Classification

November 13, 2024

[1]:
```python
try:
    # Cell 1: Import necessary libraries
    import os
    import glob
    import pandas as pd

    import matplotlib.pyplot as plt
    import seaborn as sns

    from sklearn.cluster import KMeans
    from kneed import KneeLocator
    from sklearn.metrics import silhouette_score


except Exception as e:
    print(f"Error : {e}")
```

[2]:
```python
# Cell 2: Load the dataset
# Find the CSV file in the Datasets directory
data_path = '../Datasets/*.csv'
file_list = glob.glob(data_path)

for file in file_list:
    print(f"Found file: {file}")

# Ensure there is exactly one file
if len(file_list) == 1:
    # Load the dataset
    df = pd.read_csv(file_list[0])
    print(f"Loaded dataset: {file_list[0]}")
else:
    raise FileNotFoundError("No CSV file found or multiple CSV files found in␣
 ↪the Datasets directory.")
```

```
Found file: ../Datasets/Dataset.csv
Loaded dataset: ../Datasets/Dataset.csv
```

[3]:
```python
# Cell 3: Set the model saving path
destination = '../Models/'
os.makedirs(destination, exist_ok=True)
print(f"Model will be saved to: {destination}")
```

```
Model will be saved to: ../Models/
```

[4]:
```python
# Cell 4: Assign 'Unacceptable' to parts below threshold
clf_df = df.copy()
threshold_value = 1500

# Create 'Lifetime' column and set default to None
clf_df['Lifetime'] = None

# Assign 'Unacceptable' to rows where 'Lifespan' < threshold_value
clf_df.loc[clf_df['Lifespan'] < threshold_value, 'Lifetime'] = 'Unacceptable'

clf_df.head(10)
```

[4]:

```
   Lifespan partType microstructure  coolingRate  quenchTime  forgeTime  \
0   1469.17   Nozzle      equiGrain           13        3.84       6.47
1   1793.64    Block    singleGrain           19        2.62       3.48
2    700.60    Blade      equiGrain           28        0.76       1.34
3   1082.10   Nozzle       colGrain            9        2.01       2.19
4   1838.83    Blade       colGrain           16        4.13       3.87
5    660.62    Valve       colGrain           28        4.45       3.82
6   1835.46    Block      equiGrain           19        2.76       4.27
7   1522.80    Block      equiGrain           16        1.48       9.61
8   1347.72    Blade      equiGrain           21        1.41       4.17
9    985.79    Valve       colGrain           10        3.75       6.82

   HeatTreatTime  Nickel%  Iron%  Cobalt%  Chromium%  smallDefects  \
0          46.87    65.73  16.52    16.82       0.93            10
1          44.70    54.22  35.38     6.14       4.26            19
2           9.54    51.83  35.95     8.81       3.41            35
3          20.29    57.03  23.33    16.86       2.78             0
4          16.13    59.62  27.37    11.45       1.56            10
5          18.11    50.30  33.30    12.45       3.95            21
6          56.75    63.85  17.64    16.79       1.72            18
7          51.37    52.75  37.10     9.05       1.10            21
8          53.76    58.88  20.39    16.03       4.70            24
9          23.47    66.75  14.48    18.14       0.63             0

   largeDefects  sliverDefects seedLocation    castType      Lifetime
0             0              0       Bottom         Die  Unacceptable
1             0              0       Bottom  Investment          None
2             3              0       Bottom  Investment  Unacceptable
3             1              0          Top  Continuous  Unacceptable
```

```
4               0                 0         Top         Die          None
5               4                 0         Top  Investment  Unacceptable
6               0                 0      Bottom         Die          None
7               0                 0      Bottom  Continuous          None
8               1                 0      Bottom  Continuous  Unacceptable
9               0                 0      Bottom         Die  Unacceptable
```

[5]:
```
min_lifespan = clf_df['Lifespan'].min()
max_lifespan = clf_df['Lifespan'].max()

print(f"Minimum Lifespan: {min_lifespan}")
print(f"Maximum Lifespan: {max_lifespan}")
```

```
Minimum Lifespan: 417.99
Maximum Lifespan: 2134.53
```

[6]:
```
# Cell 5: Prepare data for clustering
# Data to be clustered (Lifespan >= threshold_value)
acceptable_df = clf_df[clf_df['Lifespan'] >= threshold_value].copy()
```

[7]:
```
# Cell 6: Select 'Lifespan' feature for clustering
X_acceptable = acceptable_df[['Lifespan']]
```

[8]:
```
# Cell 7: Perform K-Means clustering
# Ensure there is enough data to cluster
if len(acceptable_df) > 1:
    # Initialize a list to store inertia values
    max_k = min(10, len(acceptable_df))
    k_values = range(1, max_k)

    inertia = []

    # Calculate inertia for each k
    for k in k_values:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(X_acceptable)
        inertia.append(kmeans.inertia_)

    # Dynamically determine the elbow point using KneeLocator
    kneedle = KneeLocator(k_values, inertia, curve='convex',␣
↪direction='decreasing')
    elbow_k = kneedle.elbow

    # If elbow_k is None, default to 3
    if elbow_k is None:
        elbow_k = 3

    # Ensure elbow_k is not greater than max_k
```

```
    elbow_k = min(elbow_k, max_k - 1)

    # Perform KMeans clustering with elbow_k clusters
    kmeans = KMeans(n_clusters=elbow_k, random_state=42)
    acceptable_df['cluster'] = kmeans.fit_predict(X_acceptable)

    # Compute mean 'Lifespan' per cluster
    cluster_means = acceptable_df.groupby('cluster')['Lifespan'].mean()

    # Sort clusters by mean lifespan
    cluster_order = cluster_means.sort_values().index.tolist()

    # Define desired labels
    desired_labels = ['Fair', 'Good', 'Excellent']

    labels = []
    for i in range(elbow_k):
        if i < len(desired_labels):
            labels.append(desired_labels[i])
        else:
            # For additional clusters, create labels based on lifespan ranges
            cluster_idx = cluster_order[i]
            min_life = acceptable_df[acceptable_df['cluster'] ==␣
 ↪cluster_idx]['Lifespan'].min()
            max_life = acceptable_df[acceptable_df['cluster'] ==␣
 ↪cluster_idx]['Lifespan'].max()
            labels.append(f'{min_life:.2f}-{max_life:.2f}')

    # Create mapping from cluster index to label
    cluster_to_label = dict(zip(cluster_order, labels))

    # Map labels to 'Lifetime' column
    acceptable_df['Lifetime'] = acceptable_df['cluster'].map(cluster_to_label)

    # Update 'Lifetime' column in clf_df
    clf_df.loc[acceptable_df.index, 'Lifetime'] = acceptable_df['Lifetime']

    # Drop 'cluster' column if not needed
    acceptable_df.drop(columns=['cluster'], inplace=True)
else:
    print("Not enough data to perform clustering on acceptable parts.")
```

[9]:
```
# Plotting the Elbow Method with all indicators
plt.figure(figsize=(10, 6))
# Plot line segments with different colors
plt.plot(k_values[:elbow_k], inertia[:elbow_k], 'bo-', label="Decreasing Phase")
```

