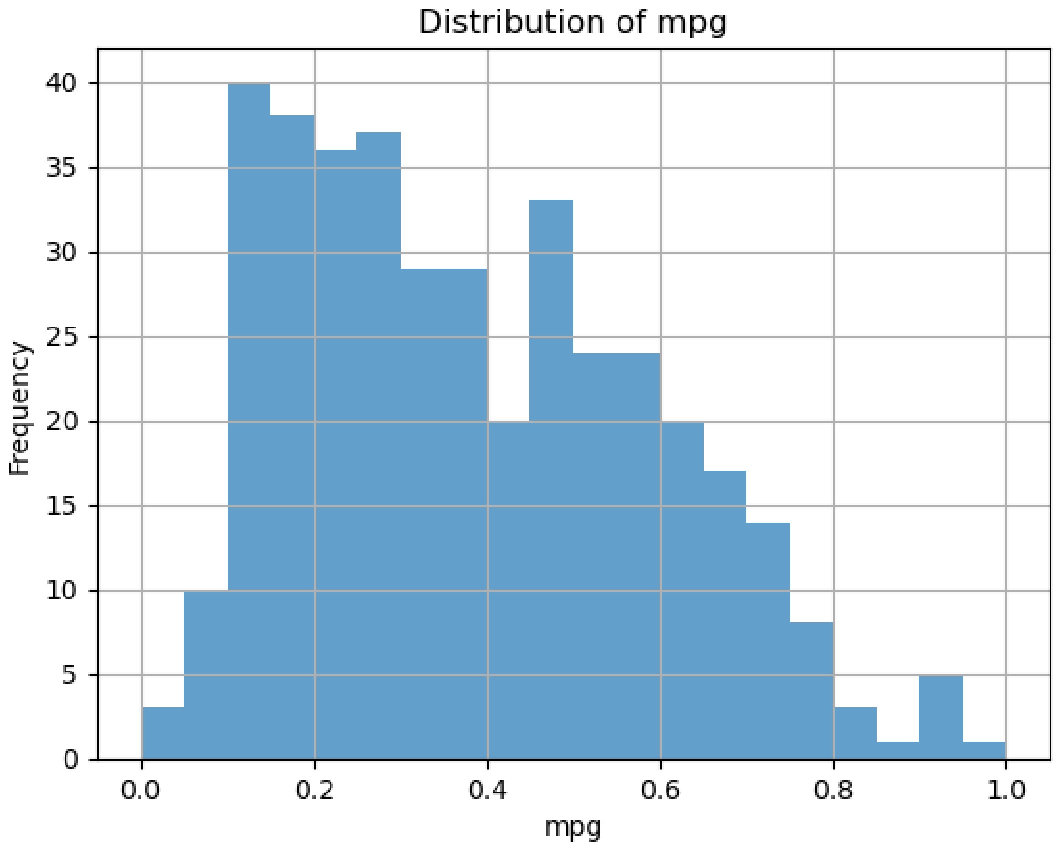
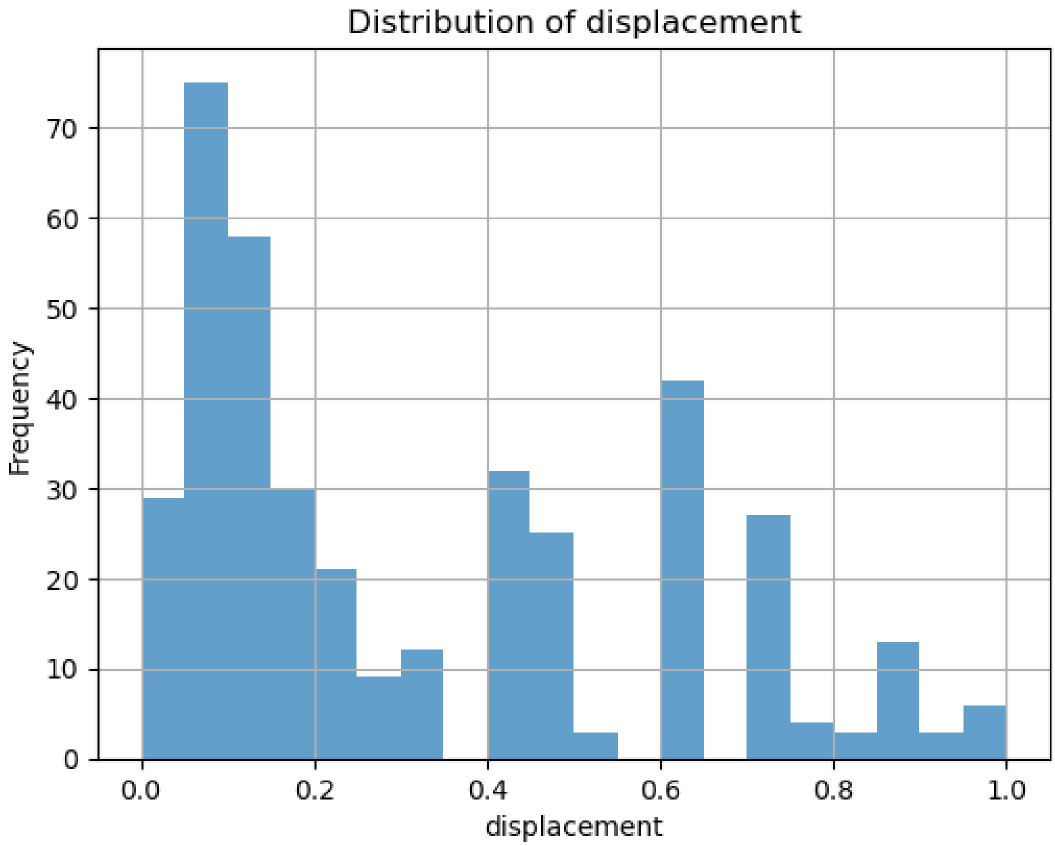
