BDA HOLIDAY ASSIGNMENT

2211CS020071
AIML-ZETA
B.AKSHAJ

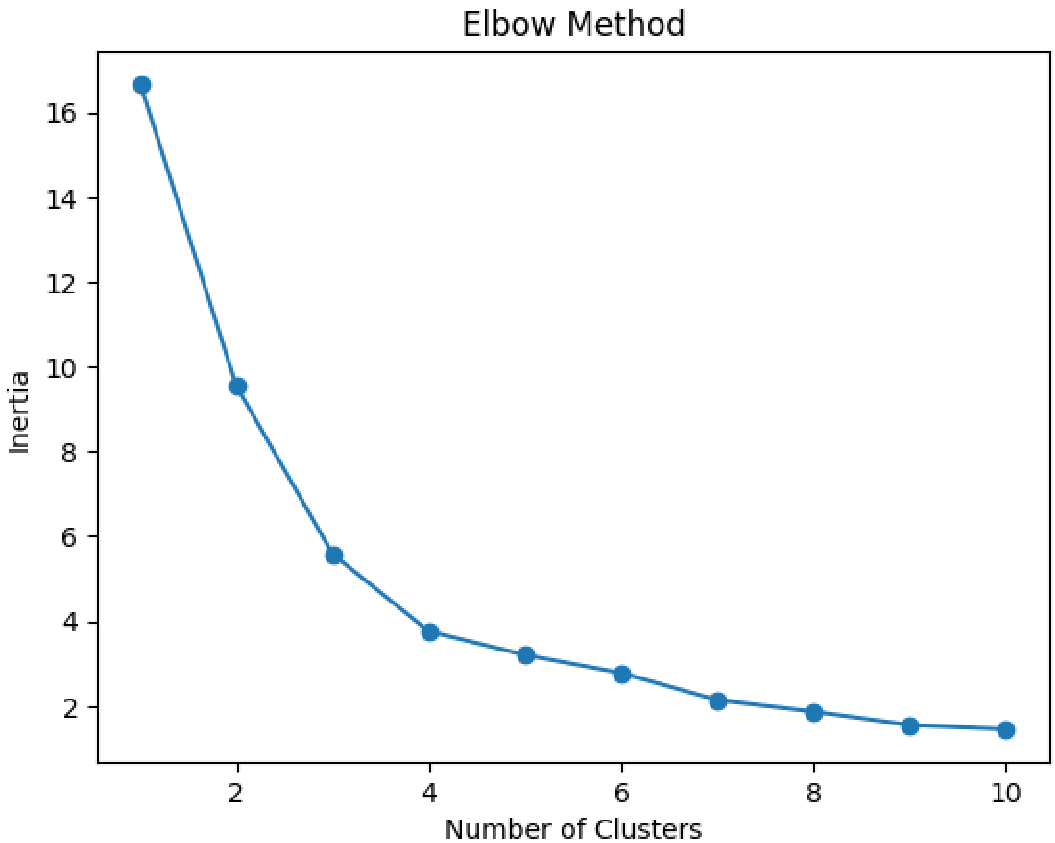
0.1 *Handle Imbalanced Datasets for Classification Tasks*

[3]:
```python
# Example: Using SMOTE to handle imbalanced data
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
import pandas as pd

# Sample data
data = pd.DataFrame({'feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                     'feature2': [5, 6, 7, 8, 9, 10, 11, 12, 13, 14],
                     'label': [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]})

X = data[['feature1', 'feature2']]
y = data['label']

# Oversampling
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Train model
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
    test_size=0.3, random_state=42)
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       1.00      0.50      0.67         2

    accuracy                           0.67         3
   macro avg       0.75      0.75      0.67         3
weighted avg       0.83      0.67      0.67         3
```

1 Optimal Number of Clusters for K-Means

[4]:
```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np

# Sample data
data = np.random.rand(100, 2)

# Elbow Method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data)
    inertia.append(kmeans.inertia_)

plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()
```



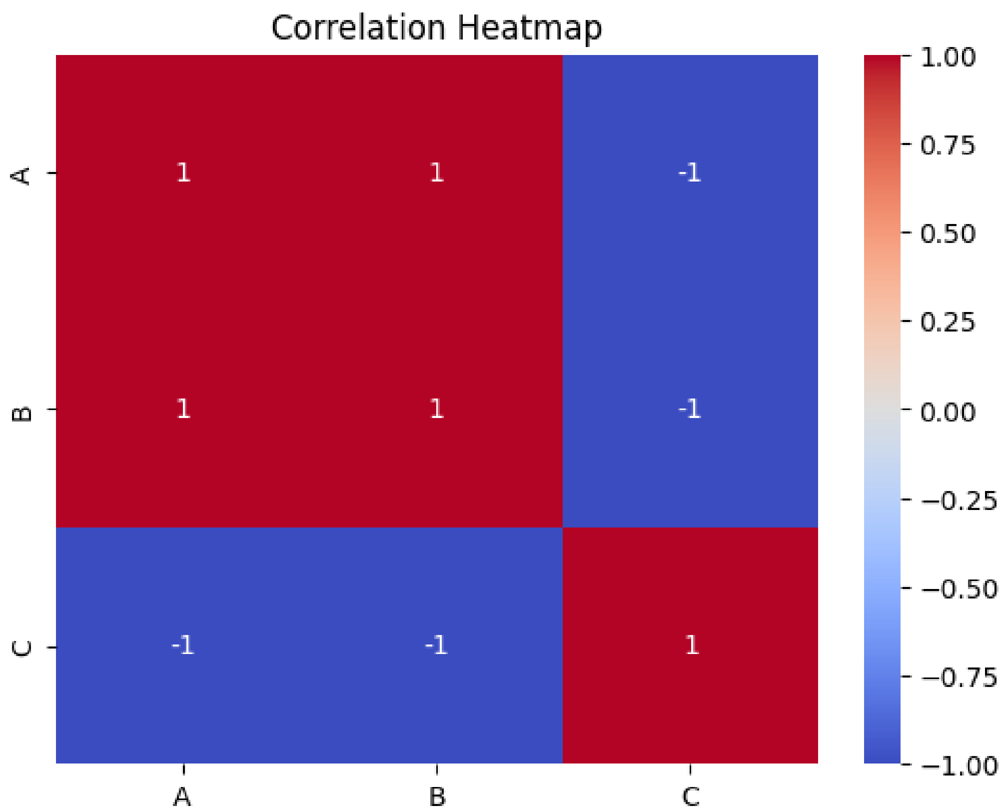
2 Dimensionality Reduction (PCA)

[5]:
```
from sklearn.decomposition import PCA
import numpy as np

# Sample data
data = np.random.rand(100, 5)

# PCA
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(data)
print("Reduced Data Shape:", reduced_data.shape)
```

```
Reduced Data Shape: (100, 2)
```

3 *Find and Visualize Correlations*

[6]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = pd.DataFrame({'A': [1, 2, 3, 4, 5],
                     'B': [5, 6, 7, 8, 9],
                     'C': [9, 8, 7, 6, 5]})

# Correlation Matrix
corr_matrix = data.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



4 *Handle Missing Values*

[7]:
```python
import pandas as pd
from sklearn.impute import SimpleImputer

# Sample data
data = pd.DataFrame({'A': [1, 2, None, 4, 5],
                     'B': [5, None, 7, 8, 9]})

# Impute missing values
imputer = SimpleImputer(strategy='mean')
data_imputed = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)
print(data_imputed)
```

```
     A     B
0  1.0  5.00
1  2.0  7.25
2  3.0  7.00
3  4.0  8.00
4  5.0  9.00
```

5 *Detect and Remove Duplicates*

[8]:
```python
import pandas as pd

# Sample data
data = pd.DataFrame({'A': [1, 2, 2, 4, 5],
                     'B': [5, 6, 6, 8, 9]})

# Detect and remove duplicates
print("Duplicates:\n", data[data.duplicated()])
data_cleaned = data.drop_duplicates()
print("Cleaned Data:\n", data_cleaned)
```

```
Duplicates:
   A  B
2  2  6
Cleaned Data:
   A  B
0  1  5
1  2  6
3  4  8
4  5  9
```