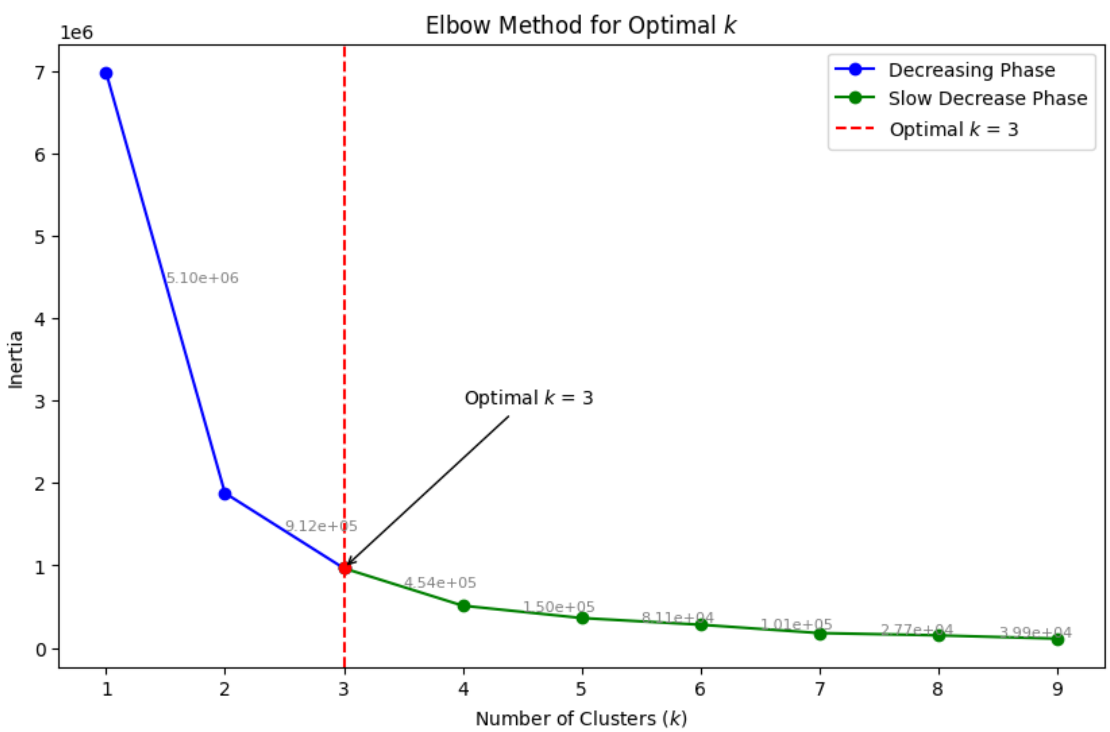
```
plt.plot(k_values[elbow_k - 1:], inertia[elbow_k - 1:], 'go-', label="Slow␣
    ↪Decrease Phase")

# Vertical line at elbow
plt.axvline(x=elbow_k, linestyle='--', color='r', label=f'Optimal $k$ =␣
    ↪{elbow_k}')

# Highlight the elbow point with a red marker and annotation
plt.plot(elbow_k, inertia[elbow_k - 1], 'ro')  # red point at elbow
plt.annotate(f"Optimal $k$ = {elbow_k}", xy=(elbow_k, inertia[elbow_k - 1]),
             xytext=(elbow_k + 1, inertia[elbow_k - 1] + 0.2e7),
             arrowprops=dict(facecolor='black', arrowstyle="->"))

# Annotate each segment with inertia differences
for i in range(1, len(k_values)):
    plt.annotate(f"{inertia[i-1] - inertia[i]:.2e}",
                 (k_values[i] - 0.5, (inertia[i-1] + inertia[i]) / 2),
                 fontsize=8, color='gray')

# Set plot labels and title
plt.xlabel(f'Number of Clusters ($k$)')
plt.ylabel('Inertia')
plt.title(f'Elbow Method for Optimal $k$')
plt.legend()
plt.show()
```