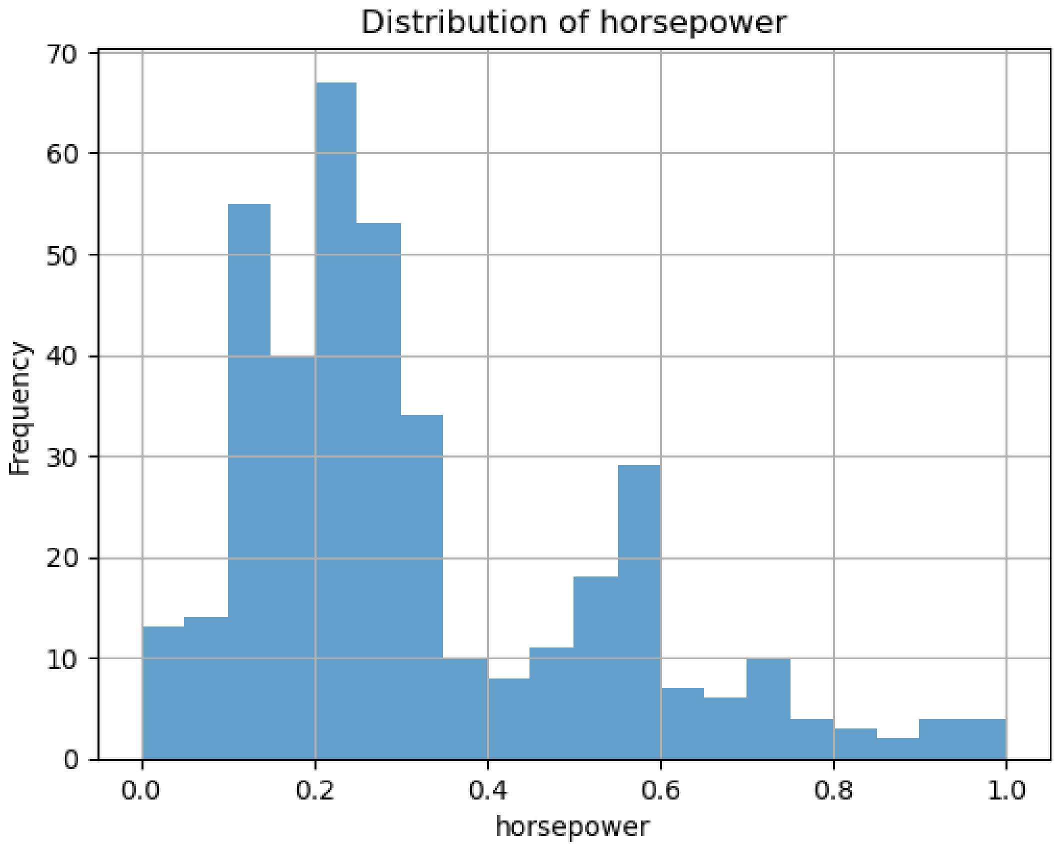
hw2

