DATA70121 Coursework

Student ID: 14199981

November 16, 2025

1 Introduction

This study analyzes customer refund-seeking behaviour in UK rail transport using the Maven-Rail dataset, comprising 30,686 passenger journeys recorded between January and April 2024. Each observation captures 13 attributes including transaction characteristics, route information, temporal features, service performance indicators, and the binary variable indicating refund requests following service disruptions.

This study aims to model refund request behaviour prediction as a function of journey characteristics and service failures. Identifying the determinants of compensation claims provides substantive operational value for rail operators seeking to optimise resource deployment and enhance passenger satisfaction management. The dataset exhibits pronounced class imbalance, with refund requests constituting only 3.6% of all observations. Consequently, the modelling strategy necessitates the use of imbalance-aware evaluation metrics and class-weighting approaches to ensure robust and unbiased predictive performance.(Chawla et al. 2002).

2 Data Description

2.1 Data Processing

Data processing procedures ensured that key variables were standardised and analytically reliable. Price, Journey.Status and Refund.Request showed complete observations, while Railcard contained substantial missingness at 66.2%. This absence was interpreted as an inherent attribute of passengers rather than an error introduced during data ingestion, and the variable was retained without imputation. The dataset also included 5,525 duplicated rows. As each row corresponds to an individual journey instance, these duplicates were preserved to maintain fidelity to the source data. Temporal attributes including Departure, Scheduled.Arrival and Actual.Arrival were cleaned, converted into consistent datetime formats and validated to support downstream feature engineering. (McKinney et al. 2010).

2.2 Descriptive Statistics

Table 1 presents summary statistics for key numeric and categorical variables. Service performance remained stable with 86.6% of journeys arriving on time, 7.5% experiencing delays and 6.0% being cancelled. Ticket pricing varied widely from £1 to £267, with a median of £11 and

a mean of £23.21, illustrating a right-skewed distribution driven by the dominance of lower-fare Advance and Standard class tickets.

Table 1: Descriptive statistics for key variables (n = 30,686)

Variable	**Mean**	**Median**	**SD**	**Min**	**Max**
Price (£)	23.21	11.00	29.98	1.00	267.00
Scheduled.Arrival (min)	70.75	80.00	36.15	15.00	260.00
Actual.Arrival (min)	74.07	80.00	40.20	15.00	288.00
Delay (min, if delayed)	42.57	37.00	34.44	1.00	180.00
Categorical Variables					
Journey Status: On Time (86.6%), Delayed (7.5%), Cancelled (6.0%)					
Refund Request: No (96.4%), Yes (3.6%)					
Ticket Type: Advance (55.5%), Off-Peak (27.1%), Anytime (17.4%)					
Ticket Class: Standard (90.3%), First Class (9.7%)					

3 Exploratory Data Analysis

3.1 Service Performance

Figure 1 summarises that On-time arrivals accounted for 86.6% of all journeys, indicating a largely reliable level of service delivery. Delays represented 7.5% and cancellations a further 6.0%, signalling service failures with potential operational and customer-experience implications. Within the subset of non-punctual journeys, refund request rates rose markedly, demonstrating a relationship between service disruption and compensation-seeking behaviour.

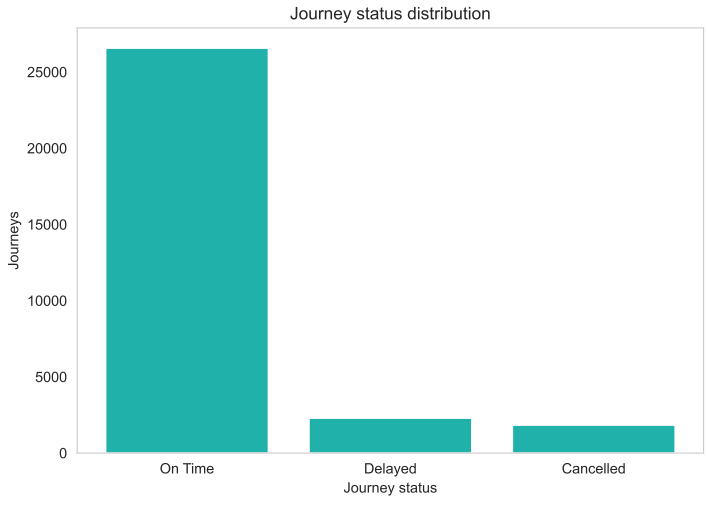

Figure 1: Service Performance Distribution.

3.2 Departure-time pattern Demand

Figure 2 exhibit a clear bimodal structure, with pronounced morning peaks between 07:00 and 09:00 and evening peaks between 17:00 and 19:00. These concentrations reflect commuter-driven travel demand, characterised by high passenger throughput during rush hours.

