Comparing Selected Clustering Algorithms with Generated Datasets

Abhijeet Sahdev (as4673)
Submitted to: Dr. Jason T.L. Wang

November 16, 2024

Contents

1 Packages

```
import pandas as pd
import docx2txt
from io import StringIO
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np
import seaborn as sns
import random
import kagglehub
import requests
from sklearn.metrics import silhouette_score, silhouette_samples
import plotly.express as px
```

2 Helper Functions

```
def pCA(X, n_components):
    X_mean = np.mean(X, axis=0)
    X_centered = X - X_mean
    cov_matrix = np.cov(X_centered, rowvar=False)
    eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
    idx = eigenvalues.argsort()[::-1]
    eigenvalues = eigenvalues[idx]
    eigenvectors = eigenvectors[:, idx]
    eigenvectors = eigenvectors[:, :n_components]
    X_pca = np.dot(X_centered, eigenvectors)
    return X_pca, eigenvalues, eigenvectors

def euclideanDistance(a, b):
  return np.sqrt(np.sum((a - b)**2))

def arrayDistances(data):
  distances = set()
  for i in range(len(data)):
    for j in range(i+1, len(data)):
      distance = euclideanDistance(data[i], data[j])
      distances.add(distance)
  return [np.array(distance) for distance in distances]

def calculateIQR(data):
  q1, q3 = np.percentile(data, [25, 75])
  iqr = q3 - q1
  return iqr
```

```
def removeOutliers(data):
  outliersSet = set()
  distances = arrayDistances(data)
  q1, q3 = np.percentile(distances, [25, 75])
  iqr = q3 - q1
  lowerBound = q1 - 1.5 * iqr
  upperBound = q3 + 1.5 * iqr
  for i in range(len(data)):
    for j in range(i+1, len(data)):
      distance = euclideanDistance(data[i], data[j])
      if distance < lowerBound or distance > upperBound:
        outliersSet.add(tuple(data[i]))
        outliersSet.add(tuple(data[j]))
  return outliersSet

def computeSilhouetteScore(df, method):
  for column in df.columns:
    if 'clusters' in column:
      labels = df[column]
      if all(col in df.columns for col in ['X (Seconds)', 'Y
         (Hours)', 'Z (Minutes)']):
        data = df[['X (Seconds)', 'Y (Hours)', 'Z (Minutes)']].values
        score = silhouette_score(data, labels, random_state=33)
        experiments.append(f'{df.name}_{column}')
        scores.append(score)

def selectRandomVectors(data, k):
  selectedVectors = []
  selectedIndices = set()
  while len(selectedVectors) < k:
      randomIndex = random.randint(0, len(data) - 1)
      if randomIndex not in selectedIndices:
        selectedVectors.append(data[randomIndex])
        selectedIndices.add(randomIndex)
  return selectedVectors
```

3 Data Generation and Wrangling

Data generated using Grok along with the code. You may view the datasets at the following links df and df2. Note that 500 lines of data can not be displayed on an Overleaf document.

```
genData = docx2txt.process('/content/drive/MyDrive/
dm634/generatedUsingGrok.docx')
df = pd.read_csv(StringIO(genData),sep=',')
df2 = df.copy()
df2['Z (Minutes)'] = df2['Y (Hours)']*60 + df2['X (Seconds)']/60
```

Kaggle's F1 dataset was processed as follows:

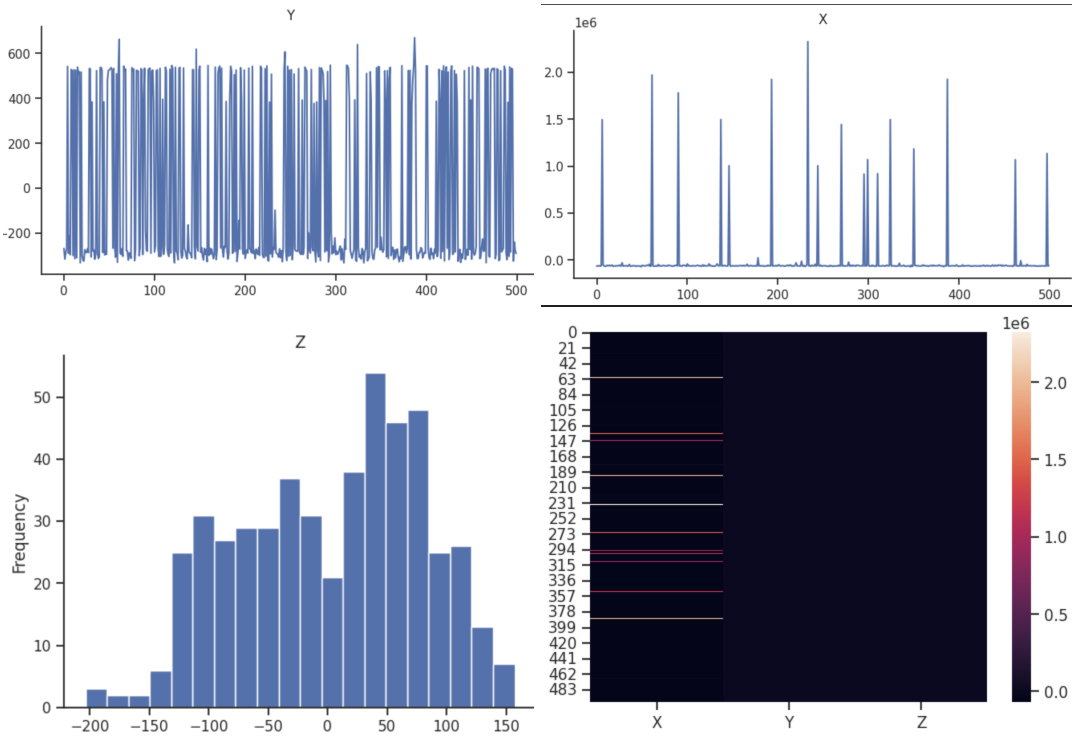
```
path = kagglehub.dataset_download("rohanrao/formula-1-world-
championship-1950-2020")
print('Path ',path)

pitStops = path + '/pit_stops.csv'
fPS = pd.read_csv(pitStops)

races = path + '/races.csv'
fR = pd.read_csv(races)

f0 = fR.merge(fPS, on='raceId', how='right')

#Dropping all non interger type columns
for column in f0.columns:
  if f0[column].dtype != 'int64':
    f0.drop(column, axis=1, inplace=True)

f0 = f0.to_numpy()
f0Reduced = pca(f0, 3)[0]

np.random.seed(33)
f0Reduced = selectRandomVectors(f0Reduced, 500)

fdf = pd.DataFrame(f0Reduced, columns=['X (Seconds)', 'Y (Hours)',
   'Z (Minutes)'])
fdf.head()

sns.set_theme(style="ticks")
sns.heatmap(fdf)
```

