Classification

November 12, 2024

[1]:
```python
try:
    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

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

[2]:
```python
# 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
```

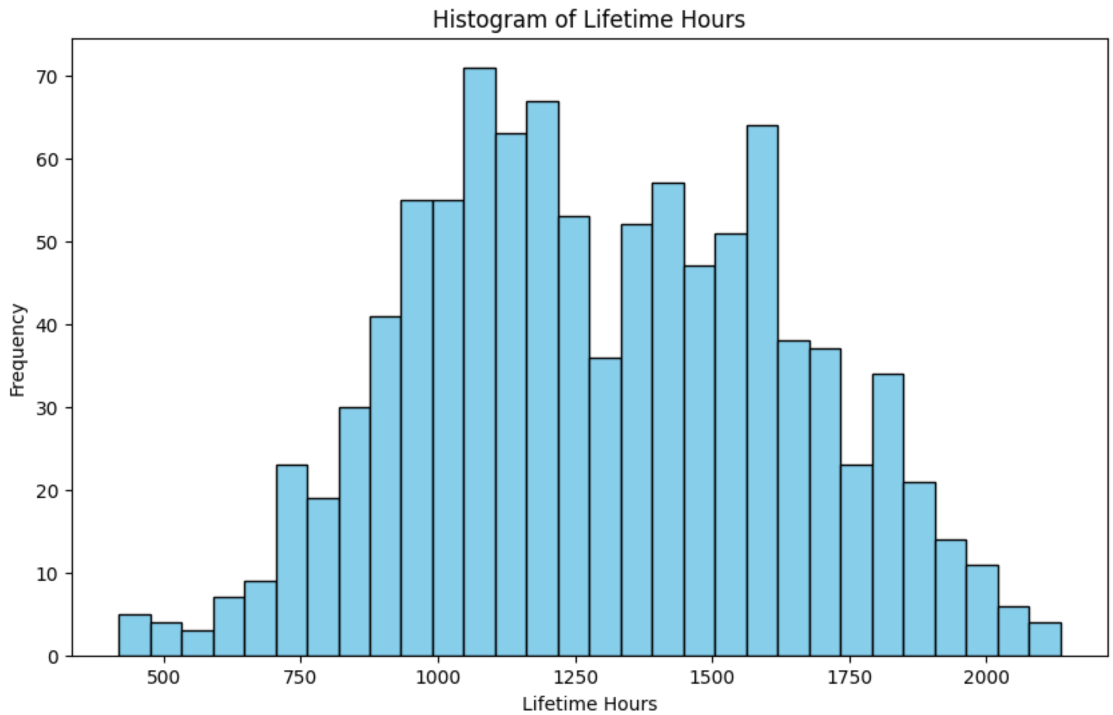
[3]:
```python
# File path to save the trained model
destination = '../Models/'
os.makedirs(destination, exist_ok=True)
print(f"Model will be saved to: {destination}")
```

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

```
[4]: clf_df = df.copy()

clf_df['1500_labels'] = clf_df['Lifespan'].apply(lambda x: 1 if x >= 1500 else␣
 ↪0)

clf_df.head()
```

```
[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

   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

   largeDefects  sliverDefects seedLocation    castType  1500_labels
0             0              0       Bottom         Die            0
1             0              0       Bottom  Investment            1
2             3              0       Bottom  Investment            0
3             1              0          Top  Continuous            0
4             0              0          Top         Die            1
```

```
[5]: # Histogram to understand the distribution of 'lifetime_hours'
plt.figure(figsize=(10, 6))
plt.hist(clf_df['Lifespan'], bins=30, color='skyblue', edgecolor='black')
plt.xlabel('Lifetime Hours')
plt.ylabel('Frequency')
plt.title('Histogram of Lifetime Hours')
plt.show()