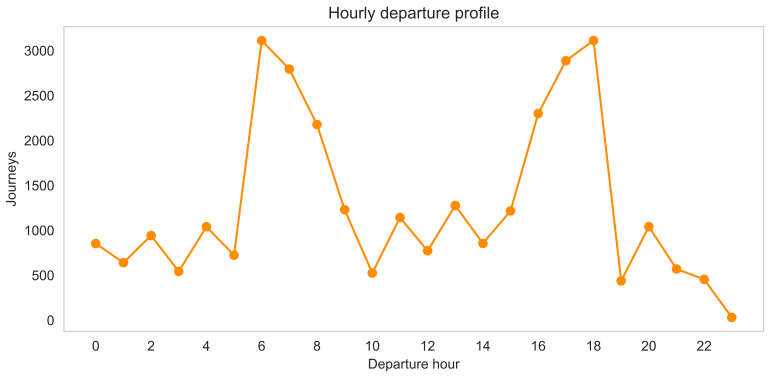

Figure 2: Departure-Time Distribution.

3.3 Price-Duration Relationship

Figure 3 demonstrates positive correlation between duration and price, with longer routes commanding premium fares. This suggests disruptions to longer journeys represent higher-value service failures, potentially elevating refund propensity.

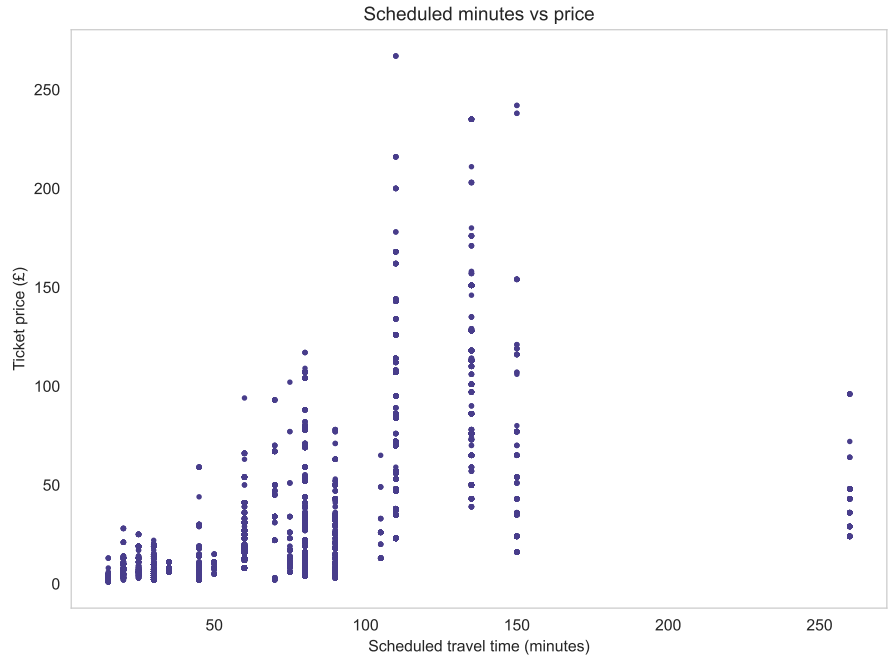

Figure 3: Price-Duration Distribution.

3.4 Service Delays Distribution

Amongst non-punctual journeys, delay durations exhibited right-skewed distribution (Figure 4) with long tail. Although most disruptions impose modest time penalties, a minority experience severe delays. Prior researcher suggests such extreme failures disproportionately drive dissatisfaction and compensatory behaviour (Olsson et al. 2012).

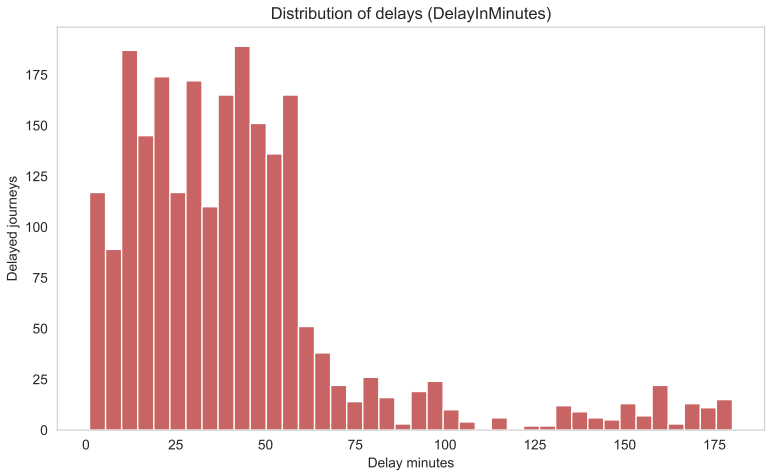

Figure 4: Service Disruption Distribution.

3.5 Multivariate Relationships Among Key Variables

Figure 5 reveals journey duration metrics, including ScheduledMinutes, ActualMinutes, exhibit positive correlation with ticket price, reflecting the distance-dependent structure of rail fares. In contrast, DelayInMinutes demonstrates much weaker alignment with price, implying that service disruptions tend to affect passengers in a broadly uniform manner across different fare levels.

