

Assignment 3

ML Data Product

08/11/2024

Group 9

Student Last Name	Student First Name	Student ID	Group Allocation
Le	Thanh Nhan (Nicholas)	11919925	Student A
Schou	Alexander	25636299	Student B
Paiynthalir	Samyappan Nallamuthu	24715435	Student C

Github	Project Repo: https://github.com/Paiynthalir/Airfare_Predict.git Streamlit App Repo: https://github.com/nicn131/airfare-streamlit.git
--------	--

36120 - Advanced Machine Learning Application
Master of Data Science and Innovation
University of Technology of Sydney

Table of Contents

1. Executive Summary	3
2. Business Understanding	4
a. Business Use Cases	4
3. Data Understanding	5
4. Data Preparation	10
a. Student A: Nicholas (Thanh Nhan) Le	10
b. Student B: Alexander Schou	12
c. Student C: Paiynthalir Samyappan Nallamuthu	12
5. Modelling	15
a. Student A: Nicholas (Thanh Nhan) Le	15
Modelling approaches	15
Results and Analysis	16
b. Student B: Alexander Schou	17
Baseline Model	17
Final Model	18
c. Student C: Paiynthalir Samyappan Nallamuthu	21
Results and Analysis	25
6. Evaluation	27
a. Evaluation Metrics	27
b. Business Impact and Benefits	27
c. Data Privacy and Ethical Concerns	27
7. Deployment	29
a. Model Serving	29
b. Web App	30
8. Collaboration	32
a. Individual Contributions	32
Nicholas (Thanh Nhan) Le	32
Alexander Schou	32
Paiynthalir Samyappan Nallamuthu	32
b. Group Dynamic	33
c. Ways of Working Together	33
d. Issues Faced	33
9. Conclusion	34
10. References	35



Appendix 1: Data Dictionary	36
Appendix 2: Installation and Running Instructions for Streamlit web app	38



1. Executive Summary

This project develops a predictive tool for estimating airfare costs, designed for both travellers and businesses. The goal is to create an accessible, reliable web application that forecasts airfares based on inputs like origin, destination, dates, and cabin type, helping users with budgeting.

Each team member contributes models, such as XGBoost, linear regression, and neural networks. Model performance is measured using RHey MSE and MAE, ensuring accurate predictions. Key improvements come from using cyclical transformations for time features and effective encoding for categorical data.

The app, deployed through Streamlit, allows users to input travel details and receive fare predictions. Despite limitations due to the dataset's three-month span, the tool offers significant advantages for fare estimation. Future improvements include incorporating seasonal data for enhanced predictions.





2. Business Understanding

a. Business Use Cases

This project creates a predictive tool that helps travellers estimate local airfare costs by entering details such as origin and destination airports, departure date and time, and cabin type. It provides a convenient way for all types of travellers to budget and plan their travels.

Airfare prices are often complex and highly variable. Hence, the primary challenge comes from the unpredictable nature of airfare pricing, influenced by seasonal trends, demand, and other factors, which makes it difficult for travellers to plan with confidence. This project addresses this gap by offering a transparent and reliable fare estimation tool. Additionally, travel companies and airlines could use this model to optimise pricing strategies, create targeted offers, and improve customer engagement.

b. Key Objectives

The main objectives are to build an accurate and user-friendly fare prediction web application that meets the needs of two primary stakeholders: travellers and travel-related businesses. Travellers require a tool that quickly provides accurate fare estimates, allowing them to plan with ease. Travel companies could be interested in data-driven insights to improve their pricing models and develop promotions, thereby enhancing customer satisfaction and loyalty.

Machine learning algorithms are essential to this solution, as they can capture complex relationships between factors influencing airfare prices. Each team member will develop a unique model that will be implemented to ensure the app provides reliable and adaptable predictions.



3. Data Understanding

A data dictionary can be found under Appendix 1. The dataset contains 27 columns, including airport for initial location, destination airport, total fare, travel duration, cabin type, departure/arrival date and time, among others. The total number of rows is 13.519.999. The most useful columns are those with information about origins, destinations, departure date/time, cabin type (coach, premium, ...) since these are the minimum required inputs that the users of the web application should be able to provide.

The data spans between 17/04/2022 and 17/07/2022, which is relatively short. This means our models may not be able to learn from trends outside this range, which poses a major limitation.