Output:
Path /root/.cache/kagglehub/datasets/rohanrao /formula-1-world-championship-1950-2020/versions/23

Resulting dataset is here.

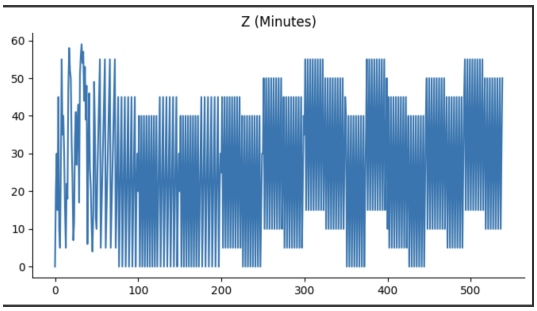
3.1 EDA

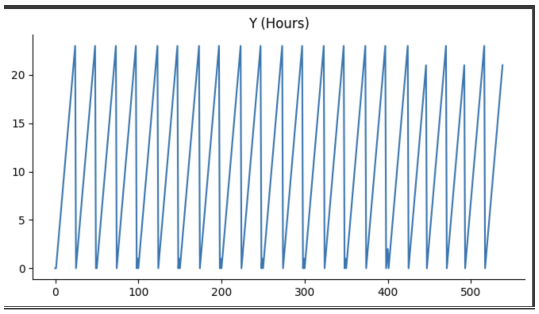
3.1.1 F1

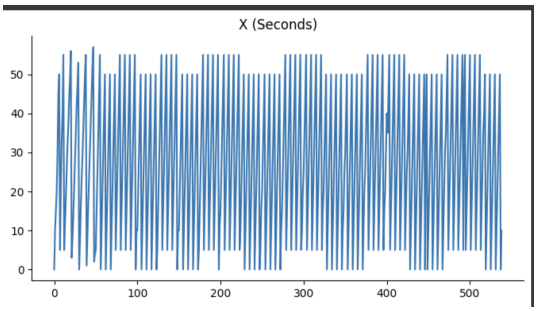


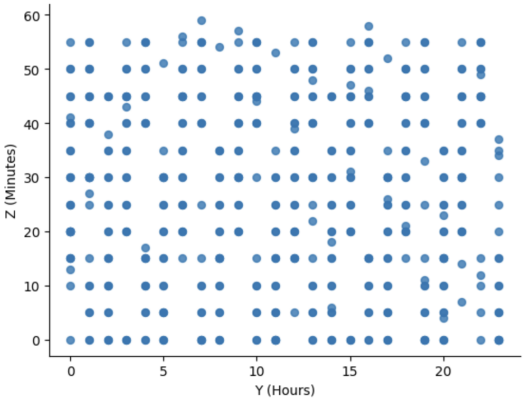

```
fdf['Y'].plot(kind='line', figsize=(8, 4), title='Y')
plt.gca().spines[['top', 'right']].set_visible(False)

fdf['X'].plot(kind='line', figsize=(8, 4), title='X')
plt.gca().spines[['top', 'right']].set_visible(False)

fdf['Z'].plot(kind='hist', bins=20, title='Z')
plt.gca().spines[['top', 'right',]].set_visible(False)

sns.set_theme(style="ticks")
sns.heatmap(fdf)
```

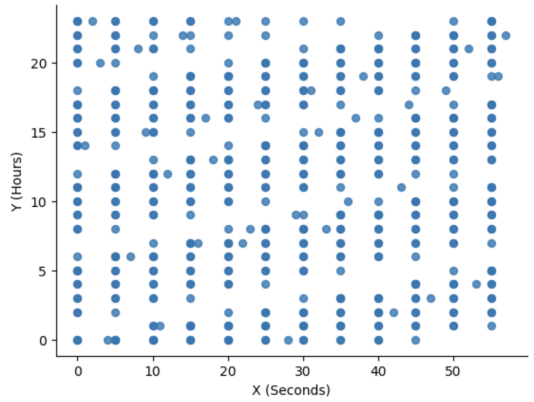
3.1.2 Df

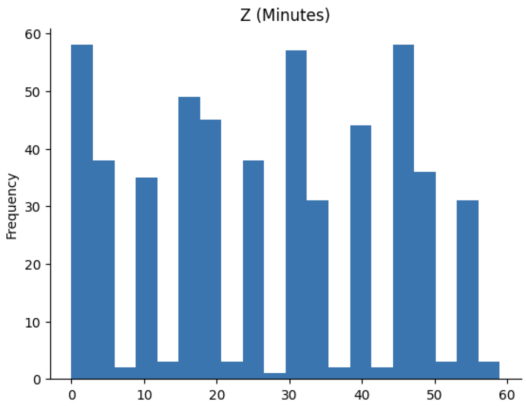

Z (Minutes)