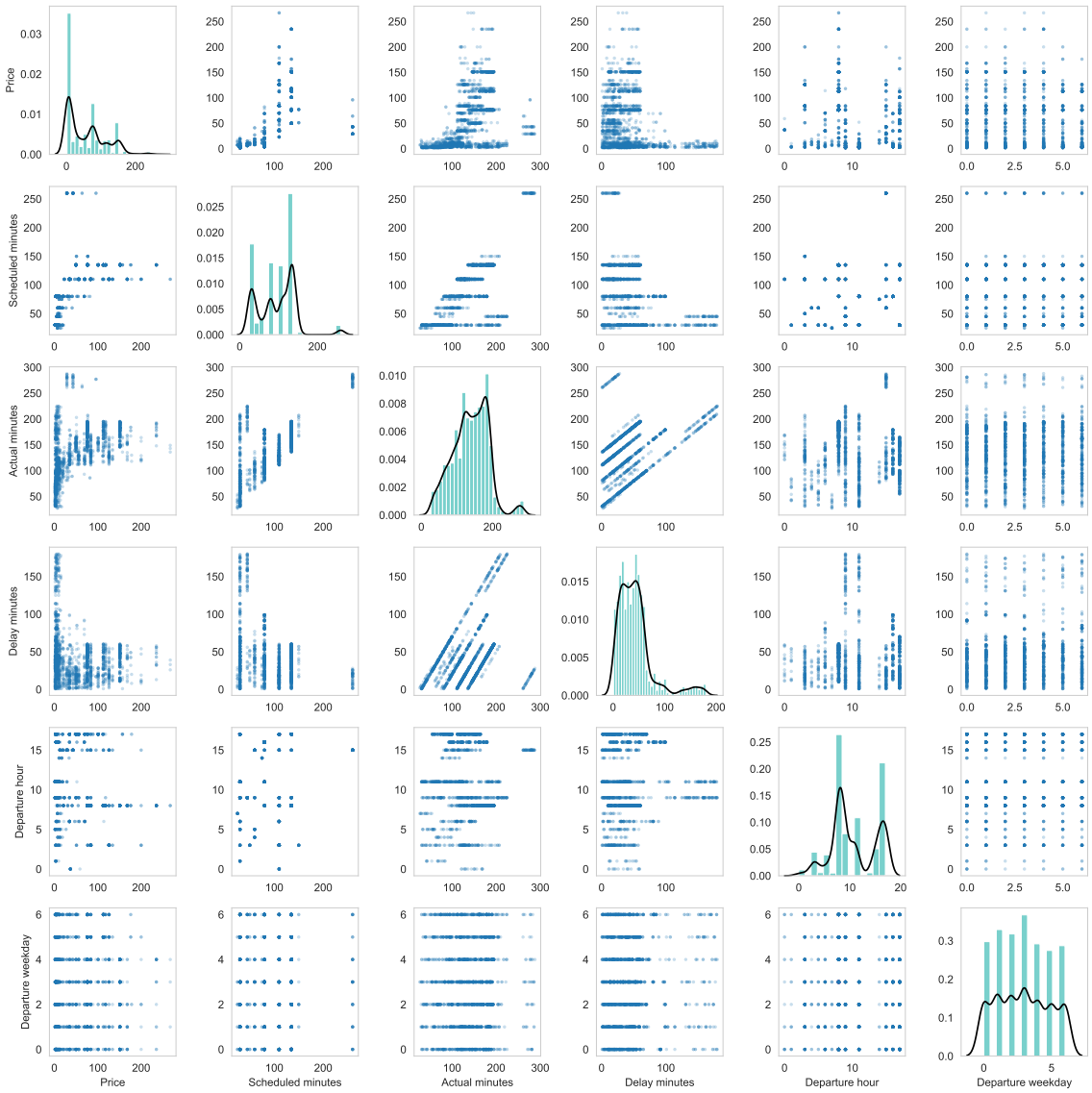

Figure 5: Multivariate Distribution and Correlation.

4 Feature Engineering

DelayInMinutes was derived by computing the temporal difference between Actual.Arrival and Scheduled.Arrival, retaining only positive values to represent true delay magnitudes. Journeys that arrived on time or early, where the difference was zero or negative, were assigned NA to maintain a clear distinction between punctual services and disrupted journeys. This processing step produced 28,416 NA entries, representing 92.6% of all observations, and yielded 2,270 valid delay measurements for subsequent analysis (Olsson et al. 2012).

5 Single Variable Logistic Regression Model

5.1 Data Subset and Feature Construction

Analysis was restricted to non-punctual journeys (Journey.Status $\neq$ "On Time"), yielding n = 4,121. A binary predictor MediumPrice was constructed to test whether mid-range fare passengers (£10 < Price $\leq$ £30) exhibit distinct refund-seeking behaviour:

$$\text{MediumPrice} = \begin{cases} 1 & \text{if } 10 < \text{Price} \leq 30 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

5.2 Model Specification

The logistic regression model is defined as:

Linear component:

$$\text{logit}(P(\text{Refund} = \text{Yes})) = \log\left(\frac{P(\text{Refund} = \text{Yes})}{1 - P(\text{Refund} = \text{Yes})}\right) = \beta_0 + \beta_1 \times \text{MediumPrice} \tag{2}$$

Probability transformation:

$$P(\text{Refund} = \text{Yes} \mid \text{MediumPrice}) = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 \times \text{MediumPrice}))} \tag{3}$$

where β_0 represents the baseline log-odds of refund requests for non-medium-priced tickets, and β_1 quantifies the change in log-odds associated with medium-priced tickets.