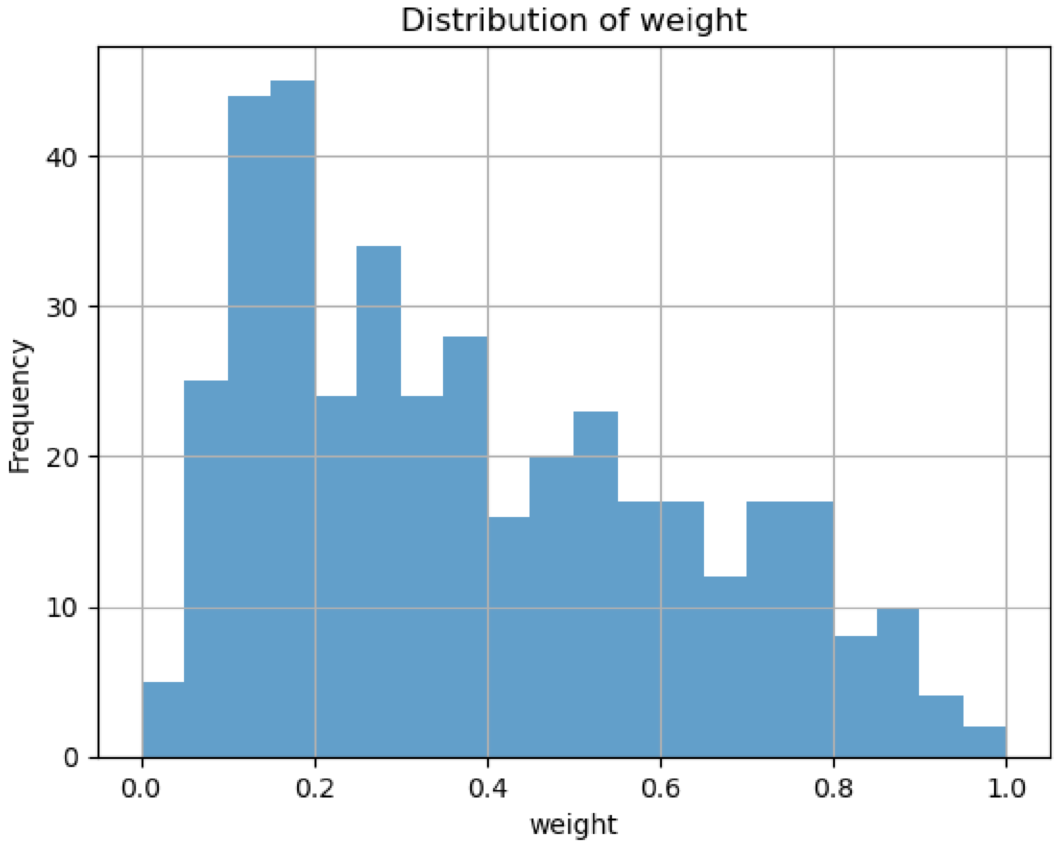
January 30, 2025

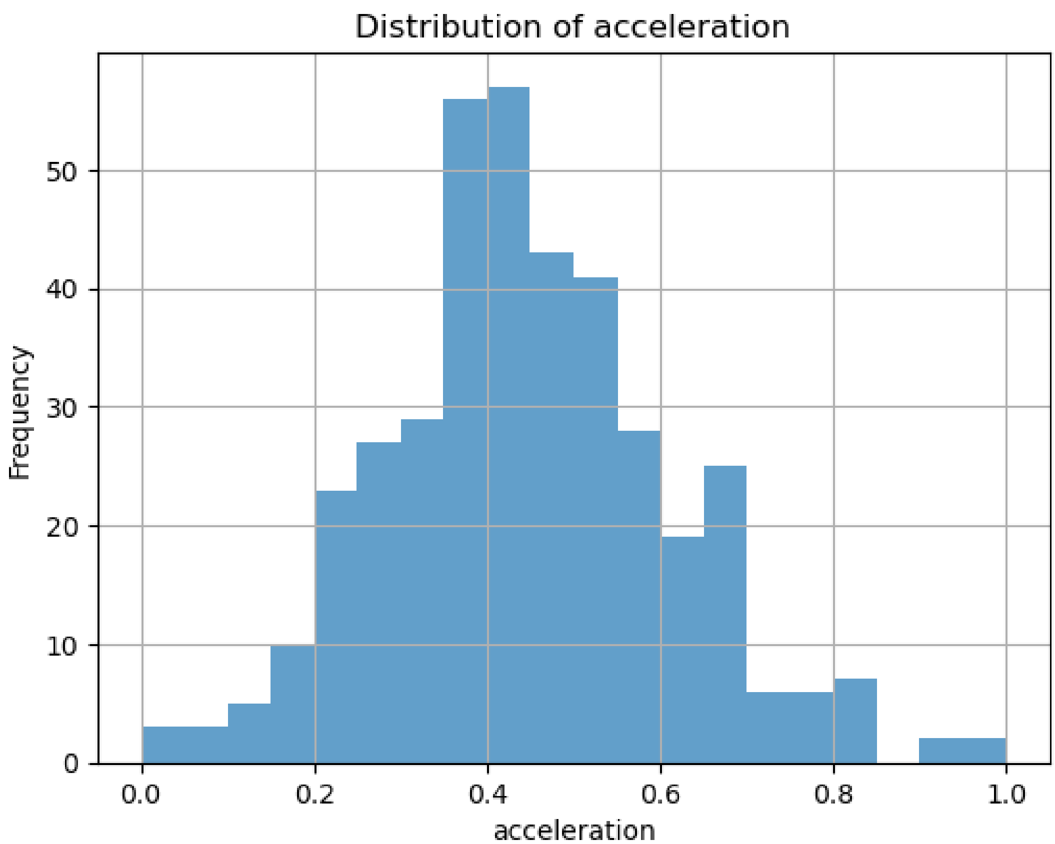
[ ]:
```
# Exercise 1

import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

data = pd.read_csv('auto-mpg.csv')

# Apply one-hot encoding
encoded_data = pd.get_dummies(data, columns=['origin'], prefix='origin')

# Normalize each field of the input data using the min-max normalization␣
↪technique
scaler = MinMaxScaler()
numerical_columns = ['mpg', 'displacement', 'horsepower', 'weight',␣
↪'acceleration']
encoded_data[numerical_columns] = scaler.
↪fit_transform(encoded_data[numerical_columns])

# Plot the data
for column in numerical_columns:
    encoded_data[column].hist(bins=20, alpha=0.7)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()

# Data Analyzation
# - mpg: skewed to right
# - displacement: skewed to right
# - horsepower: skewed to right
# - weight: skewed to right
# - acceleration: symmetric
```