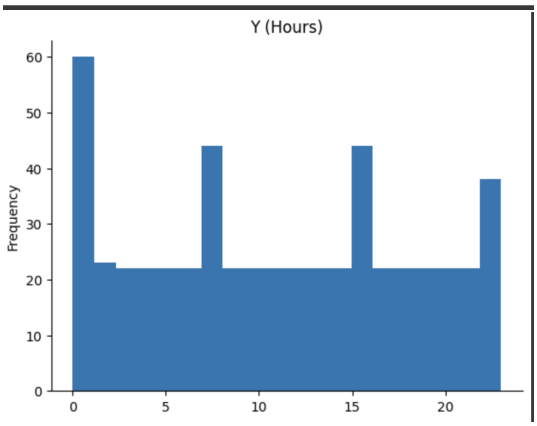

Y (Hours)

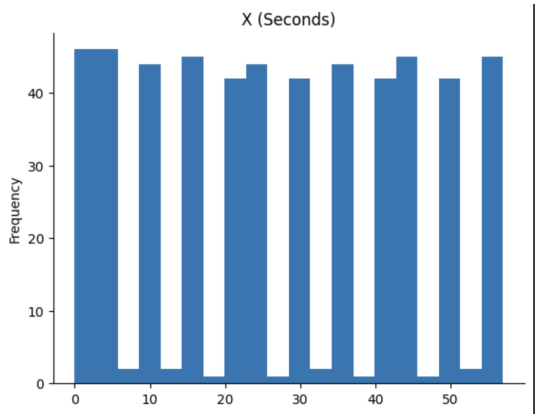

X (Seconds)

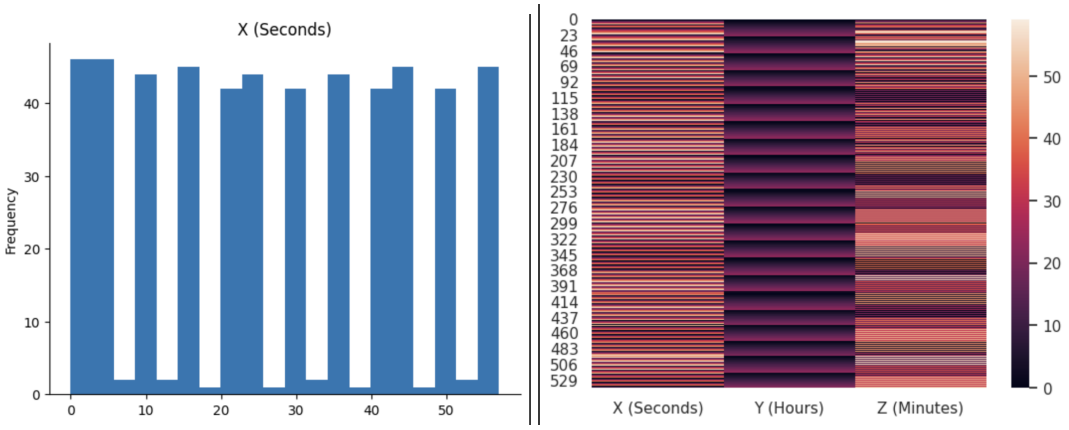

Z (Minutes)
Y (Hours)

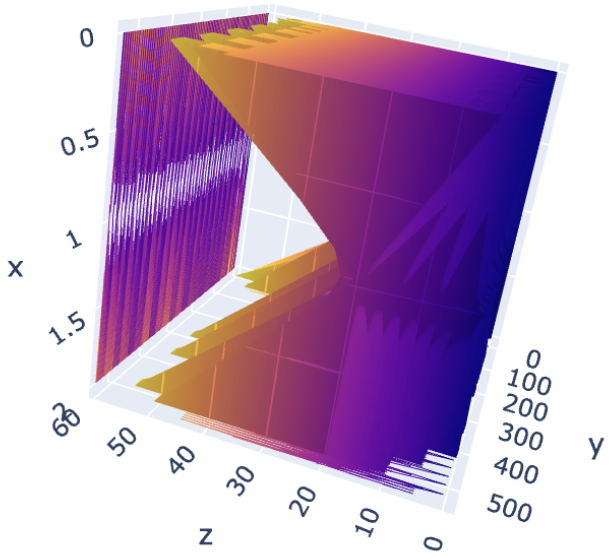

Y (Hours)
X (Seconds)