5.3 Empirical Analysis

Maximum likelihood estimation yielded: intercept $\hat{\beta}_0 = -1.084$ (SE = 0.040, p < 0.001) and slope $\hat{\beta}_1 = 0.354$ (SE = 0.088, p < 0.001). The coefficient supports the hypothesis that mid-range fare passengers exhibit elevated refund propensity. The odds ratio exp(0.354) = 1.43 indicates medium-priced ticket holders have 43% higher odds of requesting refunds compared to other fare categories, ceteris paribus.

5.4 Refund Probability Calculations

Using the fitted coefficients and the logistic transformation formula, predicted refund probabilities were computed:

For a £5 ticket (MediumPrice = 0):

$$\text{Linear predictor: } \eta = \hat{\beta}_0 + \hat{\beta}_1 \times 0 = -1.084 \tag{4}$$

$$\text{Probability: } P(\text{Refund} = \text{Yes}) = \frac{1}{1 + \exp(1.084)} = \mathbf{0.253} \tag{5}$$

For a £25 ticket (MediumPrice = 1):

$$\text{Linear predictor: } \eta = \hat{\beta}_0 + \hat{\beta}_1 \times 1 = -1.084 + 0.354 = -0.730 \tag{6}$$

$$\text{Probability: } P(\text{Refund} = \text{Yes}) = \frac{1}{1 + \exp(0.730)} = \mathbf{0.325} \tag{7}$$

These analyses demonstrate that mid-range ticket holders on disrupted services were substantially more likely to seek refunds.

6 Multivariable Refund Prediction Models

6.1 Methodological Framework

A comprehensive feature set was assembled encompassing fare structure (Price), temporal characteristics (DepartureHour, DepartureDayOfWeek), journey duration metrics (ScheduledMinutes, ActualMinutes), and categorical attributes (Payment.Method, Ticket.Class, Ticket.Type, Journey.Status, Reason.for.Delay). Categorical predictors were one-hot encoded (Hastie et al. 2009).

Three algorithms were evaluated: Logistic Regression, Decision Tree, and Random Forest. Given severe class imbalance (3.6% minority), all models used `class_weight='balanced'` to mitigate majority class bias (Chen et al. 2004). Data were partitioned 80/20 (training/testing) using stratified sampling. Let `random_state=123` ensure reproducibility.

6.2 Model Performance Comparison

Table 2 presents test set performance. All algorithms demonstrated exceptional discrimination with ROC-AUC > 0.97.

Table 2: Classification model performance on test set (20% holdout)

Model	Accuracy	Precision	Recall	F1	ROC-AUC	PR-AUC
Logistic Regression	0.941	0.375	0.973	0.541	0.984	0.669
Decision Tree	0.930	0.338	0.995	0.505	0.986	0.649
Random Forest	0.970	0.569	0.662	0.612	0.967	0.745

Decision Tree achieved highest ROC-AUC (0.986), whilst Random Forest demonstrated superior PR-AUC (0.745). Logistic Regression attained competitive discrimination (ROC-AUC = 0.984) with near-perfect recall (0.973), identifying 97.3% of refund requests.

ROC curves (Figure 6) displays sensitivity-specificity trade-offs. All models substantially outperform random classification (diagonal), with Decision Tree and Logistic Regression exhibiting nearly identical performance. PR curves (Figure 7) emphasise minority class detection. PR-AUC values (0.669–0.745) substantially exceed no-skill baseline (0.0360), confirming genuine predictive signal extraction despite severe imbalance.

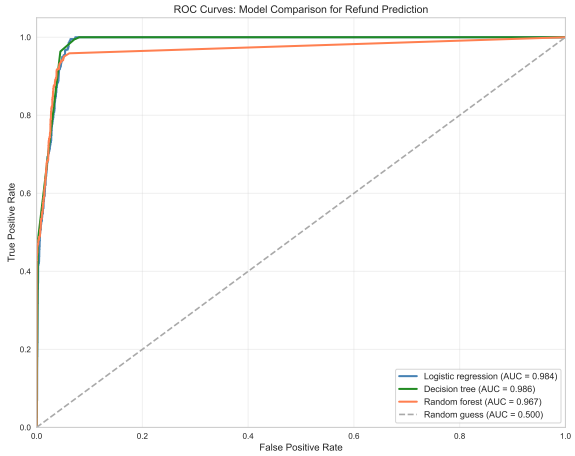

Figure 6: ROC-AUC Curves.

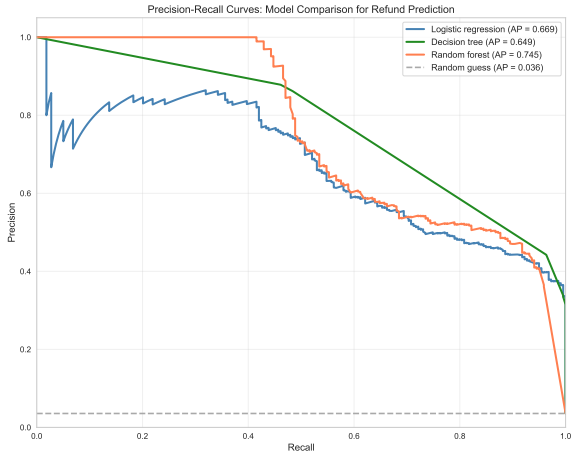

Figure 7: Precision-Recall Curves.