Distribution of mpg
Frequency
mpg
0
5
10
15
20
25
30
35
40
0.0
0.2
0.4
0.6
0.8
1.0


Distribution of displacement
Frequency
displacement
0
10
20
30
40
50
60
70
0.0
0.2
0.4
0.6
0.8
1.0


Distribution of horsepower
Frequency
horsepower
0
10
20
30
40
50
60
70
0.0
0.2
0.4
0.6
0.8
1.0


Distribution of weight
Frequency
0
10
20
30
40
0.0
0.2
0.4
0.6
0.8
1.0
weight

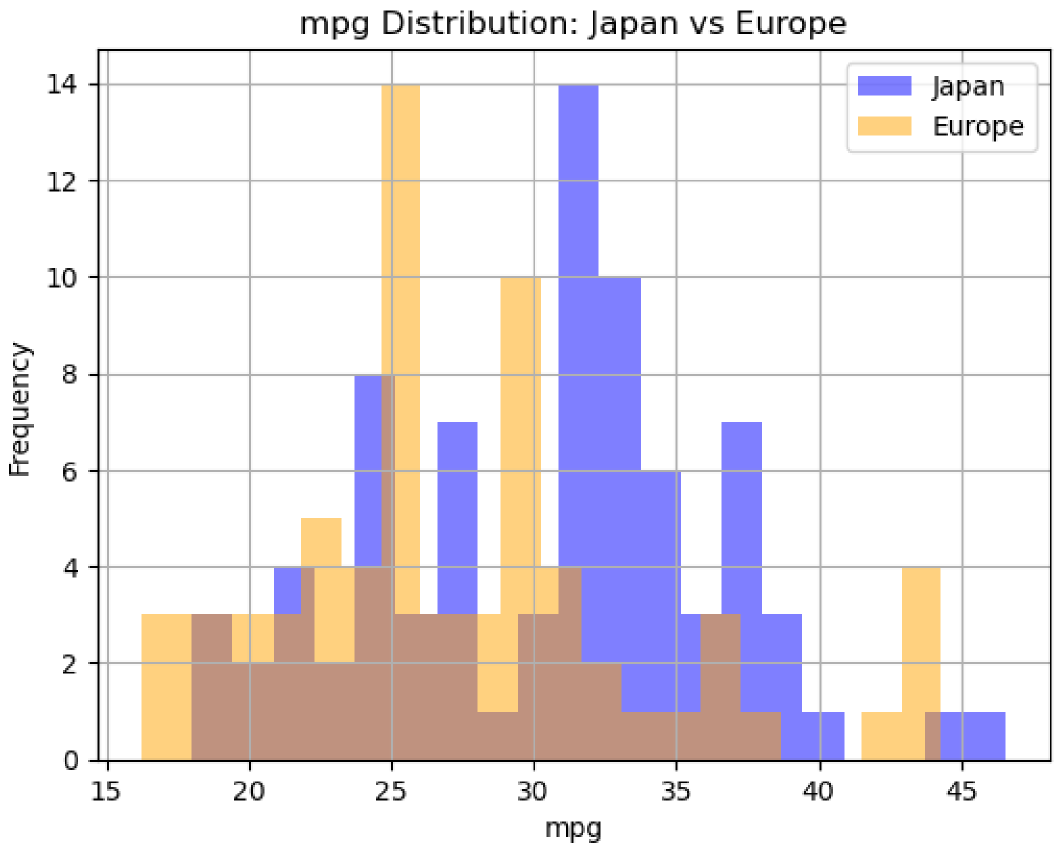


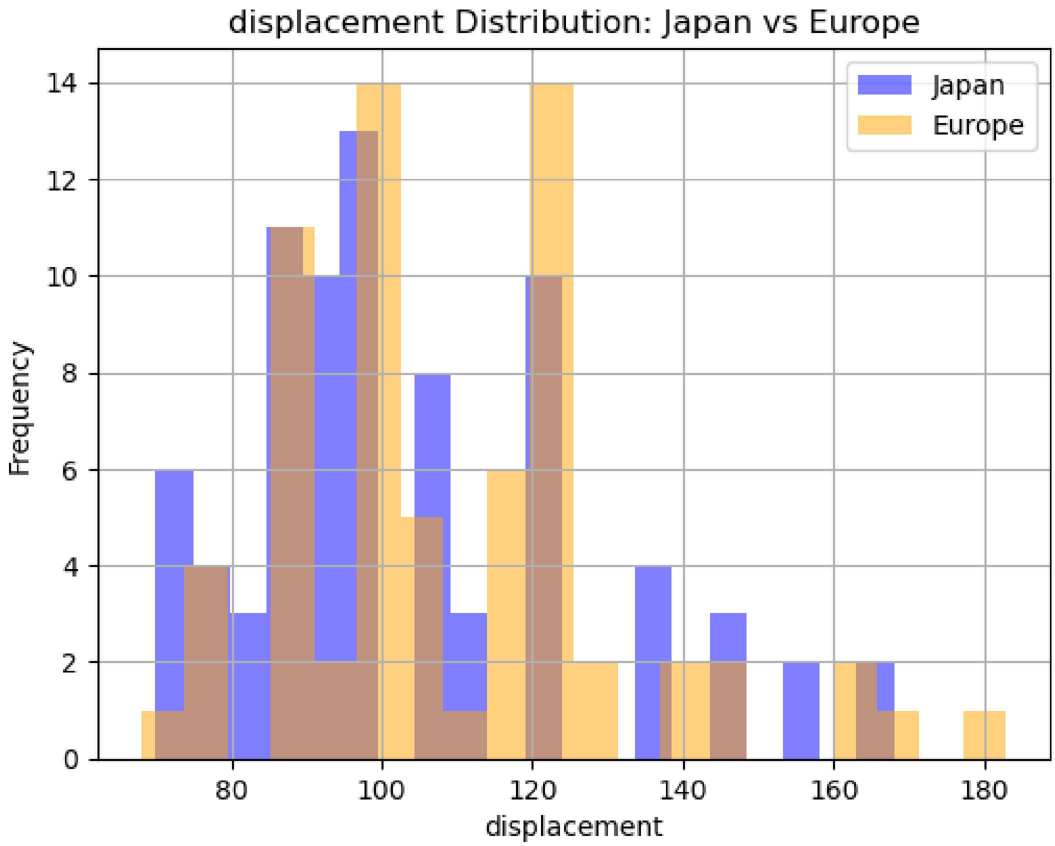
[20]:
```
# Exercise 2
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

data = pd.read_csv('auto-mpg.csv')

# Remove "Europe"
filtered_data = data[data['origin'].isin(['USA', 'Japan'])]

# Apply one-hot encoding
filtered_data = pd.get_dummies(filtered_data, columns=['origin'],␣
 ↪prefix='origin')

# Define features and targets
numerical_columns = ['mpg', 'displacement', 'horsepower', 'weight',␣
 ↪'acceleration']
```

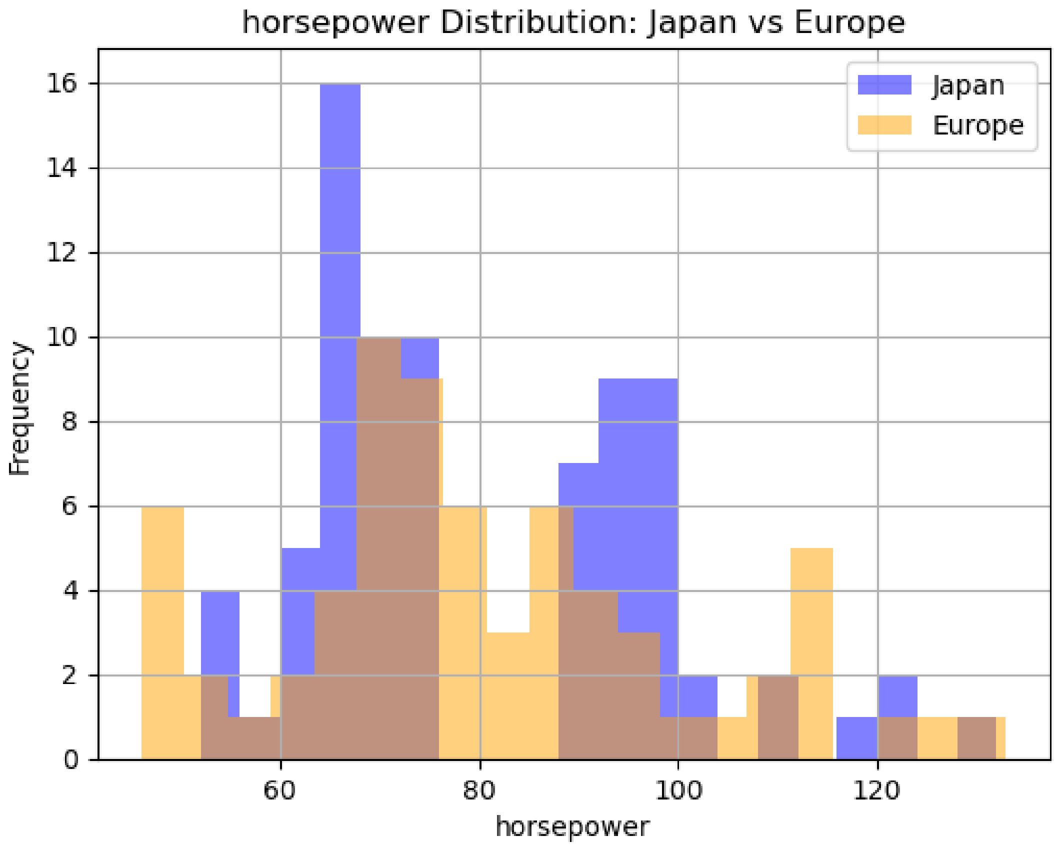
```python
X = filtered_data[numerical_columns]
y = filtered_data['origin_USA']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Build a Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Prediction
y_pred = model.predict(X_test)

# Report the precision and recall
report = classification_report(y_test, y_pred, target_names=['Japan', 'USA'])
print(report)

japan_europe_data = data[data['origin'].isin(['Japan', 'Europe'])]
japan_europe_data = pd.get_dummies(japan_europe_data, columns=['origin'],␣
↪prefix='origin')

for column in numerical_columns:
    japan_europe_data[japan_europe_data['origin_Japan'] == 1][column].
↪hist(bins=20, alpha=0.5, label='Japan', color='blue')
    japan_europe_data[japan_europe_data['origin_Japan'] == 0][column].
↪hist(bins=20, alpha=0.5, label='Europe', color='orange')
    plt.title(f'{column} Distribution: Japan vs Europe')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()

# From the histograms, the horsepower, mpg, and acceleration have clearly␣
↪different distributions,  which is highly predictive of the accuracy of a␣
↪model using them.
# On the other hand, the other features have a lot of overlap in their␣
↪distributions, which could potentially have negative effect on model␣
↪performance.
```