Z (Minutes)
Frequency


Y (Hours)
Frequency


X (Seconds)
Frequency

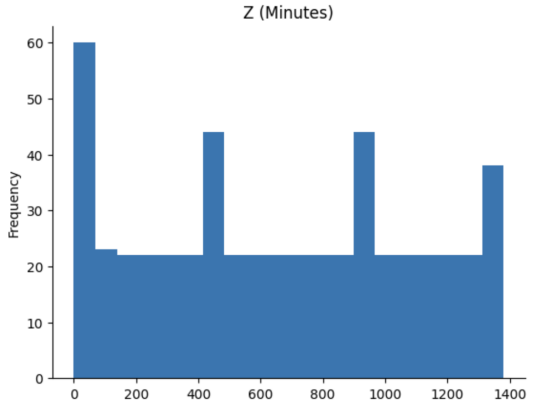


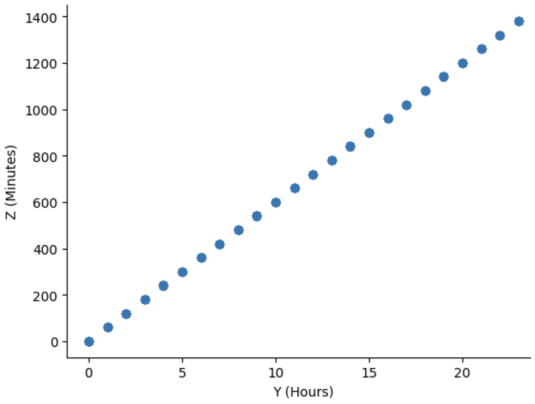


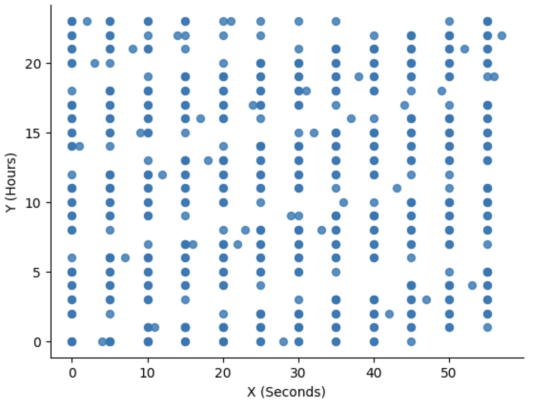

```
df['Z (Minutes)'].plot(kind='line', figsize=(8, 4), title='Z
   (Minutes)')
plt.gca().spines[['top', 'right']].set_visible(False)

df['Y (Hours)'].plot(kind='line', figsize=(8, 4), title='Y (Hours)')
plt.gca().spines[['top', 'right']].set_visible(False)

df['X (Seconds)'].plot(kind='line', figsize=(8, 4), title='X
   (Seconds)')
plt.gca().spines[['top', 'right']].set_visible(False)

df.plot(kind='scatter', x='Y (Hours)', y='Z (Minutes)', s=32,
   alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

df.plot(kind='scatter', x='X (Seconds)', y='Y (Hours)', s=32,
   alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

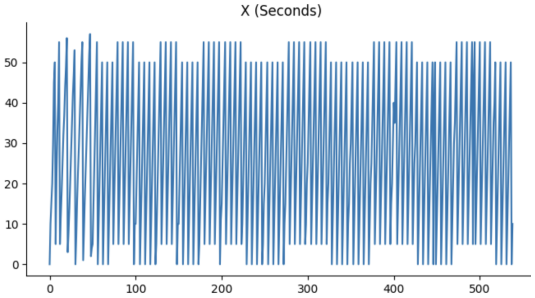
```
df['Z (Minutes)'].plot(kind='hist', bins=20, title='Z (Minutes)')
plt.gca().spines[['top', 'right',]].set_visible(False)

df['Y (Hours)'].plot(kind='hist', bins=20, title='Y (Hours)')
plt.gca().spines[['top', 'right',]].set_visible(False)

df['X (Seconds)'].plot(kind='hist', bins=20, title='X (Seconds)')
plt.gca().spines[['top', 'right',]].set_visible(False)

sns.set_theme(style="whitegrid")
sns.heatmap(df)

fig = go.Figure(data=[go.Surface(z=df.values)])
fig.update_traces(contours_z=dict(show=True, usecolormap=True,
                                  highlightcolor="limegreen",
                                     project_z=True),
                                     contours_x=dict(show=True,
                                     usecolormap=True,
                                  highlightcolor="pink",
                                     project_x=True), )
fig.update_layout(title='Surface Plot', autosize=False,
                  scene_camera_eye=dict(y=2.87,z=-0.64,x=-1),
                  width=1000, height=500,
                  margin=dict(l=65, r=50, b=65, t=90)
)

fig.show()
```

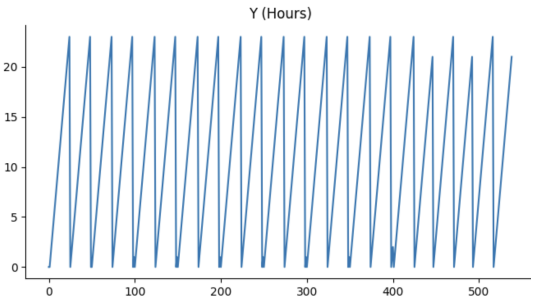
3.2 Df2

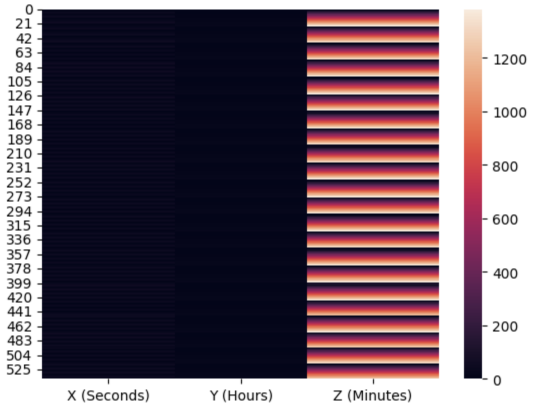

Z (Minutes)
Frequency

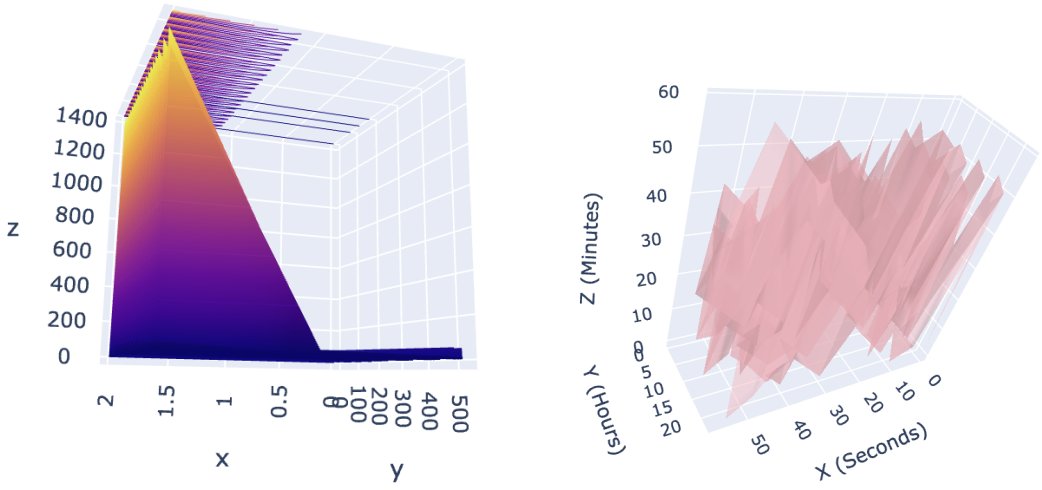

Z (Minutes)
Y (Hours)


Y (Hours)
X (Seconds)


X (Seconds)


Y (Hours)


X (Seconds)
Y (Hours)
Z (Minutes)