# Calculate key percentiles (quartiles)
percentiles = clf_df['Lifespan'].quantile([0.25, 0.5, 0.75])
print(f"Quartiles of Lifetime Hours: \n{percentiles}")
```



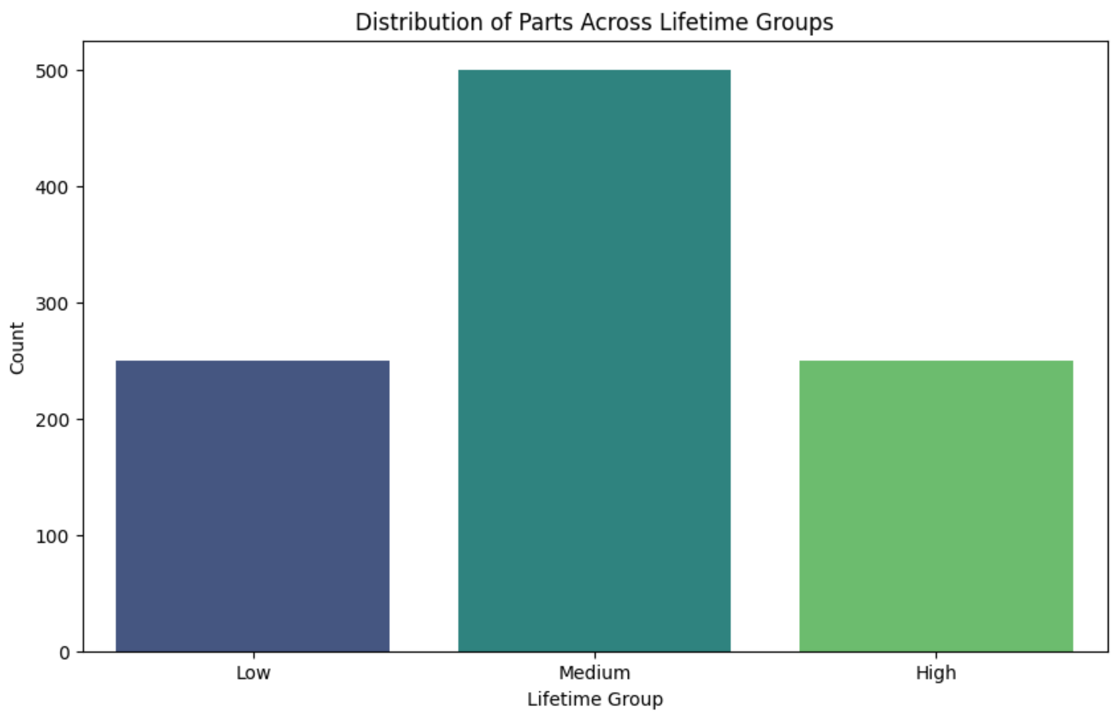

```
Quartiles of Lifetime Hours:
0.25    1047.2575
0.50    1266.0400
0.75    1563.0500
Name: Lifespan, dtype: float64
```

[6]:

```python
# Creating a new column 'Lifetime' to classify parts based on quartiles
conditions = [
    (clf_df['Lifespan'] <= percentiles[0.25]),
    (clf_df['Lifespan'] > percentiles[0.25]) & (clf_df['Lifespan'] <=
 ↪percentiles[0.75]),
    (clf_df['Lifespan'] > percentiles[0.75])
]
choices = ['Low', 'Medium', 'High']

clf_df['Lifetime'] = pd.cut(clf_df['Lifespan'], bins=[-float('inf'),
 ↪percentiles[0.25], percentiles[0.75], float('inf')], labels=choices)
print(clf_df[['Lifespan', 'Lifetime']].head())
```

```
   Lifespan Lifetime
0   1469.17   Medium
1   1793.64     High
2    700.60      Low
3   1082.10   Medium
4   1838.83     High
```

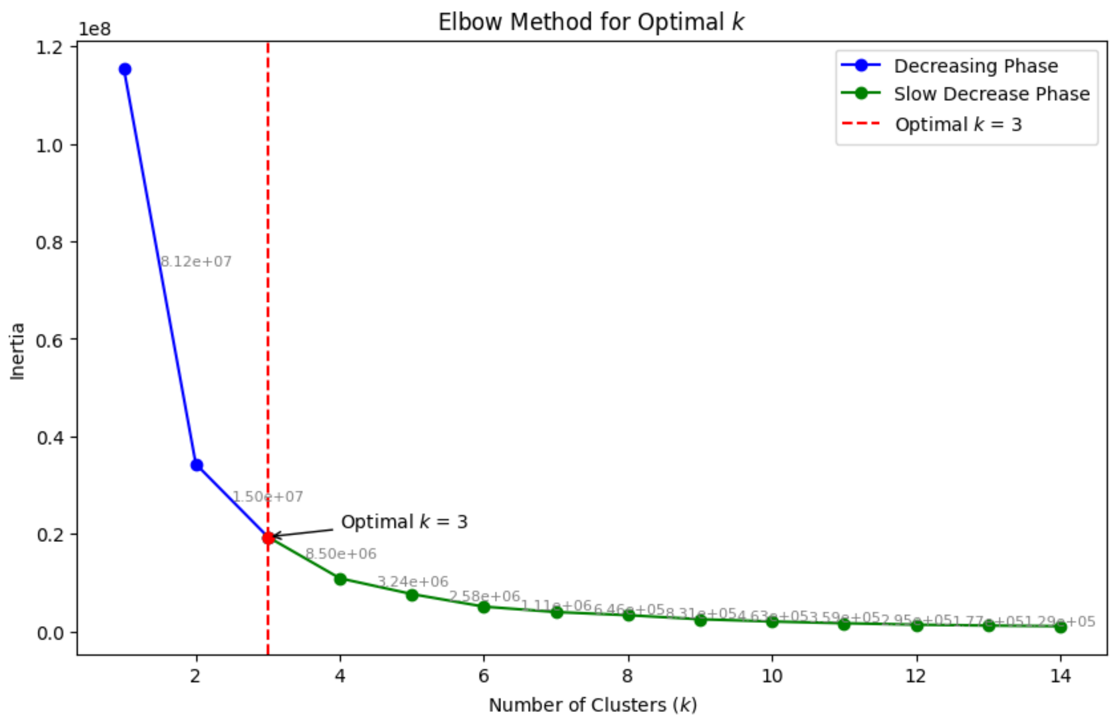
[7]:
```python
# Bar plot to visualize the distribution of parts across groups
plt.figure(figsize=(10, 6))
sns.countplot(x='Lifetime', data=clf_df, hue='Lifetime', palette='viridis',␣
 ↪legend=False)
plt.xlabel('Lifetime Group')
plt.ylabel('Count')
plt.title('Distribution of Parts Across Lifetime Groups')
plt.show()
```



[8]:
```python
# Selecting only 'Lifespan' for simplicity
X = clf_df[['Lifespan']]
```

[17]:
```python
# Initialize a list to store inertia values
inertia = []
k_values = range(1, 15)