6 Random Forest Regression for Housing Prices

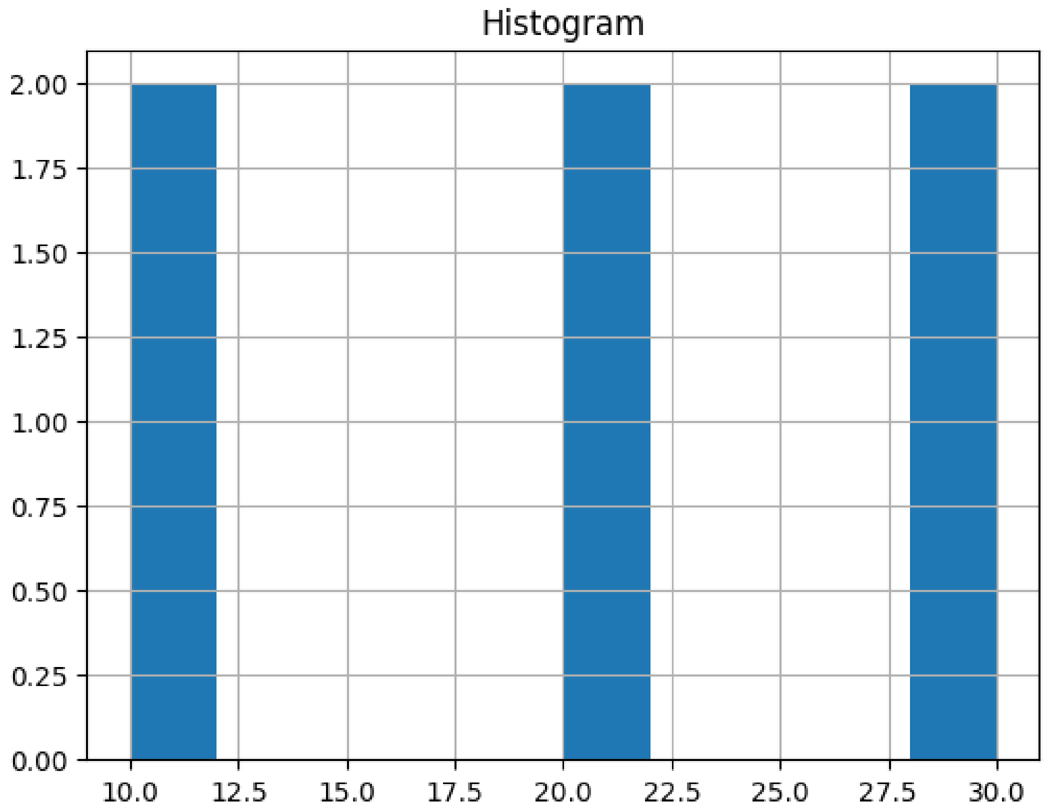
[9]:
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import pandas as pd

# Sample data
data = pd.DataFrame({'feature1': [1, 2, 3, 4, 5],
                     'feature2': [5, 6, 7, 8, 9],
                     'price': [100, 200, 300, 400, 500]})

X = data[['feature1', 'feature2']]
y = data['price']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
 random_state=42)