```
fig = go.Figure(data=[go.Surface(z=df2.values)])
fig.update_traces(contours_z=dict(show=True, usecolormap=True,
   highlightcolor="limegreen",project_z=True))
fig.update_layout(title='Surface Plot', autosize=False,
                  scene_camera_eye=dict(y=2.87,z=-0.64,x=1),
                  width=1000, height=500,
                  margin=dict(l=65, r=50, b=65, t=90))
fig.show()

df2['Z (Minutes)'].plot(kind='hist', bins=20, title='Z (Minutes)')
plt.gca().spines[['top', 'right',]].set_visible(False)

df2.plot(kind='scatter', x='Y (Hours)', y='Z (Minutes)', s=32,
   alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

df2.plot(kind='scatter', x='X (Seconds)', y='Y (Hours)', s=32,
   alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

df2['X (Seconds)'].plot(kind='line', figsize=(8, 4), title='X
   (Seconds)')
plt.gca().spines[['top', 'right']].set_visible(False)

df2['Y (Hours)'].plot(kind='line', figsize=(8, 4), title='Y (Hours)')
plt.gca().spines[['top', 'right']].set_visible(False)

sns.heatmap(df2)

x = df['X (Seconds)'].to_list()
y = df['Y (Hours)'].to_list()
z = df['Z (Minutes)'].to_list()
```

```
fig = go.Figure(data=[go.Mesh3d(x=x, y=y, z=z, color='lightpink',
   opacity=0.50,)], layout=go.Layout(scene=dict(xaxis_title='X
   (Seconds)', yaxis_title='Y (Hours)', zaxis_title='Z (Minutes)')))
fig.show()
```

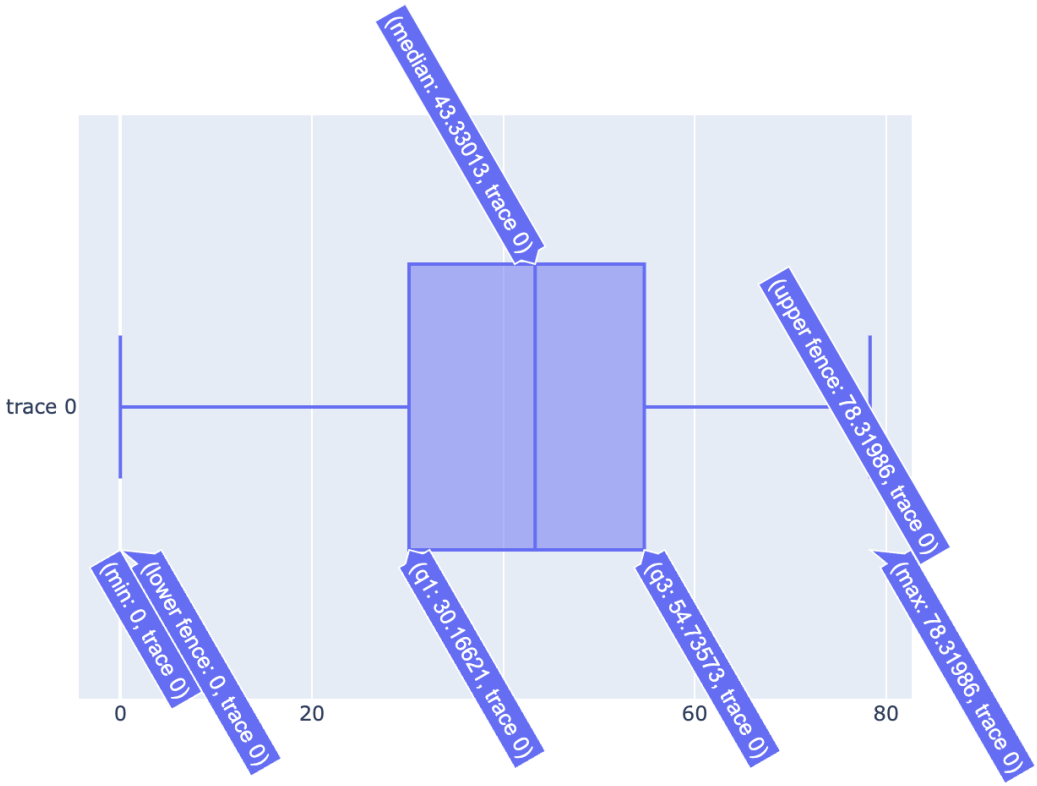
4 Removing Outliers

4.1 Dataset 1

