UK Train Rides Forecasting

Problem Statement

This project takes a practical and coding-focused approach. We aim to build machine learning models to forecast key performance indicators for the UK railway system using historical ride-level data. Specifically, we will focus on three forecasting tasks:

- Predicting the number of rides expected on future dates.
- Estimating the average delay duration for train services.
- Forecasting the number of delayed or cancelled trains.
- Forcasting the total revenue in the next 3 months

These tasks are essential for railway operations, planning, and improving passenger satisfaction.

We can now create a Pandas dataframe using the downloaded file, to view and analyze the data.

```
import pandas as pd

railway_df = pd.read_csv('railway.csv')
railway_df
```

```
                Transaction ID Date of Purchase Time of Purchase  \
0      da8a6ba8-b3dc-4677-b176       2023-12-08         12:41:11
1      b0cdd1b0-f214-4197-be53       2023-12-16         11:23:01
2      f3ba7a96-f713-40d9-9629       2023-12-19         19:51:27
3      b2471f11-4fe7-4c87-8ab4       2023-12-20         23:00:36
4      2be00b45-0762-485e-a7a3       2023-12-27         18:22:56
...                        ...              ...              ...
31648  1304623d-b8b7-4999-8e9c       2024-04-30         18:42:58
31649  7da22246-f480-417c-bc2f       2024-04-30         18:46:10
31650  add9debf-46c1-4c75-b52d       2024-04-30         18:56:41
31651  b92b047c-21fd-4859-966a       2024-04-30         19:51:47
31652  1d5d89a2-bde5-410f-8f91       2024-04-30         20:05:39

      Purchase Type Payment Method Railcard Ticket Class Ticket Type
Price  \
0            Online    Contactless    Adult     Standard     Advance
43
1           Station    Credit Card    Adult     Standard     Advance
23
2            Online    Credit Card      NaN     Standard     Advance
3
3           Station    Credit Card      NaN     Standard     Advance
13
```

```
4             Online    Contactless      NaN     Standard     Advance
76
...              ...            ...      ...          ...         ...
...
31648         Online    Credit Card      NaN     Standard    Off-Peak
4
31649         Online    Contactless      NaN     Standard    Off-Peak
10
31650        Station    Credit Card      NaN     Standard    Off-Peak
4
31651        Station    Credit Card      NaN     Standard    Off-Peak
10
31652        Station    Credit Card    Adult     Standard    Off-Peak
3

           Departure Station    Arrival Destination Date of Journey  \
0          London Paddington  Liverpool Lime Street      2024-01-01
1         London Kings Cross                   York      2024-01-01
2      Liverpool Lime Street  Manchester Piccadilly      2024-01-02
3          London Paddington                Reading      2024-01-01
4      Liverpool Lime Street          London Euston      2024-01-01
...                      ...                    ...             ...
31648  Manchester Piccadilly  Liverpool Lime Street      2024-04-30
31649          London Euston  Birmingham New Street      2024-04-30
31650  Manchester Piccadilly  Liverpool Lime Street      2024-04-30
31651          London Euston  Birmingham New Street      2024-04-30
31652  Liverpool Lime Street  Manchester Piccadilly      2024-04-30

      Departure Time Arrival Time Actual Arrival Time Journey
Status  \
0           11:00:00     13:30:00            13:30:00        On Time

1           09:45:00     11:35:00            11:40:00        Delayed

2           18:15:00     18:45:00            18:45:00        On Time

3           21:30:00     22:30:00            22:30:00        On Time

4           16:45:00     19:00:00            19:00:00        On Time

...              ...          ...                 ...            ...

31648       20:00:00     20:30:00            20:30:00        On Time

31649       20:15:00     21:35:00            21:35:00        On Time

31650       20:15:00     20:45:00            20:45:00        On Time

31651       21:15:00     22:35:00            22:35:00        On Time
```

```
31652       21:30:00    22:00:00            22:00:00       On Time


      Reason for Delay Refund Request
0                  NaN             No
1       Signal Failure             No
2                  NaN             No
3                  NaN             No
4                  NaN             No
...                ...            ...
31648              NaN             No
31649              NaN             No
31650              NaN             No
31651              NaN             No
31652              NaN             No

[31653 rows x 18 columns]
```

```
railway_df.columns
```

```
Index(['Transaction ID', 'Date of Purchase', 'Time of Purchase',
       'Purchase Type', 'Payment Method', 'Railcard', 'Ticket Class',
       'Ticket Type', 'Price', 'Departure Station', 'Arrival
Destination',
       'Date of Journey', 'Departure Time', 'Arrival Time',
       'Actual Arrival Time', 'Journey Status', 'Reason for Delay',
       'Refund Request'],
      dtype='object')
```

```
railway_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31653 entries, 0 to 31652
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Transaction ID       31653 non-null  object
 1   Date of Purchase     31653 non-null  object
 2   Time of Purchase     31653 non-null  object
 3   Purchase Type        31653 non-null  object
 4   Payment Method       31653 non-null  object
 5   Railcard             10735 non-null  object
 6   Ticket Class         31653 non-null  object
 7   Ticket Type          31653 non-null  object
 8   Price                31653 non-null  int64
 9   Departure Station    31653 non-null  object
 10  Arrival Destination  31653 non-null  object
 11  Date of Journey      31653 non-null  object
 12  Departure Time       31653 non-null  object
 13  Arrival Time         31653 non-null  object
```

```
 14  Actual Arrival Time   29773 non-null  object
 15  Journey Status        31653 non-null  object
 16  Reason for Delay      4172 non-null   object
 17  Refund Request        31653 non-null  object
dtypes: int64(1), object(17)
memory usage: 4.3+ MB
```