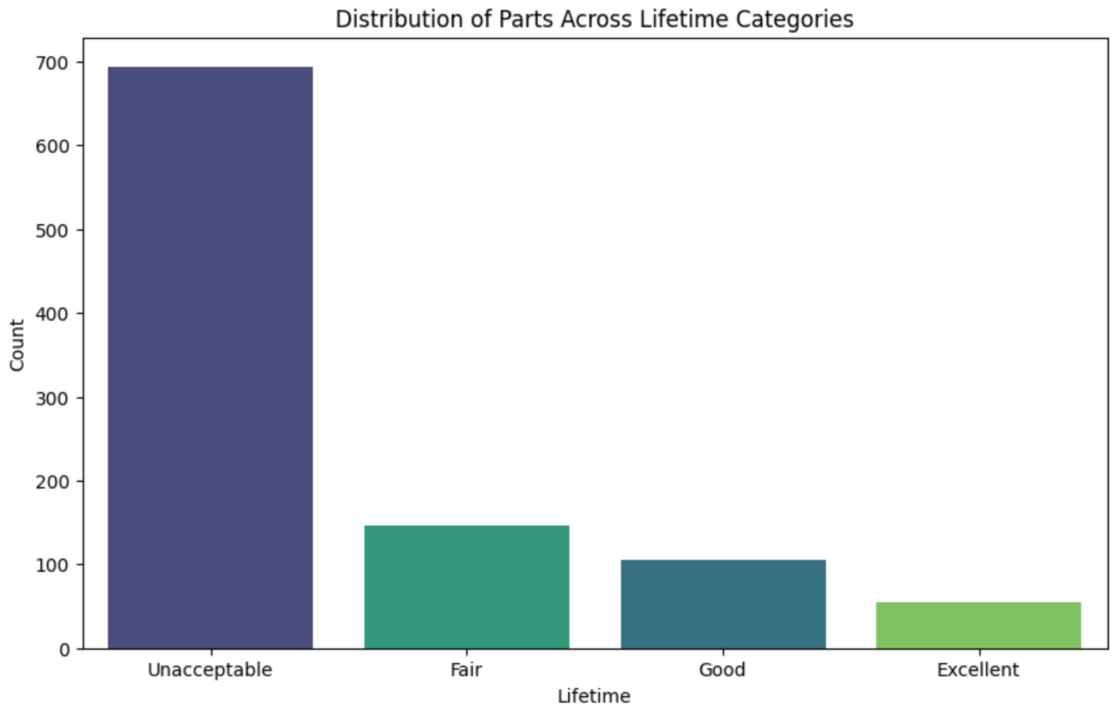


[10]:
```python
# Cell 8: Display the cluster ranges
# Group the data by 'Lifetime' and aggregate to find the min and max Lifespan␣
↪for each group
cluster_ranges = clf_df.groupby('Lifetime')['Lifespan'].agg(['min', 'max']).
↪sort_values(by='min').reset_index()

# Display the sorted DataFrame
display(cluster_ranges)
```

	Lifetime	min	max
0	Unacceptable	417.99	1499.31
1	Fair	1501.76	1661.54
2	Good	1666.64	1850.75
3	Excellent	1854.50	2134.53

[11]:
```python
# Cell 9: Visualize the distribution of parts across Lifetime categories
plt.figure(figsize=(10, 6))
sns.countplot(
    x='Lifetime',
    data=clf_df,
    hue='Lifetime',
    order=cluster_ranges['Lifetime'].unique(),
    palette='viridis',
    dodge=False,
    legend=False
)
plt.xlabel('Lifetime')
plt.ylabel('Count')
plt.title('Distribution of Parts Across Lifetime Categories')
plt.show()
```



[12]:
```
# Cell 10: Calculate the silhouette score
if len(acceptable_df['Lifetime'].unique()) > 1:
    # Map Lifetime labels back to cluster numbers for silhouette score
    label_to_cluster = {v: k for k, v in cluster_to_label.items()}
    acceptable_clusters = acceptable_df['Lifetime'].map(label_to_cluster)
    silhouette_avg = silhouette_score(X_acceptable, acceptable_clusters)
    print(f'Silhouette Score for k={elbow_k}: {silhouette_avg:.2f}')
else:
    print("Cannot compute silhouette score with only one cluster.")
```

```
Silhouette Score for k=3: 0.58
```

[13]:
```
clf_df.head(10)
```

[13]:

	Lifespan	partType	microstructure	coolingRate	quenchTime	forgeTime	\
0	1469.17	Nozzle	equiGrain	13	3.84	6.47	
1	1793.64	Block	singleGrain	19	2.62	3.48	
2	700.60	Blade	equiGrain	28	0.76	1.34	
3	1082.10	Nozzle	colGrain	9	2.01	2.19	
4	1838.83	Blade	colGrain	16	4.13	3.87	
5	660.62	Valve	colGrain	28	4.45	3.82	
6	1835.46	Block	equiGrain	19	2.76	4.27	
7	1522.80	Block	equiGrain	16	1.48	9.61	
8	1347.72	Blade	equiGrain	21	1.41	4.17	

```
9    985.79    Valve       colGrain           10         3.75       6.82

   HeatTreatTime  Nickel%  Iron%  Cobalt%  Chromium%  smallDefects  \
0          46.87    65.73  16.52    16.82       0.93            10
1          44.70    54.22  35.38     6.14       4.26            19
2           9.54    51.83  35.95     8.81       3.41            35
3          20.29    57.03  23.33    16.86       2.78             0
4          16.13    59.62  27.37    11.45       1.56            10
5          18.11    50.30  33.30    12.45       3.95            21
6          56.75    63.85  17.64    16.79       1.72            18
7          51.37    52.75  37.10     9.05       1.10            21
8          53.76    58.88  20.39    16.03       4.70            24
9          23.47    66.75  14.48    18.14       0.63             0

   largeDefects  sliverDefects seedLocation    castType      Lifetime
0             0              0       Bottom         Die  Unacceptable
1             0              0       Bottom  Investment          Good
2             3              0       Bottom  Investment  Unacceptable
3             1              0          Top  Continuous  Unacceptable
4             0              0          Top         Die          Good
5             4              0          Top  Investment  Unacceptable
6             0              0       Bottom         Die          Good
7             0              0       Bottom  Continuous          Fair
8             1              0       Bottom  Continuous  Unacceptable
9             0              0       Bottom         Die  Unacceptable
```