```
dataArray = df.to_numpy()
distances = arrayDistances(dataArray)
fig = go.Figure(data=[go.Box(x=distances)])
fig.show()
print(len(outliers))
outliers = removeOutliers(dataArray)
print(len(outliers))
```

Output: 24.556671441748193, 0

4.1.1 Image

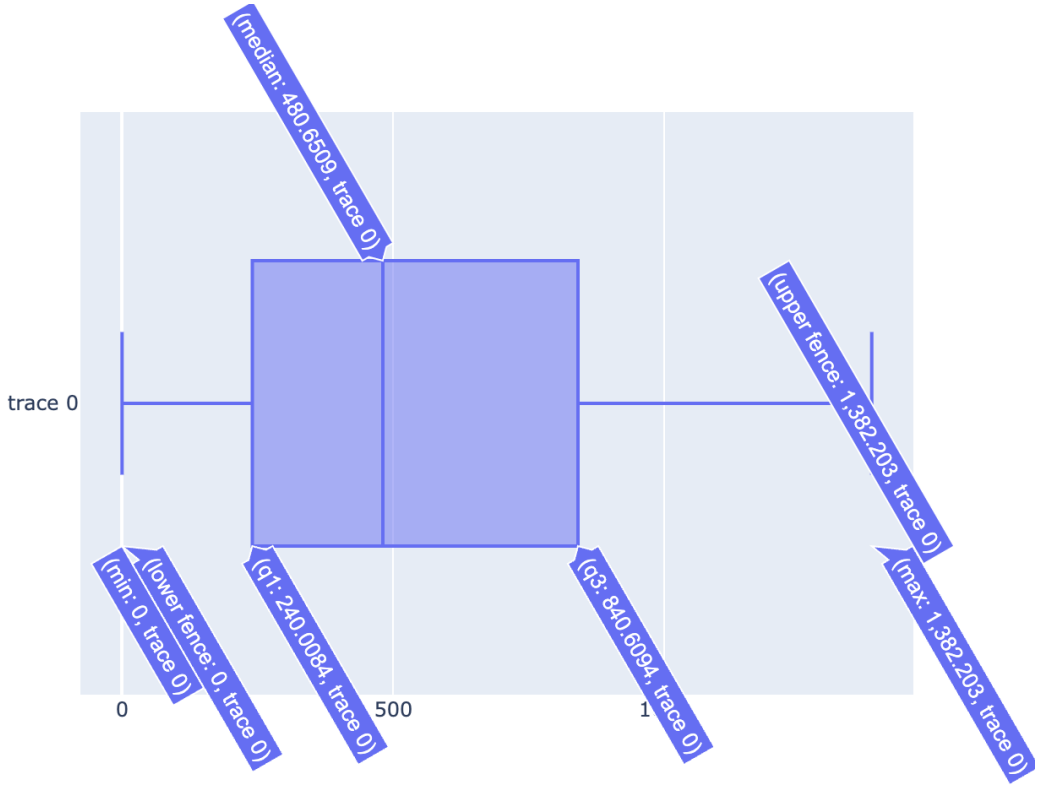


4.2 Dataset 2

```
dataArray = df2.to_numpy()
distances = arrayDistances(dataArray)
fig = go.Figure(data=[go.Box(x=distances)])
fig.show()
print(len(outliers))
outliers = removeOutliers(dataArray)
print(len(outliers))
```

Output: 600.6010023387444, 0

4.2.1 Image

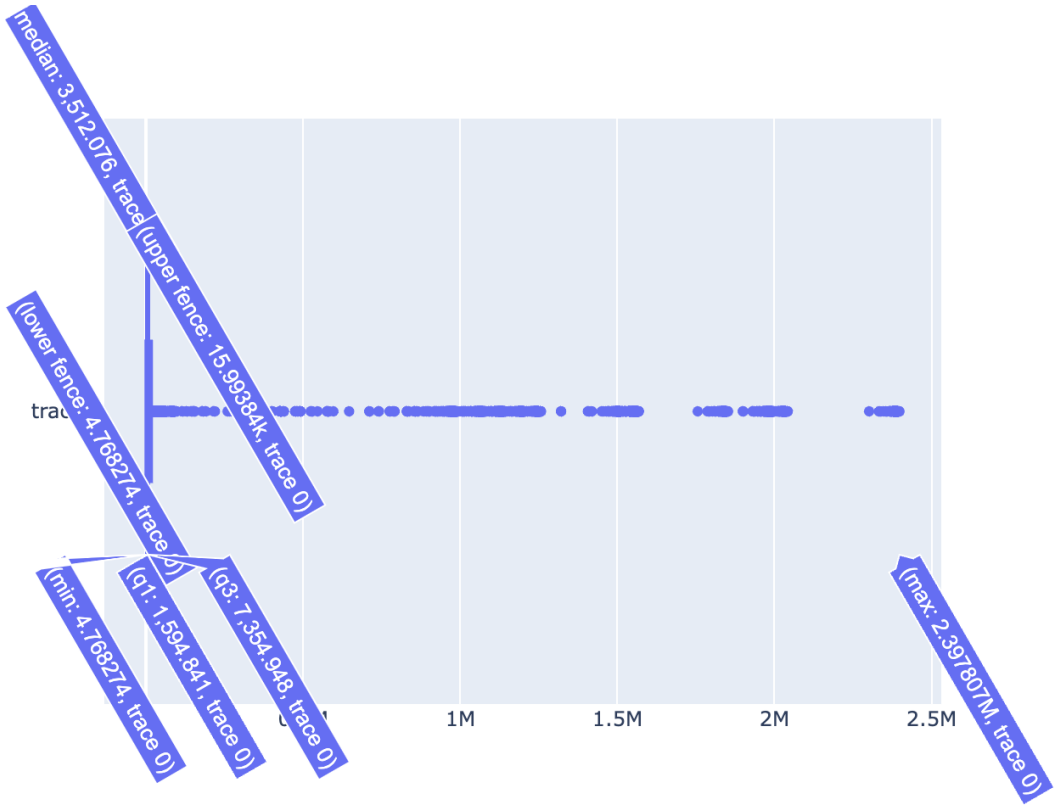


4.3 Dataset 3

```
dataArray = f0.to_numpy()
distances = arrayDistances(dataArray)
fig = go.Figure(data=[go.Box(x=distances)])
fig.show()
print(len(outliers))
outliers = removeOutliers(dataArray)
print(len(outliers))
```

Output: 5760.0896292870675, 500 - taken as 0 since that is the length of the entire dataset. This dataset has imaginary values.

4.3.1 Image



5 Placeholders

```
dfKmeans = df.copy()
df2Kmeans = df2.copy()
fdfKmeans = fdf.copy()

dataframesForKmeans = [dfKmeans, df2Kmeans, fdfKmeans]

dfAgglomerative = df.copy()
df2Agglomerative = df2.copy()
fdfAgglomerative = fdf.copy()

dataframesForAgglomerative = [dfAgglomerative, df2Agglomerative,
   fdfAgglomerative]
```

6 Clustering Algorithms

6.1 Kmeans

6.1.1 Code

```
def kmeans(data, k, max_iterations=100):
  random.seed(33)
  centroids = selectRandomVectors(data,k) #random initialization
  for _ in range(max_iterations):
    clusters = [[] for _ in range(k)]

