In [3]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
```

In [4]:
```
# Load data
air = pd.read_excel("Airfares.xlsx", sheet_name="data")

# Drop first four predictors (assignment allows ignoring these)
air = air.drop(columns=["S_CODE", "S_CITY", "E_CODE", "E_CITY"])

# Optional: inspect columns
air.head()
```

Out[4]:

	COUPON	NEW	VACATION	SW	HI	S_INCOME	E_INCOME	S_POP	
0	1.00	3	No	Yes	5291.991341	28637	21112	3036732	
1	1.06	3	No	No	5419.160907	26993	29838	3532657	7
2	1.06	3	No	No	9185.283234	30124	29838	5787293	7
3	1.06	3	No	Yes	2657.351987	29260	29838	7830332	7
4	1.06	3	No	Yes	2657.351987	29260	29838	7830332	7

In [5]:
```
cat_vars = ["VACATION", "SW", "SLOT", "GATE"]

air = pd.get_dummies(air, columns=cat_vars, drop_first=True)

air = air.astype({col: int for col in air.select_dtypes(include="bool").colu
```

In [7]:
```
train, temp = train_test_split(air, test_size=0.30, random_state=123)
validation, test = train_test_split(temp, test_size=1/3, random_state=123)
```

In [10]:
```
X_train = train.drop(columns=["FARE"])
y_train = train["FARE"]

X_train = sm.add_constant(X_train)

X_train = X_train.apply(pd.to_numeric, errors="coerce")
y_train = pd.to_numeric(y_train, errors="coerce")

data_clean = pd.concat([X_train, y_train], axis=1).dropna()

X_train = data_clean.drop(columns=["FARE"])
y_train = data_clean["FARE"]
```

In [11]:
```
def backward_elimination(X, y, alpha=0.05):
    model = sm.OLS(y, X).fit()

    while model.pvalues.max() > alpha:
```

```
        worst_feature = model.pvalues.idxmax()
        if worst_feature == "const":
            break
        X = X.drop(columns=[worst_feature])
        model = sm.OLS(y, X).fit()

    return model, X

final_model, X_train_selected = backward_elimination(X_train, y_train)
print(final_model.summary())
```

OLS Regression Results

Dep. Variable:	FARE	R-squared:	0.777
Model:	OLS	Adj. R-squared:	0.771
Method:	Least Squares	F-statistic:	116.0
Date:	Tue, 10 Feb 2026	Prob (F-statistic):	8.46e-132
Time:	11:00:58	Log-Likelihood:	-2233.8
No. Observations:	446	AIC:	4496.
Df Residuals:	432	BIC:	4553.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	12.3301	33.335	0.370	0.712	-53.188	77.849
COUPON	11.4786	14.886	0.771	0.441	-17.779	40.736
NEW	-4.4448	2.375	-1.872	0.062	-9.112	0.223
HI	0.0085	0.001	6.778	0.000	0.006	0.011
S_INCOME	0.0013	0.001	2.101	0.036	8.45e-05	0.003
E_INCOME	0.0012	0.000	2.745	0.006	0.000	0.002
S_POP	3.412e-06	7.91e-07	4.314	0.000	1.86e-06	4.97e-06
E_POP	4.12e-06	9.14e-07	4.508	0.000	2.32e-06	5.92e-06
DISTANCE	0.0750	0.004	16.774	0.000	0.066	0.084
PAX	-0.0008	0.000	-4.578	0.000	-0.001	-0.000
VACATION_Yes	-36.2304	4.407	-8.222	0.000	-44.891	-27.569
SW_Yes	-39.5331	4.761	-8.304	0.000	-48.890	-30.176
SLOT_Free	-17.1268	4.790	-3.575	0.000	-26.542	-7.712
GATE_Free	-21.9189	5.134	-4.269	0.000	-32.009	-11.828

Omnibus:	0.431	Durbin-Watson:	1.9

```
06
Prob(Omnibus):                  0.806   Jarque-Bera (JB):                0.4
82
Skew:                           0.074   Prob(JB):                        0.7
86
Kurtosis:                       2.934   Cond. No.                     1.21e+
08
==============================================================================
==

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
[2] The condition number is large, 1.21e+08. This might indicate that there
are
strong multicollinearity or other numerical problems.
```

In [12]:
```
X_val = validation[X_train_selected.columns.drop("const")]
X_val = sm.add_constant(X_val)
y_val = validation["FARE"]

val_preds = final_model.predict(X_val)
val_model = sm.OLS(y_val, val_preds).fit()
print("Validation Adjusted R²:", val_model.rsquared_adj)
```

```
Validation Adjusted R²: 0.9683545731130829
```

In [13]:
```
X_test = test[X_train_selected.columns.drop("const")]
X_test = sm.add_constant(X_test)
y_test = test["FARE"]

test_preds = final_model.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, test_preds))
avg_error = np.mean(test_preds - y_test)

print("Test RMSE:", rmse)
print("Test Average Error:", avg_error)
```

```
Test RMSE: 34.090129394111514
Test Average Error: 0.9582018609527441
```

In [15]:
```
new_route = pd.DataFrame({
    "COUPON": [1],
    "NEW": [3],
    "HI": [4442.141],
    "S_INCOME": [28760],
    "E_INCOME": [27664],
    "S_POP": [4557004],
    "E_POP": [3195503],
    "DISTANCE": [1976],
    "PAX": [12782],
    "VACATION_Yes": [0],
    "SW_Yes": [0],
    "SLOT_Free": [0],
    "GATE_Free": [0]
})
```

```
# Reindex to exactly match X_train_selected
new_route = new_route.reindex(columns=X_train_selected.columns, fill_value=0

# No need to add_constant — already included in X_train_selected
predicted_fare = final_model.predict(new_route)
print("Predicted fare for new route:", predicted_fare.values[0])
```

```
Predicted fare for new route: 274.08931978342594
```

In [ ]: