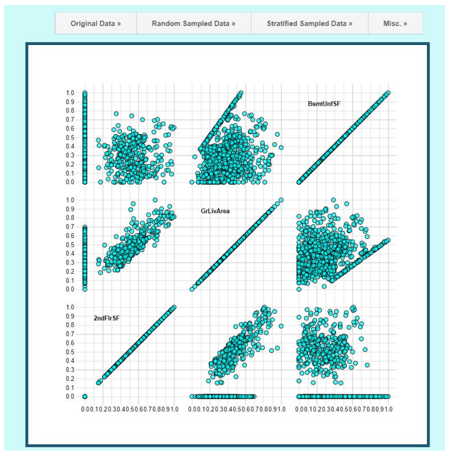
Below are the representations for correlation based MDS for original data, randomly sampled data and stratified sampled data respectively. We can clearly observe that the plots for sampled data have fewer data points as compared to that of original data.

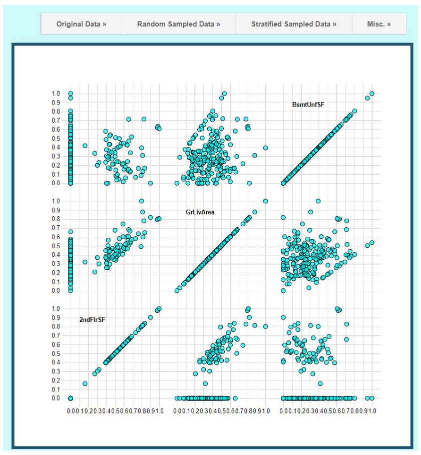


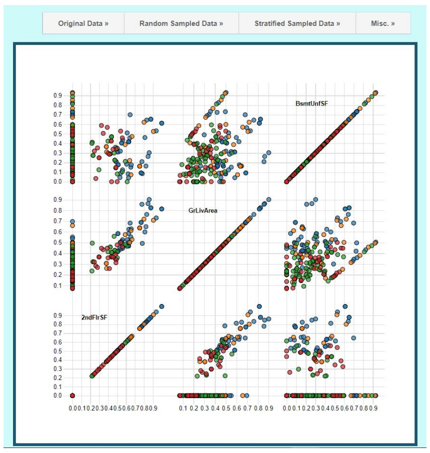




6. Visualizing Scatterplot matrix: We have also visualized a scatterplot matrix for the three highest PCA loaded attributes (“BsmtUnfSF”, “GrLivArea” and “2ndFlrSF”). The matrix offers a good visualization of correlation between each of the two attributes from the set of three attributes.







We can see the scatterplot matrix for original data, random sampled data and stratified sampled data in their respective order. We can clearly see fewer data points plotted for random and stratified sampled data as compared to the original data.

## Code set up and execution:

The code will require a python flask server to execute. Please follow the following steps to set up and run the code:

First Time setup:

1. Install python, pip
2. Do “pip install virtualenv” and “pip install virtualenvwrapper-win”
3. Extract the zip to some local folder CSE564_Project2.
4. Go to the Project folder from cmd (cd CSE564_Project2)
5. Run “mkvirtualenv CSE564_Project2” on cmd.
6. Run “setprojectdir .” on cmd.
7. Run “pip install flask” on cmd.

8. Install any necessary python packages using pip (pandas, sklearn, matplotlib).
9. Execute “python app.py”
10. Go to http://127.0.0.1:5000/ after the server initializes

Normal code run:

1. Go to the Project folder from cmd (cd CSE564_Project2)
2. Execute “workon CSE564_Project2”
3. Execute “python app.py”
4. Go to http://127.0.0.1:5000/ after the server initializes

Please refer to the video demonstration of the project here.