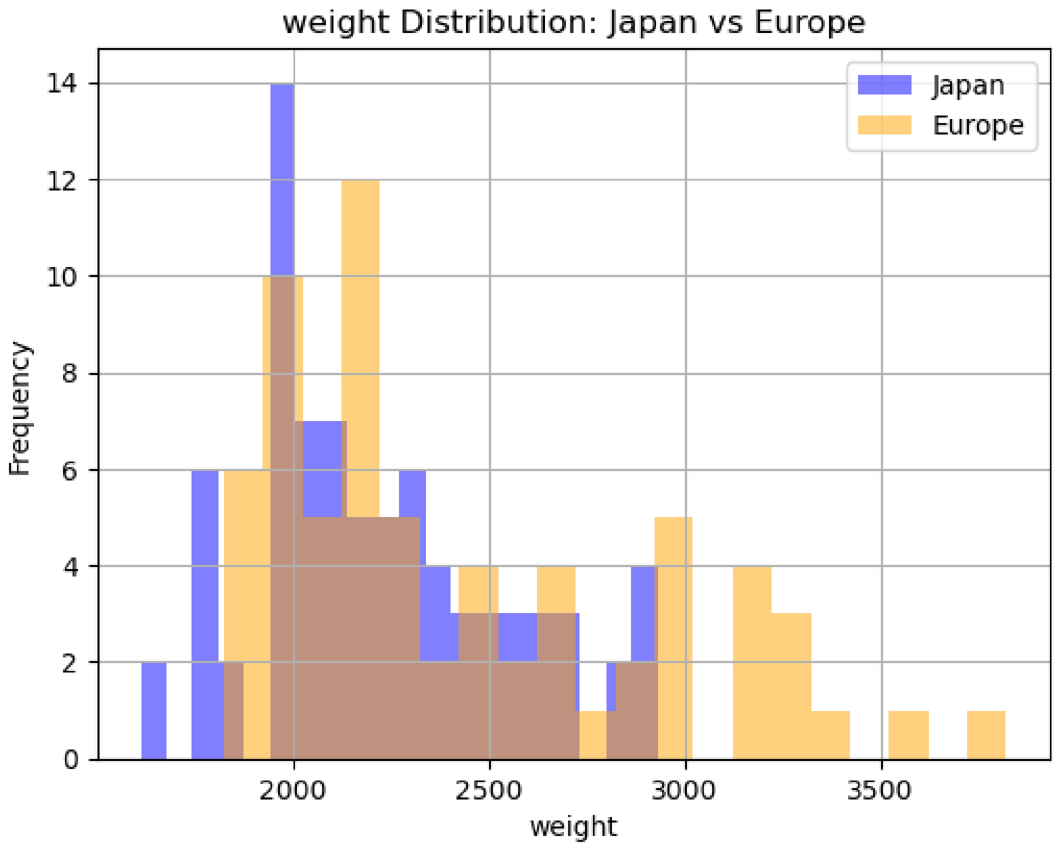
```
              precision    recall  f1-score   support

       Japan       0.76      0.94      0.84        17
         USA       0.98      0.90      0.93        48

    accuracy                           0.91        65
   macro avg       0.87      0.92      0.89        65
```

