Ekta Shukla - A20567127

Rithika Kavitha Suresh - A20564346

Roger Kewin Samson - A20563057

Jude Rosun - A20564339

Model Selection k-fold cross-validation and bootstrapping

In [11]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import OneHotEncoder, StandardScaler, PolynomialFeatures
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.utils import resample
```

Load the dataset

In [12]:

```
file_path = "flight prediction.csv"  # Replace with the correct file path
data = pd.read_csv(file_path)

# Drop unnecessary columns and clean missing values
data = data.drop(columns=['Unnamed: 0'])  # Drop index-like column
data = data.dropna()  # Remove rows with missing values

X = data.drop(columns=['price'])  # Features
y = data['price']  # Target variable
```

In [13]:

```
categorical_features = ['airline', 'source_city', 'departure_time', 'stops',
                        'arrival_time', 'destination_city', 'class']
numerical_features = ['duration', 'days_left']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)

# Transform the dataset
X_transformed = preprocessor.fit_transform(X)
```

Split the data into training and testing sets

In [14]:

```
X_train, X_test, y_train, y_test = train_test_split(X_transformed, y, test_size=0.2, random_state=42)

X_train_dense = X_train.toarray()
X_test_dense = X_test.toarray()
```

Initialize Linear Regression

In [15]:

```
lr = LinearRegression()

# Implementing K-Fold Cross-Validation
kf = KFold(n_splits=10, shuffle=True, random_state=0)
cv_scores = []

for train_index, test_index in kf.split(X_train_dense):
    # Split data manually for each fold
    X_train_kf, X_test_kf = X_train_dense[train_index], X_train_dense[test_index]
    y_train_kf, y_test_kf = y_train.iloc[train_index], y_train.iloc[test_index]

    lr.fit(X_train_kf, y_train_kf)
    y_pred_kf = lr.predict(X_test_kf)

    rss = np.sum((y_test_kf - y_pred_kf)**2)
    tss = np.sum((y_test_kf - np.mean(y_test_kf))**2)
    r2_kf = 1 - (rss / tss)
    cv_scores.append(r2_kf)

# Calculate the mean R² score across folds
```

```
cv_mean_r2 = np.mean(cv_scores)
print(f"K-Fold Cross-Validation Mean R² Score: {cv_mean_r2:.4f}")
```

```
K-Fold Cross-Validation Mean R² Score: 0.9115
```

Bootstrapping Implementation

In [16]:

```
n_bootstraps = 50  # Number of bootstrap iterations
bootstrap_sample_size = int(0.5 * len(X_train_dense))  # Use 50% of the training data per iteration
bootstrap_scores = []

for _ in range(n_bootstraps):
    indices = np.random.choice(len(X_train_dense), size=bootstrap_sample_size, replace=True)
    X_bootstrap, y_bootstrap = X_train_dense[indices], y_train.iloc[indices]

    # Train and predict
    lr.fit(X_bootstrap, y_bootstrap)
    y_pred_bootstrap = lr.predict(X_test_dense)

    rss = np.sum((y_test - y_pred_bootstrap)**2)
    tss = np.sum((y_test - np.mean(y_test))**2)
    r2_bootstrap = 1 - (rss / tss)
    bootstrap_scores.append(r2_bootstrap)

# Calculate the mean and standard deviation of R²
bootstrap_mean_r2 = np.mean(bootstrap_scores)
bootstrap_std_r2 = np.std(bootstrap_scores)
print(f"Bootstrapping Mean R² Score: {bootstrap_mean_r2:.4f}")
```

```
Bootstrapping Mean R² Score: 0.9113
```

Print final results

In [24]:

```
print(f"Final Results:")
print(f"  K-Fold Cross-Validation Mean R²: {cv_mean_r2:.4f}")
print(f"  Bootstrapping Mean R²: {bootstrap_mean_r2:.4f}")
print(f"  Bootstrapping R² Std Dev: {bootstrap_std_r2:.4f}")
```

```
Final Results:
  K-Fold Cross-Validation Mean R²: 0.9115
  Bootstrapping Mean R²: 0.9113
  Bootstrapping R² Std Dev: 0.0000
```

In [17]:

```
# AIC
def calculate_aic(n, rss, k):
    return n * np.log(rss / n) + 2 * k
```

In [20]:

```
kf_aic_scores = []

for train_index, test_index in kf.split(X_train_dense):
    # Split data manually for each fold
    X_train_kf, X_test_kf = X_train_dense[train_index], X_train_dense[test_index]
    y_train_kf, y_test_kf = y_train.iloc[train_index], y_train.iloc[test_index]
    lr.fit(X_train_kf, y_train_kf)
    y_pred_kf = lr.predict(X_test_kf)

    rss = np.sum((y_test_kf - y_pred_kf)**2)
    tss = np.sum((y_test_kf - np.mean(y_test_kf))**2)
    r2_kf = 1 - (rss / tss)
    cv_scores.append(r2_kf)
    n = len(y_test_kf)
    k = X_train_kf.shape[1] + 1
    aic = calculate_aic(n, rss, k)
    kf_aic_scores.append(aic)
# Print average AIC for K-Fold
mean_aic_kf = np.mean(kf_aic_scores)
print(f"K-Fold Cross-Validation Mean AIC: {mean_aic_kf:.4f}")
```

```
K-Fold Cross-Validation Mean AIC: 423538.1198
```

In [22]:

```
bootstrap_aic_scores = []

for _ in range(n_bootstraps):
    indices = np.random.choice(len(X_train_dense), size=bootstrap_sample_size, replace=True)
    X_bootstrap, y_bootstrap = X_train_dense[indices], y_train.iloc[indices]
    lr.fit(X_bootstrap, y_bootstrap)
    y_pred_bootstrap = lr.predict(X_test_dense)
    rss = np.sum((y_test - y_pred_bootstrap)**2)
    tss = np.sum((y_test - np.mean(y_test))**2)
    r2_bootstrap = 1 - (rss / tss)
    bootstrap_scores.append(r2_bootstrap)
    n = len(y_test)
    k = X_bootstrap.shape[1] + 1
    aic = calculate_aic(n, rss, k)
    bootstrap_aic_scores.append(aic)

# Print average AIC for Bootstrapping
mean_aic_bootstrap = np.mean(bootstrap_aic_scores)
print(f"Bootstrapping Mean AIC: {mean_aic_bootstrap:.4f}")
```

```
Bootstrapping Mean AIC: 1058920.7911
```

```
In [ ]:
```