    # assigning tuples to nearest centroid
    for point in data:
      distances = [euclideanDistance(point, centroid) for centroid
         in centroids]
      clusterAssignment = distances.index(min(distances))
      clusters[clusterAssignment].append(point)

    # update centroids if a cluster is edited during the assignment
       loop
    newCentroids = []
    for cluster in clusters:
      if cluster:
        newCentroid = [sum(coord) / len(cluster) for coord in
           zip(*cluster)]
        newCentroids.append(newCentroid)
      else:
        newCentroids.append(centroids[clusters.index(cluster)])

    # convergence check
    if np.array_equal(np.array(centroids), np.array(newCentroids)):
      break

    centroids = newCentroids

  # assignLabels
  clusterLabels = []
  for point in data:
    distances = [euclideanDistance(point, centroid) for centroid in
       centroids]
    clusterLabels.append(distances.index(min(distances)))

return clusterLabels

for k in range(2, 7):
  for dataframe in dataframesForKmeans:
    clusterLabels = kmeans(dataframe.to_numpy(), k)
    dataframe[f'{k} clusters'] = clusterLabels

dfKmeans.to_csv('/content/drive/MyDrive/dm634/
dfKmeans.csv',index=True)
df2Kmeans.to_csv('/content/drive/MyDrive/dm634/
```

```
df2Kmeans.csv',index=True)
fdfKmeans.to_csv('/content/drive/MyDrive/dm634/
fdfKmeans.csv',index=True)
```

6.1.2 Output

6.2 Hierarchial Agglomerative

6.2.1 Code

```
def calculateDistance(cluster1, cluster2, linkage):
c1 = np.array(cluster1)
c2 = np.array(cluster2)
if linkage == 'single':
return np.min([euclideanDistance(p1, p2) for p1 in c1 for p2 in c2])
elif linkage == 'complete':
return np.max([euclideanDistance(p1, p2) for p1 in c1 for p2 in c2])
elif linkage == 'average':
return np.mean([euclideanDistance(p1, p2) for p1 in c1 for p2 in c2])
elif linkage == 'centroid':
center1 = np.mean(c1, axis=0)
center2 = np.mean(c2, axis=0)
return euclideanDistance(center1, center2)
else:
  raise ValueError("Unknown linkage method")

def agglomerativeClustering(data, k=7, linkage='average'):
# Start with each point as its own cluster
clusters = {i: [point] for i, point in enumerate(data)}

# Keep track of cluster assignments
labels = list(range(len(data)))

while len(clusters) > k:
min_distance = np.inf
to_merge = (0, 1)
for i in range(len(clusters)):
  for j in range(i + 1, len(clusters)):
    distance = calculateDistance(clusters[i], clusters[j], linkage)
    if distance < min_distance:
      min_distance = distance
      to_merge = (i, j)

# Merge clusters and update labels
clusters[to_merge[0]] = clusters[to_merge[0]] + clusters[to_merge[1]]
for point in clusters[to_merge[1]]:
```

```
    labels[data.tolist().index(point.tolist())] = to_merge[0]  #
        Update label to merged cluster index

del clusters[to_merge[1]]

# Map labels to be 0, 1, ..... k-1
unique_labels = sorted(list(set(labels)))
remapped_labels = {old: new for new, old in enumerate(unique_labels)}
labels = [remapped_labels[label] for label in labels]

return labels

for k in range(2, 7):
for linkage in ['single','average','complete','centroid']:
for dataframe in dataframesForAgglomerative:
  clusterLabels = kmeans(dataframe.to_numpy(), k)
  dataframe[f'{k} {linkage} clusters'] = clusterLabels

dfAgglomerative.to_csv('/content/drive/MyDrive/dm634
/dfAgglomerative.csv',index=True)
df2Agglomerative.to_csv('/content/drive/MyDrive/dm634
/df2Agglomerative.csv',index=True)
fdfAgglomerative.to_csv('/content/drive/MyDrive/dm634
/fdfAgglomerative.csv',index=True)
```

6.2.2 Output

Various output files for k between (1,7) for Kmeans and HAC along with four different linkages that are, single, complete, average and centroid are as follows:

- Kmeans Clustering: fdf, df and df2.
- Hierarchial Agglomerative Clustering: df2, df and fdf.

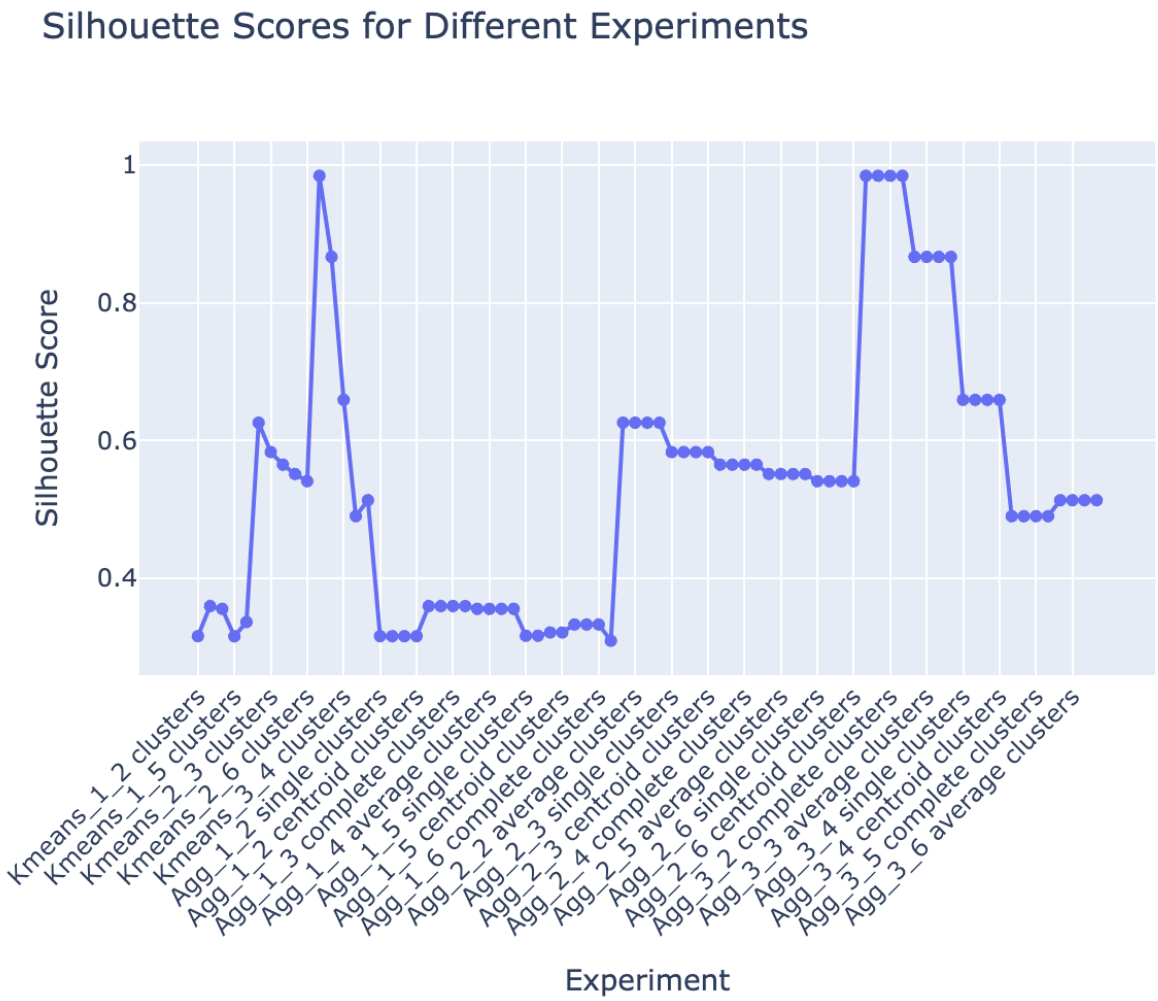
7 Comparision Based on Mean Silhouette Values

In total, **75** experiments were conducted to generate the results discussed in the following subsections. Moreover, to help with reproducibility, random seeds were used along with storing results with assigned labels during labeling in the above links.