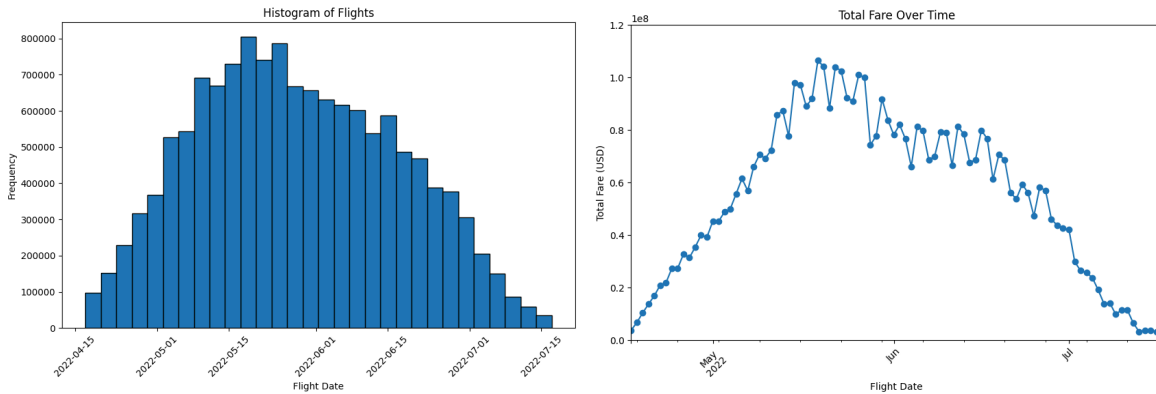


Figure 1: Histogram of flights and line plot of total fare over time

Figure 1 shows a histogram of all the flights. The data indicates that flight activity gradually increased from April, reaching its highest frequency in May, and then started to decline by mid-June. This pattern suggests a seasonal trend, possibly due to increased travel demand during late spring/early summer months, followed by a decrease as summer progresses. The line plot captures the total fare over time, capturing a clear trend that peaks and then declines. Starting from an initial rise in fares, the plot reveals a steady increase up to a peak in late May to early June. This period marks the highest total fares, possibly indicating a surge in travel demand. Following this peak, the total fare begins to decline gradually, with a noticeable decrease in July, eventually returning to lower levels.

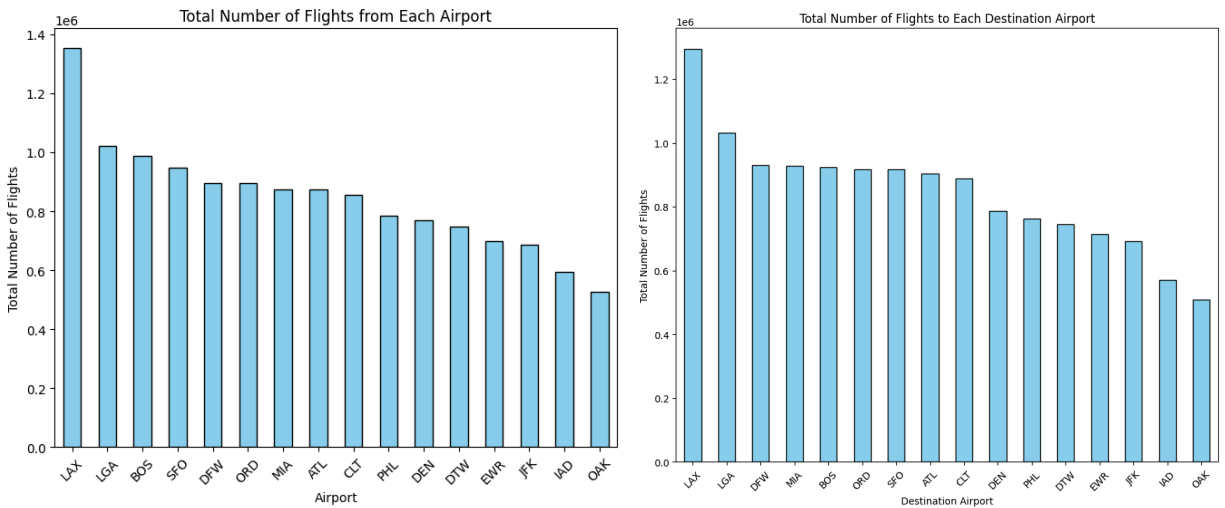


Figure 2: Total number of flights departing and arriving at each airport

Figure 2 shows two bar charts representing the total number of flights departing from and arriving at each airport. The left chart shows that Los Angeles (LAX) has the highest volume of departures, followed by other major hubs such as LaGuardia (LGA), Boston (BOS), and San Francisco (SFO). The departure volumes then gradually decrease across other airports, with Oakland (OAK) having the lowest number of departures in this set. The right chart shows that similar to the departure chart, LAX again leads with the highest volume of arrivals, indicating its role as a significant hub in the network. LGA and Dallas-Fort Worth (DFW) also have high arrival volumes, with other airports like Miami (MIA) and Boston (BOS) following closely. The number of arrivals decreases across the remaining airports, with OAK also showing the lowest arrival volume.

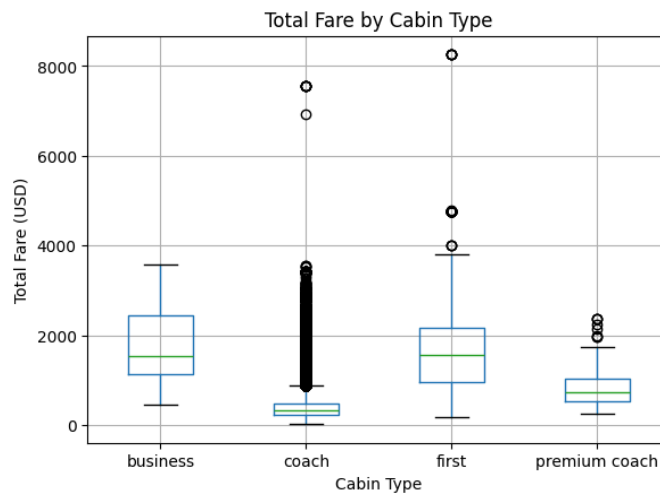


Figure 3: Boxplot of total fare by cabin type

Figure 3 shows the distribution of total fares across different cabin types. Business class fares display a relatively high median and a wider range of fare values compared to coach and premium coach, with a few high outliers indicating occasionally expensive tickets. Coach class has the lowest median fare and a narrow IQR, suggesting less variability in fares, though there are numerous outliers, which likely reflect higher prices due to peak demand or last-minute bookings. First class exhibits the highest median fare along with a broad IQR, indicating significant variability in fares and some high outliers. Premium coach fares are closer to coach in distribution, but with a slightly higher median and more variability.