6.3 Final Model Selection and Justification

Building on the comparative results above, the final model was selected using three criteria: discrimination, interpretability, and minority-class sensitivity. Logistic regression and decision trees perform similarly in ROC-AUC and provide interpretable log-odds ratios (Hosmer Jr et al. 2013). While random forests have higher accuracy, their recall is significantly lower than that of logistic regression.

6.4 Predicting Refund Probabilities

Using the previously validated logistic regression model, refund probabilities were generated for the unlabelled journeys, with the results summarised in Table 3.

Table 3: Predicted refund probabilities for unlabelled journeys

Price (£)	Status	Ticket Type	Class	Payment	Refund	Prob. (%)
54	On Time	Advance	First Class	Debit Card	NO	0.7
7	On Time	Advance	Standard	Credit Card	NO	0.0
113	Delayed	Off-Peak	Standard	Debit Card	NO	97.8
3	Delayed	Off-Peak	Standard	Contactless	NO	88.9
4	Cancelled	Off-Peak	Standard	Credit Card	NO	92.8
3	Cancelled	Advance	Standard	Contactless	NO	90.3
126	Delayed	Off-Peak	Standard	Debit Card	NO	98.2
22	Cancelled	Advance	Standard	Credit Card	YES	91.5

7 Conclusion

This study demonstrates that logistic regression is effective in predicting refund behavior among UK rail passengers. The model achieves robust predictive performance based on journey char-

acteristics and service performance indicators, striking a good balance between accuracy and interpretability. A comparison of the three models further shows that while tree-based models are similar in discrimination, their recall is insufficient and interpretability is weak, thus confirming logistic regression as the most reliable and operational choice. Applying the final model to unlabeled journeys yields probability distributions consistent with the preceding empirical analysis: journeys with severe delays, higher fares, and those using flexible tickets have a higher propensity to request refunds.

References

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. (2002), 'Smote: synthetic minority over-sampling technique', *Journal of Artificial Intelligence Research* **16**, 321–357.

Chen, C., Liaw, A. & Breiman, L. (2004), Using random forest to learn imbalanced data, Technical Report Technical Report 666, Department of Statistics, University of California, Berkeley.
URL: *https://statistics.berkeley.edu/sites/default/files/tech-reports/666.pdf*

Hastie, T., Tibshirani, R. & Friedman, J. (2009), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd edn, Springer, New York.

Hosmer Jr, D. W., Lemeshow, S. & Sturdivant, R. X. (2013), *Applied Logistic Regression*, 3rd edn, John Wiley & Sons, Hoboken, NJ.

McKinney, W. et al. (2010), Data structures for statistical computing in python, *in* 'Proceedings of the 9th Python in Science Conference', Vol. 445, Austin, TX, pp. 51–56.

Olsson, N. O., Friman, M. & Lättman, K. (2012), 'Satisfaction with train travel: A study of the relationship between service quality, journey satisfaction and customer loyalty', *Transportation Research Part A: Policy and Practice* **46**(9), 1444–1454.

