N PROGRESS - Clustering the NASA Exoplanet Archive With K-Means to Find New Earth Candidates

In [1]:
```
import numpy as np
import pandas as pd
from matplotlib.pyplot import *
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.cm as cm
import matplotlib.pyplot as plt
%matplotlib inline, %matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Read exoplanet dataset ###
df = pd.read_csv('/kaggle/input/nasa-exoplanets-data/')
df.shape
```

Out[2]:

	pl_name	hostname	pl_letter	pl_refname	pl_orbper	pl_orbsmax	pl_orbeccen	pl_orbinclin	pl_orbtperd	...	pl_maxmass3	pl_maxmdn3	pl_maxmass4	pl_maxmdn4
0	11 Com b	11 Com	b	Netto et al. (2020)	326.03	1.29	0.231	109.96	2455117.2	...	NaN	NaN	NaN	NaN
1	11 Oph b	11 Oph	b	Sana et al. (2014)	1739.5	3.09	0.08	NaN	NaN	...	NaN	NaN	NaN	NaN
2	11 UMi b	11 UMi	b	Wittenmyer et al. (2017)	516.22	1.53	0.08	121.03	2457097.0	...	NaN	NaN	NaN	NaN
3	14 And b	14 And	b	Wittenmyer et al. (2007)	185.84	0.83	0.15	NaN	2450138.6	...	NaN	NaN	NaN	NaN
4	14 Her b	14 Her	b	Wittenmyer et al. (2007)	1912.1	3.35	0.37	147.6	2450331.0	...	NaN	NaN	NaN	NaN
5	16 Cyg B b	16 Cyg B	b	Lewis et al. (2014)	798.5	1.68	0.68	116.76	2456642.6	...	NaN	NaN	NaN	NaN
6	17 UMa b	17 UMa	b	Vogt et al. (2014)	1079.1	2.15	0.43	153.4	2456811.2	...	NaN	NaN	NaN	NaN
7	18 Del b	18 Del	b	Lee et al. (2014)	994.6	2.0	0.0	38.0	2456945.0	...	NaN	NaN	0.000	0.000
8	19 Dra b	19 Dra	b	Jones et al. (2016)	625.18	1.39	0.28	88.86	2457504.4	...	NaN	NaN	0.000	0.000
9	1 Leo b	1 Leo	b	Trifonov et al. (2019)	437.6	1.29	0.14	108.1	2457099.6	...	NaN	NaN	NaN	NaN
10	1 Pic b	1 Pic	b	Nowak et al. (2020)	3.35	0.0203	0.07	89.1	2457849.0	...	NaN	NaN	NaN	NaN
11	2 Cnc b	2 Cnc	b	Fischer et al. (2016)	437.5	1.94	0.12	154.0	2457024.9	...	NaN	NaN	NaN	NaN
12	2 Cnc c	2 Cnc	c	Fischer et al. (2016)	5360.1	6.0	0.34	154.0	2457024.9	...	NaN	NaN	NaN	NaN
13	2 Cnc d	2 Cnc	d	Fischer et al. (2016)	261.9	0.82	0.12	154.0	2457024.9	...	NaN	NaN	NaN	NaN
14	2 Cnc e	2 Cnc	e	Fischer et al. (2016)	2.2	0.0153	0.12	154.0	2457024.9	...	NaN	NaN	NaN	NaN
15	2 Cnc f	2 Cnc	f	Fischer et al. (2016)	4909.0	7.0	0.09	154.0	2457024.9	...	NaN	NaN	NaN	NaN
16	2 Cnc g	2 Cnc	g	Fischer et al. (2016)	10000.0	9.0	0.09	154.0	2457024.9	...	NaN	NaN	NaN	NaN
17	2 Cnc h	2 Cnc	h	Fischer et al. (2016)	1.4	0.0244	0.09	154.0	2457024.9	...	NaN	NaN	NaN	NaN
18	2 MASS J0746+2000 B b	2 MASS J0746+2000 B	b	Bardalez Gagliuffi et al. (2019)	3.3	0.0	0.0	90.0	2458500.0	...	NaN	NaN	NaN	NaN
19	2 MASS J0850+1705 A b	2 MASS J0850+1705 A	b	Bardalez Gagliuffi et al. (2019)	1.8	0.0	0.0	90.0	2458500.0	...	NaN	NaN	NaN	NaN