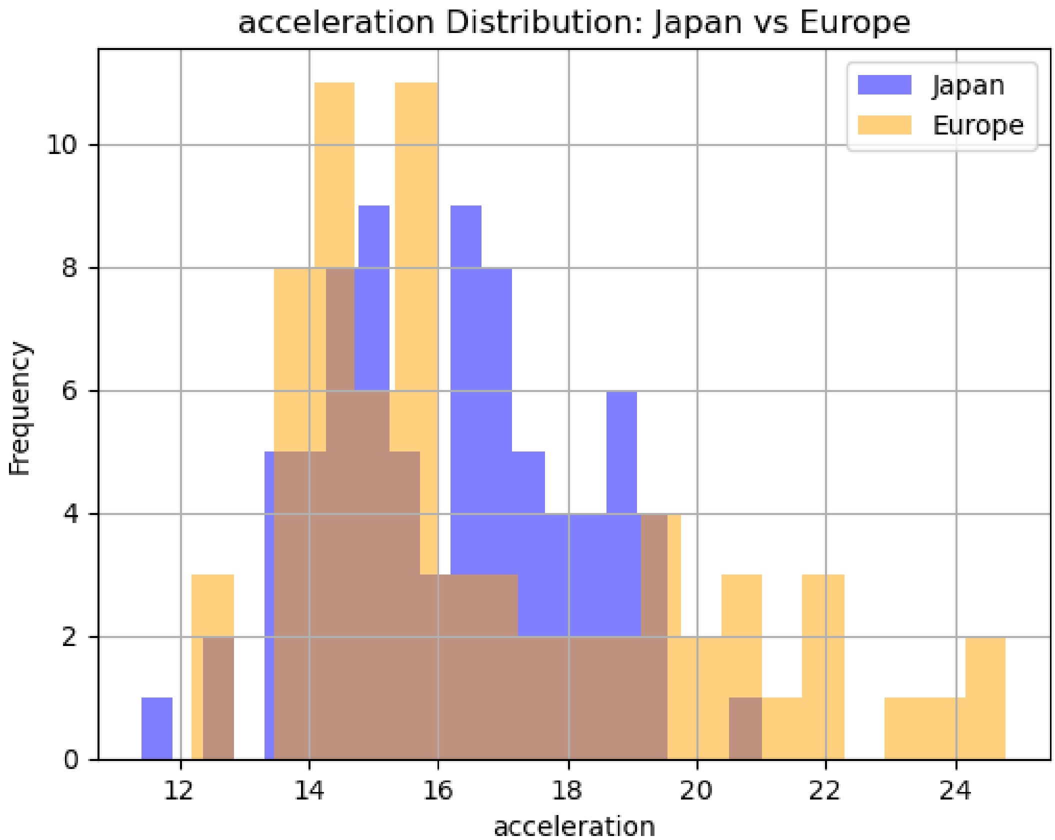
```
weighted avg       0.92      0.91      0.91        65
```




displacement Distribution: Japan vs Europe
Japan
Europe
Frequency
0
2
4
6
8
10
12
14
80
100
120
140
160
180
displacement


horsepower Distribution: Japan vs Europe
Japan
Europe
Frequency
horsepower
0
2
4
6
8
10
12
14
16
60
80
100
120


weight Distribution: Japan vs Europe
Japan
Europe
Frequency
weight
0
2
4
6
8
10
12
14
2000
2500
3000
3500