Appendix A: Python Code

```
# ===== Section 1: Imports & Settings =====
import warnings
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    average_precision_score,
    f1_score,
    precision_recall_curve,
    precision_score,
    recall_score,
    roc_auc_score,
    roc_curve,
)
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

warnings.simplefilter(action="ignore", category=FutureWarning)
sns.set_style("whitegrid", {"axes.grid": False})

BASE_DIR = Path(__file__).resolve().parent
FIG_DIR = BASE_DIR / "Figures"
DATA_PATH = BASE_DIR / "MavenRail_cleaned2.csv"
PREDICTION_PATH = BASE_DIR / "ToPredict.csv"
RANDOM_STATE = 123
np.random.seed(RANDOM_STATE)

if not FIG_DIR.exists():
    FIG_DIR.mkdir(parents=True)
print(f"Figures directory: {FIG_DIR.resolve()}")

# ===== Section 2: Dataset Loading =====
if not DATA_PATH.exists():
    raise FileNotFoundError(f"Dataset not found at {DATA_PATH.resolve()}")
```

```
rail_df = pd.read_csv(DATA_PATH)
print(f"Loaded dataset shape: {rail_df.shape}")

duplicate_count = rail_df.duplicated().sum()
print(f"Duplicate rows retained: {duplicate_count}")

critical_cols = [
    "Price",
    "Journey.Status",
    "Refund.Request",
    "Departure",
    "Scheduled.Arrival",
    "Actual.Arrival",
]
missing_summary = rail_df[critical_cols].isna().sum().to_frame(name="missing_count")
missing_summary["missing_pct"] = (
    missing_summary["missing_count"] / len(rail_df) * 100
).round(2)
print("Missingness summary for critical columns:\n", missing_summary)

rows_before = len(rail_df)
rail_df = rail_df.dropna(subset=["Price", "Journey.Status", "Refund.Request"])
print(
    f"Rows removed due to missing critical fields: {rows_before - len(rail_df)}"
)
print(f"Dataset after cleaning shape: {rail_df.shape}")

# ===== Section 3: Data Cleaning & Feature Engineering =====
datetime_cols = ["Departure", "Scheduled.Arrival", "Actual.Arrival"]
for col in datetime_cols:
    rail_df[col] = pd.to_datetime(rail_df[col], format="%Y-%m-%d %H:%M")

durations = rail_df["Actual.Arrival"] - rail_df["Scheduled.Arrival"]
delay_minutes = durations.dt.total_seconds() / 60
rail_df["DelayInMinutes"] = np.where(delay_minutes > 0, delay_minutes, np.nan)

print(f"Total journeys: {len(rail_df):,}")
print(f"Journeys on time or early: {(delay_minutes <= 0).sum():,}")
print(f"DelayInMinutes NA count: {rail_df['DelayInMinutes'].isna().sum():,}")

rail_df["ScheduledMinutes"] = (
    rail_df["Scheduled.Arrival"] - rail_df["Departure"]
```

```
).dt.total_seconds() / 60
rail_df["ActualMinutes"] = (
    rail_df["Actual.Arrival"] - rail_df["Departure"]
).dt.total_seconds() / 60
rail_df["DepartureHour"] = rail_df["Departure"].dt.hour
rail_df["DepartureDayOfWeek"] = rail_df["Departure"].dt.dayofweek

print("Summary of key numeric variables:")
print(
    rail_df[["Price", "ScheduledMinutes", "ActualMinutes"]]
    .describe()
    .round(2)
)

numeric_specs = [
    ("Price (£)", "Price"),
    ("Scheduled.Arrival (min)", "ScheduledMinutes"),
    ("Actual.Arrival (min)", "ActualMinutes"),
    ("Delay (min, if delayed)", "DelayInMinutes"),
]
stats_rows = []
for label, col in numeric_specs:
    series = rail_df[col].dropna()
    if series.empty:
        continue
    stats_rows.append(
        {
            "Variable": label,
            "Mean": series.mean(),
            "Median": series.median(),
            "SD": series.std(),
            "Min": series.min(),
            "Max": series.max(),
        }
    )
stats_df = pd.DataFrame(stats_rows)
stats_df[["Mean", "Median", "SD", "Min", "Max"]] = stats_df[
    ["Mean", "Median", "SD", "Min", "Max"]
].round(2)
print("\nDescriptive statistics used in LaTeX table:\n", stats_df)

categorical_mapping = {
```

```
    "Journey Status": "Journey.Status",
    "Refund Request": "Refund.Request",
    "Ticket Type": "Ticket.Type",
    "Ticket Class": "Ticket.Class",
}
print("\nCategorical distributions (%):")
for label, col in categorical_mapping.items():
    distribution = rail_df[col].value_counts(normalize=True).mul(100).round(1)
    formatted = ", ".join(f"{idx} ({pct}%)" for idx, pct in distribution.items())
    print(f"{label}: {formatted}")

# ===== Section 4: Exploratory Data Analysis =====
scatter_df = rail_df[["ScheduledMinutes", "Price"]].dropna()
plt.figure(figsize=(8, 6))
plt.scatter(scatter_df["ScheduledMinutes"], scatter_df["Price"], s=6, color="darkslateblue")
plt.xlabel("Scheduled travel time (minutes)")
plt.ylabel("Ticket price (£)")
plt.title("Scheduled minutes vs price")
plt.tight_layout()
plt.savefig(FIG_DIR / "eda_sched_vs_price.pdf", format="pdf")
plt.close()
print("Saved eda_sched_vs_price.pdf")

hour_counts = rail_df["DepartureHour"].value_counts().sort_index()
plt.figure(figsize=(8, 4))
plt.plot(hour_counts.index, hour_counts.values, marker="o", color="darkorange")
plt.xlabel("Departure hour")
plt.ylabel("Journeys")
plt.title("Hourly departure profile")
plt.xticks(range(0, 24, 2))
plt.tight_layout()
plt.savefig(FIG_DIR / "eda_hour_profile.pdf", format="pdf")