1) Predicting the number of rides expected on future dates.

Date and Time Preprocessing

We convert all relevant date and time columns to datetime format to ensure consistency and enable time-based analysis. Combined columns like Departure DateTime and Actual Arrival DateTime are created to support delay and scheduling calculations.

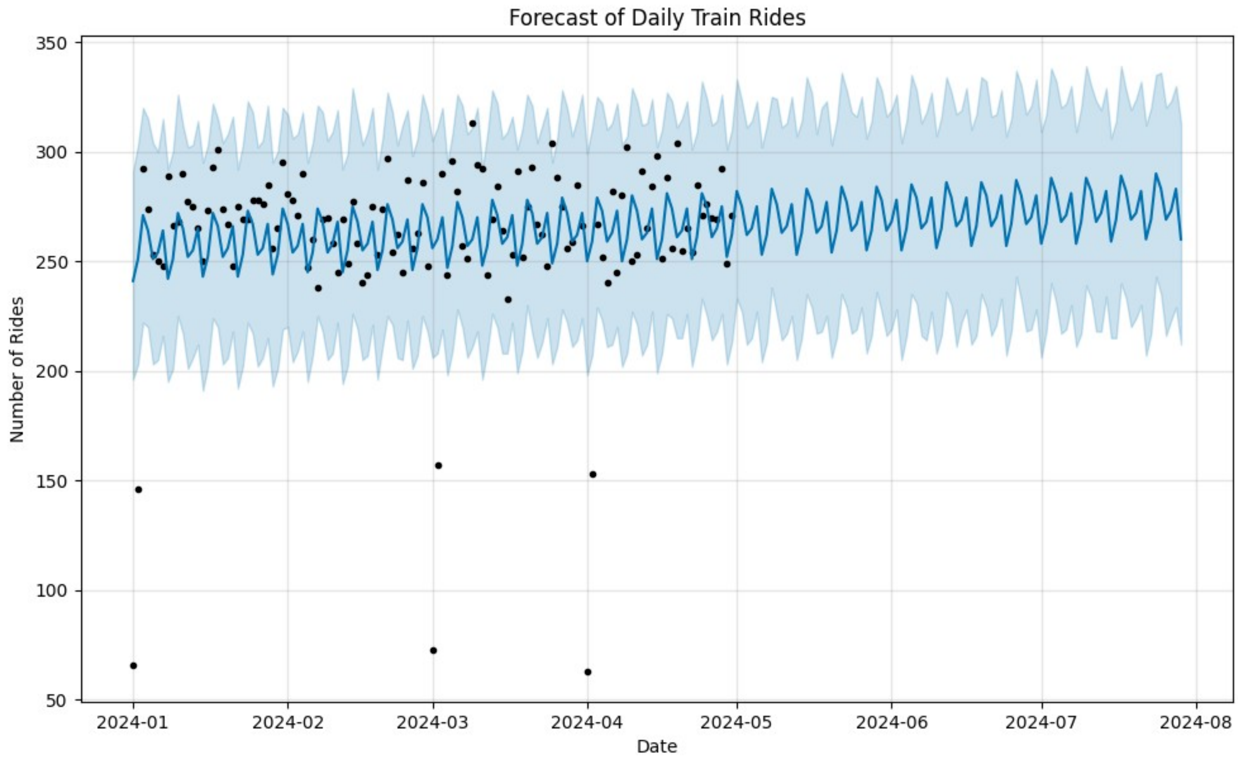
```
railway_df['Date of Purchase'] = pd.to_datetime(railway_df['Date of 
Purchase'], errors='coerce')
railway_df['Time of Purchase'] = pd.to_datetime(railway_df['Time of 
Purchase'], format='%H:%M:%S', errors='coerce')

railway_df['Date of Journey'] = pd.to_datetime(railway_df['Date of 
Journey'], errors='coerce')
railway_df['Departure DateTime'] = pd.to_datetime(railway_df['Date of 
Journey'].dt.strftime('%Y-%m-%d') + ' ' + railway_df['Departure 
Time'], errors='coerce')

railway_df['Arrival DateTime'] = pd.to_datetime(
    railway_df['Date of Journey'].dt.strftime('%Y-%m-%d') + ' ' + 
railway_df['Arrival Time'].astype(str),
    errors='coerce'
)

railway_df['Actual Arrival DateTime'] = pd.to_datetime(
    railway_df['Date of Journey'].dt.strftime('%Y-%m-%d') + ' ' + 
railway_df['Actual Arrival Time'].astype(str),
    errors='coerce'
)
```