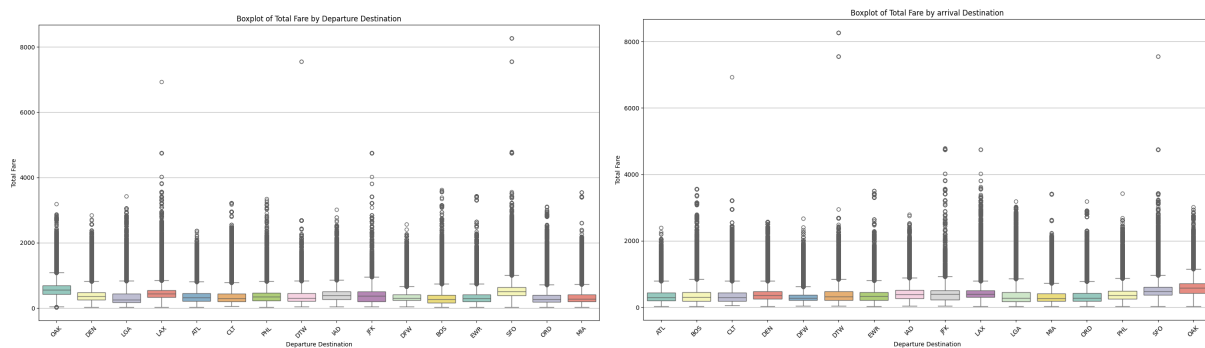


Figure 4: Boxplot of total fare by departure destination and arrival destination

Figure 4 presents two boxplots displaying the distribution of total fares by departure destination (left) and arrival destination (right). The distribution of fares is relatively consistent across most airports, with the median and IQRs generally at lower fare levels. However, certain airports show a higher number of outliers, suggesting more variability in pricing for those locations. In both departure and arrival plots, airports like JFK, LAX, and ORD display a wider spread of outliers, likely reflecting instances of premium flights or high-demand pricing.