5 rows × 26 columns

In [3]:
```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4879 entries, 0 to 4878
Data columns (total 26 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   pl_name     4879 non-null   object
 1   hostname    4879 non-null   object
 2   pl_letter   4879 non-null   object
 3   pl_refname  4879 non-null   object
 4   pl_orbper   4592 non-null   float64
 5   pl_orbsmax  3918 non-null   float64
 6   pl_orbeccen 3215 non-null   float64
 7   pl_orbinclin 3211 non-null  float64
 8   pl_orbtperd 2455 non-null   float64
 9   pl_orbtperd_lim 2455 non-nul float64
 10  pl_rvamp    1115 non-null   float64
 11  pl_radj     2197 non-null   float64
 12  pl_radjerr1 2197 non-null   float64
 13  pl_radjerr2 2197 non-null   float64
 14  pl_radjlim  2197 non-null   float64
 15  pl_bmasse   3515 non-null   float64
 16  pl_bmasseerr1 3515 non-null float64
 17  pl_bmasseerr2 3515 non-null float64
 18  pl_bmasselim 3547 non-null  float64
 19  pl_maxmass3 3011 non-null   float64
 20  pl_maxmass4 3011 non-null   float64
 21  pl_maxmass5 3011 non-null   float64
 22  pl_orbsmaxlim 2679 non-null float64
 23  pl_rvamp3   1791 non-null   float64
 24  pl_maxmass1 1791 non-null   float64
 25  pl_maxmass2 1791 non-null   float64
dtypes: float64(21), object(5)
memory usage: 9.9 MB
```

```
### drop rows with null values ###
df.dropna(inplace=True)

### keep only relevant columns ###
columns_to_keep = ['pl_name', 'hostname', 'pl_orbper', 'pl_orbsmax',
                   'pl_radj', 'pl_bmasse', 'pl_orbeccen', 'pl_eqt']
df = df[columns_to_keep]

df.head()
```

Out[3]:

	pl_name	hostname	pl_orbper	pl_orbsmax	pl_radj	pl_bmasse	pl_orbeccen	pl_eqt
163	55 Cnc e	55 Cnc	0.736544	0.01544	0.1841	12.930	0.041	1958.0
186	CoRoT-1 b	CoRoT-1	1.508477	0.02556	1.465	347.9230	0.071	1814.0
232	CoRoT-7 b	CoRoT-7	0.8536600	0.01681	0.1875	674.2000	0.130	1832.0
252	CoRoT-9 b	CoRoT-9	95.27400	0.38290	0.8456	269.2000	0.0725	766.0
348	CoRoT-18 b	CoRoT-18	0.8414905	0.01274	1.21	3.47000	0.0067	2056.0

```
### Use one of data for each ###
new_df = df[df['pl_name'] == '55 Cnc e']
new_df = pd.concat([new_df, df[df['pl_name'] == 'GJ 1214 b']])

### set the new row using the loc indexer ###
df.loc[163:251] = new_df

df.tail()
```

Out[4]:

	pl_name	hostname	pl_orbper	pl_orbsmax	pl_radj	pl_bmasse	pl_orbeccen	pl_eqt
3478	WASP-85 b	WASP-85	2.656433	0.04270	0.857	975.7070	0.300	1020.0
3479	WASP-91 b	WASP-91	2.852970	0.03772	1.043	2.7000	0.4000	1560.0
3480	WASP-93 b	WASP-93	6.831979	0.05706	1.043	6.2600	0.4100	1760.0
3481	K2-79 b	K2-79	2.890442	0.04357	16.380	225.3947	0.0380	1763.0
3482	GJ 1214 b	GJ 1214	1.580404	0.01490	0.2427	6.1000	0.0000	604.0

```
### drop rows with gravity ###
df = df[df['pl_bmasse'] > 0]

fig = px.line(scipy_values, y=pca.explained_variance_ratio_, markers=True,
              title='Scree Plot', labels={'x': 'Principal Component', 'y': 'Variance Explained'})
fig.update_traces(line=dict(color='blue', width=2))

fig.show()
```

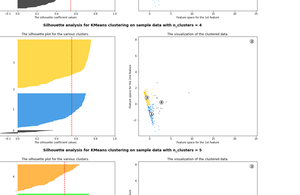
Scree Plot

In [24]:
```
print(pca.explained_variance_ratio_)

[0.44011392 0.2728594 0.18876428 0.07701437 0.05254112 0.0116135]
```