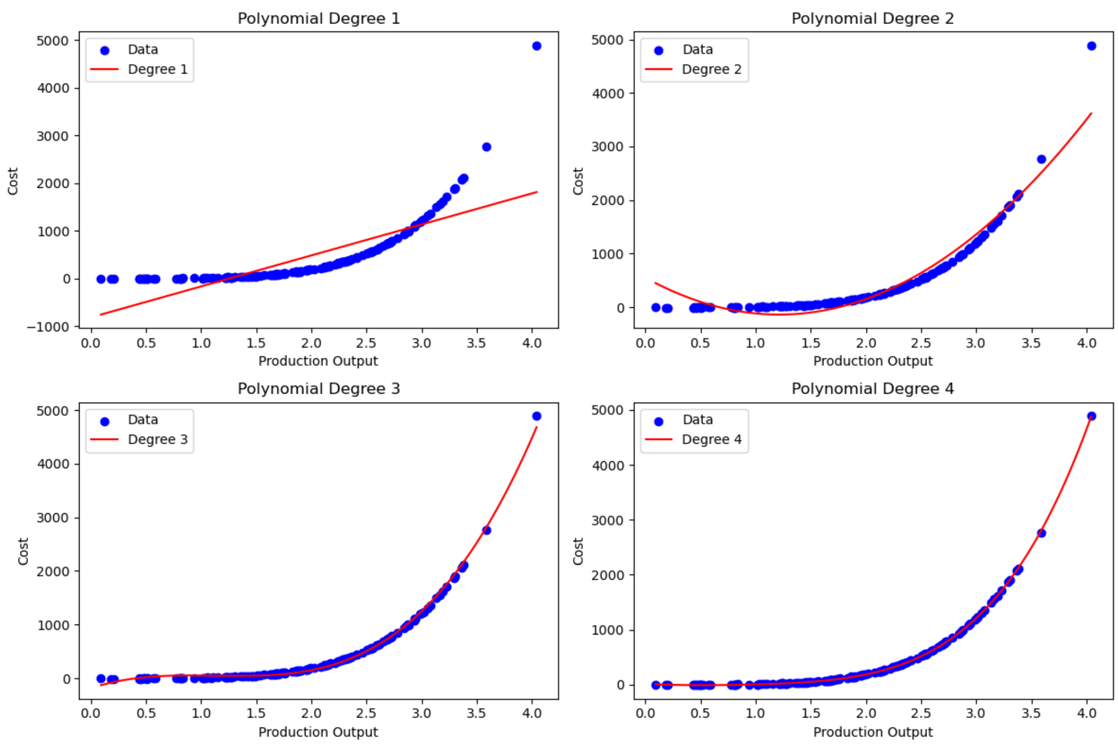


[23]:
```
# Exercise 3

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score


data = pd.read_csv('cost.csv')

# Define features and targets
X_cost = data[['production_output']]
y_cost = data['cost']

# Split the data
X_train_cost, X_test_cost, y_train_cost, y_test_cost = train_test_split(X_cost,␣
↪y_cost, test_size=0.3, random_state=42)
```

```python
print("Training set size:", X_train_cost.shape[0])
print("Testing set size:", X_test_cost.shape[0])

degrees = [1, 2, 3, 4]

results = []

plt.figure(figsize=(12, 8))

for degree in degrees:
    poly = PolynomialFeatures(degree)
    X_train_poly = poly.fit_transform(X_train_cost)
    X_test_poly = poly.fit_transform(X_test_cost)

    model = LinearRegression()
    model.fit(X_train_poly, y_train_cost)

    y_train_pred = model.predict(X_train_poly)
    y_test_pred = model.predict(X_test_poly)

    # Calculate the RMSE and R^2
    rmse_train = np.sqrt(mean_squared_error(y_train_cost, y_train_pred))
    rmse_test = np.sqrt(mean_squared_error(y_test_cost, y_test_pred))
    r2_train = r2_score(y_train_cost, y_train_pred)
    r2_test = r2_score(y_test_cost, y_test_pred)

    # Store the results
    results.append({
        'degree': degree,
        'rmse_train': rmse_train,
        'rmse_test': rmse_test,
        'r2_train': r2_train,
        'r2_test': r2_test
    })

    # Plotting and fitting line
    plt.subplot(2, 2, degree)
    plt.scatter(X_cost, y_cost, color='blue', label='Data')
    X_range = np.linspace(X_cost.min(), X_cost.max(), 100).reshape(-1, 1)
    X_range_poly = poly.fit_transform(X_range)
    y_range_pred = model.predict(X_range_poly)
    plt.plot(X_range, y_range_pred, color='red', label=f'Degree {degree}')
    plt.title(f'Polynomial Degree {degree}')
    plt.xlabel('Production Output')
    plt.ylabel('Cost')
    plt.legend()
```

```
plt.tight_layout()
plt.show()

# Show the result
results_df = pd.DataFrame(results)
print(results_df)

# From the result, we can see that the Polynomial Degree 4 has the lowest RMSE test score.
# Threrefore, I would say this one provides the most qppropreate prediction between models.
```

```
Training set size: 105
Testing set size: 45
```




```
   degree  rmse_train   rmse_test  r2_train   r2_test
0       1  438.842780  281.084793  0.607279  0.529909
1       2  181.973847  115.351446  0.932472  0.920831
2       3   39.467718   26.623867  0.996823  0.995783
3       4    5.995418    5.998611  0.999927  0.999786
```

[25]:
```
# Exercise 4
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt

data = pd.DataFrame({
    'Weight': [7.0, 6.0, 8.0],
    'Length': [50, 55, 56],
    'Actual_Height': [5.80, 5.70, 6.00]
})

# define the parameter of the model
coefficient_weight = 0.1
coefficient_length = 0.1
# Assume distribution to be 0.1
sigma_squared = 0.1

# Calculate the predicted height
data['Predicted_Height'] = coefficient_weight * data['Weight'] +
 ↪coefficient_length * data['Length']

# Calculate the Squared Residual
data['Squared_Residual'] = (data['Actual_Height'] - data['Predicted_Height'])
 ↪** 2

# Function to calculate the Log-Likelihood
def log_likelihood(y, y_pred, sigma_squared):
    residual_squared = (y - y_pred) ** 2
    return -0.5 * np.log10(2 * np.pi * sigma_squared) - (residual_squared / (2
 ↪* sigma_squared * np.log(10)))

# Calculate the Log-Likelihood for each data point
data['Log_Likelihood'] = data.apply(
    lambda row: log_likelihood(row['Actual_Height'], row['Predicted_Height'],
 ↪sigma_squared), axis=1
)

# Sum the Log-Likelihood
log_likelihood_sum = data['Log_Likelihood'].sum()

print(data[['Weight', 'Length', 'Actual_Height', 'Predicted_Height',
 ↪'Squared_Residual', 'Log_Likelihood']])
print(f"\nTotal Log-Likelihood: {log_likelihood_sum:.4f}")
```

```
   Weight  Length  Actual_Height  Predicted_Height  Squared_Residual  \
0     7.0      50            5.8               5.7              0.01
1     6.0      55            5.7               6.1              0.16
2     8.0      56            6.0               6.4              0.16
```

```
   Log_Likelihood
0        0.079195
1       -0.246526
2       -0.246526

Total Log-Likelihood: -0.4139
```