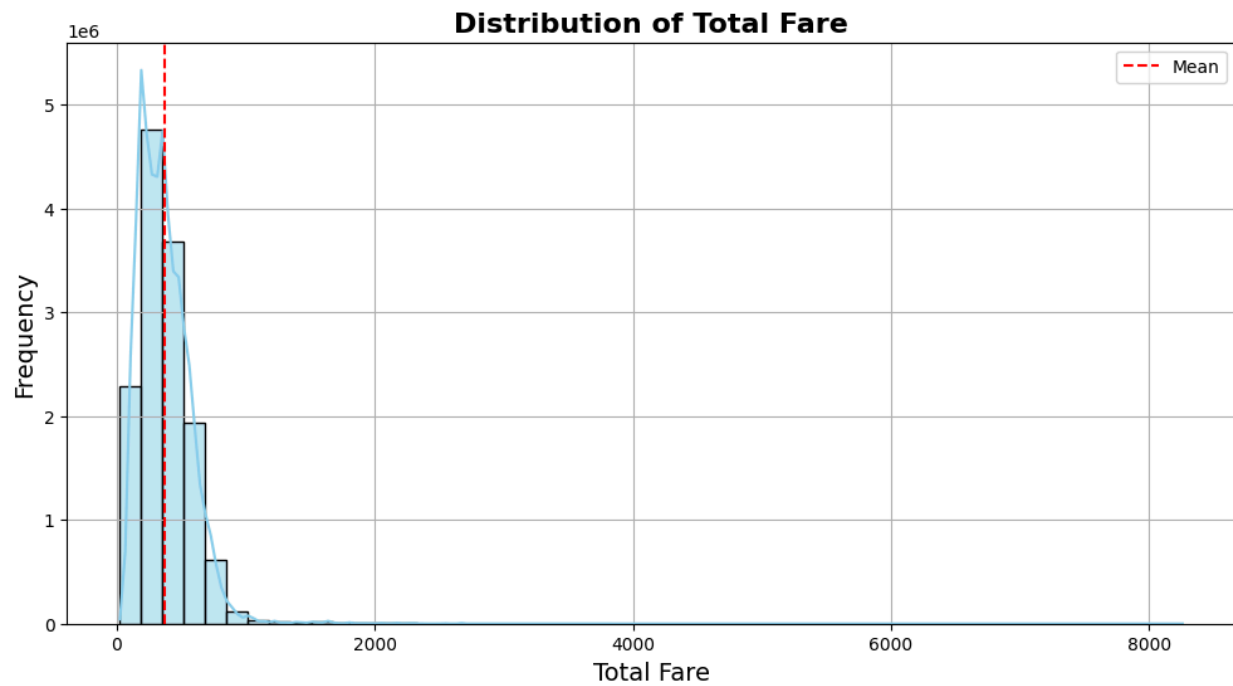


Figure 5: Distribution of total fare

Figure 5 shows the distribution of total fares. The distribution is heavily right-skewed, with most fares clustered at lower values, indicating that the majority of fares fall within a low-to-moderate range. A few observations have much higher fares, as seen by the tail extending to the right, suggesting occasional premium or long-distance fares that raise the total significantly. The red dashed line marks the mean fare, which is positioned to the right of the peak. This shift reflects the impact of the high-value outliers, which pull the mean above the most common fare values.

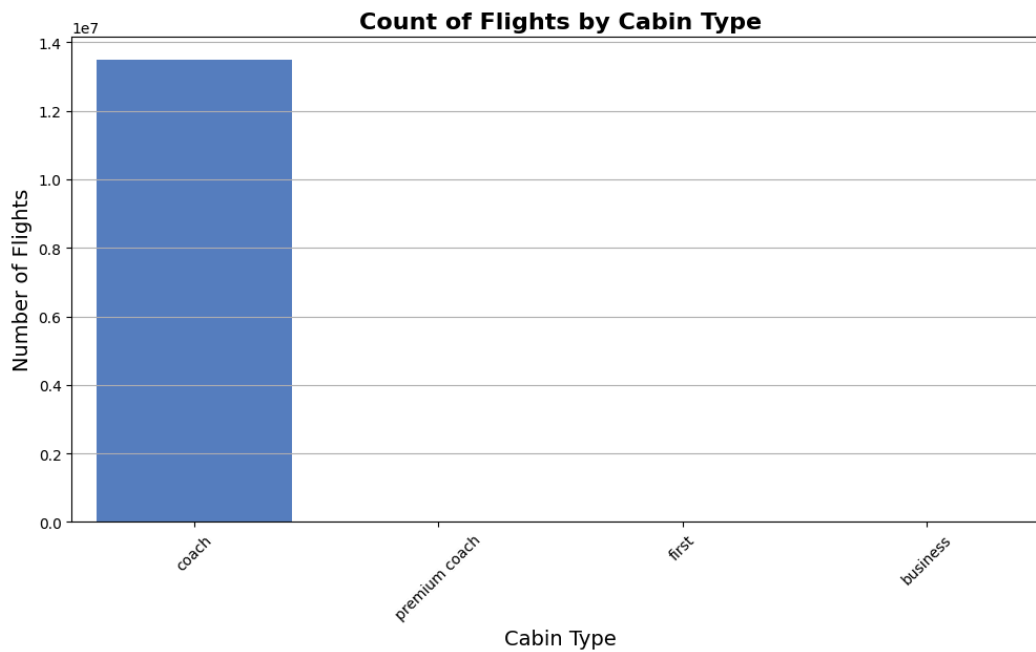


Figure 6: Count of flights by cabin type

Figure 6 displays the count of flights by cabin type. The vast majority of flights are in coach (over 13M). Other cabin types have significantly lower counts, barely visible in comparison. This suggests a common customer mentality when booking domestic travels: they want only the cheapest tickets.

4. Data Preparation

Our group takes an Agile approach to the entire development pipeline, including data preparation. This means each member is responsible for preparing their own versions of the dataset for modelling purposes.

The following steps are shared between team members:

- Extract the ZIP file containing the different parts of the dataset into the **data/interim/** folder. The result is a flattened view of all CSV files in a single folder.
- Join the CSV files together to form the complete dataset.
- (Optional) Save the full dataset in **parquet** format for storage efficiency.

We also share feature engineering for the following columns:

- *flightDate*: cyclical transformation is performed to extract the **sine/cosine information** from the different components, such as *month*, *hour* and *minute*. The idea is to bridge the gap between the last instance of the previous month/hour/minute and the first instance of the next, so the models better learn this characteristic. The transformation formulas are below:

$$x_{sin} = \sin \frac{2\pi x}{\max(x)}$$

$$x_{cos} = \cos \frac{2\pi x}{\max(x)}$$

a. Student A: Nicholas (Thanh Nhan) Le

The columns chosen for modelling are summarised in Table 1.

Column name	Rationale/Notes
<i>searchDate</i>	May be used with flight date to calculate the time delta between flight date and search date. Small time deltas likely lead to larger fares
<i>flightDate</i>	User-required
<i>startingAirport</i>	User-required
<i>destinationAirport</i>	User-required

<i>isBasicEconomy</i>	Directly linked to fares
<i>isRefundable</i>	Directly linked to fares
<i>isNonStop</i>	Directly linked to fares
<i>totalTravelDistance</i>	Directly linked to fares
<i>travelDuration</i>	Directly linked to fares, and may include extra layover. Long layover flights tend to be cheaper
<i>segmentsDepartureTimeEpochSeconds</i>	The entry for the first leg can be used to get detailed timestamps for departure times, which is useful for feature engineering
<i>segmentsCabinCode</i>	Directly linked to fares
<i>totalFare</i>	Target variable

Table 1: Columns chosen by Nicholas, and rationale for choice.