```
dataframesForKmeans = [dfKmeans, df2Kmeans, fdfKmeans]
dataframesForAgglomerative = [dfAgglomerative, df2Agglomerative,
   fdfAgglomerative]

experiments = []
scores = []
```

```
for i, df in enumerate(dataframesForKmeans, 1):
  df.name = f'Kmeans_{i}'
  computeSilhouetteScore(df, 'Kmeans')

for i, df in enumerate(dataframesForAgglomerative, 1):
    df.name = f'Agg_{i}'
    computeSilhouetteScore(df, 'Agg')

cdf = pd.DataFrame({
    'Experiment': experiments,
    'Silhouette Score': scores
})

fig = px.line(cdf, x='Experiment', y='Silhouette Score',
   title='Silhouette Scores for Different Experiments', markers=True)
fig.update_traces(hovertemplate='Experiment: %{x}<br>Silhouette
   Score: %{y}')
fig.update_layout(xaxis_tickangle=-45, xaxis_title='Experiment',
   yaxis_title='Silhouette Score')
fig.show()

cdf['Dataset'] = cdf['Experiment'].str.split('_').str[1]
cdf['Algorithm'] = cdf['Experiment'].str.split('_').str[0]

bestScoresForEachDataset = cdf.groupby('Dataset')['Silhouette
   Score'].agg(['max', 'idxmax'])
bestScoresForEachAlgorithmPerDataset = cdf.groupby(['Algorithm',
   'Dataset'])['Silhouette Score'].agg(['max', 'idxmax'])

print("Best Scores for Each Dataset:")
for dataset, (maxScore, index) in
   bestScoresForEachDataset.iterrows():
  bestExperiment = cdf.loc[index, 'Experiment']
  print(f"  Dataset {dataset}: Best Silhouette Score =
     {maxScore:.8f}, Experiment = {bestExperiment}")

print("\nBest Scores for Each Algorithm Per Dataset:")
for (algorithm, dataset), (maxScore, index) in
   bestScoresForEachAlgorithmPerDataset.iterrows():
  bestExperiment = cdf.loc[index, 'Experiment']
  print(f"  Algorithm: {algorithm}, Dataset {dataset}: Best
     Silhouette Score = {maxScore:.8f}, Experiment =
     {bestExperiment}")
```

7.1 Graph



7.2 Reported Values

Best Scores for Each Dataset:
Dataset 1: Best Silhouette Score = 0.35944931, Experiment = Kmeans_1_3 clusters Dataset 2: Best Silhouette Score = 0.62581555, Experiment = Kmeans_2_2 clusters Dataset 3: Best Silhouette Score = 0.98445955, Experiment = Agg_3_2 single clusters

Best Scores for Each Algorithm Per Dataset: Algorithm: Agg, Dataset 1: Best Silhouette Score = 0.35944931, Experiment = Agg_1_3 single clusters Algorithm: Agg, Dataset 2: Best Silhouette Score = 0.62581555, Experiment = Agg_2_2 single clusters Algorithm: Agg, Dataset 3: Best Silhouette Score = 0.98445955, Experiment = Agg_3_2 single clusters Algorithm: Kmeans, Dataset 1: Best Silhouette Score = 0.35944931, Experiment = Kmeans_1_3 clusters Algorithm: Kmeans, Dataset 2: Best Silhouette Score = 0.62581555, Experiment = Kmeans_2_2 clusters Algorithm: Kmeans, Dataset 3: Best Silhouette Score = 0.98445955, Experiment = Kmeans_3_2 clusters

where, HAC experiments are represented as *Algorithm_Dataset number_NumberOfClusters typeOfLinking* and Kmeans experiments as: *Algorithm_Dataset number_NumberOfClusters*.