We use Facebook Prophet model to forecast daily train ride counts based on historical data. The model is trained on ride frequencies grouped by journey date and predicts the next 90 days, including confidence intervals. The results are visualized and exported for further analysis.

```
from prophet import Prophet
import matplotlib.pyplot as plt

ride_counts = railway_df[railway_df['Date of Journey'].notnull()]
ride_counts_grouped = ride_counts.groupby('Date of 
Journey').size().reset_index(name='Ride Count')
```

```python
prophet_df = ride_counts_grouped.rename(columns={'Date of Journey': 'ds', 'Ride Count': 'y'})

model = Prophet()
model.fit(prophet_df)

future = model.make_future_dataframe(periods=90)
forecast = model.predict(future)

forecast['yhat'] = forecast['yhat'].round(0)
forecast['yhat_lower'] = forecast['yhat_lower'].round(0)
forecast['yhat_upper'] = forecast['yhat_upper'].round(0)

fig1 = model.plot(forecast)
plt.title("Forecast of Daily Train Rides")
plt.xlabel("Date")
plt.ylabel("Number of Rides")
plt.grid(True)
plt.show()

forecast_output = forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].rename(columns={
    'ds': 'Date of Journey',
    'yhat': 'Ride Count',
    'yhat_lower': 'Lower Limit',
    'yhat_upper': 'Upper Limit'
})
```

```
14:54:41 - cmdstanpy - INFO - Chain [1] start processing
14:54:44 - cmdstanpy - INFO - Chain [1] done processing
```



To evaluate model performance on peak travel days, we extract forecasts for Sundays, Mondays, and Fridays after April 30, 2024. This provides a clear tabular view of expected ride volumes during typical rush periods.

```
forecast_output['Day of the Week'] = forecast_output['Date of Journey'].dt.strftime('%A')

filtered_forecast_output = forecast_output.loc[
    (forecast_output['Date of Journey'] > '2024-04-30') &
    (forecast_output['Date of Journey'].dt.weekday.isin([6, 0, 4]))
]

filter_forecast_output = filtered_forecast_output[['Day of the Week', 'Date of Journey', 'Ride Count', 'Lower Limit', 'Upper Limit']]
filter_forecast_output.head(10)
```

	Day of the Week	Date of Journey	Ride Count	Lower Limit	Upper Limit
123	Friday	2024-05-03	262.0	212.0	311.0
125	Sunday	2024-05-05	275.0	227.0	323.0
126	Monday	2024-05-06	253.0	204.0	302.0
130	Friday	2024-05-10	263.0	214.0	311.0
132	Sunday	2024-05-12	276.0	225.0	

```
325.0
133          Monday     2024-05-13     253.0      205.0
307.0
137          Friday     2024-05-17     263.0      217.0
308.0
139          Sunday     2024-05-19     277.0      225.0
323.0
140          Monday     2024-05-20     254.0      206.0
303.0
144          Friday     2024-05-24     264.0      217.0
318.0
```