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

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

```python
elbow_k = kneedle.elbow

# 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()
```

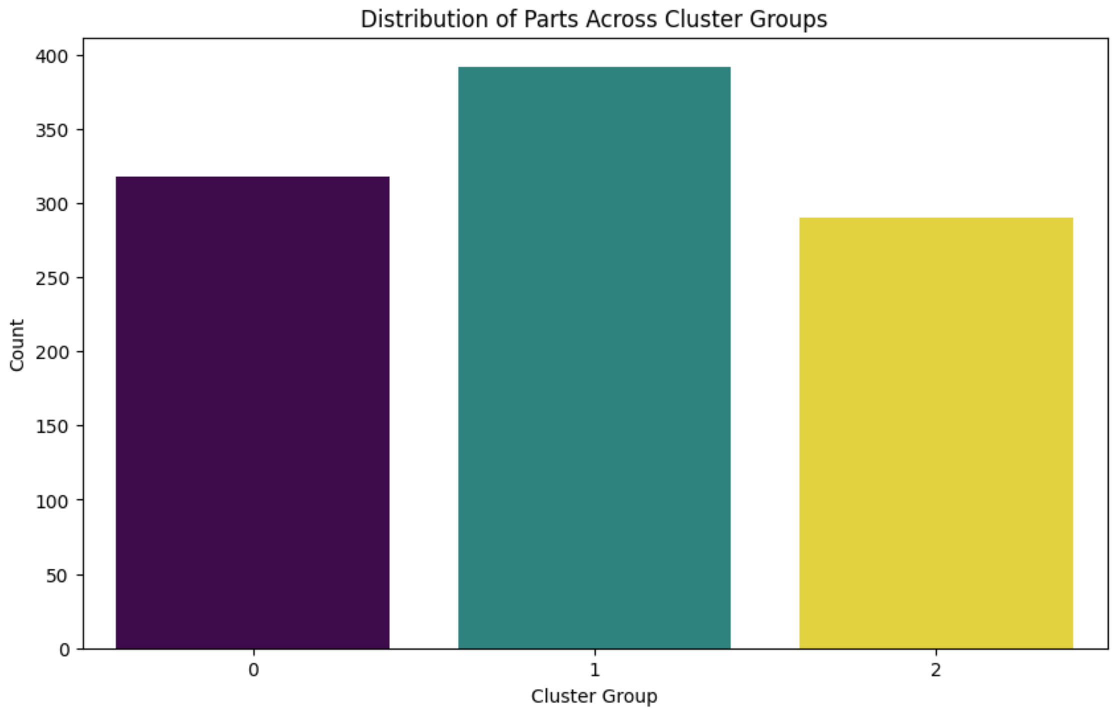


[18]:

```
kmeans = KMeans(n_clusters=elbow_k, random_state=42)
clf_df['cluster_group'] = kmeans.fit_predict(X)
```

[19]:

```
plt.figure(figsize=(10, 6))
sns.countplot(x='cluster_group', data=clf_df, hue='cluster_group',␣
↪palette='viridis', legend=False)
plt.xlabel('Cluster Group')
plt.ylabel('Count')
plt.title('Distribution of Parts Across Cluster Groups')
plt.show()
```



[20]:
```
from sklearn.metrics import silhouette_score

silhouette_avg = silhouette_score(X, kmeans.labels_)
print(f'Silhouette Score for k={elbow_k}: {silhouette_avg:.2f}')
```

```
Silhouette Score for k=3: 0.52
```