From here, the observations are:

- The *totalTravelDistance* column has missing values, so I use a reference dataset from the US Bureau of Transportation Statistics [1] to get the total distance between any two airports to fill this information.
- The full dataset has 55 duplicated rows, so these are dropped before being saved as the final dataset in the **data/interim/** folder.

I also perform more feature engineering:

- Cyclical transformation for *day of week*, where this is originally represented as numbers from 0 to 6 (i.e. from Monday to Sunday). The resulting columns are *flightDayOfWeekSin* and *flightDayOfWeekCos*.
- *deltaDays*: the difference in days between the search date and the flight date. The assumption is that flights closer to the search date are more expensive.
- *travelDurationDays*: extract the *hour* and *minute* information from the *travelDuration* column, and transform it into days to reduce the range for this data.
- *isBasicEconomy*, *isRefundable*, *isNonstop*: perform **effect encoding** for these features. In a binary encoder, values are encoded as **[0, 1]**, whereas with the effect encoder, values are encoded as **[-1, 1]**. This strategy stands to benefit convergence especially for neural networks when there are many binary variables. [2]
- *numLegs*: the number of legs for each trip, extracted from any column with the *segments* prefix.

- *segmentsCabinCode*: I binarise the cabin information for each leg of the trip, resulting in dummy variables. These are then **effect-encoded**.

I also export the following as reference data for use in the app:

- *Airport names*: the airports' full names and IATA codes, extracted from [3]. This is so that the airport choices are more user-friendly in the UI. For example, a choice of airport can be **Birmingham International Airport (BHM)** instead of just **"BHM"**.
- *Distance data*: the distance data for each pair of airports appearing in the dataset. This is for calculating the *totalTravelDistance* feature.
- *Travel duration data*: the total duration (in days) for each pair of airports appearing in the dataset. This is for calculating the *travelDurationDays* feature.

b. Student B: Alexander Schou

The following steps are taken to prepare the data:

- "startingAirport", "destinationAirport", "cabin_type" converted to categorical types
- Date/time fields converted to appropriate datetime formats. Any errors/missing values were handled to maintain data consistency.

The processed dataset was split into training, validation and test using a **chronological split**. 70% of the data is for training, 15% for validation, and 15% for testing. This split avoids data leakage, as future observations in the timeline were not included in the training data, allowing for a realistic assessment of model performance.

c. Student C: Paiynthalir Samyappan Nallamuthu

My data preparation steps are as follows:

- Chose the following columns of interest and dropped the rest, since the goal is to predict the airfare for given start and end airport destination, date of travel and cabin type:
 - *startingAirport*
 - *destinationAirport*
 - *totalFare*
 - *segmentsDepartureTimeRaw*
 - *segmentsCabinCode*
- Observed the values in the columns are single or multiple based on the number of legs in the travel itinerary. And these values are separated with '||'. So, this was handled during feature extraction from these columns.
- Feature engineering on departure time:

- Extract first leg of the 'segmentsDepartureTimeRaw' and saved as *departure_time*
- Convert the UTC time value in datetime format
- Extract the following features:
 - *departure_day*
 - *departure_dayofweek*
 - *departure_month*
 - *departure_hour*
 - *departure_minute*
- Feature engineered on cabin type: extract first leg of the *segmentsCabinCode* and saved as *cabin_type*
- Performed the above feature engineering and dropped the first set of duplicate rows, even before combining the records from each itineraries CSV file to avoid memory issues handling larger dataset later.
- Iterated over all the itineraries CSV files and saved the extracted records into a separate parquet file. And deleted the CSV files.
- Extracted the data from separate parquet files and merged them into single data frame
- Null values are seen in 'cabin_type', because I have extracted the value only if all the legs of the travel have the same cabin type. 0.4% of the records do not have the same cabin types in all legs of the travel. As part of the experiments, I later dropped this 0.4% of records to see if the performance of the prediction model improves.

Shape of the dataset:

Rows: 13154778, Columns: 12

Columns of the dataset:

```
Index(['startingAirport', 'destinationAirport', 'totalFare',  
      'segmentsDepartureTimeRaw', 'segmentsCabinCode', 'departure_day',  
      'departure_dayofweek', 'departure_month', 'departure_hour',  
      'departure_minute', 'departureTime', 'cabin_type'],  
      dtype='object')
```

Figure 7: Dataset shape and columns.

```
-----  
No of duplicated rows in the dataset:  
8040247  
-----
```

```
Percentage of missing values in each column:  
cabin_type    0.283836  
dtype: float64
```

```
-----  
No of duplicated rows in the dataset:  
0  
-----
```

```
Percentage of missing values in each column:  
cabin_type    0.37587  
dtype: float64
```

■ ■ ■

5. Modelling

a. Student A: Nicholas (Thanh Nhan) Le

Modelling approaches

The models considered are summarised in **Table 2**. For each model, all engineered features from those in **Table 1** are utilised. All steps with random components have a fixed seed of **42** for reproducibility.