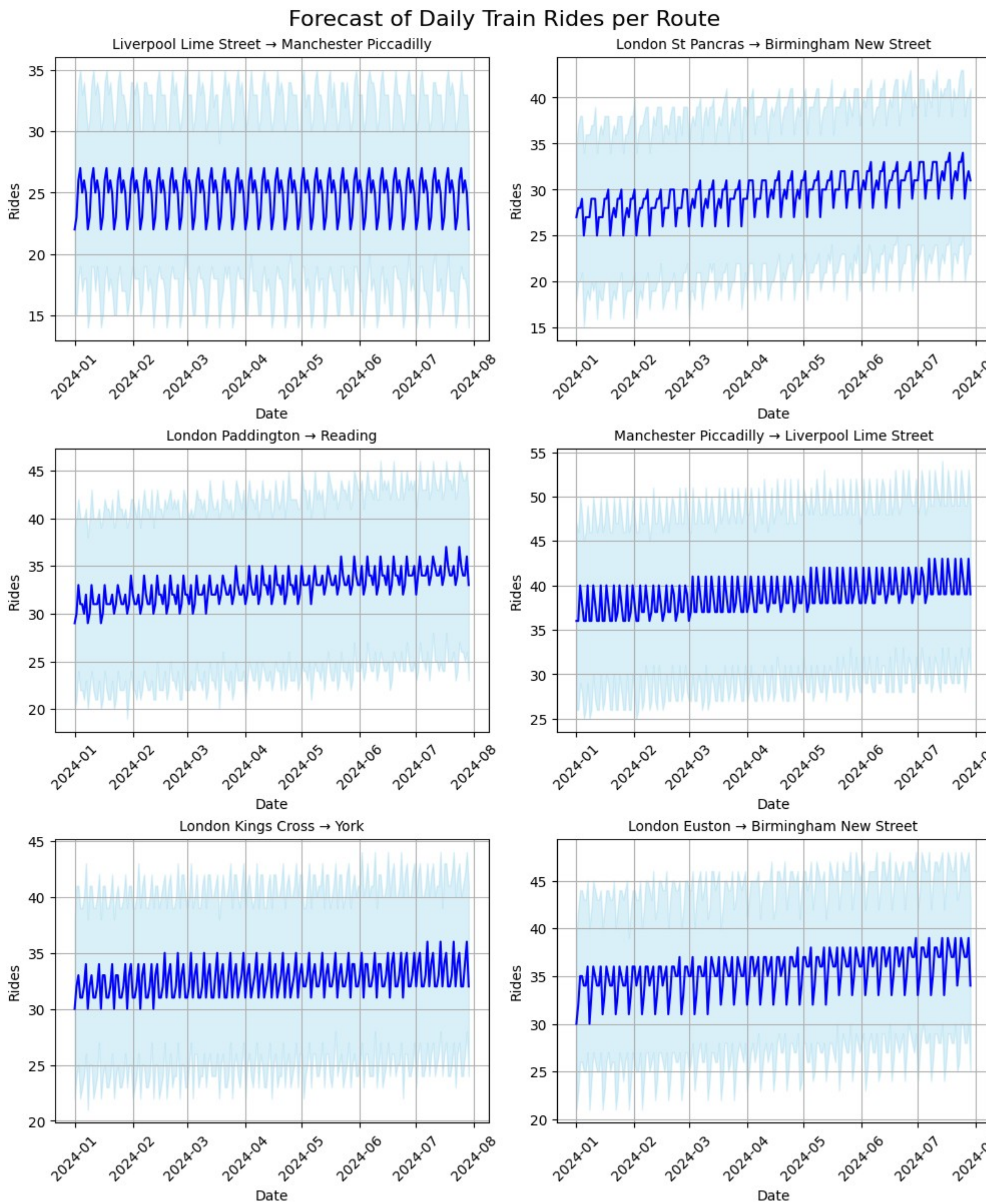
Forecasting Top Routes

We identify the six most frequent train routes and forecast daily ride counts for each using Prophet. The results are visualized in subplots, helping reveal route-specific trends and demand patterns over the next 90 days.

```
import math

railway_df['Route'] = railway_df['Departure Station'] + " → " +
railway_df['Arrival Destination']

valid_rides = railway_df[
    railway_df['Date of Journey'].notnull() &
    railway_df['Departure Station'].notnull() &
    railway_df['Arrival Destination'].notnull()
]

grouped_routes = valid_rides.groupby(['Route', 'Date of
Journey']).size().reset_index(name='Ride Count')

top_routes =
grouped_routes['Route'].value_counts().head(6).index.tolist()

num_routes = len(top_routes)
cols = 2
rows = math.ceil(num_routes / cols)
fig, axes = plt.subplots(rows, cols, figsize=(5 * cols, 4 * rows),
constrained_layout=True)

axes = axes.flatten()

for i, route in enumerate(top_routes):
    ax = axes[i]

    route_df = grouped_routes[grouped_routes['Route'] == route]
    prophet_df = route_df.rename(columns={'Date of Journey': 'ds',
'Ride Count': 'y'})
```

```
    model = Prophet()
    model.fit(prophet_df)

    future = model.make_future_dataframe(periods=90)
    forecast = model.predict(future)

    forecast['yhat'] = forecast['yhat'].round()
    forecast['yhat_lower'] = forecast['yhat_lower'].round()
    forecast['yhat_upper'] = forecast['yhat_upper'].round()

    ax.plot(forecast['ds'], forecast['yhat'], label='Forecast',
color='blue')
    ax.fill_between(forecast['ds'], forecast['yhat_lower'],
forecast['yhat_upper'], color='skyblue', alpha=0.3)
    ax.set_title(route, fontsize=10)
    ax.set_xlabel('Date')
    ax.set_ylabel('Rides')
    ax.tick_params(axis='x', rotation=45)
    ax.grid(True)

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.suptitle('Forecast of Daily Train Rides per Route', fontsize=16)
plt.show()
```

```
14:54:48 - cmdstanpy - INFO - Chain [1] start processing
14:54:49 - cmdstanpy - INFO - Chain [1] done processing
14:54:50 - cmdstanpy - INFO - Chain [1] start processing
14:54:50 - cmdstanpy - INFO - Chain [1] done processing
14:54:52 - cmdstanpy - INFO - Chain [1] start processing
14:54:52 - cmdstanpy - INFO - Chain [1] done processing
14:54:54 - cmdstanpy - INFO - Chain [1] start processing
14:54:54 - cmdstanpy - INFO - Chain [1] done processing
14:54:55 - cmdstanpy - INFO - Chain [1] start processing
14:54:55 - cmdstanpy - INFO - Chain [1] done processing
14:54:57 - cmdstanpy - INFO - Chain [1] start processing
14:54:57 - cmdstanpy - INFO - Chain [1] done processing
```



2) Estimating the average delay duration for train services.

Feature Engineering and Preprocessing

```
railway_df['Reason for Delay'].unique()
```

```
array([nan, 'Signal Failure', 'Technical Issue', 'Weather Conditions',
       'Weather', 'Staffing', 'Staff Shortage', 'Signal failure',