plt.close()
print("Saved eda_hour_profile.pdf")

delay_only = rail_df["DelayInMinutes"].dropna()
plt.figure(figsize=(8, 5))
plt.hist(delay_only, bins=40, color="firebrick", alpha=0.7)
plt.xlabel("Delay minutes")
plt.ylabel("Delayed journeys")
plt.title("Distribution of delays (DelayInMinutes)")
plt.tight_layout()
```

```
plt.savefig(FIG_DIR / "eda_delay_hist.pdf", format="pdf")
plt.close()
print("Saved eda_delay_hist.pdf")

status_counts = rail_df["Journey.Status"].value_counts()
plt.figure(figsize=(7, 5))
plt.bar(status_counts.index, status_counts.values, color="lightseagreen")
plt.xlabel("Journey status")
plt.ylabel("Journeys")
plt.title("Journey status distribution")
plt.tight_layout()
plt.savefig(FIG_DIR / "eda_status_counts.pdf", format="pdf")
plt.close()
print("Saved eda_status_counts.pdf")

numeric_cols = [
    "Price",
    "ScheduledMinutes",
    "ActualMinutes",
    "DelayInMinutes",
    "DepartureHour",
    "DepartureDayOfWeek",
]
pairplot_df = rail_df[numeric_cols].dropna()
plot_mat = pairplot_df.values
if plot_mat.shape[0] > 6000:
    rng = np.random.default_rng(RANDOM_STATE)
    idx = rng.choice(plot_mat.shape[0], size=6000, replace=False)
    plot_mat = plot_mat[idx]

labels = [
    "Price",
    "Scheduled minutes",
    "Actual minutes",
    "Delay minutes",
    "Departure hour",
    "Departure weekday",
]
n = len(labels)
plt.figure(figsize=(14, 14))
for i in range(n):
    for j in range(n):
```

```
        plt.subplot(n, n, 1 + i + (n * j))
        if i == j:
            sns.histplot(plot_mat[:, i], stat="density", color="lightseagreen", alpha=0.6)
            sns.kdeplot(plot_mat[:, i], color="black")
        else:
            plt.scatter(plot_mat[:, i], plot_mat[:, j], s=4, alpha=0.25, color="tab:blue")
        if j == n - 1:
            plt.xlabel(labels[i])
        else:
            plt.xlabel("")
        if i == 0:
            plt.ylabel(labels[j])
        else:
            plt.ylabel("")
plt.tight_layout()
plt.savefig(FIG_DIR / "eda_pairplot_custom.pdf", format="pdf")
plt.close()
print("Saved eda_pairplot_custom.pdf")

# ===== Section 5: Single Variable Logistic Regression =====
late_df = rail_df[rail_df["Journey.Status"] != "On Time"].copy()
late_df = late_df.dropna(subset=["Refund.Request"])
print(f"Non-punctual journeys for single-variable model: {len(late_df):,}")

late_df["RefundBinary"] = np.where(
    late_df["Refund.Request"].str.strip().str.lower() == "yes", 1, 0
)
late_df["MediumPrice"] = np.where(
    (late_df["Price"] > 10) & (late_df["Price"] <= 30), 1, 0
)

X_single = sm.add_constant(late_df["MediumPrice"])
logit_model_single = sm.Logit(late_df["RefundBinary"], X_single)
logit_result_single = logit_model_single.fit(disp=False)
print(logit_result_single.summary2())

coef_table_single = pd.DataFrame(
    {
        "Coefficient": logit_result_single.params,
        "Std.Err": logit_result_single.bse,
        "p-value": logit_result_single.pvalues,
        "Odds Ratio": np.exp(logit_result_single.params),
```

```
    }
).round(3)
print("\nSingle-variable logistic regression coefficients:\n", coef_table_single)

beta0 = logit_result_single.params["const"]
beta1 = logit_result_single.params["MediumPrice"]
prob_table = pd.DataFrame(
    {
        "Price (£)": [5, 25],
        "MediumPrice": [0, 1],
        "Predicted Refund Probability": [
            1 / (1 + np.exp(-beta0)),
            1 / (1 + np.exp(-(beta0 + beta1))),
        ],
    }
)
print("\nReference refund probabilities:\n", prob_table.round(3))

# ===== Section 6: Multivariable Models =====
FEATURE_COLS = [
    "Price",
    "DelayInMinutes_filled",
    "Journey.Status",
    "Ticket.Type",
    "Ticket.Class",
    "Payment.Method",
    "Railcard",
    "DepartureHour",
    "DepartureDayOfWeek",
]
CATEGORICAL_FEATURES = [
    "Journey.Status",
    "Ticket.Type",
    "Ticket.Class",
    "Payment.Method",
    "Railcard",
]

rail_df["RefundBinary"] = (rail_df["Refund.Request"] == "Yes").astype(int)
rail_df["DelayInMinutes_filled"] = rail_df["DelayInMinutes"].fillna(0)

X_raw = rail_df[FEATURE_COLS].copy()
```

```
X_raw[CATEGORICAL_FEATURES] = X_raw[CATEGORICAL_FEATURES].fillna("Unknown")
X_raw["DepartureHour"] = X_raw["DepartureHour"].fillna(
    X_raw["DepartureHour"].median()
)
X_raw["DepartureDayOfWeek"] = X_raw["DepartureDayOfWeek"].fillna(
    X_raw["DepartureDayOfWeek"].median()
)

X = pd.get_dummies(X_raw, columns=CATEGORICAL_FEATURES, drop_first=True)
y = rail_df["RefundBinary"]
print(f"Feature matrix shape: {X.shape}")
print(f"Refund request rate: {y.mean():.3f}")

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=RANDOM_STATE, stratify=y
)
print(
    f"Training rows: {X_train.shape[0]:,}, Testing rows: {X_test.shape[0]:,}"
)

models = [
    (
        "Logistic Regression",
        LogisticRegression(max_iter=1000, class_weight="balanced", random_state=RANDOM_STATE
    ),
    (
        "Decision Tree",
        DecisionTreeClassifier(max_depth=4, class_weight="balanced", random_state=RANDOM_STA