For all models except neural network:

- 5-fold cross validation is performed together with hyperparameter tuning. For the neural network, since validation is already performed per epoch for early stopping, cross-validation is not necessary.
- The objective function to minimise is the validation root mean squared error (RMSE). For the neural network, the loss function is based on the MSE, which is similar.
- Standard scaling is performed for *timeDeltaDays*, *travelDurationDay*, and *totalTravelDistance*. This is because they are of a vastly different scale from the rest. For the neural network, there is not an easy way to incorporate this into the pipeline in a way that prevents data leaking (for this reason I refrain from scaling the whole dataset before training), so I leave it as-is.

For all models, a sample of **50,000 data points** are chosen for model building. For each drawn sample, a **two-sample Kolmogorov-Smirnov (KS) test** is performed to ensure the sample datasets used for building models are as distributionally close to the population as possible. The hypotheses are:

H0: the sample and population sets come from the same probability distribution,

H1: the sample and population sets come from different probability distributions,

where the significance level is 0.05, and only samples with a **p-value over 0.5** are chosen.

Approach	Rationale for choice	Model selection process
<i>Base (mean predicting)</i>	Benchmark	—
<i>Support Vector</i>	Most engineered features carry a geometric	Hyperparameter space: <ul style="list-style-type: none">• C: 0.001, 0.01, 0.1, 1, 10, 100, 1000

<i>Regression</i>	interpretation (e.g. cyclical features), so a geometry-based algorithm would best suit the purpose.	<ul style="list-style-type: none"> <i>epsilon</i>: 0.001, 0.01, 0.1, 1, 10, 100, 1000 Rationale: C controls regularisation, and <i>epsilon</i> controls the tube around the separating hyperplane within which no training penalties apply.
<i>Random Forest Regression</i>	A bagging ensemble algorithm for variety, since other group members have chosen boosting algorithms.	Hyperparameter space: <ul style="list-style-type: none"> <i>n_estimators</i>: 100, 120, ..., 200 <i>max_depth</i>: 3, 5, 7, 9, 11 <i>min_samples_leaf</i>: 1, 10, 30, 50, 70, 90 <i>max_features</i>: sqrt, log2 Rationale: chosen among the most influential features for controlling overfitting.
<i>Artificial Neural Network</i>	Powerful class of algorithms with different non-linearities which can estimate almost any function to an arbitrary degree of accuracy.	Hyperparameter space: <ul style="list-style-type: none"> <i>Number of hidden layers</i>: 1, 2 <i>Neurons per hidden layer</i>: 4, 6, 8, ..., 18 <i>Activation for hidden layers</i>: relu <i>Learning rate</i>: 0.01, 0.001, 0.0001 <i>Max epoch</i>: 1000 Rationale:

Table 2: Nicholas' modelling approach.

Results and Analysis

Table 3 shows the train, validation and test RMSEs of all models on the sample dataset, with only the winning model in the validation RMSE getting to be retrained and tested on the full dataset.

From these results, the neural network shows great potential in the validation RMSE, without too much overfitting on the **full test set**, considering that the test set is 20% of the full dataset (i.e. 2.6 million rows), and the model is only trained/validated on a 50,000-row sample.

Random Forest, being a good ensemble, exhibits a greater amount of overfitting than other models. This shows how difficult it is to tune a tree model.

The Support Vector Regression shows almost no overfitting, albeit being the worst-performing of all three. This is still a promising algorithm, provided a simpler dataset.

Model	RMSE (train)	RMSE (validation)	RMSE (test, best model only)	Parameter selection
Base	208.4185*	208.4227*	—	—

Support Vector Regression	167.2884*	167.7360*	–	C: 1000 <i>epsilon</i> : 100
Random Forest Regression	145.3889*	162.8335*	–	<i>max_depth</i> : 11 <i>max_features</i> : log2 <i>min_samples_leaf</i> : 1 <i>n_estimators</i> : 180
Artificial neural network	144.5450*	144.4085*	146.9672**	<i>Number of hidden layers</i> : 1 <i>Neurons per hidden layer</i> : 14 <i>Activation in hidden layer</i> : relu

Table 3: Nicholas' train, validation and test results.
*: trained on sample dataset; **: trained on full dataset

During modelling, it was shown that features relating to **airlines** contributed to approximately **9% reduction** in all RMSEs; however, incorporating this information into the app causes unnecessary complications for team members who already had their models trained and ready to be deployed, so this feature was eventually left out. This stands as an area for improvement in the next iteration.

b. Student B: Alexander Schou

Baseline Model

The model used as a baseline model was linear regression. This algorithm was chosen due to its simplicity and interpretability. Additionally, linear regression performs well with features that have a linear relationship with the target variable, making it suitable for initial model evaluations before moving to more complex algorithms if necessary. We know from our data exploration that we mostly have categorical variables so other models will be tested after but the linear regression model can still provide some insights.

For linear regression, no significant hyperparameter tuning was necessary since it is a relatively straightforward algorithm with minimal parameters. The implementation involved the `LinearRegression` class from the Scikit-Learn library. Key hyperparameters for this model include the fit intercept and normalisation options, though in this case, the basic settings were sufficient as this model primarily served as a baseline.