       'Traffic'], dtype=object)
```

```
railway_df.head(5)
```

```
             Transaction ID Date of Purchase    Time of Purchase
Purchase Type  \
0  da8a6ba8-b3dc-4677-b176       2023-12-08 1900-01-01 12:41:11
Online
1  b0cdd1b0-f214-4197-be53       2023-12-16 1900-01-01 11:23:01
Station
2  f3ba7a96-f713-40d9-9629       2023-12-19 1900-01-01 19:51:27
Online
3  b2471f11-4fe7-4c87-8ab4       2023-12-20 1900-01-01 23:00:36
Station
4  2be00b45-0762-485e-a7a3       2023-12-27 1900-01-01 18:22:56
Online

  Payment Method Railcard Ticket Class Ticket Type  Price  \
0    Contactless    Adult     Standard     Advance     43
1    Credit Card    Adult     Standard     Advance     23
2    Credit Card      NaN     Standard     Advance      3
3    Credit Card      NaN     Standard     Advance     13
4    Contactless      NaN     Standard     Advance     76

       Departure Station  ... Departure Time Arrival Time Actual
Arrival Time  \
0      London Paddington  ...       11:00:00     13:30:00
13:30:00
1     London Kings Cross  ...       09:45:00     11:35:00
11:40:00
2  Liverpool Lime Street  ...       18:15:00     18:45:00
18:45:00
3      London Paddington  ...       21:30:00     22:30:00
22:30:00
4  Liverpool Lime Street  ...       16:45:00     19:00:00
19:00:00

  Journey Status Reason for Delay Refund Request  Departure
DateTime  \
0        On Time              NaN             No 2024-01-01 11:00:00

1        Delayed   Signal Failure             No 2024-01-01 09:45:00

2        On Time              NaN             No 2024-01-02 18:15:00

3        On Time              NaN             No 2024-01-01 21:30:00

4        On Time              NaN             No 2024-01-01 16:45:00
```

```
     Arrival DateTime Actual Arrival DateTime  \
0 2024-01-01 13:30:00     2024-01-01 13:30:00
1 2024-01-01 11:35:00     2024-01-01 11:40:00
2 2024-01-02 18:45:00     2024-01-02 18:45:00
3 2024-01-01 22:30:00     2024-01-01 22:30:00
4 2024-01-01 19:00:00     2024-01-01 19:00:00

                                            Route
0      London Paddington → Liverpool Lime Street
1                        London Kings Cross → York
2  Liverpool Lime Street → Manchester Piccadilly
3                      London Paddington → Reading
4          Liverpool Lime Street → London Euston

[5 rows x 22 columns]
```

```
railway_df['Arrival Delay (min)'] = (
    railway_df['Actual Arrival DateTime'] - railway_df['Arrival DateTime']
).dt.total_seconds() / 60

railway_df['Departure Hour'] = railway_df['Departure DateTime'].dt.hour

railway_df['Journey Day'] = pd.to_datetime(railway_df['Date of Journey']).dt.dayofweek
railway_df['Is Weekend'] = railway_df['Journey Day'].isin([5, 6]).astype(int)

railway_df['Scheduled Duration'] = (railway_df['Arrival DateTime'] - railway_df['Departure DateTime']).dt.total_seconds() / 60


forecast_df = railway_df.dropna(subset=[
    'Arrival Delay (min)',
    'Departure Station', 'Arrival Destination',
])
```