# Random Forest
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print("MSE:", mean_squared_error(y_test, y_pred))
```

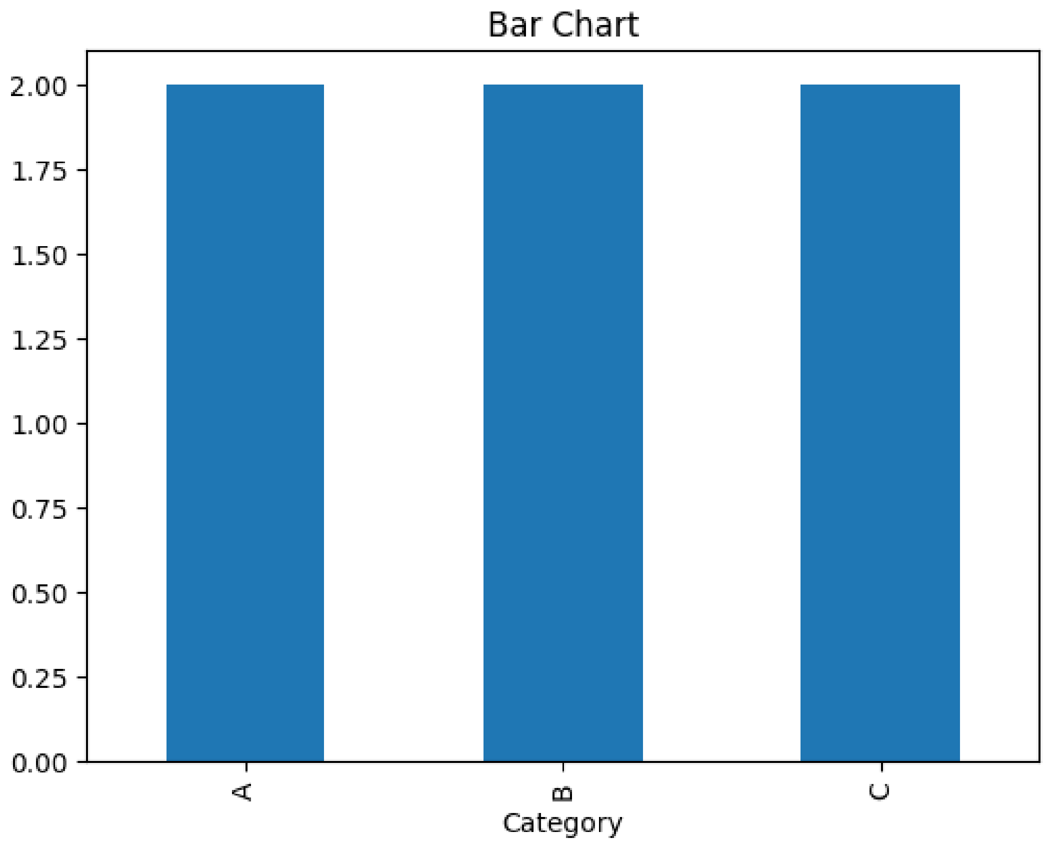
MSE: 11108.0

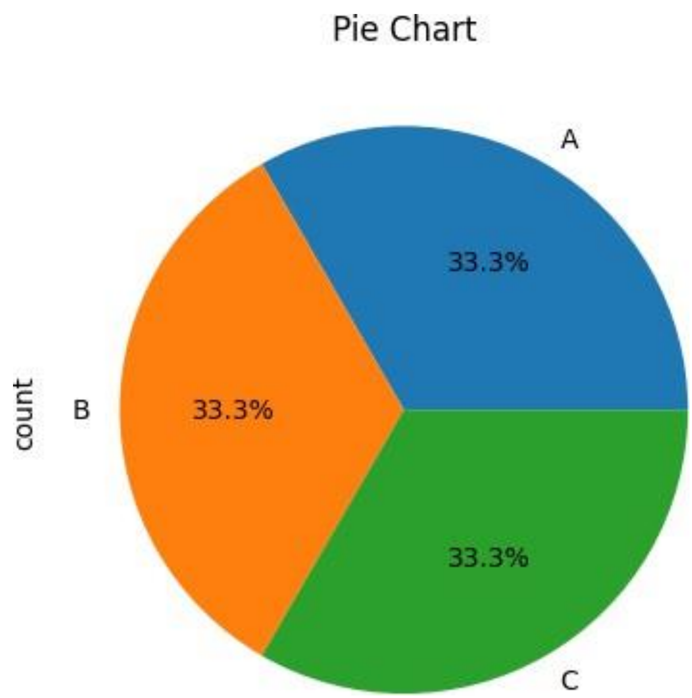
7 Plot Histogram, Bar Chart, and Pie Chart

[10]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# Sample data
data = pd.DataFrame({'Category': ['A', 'B', 'C', 'A', 'B', 'C'],
                     'Values': [10, 20, 30, 10, 20, 30]})

# Histogram
data['Values'].hist()
plt.title('Histogram')
plt.show()

# Bar Chart
data['Category'].value_counts().plot(kind='bar')
plt.title('Bar Chart')
plt.show()
```

```python
# Pie Chart
data['Category'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title('Pie Chart')
plt.show()
```




Bar Chart
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
A
B
C
Category



8 *Linear and Logistic Regression*

[11]:
```
#LINEAR REGRESSION
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import pandas as pd

# Sample data
data = pd.DataFrame({'feature': [1, 2, 3, 4, 5],
                     'target': [2, 4, 6, 8, 10]})

X = data[['feature']]
y = data['target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Linear Regression
lr = LinearRegression()
```

```
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
print("MSE:", mean_squared_error(y_test, y_pred))
```

```
MSE: 0.0
```

[12]:
```
#LOGISTIC REGRESSION
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd

# Sample data
data = pd.DataFrame({'feature': [1, 2, 3, 4, 5],
                     'label': [0, 0, 1, 1, 1]})

X = data[['feature']]
y = data['label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Logistic Regression
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5
```

9 Lag Features for Time-Series Data

[13]:
```
import pandas as pd

# Sample data
data = pd.DataFrame({'date': pd.date_range(start='1/1/2023', periods=10),
                     'value': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]})

data['lag_1'] = data['value'].shift(1)
data['lag_2'] = data['value'].shift(2)
print(data)
```

```
        date  value  lag_1  lag_2
0 2023-01-01     10    NaN    NaN
1 2023-01-02     20   10.0    NaN
2 2023-01-03     30   20.0   10.0
```

```
3 2023-01-04     40    30.0    20.0
4 2023-01-05     50    40.0    30.0
5 2023-01-06     60    50.0    40.0
6 2023-01-07     70    60.0    50.0
7 2023-01-08     80    70.0    60.0
8 2023-01-09     90    80.0    70.0
9 2023-01-10    100    90.0    80.0
```