Feature engineering involved cyclical encoding of time-related features (hour and month) to capture the temporal periodicity in the data. This was done using sine and cosine functions. This

encoding allowed the model to use these cyclical patterns effectively. Additionally, categorical variables like startingAirport, destinationAirport, and cabin_type were encoded using one-hot encoding to represent each category as binary features. These transformations were applied in a Pipeline object, which allowed for a streamlined and consistent preprocessing process during both training and inference.

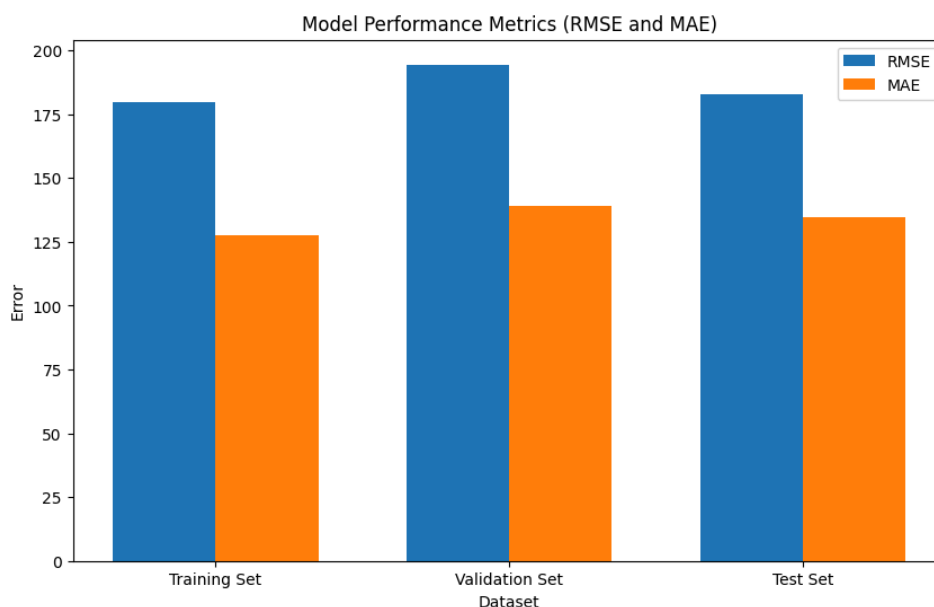


Figure 7: Performance Metrics for Linear Regression

Model performance can be seen in figure 7. The RMSE values are consistently higher than the MAE values across all datasets, which is expected, as RMSE penalises larger errors more heavily than MAE. The validation and test sets show similar RMSE and MAE values, which suggests that the model generalises well to new data and that the performance observed during validation is representative of the test performance.

Final Model

The final model that was implemented was XGBoost. This model was selected based on its ability to handle structured, tabular data with complex feature interactions. Given the objective of achieving high predictive accuracy, XGBoost was chosen for its robustness in capturing non-linear relationships between features and the target variable. Furthermore, it is effective in regression tasks with structured data and it is able to handle missing data and categorical variables effectively.

Parameter tuning was performed to optimise the XGBoost model's predictive power. Using the Hyperopt library, a search space was defined for key parameters, including learning_rate,

max_depth, n_estimators, subsample, and colsample_bytree. The Bayesian optimization method, as implemented in Hyperopt, was used to explore the parameter space efficiently, optimising for the lowest RMSE on the validation set. After selecting the best parameters, the final XGBoost model was trained on the combined dataset, incorporating training, validation, and test data.

The XGBoost model was trained on the processed training data and evaluated on both validation and test sets. The evaluation metrics used were RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error), providing insights into the model's prediction accuracy across different datasets.

Different models were tested. In the first approach, XGBoost was used with cyclical encoding of temporal features to handle the cyclic nature of hour and month. The second model used XGBoost without cyclical encoding, applying only one-hot encoding to categorical features. In the third approach, additional one-hot encoding was applied to the temporal features (month, day, hour, minute). The fourth model replaced XGBoost with CatBoost, using cyclical encoding for temporal features, similar to the first model. The final approach involved hyperparameter tuning on the XGBoost model using Hyperopt.