Encoding Categorical Data

Since machine learning models can only be trained with numeric data, we need to convert categorical data to numbers. A common technique is to use one-hot encoding for categorical columns.

One hot encoding involves adding a new binary (0/1) column for each unique category of a categorical column.

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

categorical_features = ['Departure Station', 'Arrival Destination']
numerical_features = ['Departure Hour', 'Journey Day']

X = forecast_df[categorical_features + numerical_features]
y = forecast_df['Arrival Delay (min)']

preprocessor = ColumnTransformer(transformers=[
    ('cat', OneHotEncoder(handle_unknown='ignore'),
categorical_features)
], remainder='passthrough')
```

Model Training and Evaluation

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

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

model = Pipeline(steps=[
    ('preprocess', preprocessor),
    ('regressor', RandomForestRegressor())
])

model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error: {mae:.2f} minutes')
```

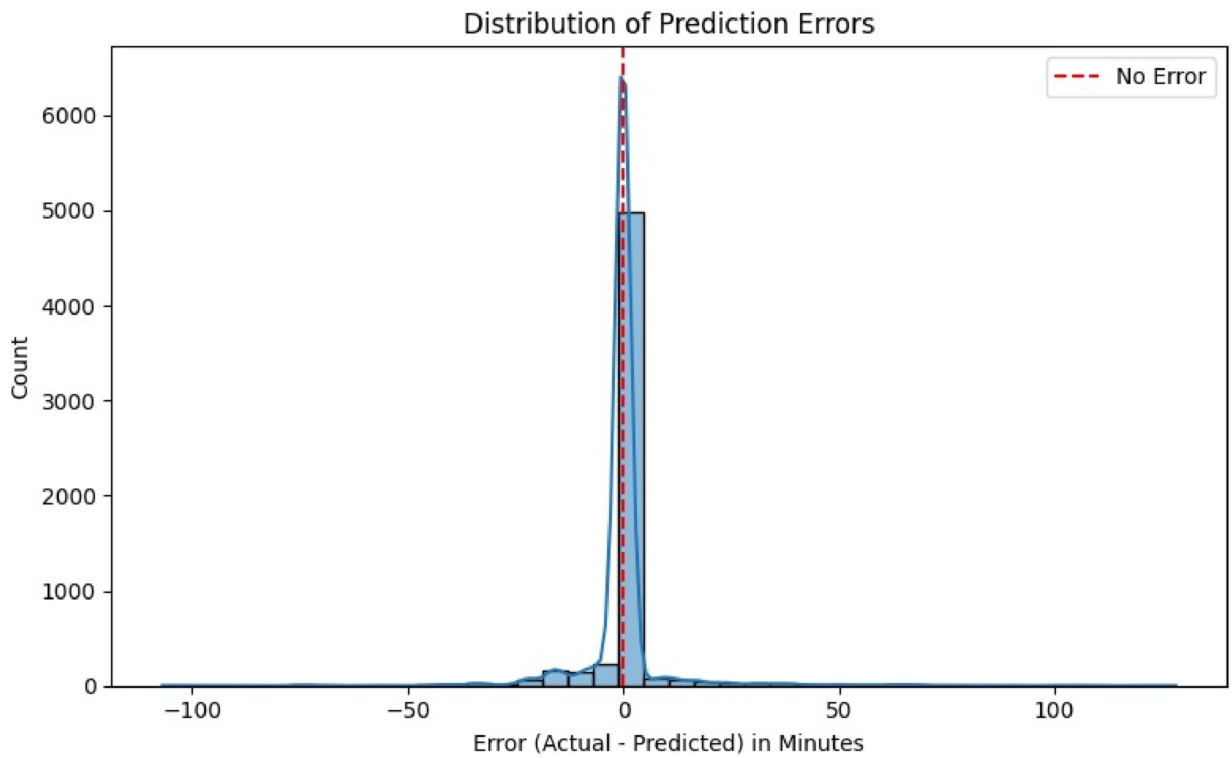
```
Mean Absolute Error: 2.86 minutes
```