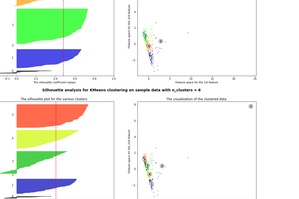
```
### calculate cumulative explained variance ###
cumulative_explained_variance = np.cumsum(pca.explained_variance_ratio_)

print(cumulative_explained_variance)

[0.44011392 0.7139186 0.9026829 0.9796973 0.9922085 1.        ]
```

```
### K-Means ###
k = [2, 3, 4, 5, 6]

# Keep the wcss values
wcss = []

for k_clusters in range(len(k)):
    ### fit the model with 2, 3, 4, 5, and 6 clusters ###
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    ### Initialize the clusterer with n_clusters ###
    ### The seed of 42 for reproducible results ###
    ### Use n_init to specify the number of times the algorithm will run
    ### with different centroid seeds. The final results will
    ### be the best output of n_init consecutive runs in terms of inertia.
    clusterer = KMeans(n_clusters=k[k_clusters],
                       n_init=10,
                       random_state=42)
    cluster_labels = clusterer.fit_predict(X)

    ### The silhouette_score gives the average value for all the samples. ###
    ### This gives a perspective into the density and separation of the formed ###
    ### clusters ###
    silhouette_avg = silhouette_score(X, cluster_labels)
    print(
        "For n_clusters =",
        k[k_clusters],
        "The average silhouette_score is :",
        silhouette_avg,
        )

    ### Compute the silhouette scores for each sample ###
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(k[k_clusters]):
        ### Aggregate the silhouette scores for samples belonging to ###
        ### cluster i, and sort them ###
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / k[k_clusters])
        ax1.fill_betweenx(
            np.arange(y_lower, y_upper),
            0,
            ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
            )

        ### Label the silhouette plots with their cluster numbers at the middle ###
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        ### Compute the new y_lower for next plot ###
        y_lower = y_upper + 10  ### 10 for the 0 samples ###

    ### Get the average silhouette score
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    ### Get the colors from the various clusters
    colors = cm.nipy_spectral(cluster_labels.astype(float) / k[k_clusters])
    ax2.scatter(
        X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7, c=colors, edgecolor='k'
        )

    ### Labeling the clusters ###
    centers = clusterer.cluster_centers_
    ### Draw white circles at cluster centers ###
    ax2.scatter(
        centers[:, 0],
        centers[:, 1],
        marker='o',
        c="white",
        alpha=1,
        s=200,
        edgecolor='k',
        )

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1, s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(
        "Silhouette analysis for KMeans clustering on sample data with n_clusters = %d"
        % k[k_clusters],
        fontsize=14,
        fontweight='bold',
        )

plt.show()

For n_clusters = 2 The average silhouette_score is : 0.7139186395857792
For n_clusters = 3 The average silhouette_score is : 0.5491024889944651
For n_clusters = 4 The average silhouette_score is : 0.4911111111111111
For n_clusters = 5 The average silhouette_score is : 0.4911111111111111
For n_clusters = 6 The average silhouette_score is : 0.4131534201024006
```

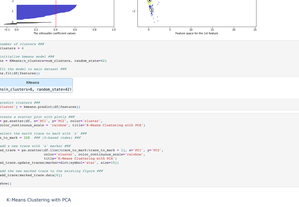
Silhouette analysis for KMeans clustering on sample data with n_clusters = 2



Silhouette analysis for KMeans clustering on sample data with n_clusters = 3



Silhouette analysis for KMeans clustering on sample data with n_clusters = 4



Silhouette analysis for KMeans clustering on sample data with n_clusters = 5

Silhouette analysis for KMeans clustering on sample data with n_clusters = 6

In [25]:
```
### Choose one of clusters ###
num_clusters = 4

### Initialize Kmeans model ###
kmeans = KMeans(n_clusters=num_clusters, random_state=42)

### Fit the model to main dataset ###
kmeans.fit(df[features])

# Output
KMeans(n_clusters=4, random_state=42)
```

In [26]:
```
### predict clustering ###
df['cluster'] = kmeans.predict(df[features])

### create a scatter plot with plotly ###
fig = px.scatter(df, x='PC1', y='PC2', color='cluster',
                 color_continuous_scale = 'viridis', title='K-Means Clustering with PCA')

### select the earth cluster to work with ###
true_to_work = df[df['cluster'] == 2]

### add a new trace with 'x' marker ###
fig.add_trace(px.scatter(x=true_to_work.loc[true_to_work.'PC1', y='PC2',
                         color_continuous_scale = 'viridis', title='K-Means Clustering with PCA').
              marker=dict(symbol='x', size=7, opacity=0.7, color='red'),
              name='Earth Candidates')

### add the new marked trace to the existing figure ###
fig.add_trace(marked_trace.data)

fig.show()
```

K-Means Clustering with PCA

In [27]:
```
### create a scatter plot with plotly ###
fig = px.scatter(df, x='PC1', y='PC2', color='cluster',
                 color_continuous_scale = 'viridis', title='K-Means Clustering with PCA')

### select the earth cluster to work with ###
true_to_work = df[df['cluster'] == 2]

### add a new trace with 'x' marker ###
fig.add_trace(px.scatter(x=true_to_work.loc[true_to_work.'PC1', y='PC2',
                         color_continuous_scale = 'viridis', title='K-Means Clustering with PCA').
              marker=dict(symbol='x', size=7, opacity=0.7, color='red'),
              name='Earth Candidates')

### add the new marked trace to the existing figure ###
fig.add_trace(marked_trace.data)

fig.show()
```

K-Means Clustering with PCA