    ),
    (
        "Random Forest",
        RandomForestClassifier(
            n_estimators=100,
            class_weight="balanced",
            random_state=RANDOM_STATE,
            n_jobs=-1,
        ),
    ),
]

model_results = []
for name, model in models:
```

```
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]
    metrics = {
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred, zero_division=0),
        "Recall": recall_score(y_test, y_pred),
        "F1": f1_score(y_test, y_pred),
        "ROC-AUC": roc_auc_score(y_test, y_proba),
        "PR-AUC": average_precision_score(y_test, y_proba),
        "Probas": y_proba,
        "Estimator": model,
    }
    model_results.append(metrics)

results_df = pd.DataFrame(model_results)[
    ["Model", "Accuracy", "Precision", "Recall", "F1", "ROC-AUC", "PR-AUC"]
]
print("\nTest set metrics:\n", results_df.round(3).to_string(index=False))

plt.figure(figsize=(7, 6))
for res in model_results:
    fpr, tpr, _ = roc_curve(y_test, res["Probas"])
    plt.plot(fpr, tpr, label=f"{res['Model']} (AUC = {res['ROC-AUC']:.3f})")
plt.plot([0, 1], [0, 1], linestyle="--", color="gray", label="Chance")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves: Refund Prediction")
plt.legend(loc="lower right")
plt.tight_layout()
plt.savefig(FIG_DIR / "model_roc_comparison.pdf", format="pdf")
plt.close()
print("Saved model_roc_comparison.pdf")

baseline = y_test.mean()
plt.figure(figsize=(7, 6))
for res in model_results:
    precision, recall, _ = precision_recall_curve(y_test, res["Probas"])
    plt.plot(recall, precision, label=f"{res['Model']} (AP = {res['PR-AUC']:.3f})")
plt.hlines(baseline, xmin=0, xmax=1, linestyles="--", colors="gray", label=f"Baseline (AP =
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curves: Refund Prediction")
plt.legend(loc="best")
plt.tight_layout()
plt.savefig(FIG_DIR / "model_pr_comparison.pdf", format="pdf")
plt.close()
print("Saved model_pr_comparison.pdf")

best_by_auc = results_df.loc[results_df["ROC-AUC"].idxmax()]
logit_auc = results_df.loc[
    results_df["Model"] == "Logistic Regression", "ROC-AUC"
].iloc[0]
if (best_by_auc["Model"] != "Logistic Regression") and (
    best_by_auc["ROC-AUC"] - logit_auc > 0.01
):
    final_model_name = best_by_auc["Model"]
else:
    final_model_name = "Logistic Regression"
final_model = next(res["Estimator"] for res in model_results if res["Model"] == final_model_
final_model.fit(X_train, y_train)
final_feature_columns = X.columns.tolist()
print(f"Final model selected: {final_model_name}")

# ===== Section 7: Final Outputs =====
if not PREDICTION_PATH.exists():
    raise FileNotFoundError(
        f"Prediction dataset not found at {PREDICTION_PATH.resolve()}"
    )
df_pred = pd.read_csv(PREDICTION_PATH)

time_cols = ["Departure", "Scheduled.Arrival", "Actual.Arrival"]
for col in time_cols:
    if col in df_pred.columns:
        df_pred[col] = pd.to_datetime(
            df_pred[col], format="%Y-%m-%d %H:%M", errors="coerce"
        )

if "DelayInMinutes" not in df_pred.columns:
    delay_minutes_pred = (
        df_pred["Actual.Arrival"] - df_pred["Scheduled.Arrival"]
```

```
    ).dt.total_seconds() / 60
    df_pred["DelayInMinutes"] = delay_minutes_pred.where(delay_minutes_pred > 0)

if "DepartureHour" not in df_pred.columns:
    df_pred["DepartureHour"] = df_pred["Departure"].dt.hour
if "DepartureDayOfWeek" not in df_pred.columns:
    df_pred["DepartureDayOfWeek"] = df_pred["Departure"].dt.dayofweek

df_pred["DelayInMinutes_filled"] = df_pred["DelayInMinutes"].fillna(0)
df_pred["DepartureHour"] = df_pred["DepartureHour"].fillna(
    df_pred["DepartureHour"].median()
)
df_pred["DepartureDayOfWeek"] = df_pred["DepartureDayOfWeek"].fillna(
    df_pred["DepartureDayOfWeek"].median()
)

missing_cols = set(FEATURE_COLS) - set(df_pred.columns)
if missing_cols:
    raise ValueError(
        f"ToPredict dataset is missing required columns: {sorted(missing_cols)}"
    )

X_pred_raw = df_pred[FEATURE_COLS].copy()
X_pred_raw[CATEGORICAL_FEATURES] = X_pred_raw[CATEGORICAL_FEATURES].fillna("Unknown")
X_pred = pd.get_dummies(X_pred_raw, columns=CATEGORICAL_FEATURES, drop_first=True)
for col in final_feature_columns:
    if col not in X_pred.columns:
        X_pred[col] = 0
X_pred = X_pred[final_feature_columns]

pred_proba = final_model.predict_proba(X_pred)[:, 1]
df_pred["PredRefundProb"] = pred_proba

pred_display_cols = [
    "Price",
    "Journey.Status",
    "Ticket.Type",
    "Ticket.Class",
    "Payment.Method",
    "PredRefundProb",
]
print(
```

```
    "\nPredicted refund probabilities (decimal):\n",
    df_pred[pred_display_cols].round({"PredRefundProb": 4}).to_string(index=False),
)
percent_view = df_pred[pred_display_cols[:-1]].copy()
percent_view["Predicted Probability (%)"] = (
    df_pred["PredRefundProb"] * 100
).round(1)
print(
    "\nPredicted refund probabilities (%):\n",
    percent_view.to_string(index=False),
)
```