After training the model and making predictions, the **Mean Absolute Error (MAE)** is calculated to evaluate the model's performance:

The MAE value of 2.86 minutes indicates that, on average, the model's predictions are off by 2.86 minutes from the actual values. This suggests that the model is relatively accurate

```
import seaborn as sns

errors = y_test - y_pred

plt.figure(figsize=(8, 5))
sns.histplot(errors, bins=40, kde=True)
plt.axvline(0, color='red', linestyle='--', label='No Error')
```

```
plt.title('Distribution of Prediction Errors')
plt.xlabel('Error (Actual - Predicted) in Minutes')
plt.legend()
plt.tight_layout()
plt.show()
```



we visualize the distribution of prediction errors to better understand the performance of the model. The histogram and kernel density estimate (KDE) plot show the spread of errors

3) Forecasting the number of delayed or cancelled trains.

```
railway_df['Is Cancelled'] = (railway_df['Journey Status'] ==
'Cancelled').astype(int)
railway_df['Is Delayed'] = (railway_df['Journey Status'] ==
'Delayed').astype(int)
```

In this section, two separate models are built to predict train cancellations and delays using a **Random Forest Classifier**. The process involves preparing the data, training the models, and splitting the data for both cancellation and delay predictions.

```
from sklearn.ensemble import RandomForestClassifier

categorical_columns = ['Route', 'Ticket Class', 'Ticket Type']

categorical_features = pd.get_dummies(railway_df[categorical_columns],
drop_first=True)
```

```
features = pd.concat([
    railway_df[['Price', 'Scheduled Duration', 'Journey Day', 'Is
Weekend']],
    categorical_features
], axis=1)

X_status = features.copy()

y_cancel = railway_df['Is Cancelled']
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(X_status,
y_cancel, test_size=0.2, random_state=42)

cancel_model = RandomForestClassifier()
cancel_model.fit(X_train_c, y_train_c)

y_delay = railway_df['Is Delayed']
X_train_d, X_test_d, y_train_d, y_test_d = train_test_split(X_status,
y_delay, test_size=0.2, random_state=42)

delay_model = RandomForestClassifier()
delay_model.fit(X_train_d, y_train_d)

RandomForestClassifier()

cancel_predictions = cancel_model.predict(X_test_c)

delay_predictions = delay_model.predict(X_test_d)

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

cancel_accuracy = accuracy_score(y_test_c, cancel_predictions)
cancel_conf_matrix = confusion_matrix(y_test_c, cancel_predictions)
cancel_class_report = classification_report(y_test_c,
cancel_predictions)

print("Cancellation Model - Accuracy:", cancel_accuracy)
print("Cancellation Model - Confusion Matrix:\n", cancel_conf_matrix)
print("Cancellation Model - Classification Report:\n",
cancel_class_report)

delay_accuracy = accuracy_score(y_test_d, delay_predictions)
delay_conf_matrix = confusion_matrix(y_test_d, delay_predictions)
delay_class_report = classification_report(y_test_d,
delay_predictions)

print("Delay Model - Accuracy:", delay_accuracy)
print("Delay Model - Confusion Matrix:\n", delay_conf_matrix)
print("Delay Model - Classification Report:\n", delay_class_report)
```

```
Cancellation Model - Accuracy: 0.9412415100300111
Cancellation Model - Confusion Matrix:
 [[5958   17]
 [ 355    1]]
Cancellation Model - Classification Report:
               precision    recall  f1-score   support

           0       0.94      1.00      0.97      5975
           1       0.06      0.00      0.01       356

    accuracy                           0.94      6331
   macro avg       0.50      0.50      0.49      6331
weighted avg       0.89      0.94      0.92      6331

Delay Model - Accuracy: 0.9527720739219713
Delay Model - Confusion Matrix:
 [[5787   85]
 [ 214  245]]
Delay Model - Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.99      0.97      5872
           1       0.74      0.53      0.62       459

    accuracy                           0.95      6331
   macro avg       0.85      0.76      0.80      6331
weighted avg       0.95      0.95      0.95      6331
```

Model Evaluation

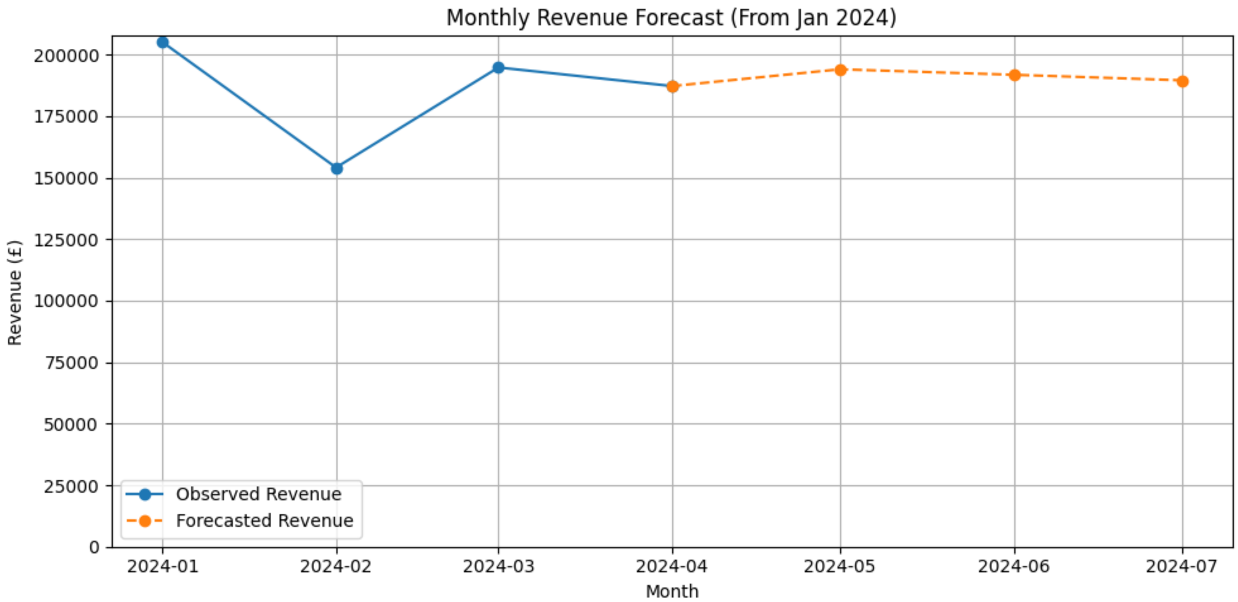
1. **Cancellation Model**
   - **Accuracy**: 94%
   - The model performs excellently for predicting non-cancelled trains (Class 0), with a high precision of 0.94 and a perfect recall of 1.00. This shows that it is very effective in identifying trains that are not cancelled.
2. **Delay Model**
   - **Accuracy**: 95.2%
   - The model has high accuracy and performs exceptionally well in identifying non-delayed trains (Class 0), with precision of 0.96 and recall of 0.99.
   - It demonstrates strong overall performance with a weighted F1-score of 0.95, which suggests it’s highly effective in real-world predictions.

4) Forcasting the total revenue in the next 3 months

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

monthly_revenue = railway_df.groupby(railway_df['Date of Purchase'].dt.to_period('M'))['Price'].sum().to_timestamp()
```

```python
monthly_revenue_filtered = monthly_revenue[monthly_revenue.index >=
'2024-01-01']

model = ExponentialSmoothing(monthly_revenue, trend='add',
seasonal=None, initialization_method='estimated')
fit = model.fit()

forecast = fit.forecast(3)

extended_forecast = pd.concat([monthly_revenue_filtered[-1:],
forecast])

combined_df = pd.concat([
    monthly_revenue_filtered.rename("Observed Revenue"),
    forecast.rename("Forecasted Revenue")
], axis=1)

plt.figure(figsize=(10, 5))
plt.plot(monthly_revenue_filtered, label='Observed Revenue',
marker='o')
plt.plot(extended_forecast, label='Forecasted Revenue', marker='o',
linestyle='--')
plt.title('Monthly Revenue Forecast (From Jan 2024)')
plt.xlabel('Month')
plt.ylabel('Revenue (£)')
plt.ylim(bottom=0)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()


print("\nRevenue Table (Observed and Forecasted):\n")
print(combined_df.round(2).to_string())
```

```
c:\Users\gharib\AppData\Local\Programs\Python\Python312\Lib\site-
packages\statsmodels\tsa\holtwinters\model.py:918: ConvergenceWarning:
Optimization failed to converge. Check mle_retvals.
  warnings.warn(
```




```
Revenue Table (Observed and Forecasted):

            Observed Revenue  Forecasted Revenue
2024-01-01          205091.0                 NaN
2024-02-01          154118.0                 NaN
2024-03-01          194789.0                 NaN
2024-04-01          187231.0                 NaN
2024-05-01               NaN           194048.30
2024-06-01               NaN           191814.05
2024-07-01               NaN           189579.80
```

End