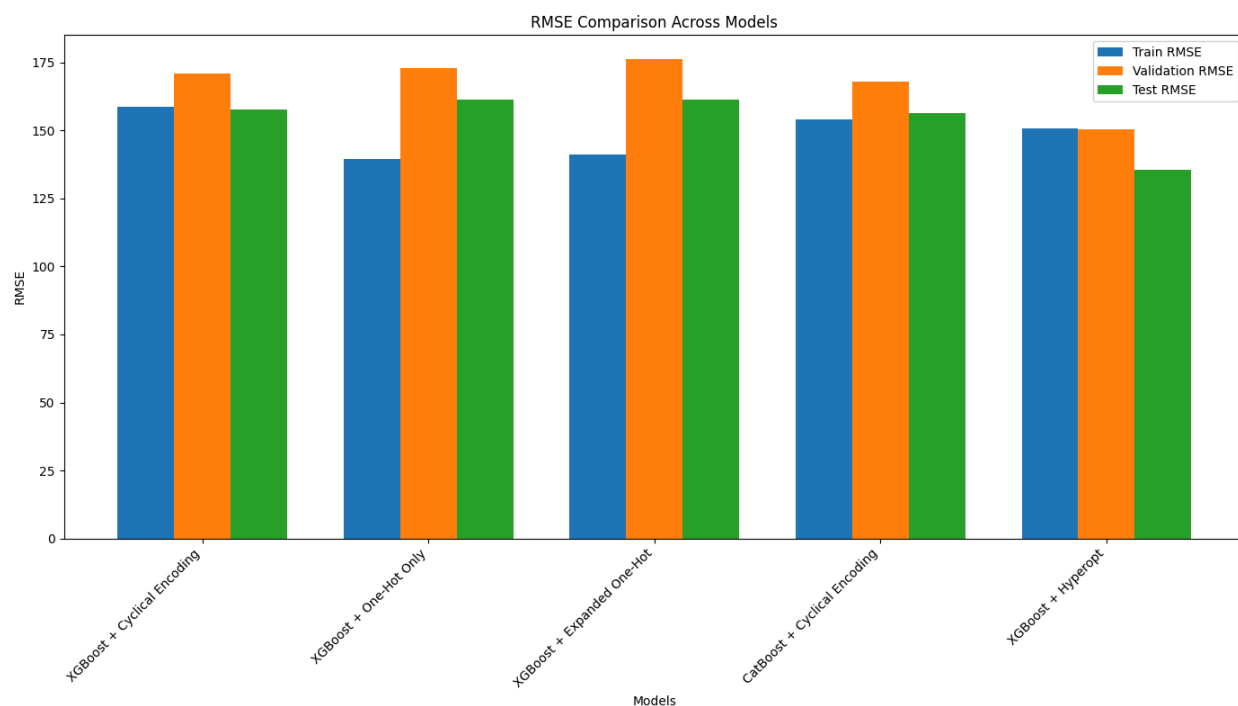


Figure 8: RMSE for the different models

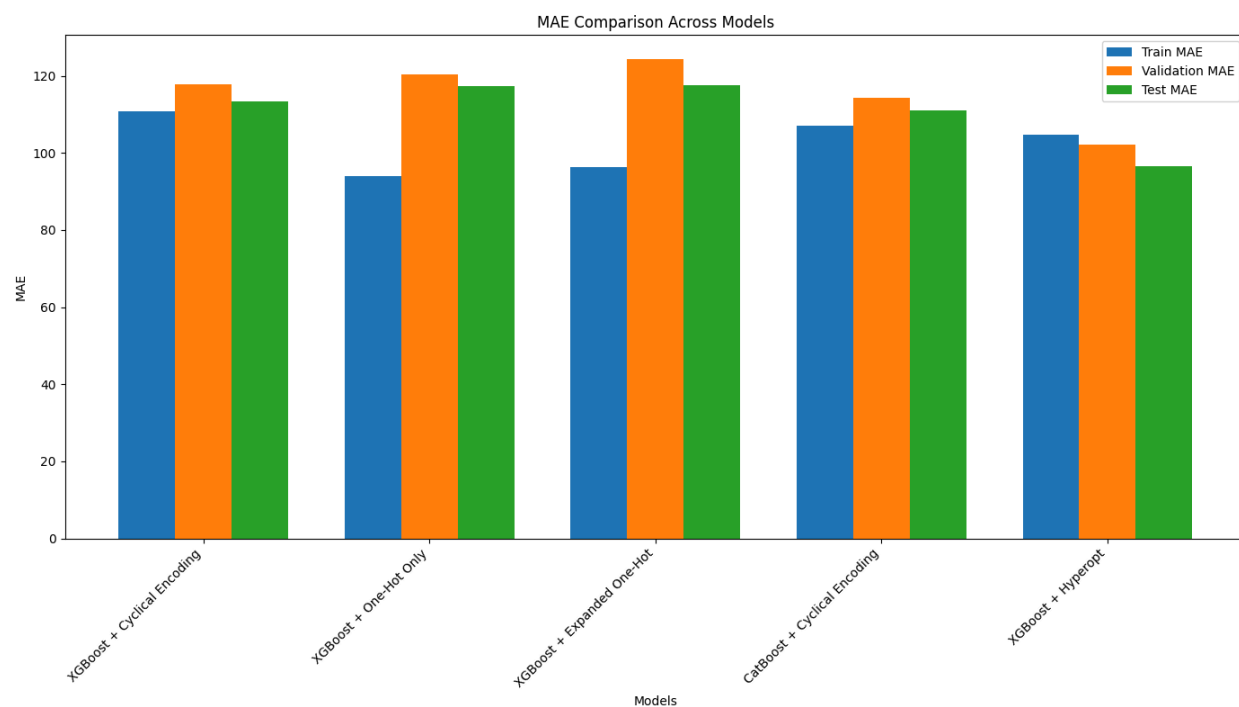


Figure 9: MAE for the different models

Figure 8 and 9 provide a comparative analysis of the performance of different models across the training, validation, and test sets, using RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) as evaluation metrics. Each figure highlights the models' abilities to generalise to unseen data, which is crucial for selecting the best model for totalFare predictions.

These plots reveal that the final XGBoost model, enhanced through hyperparameter tuning with Hyperopt, consistently outperformed other models on both RMSE and MAE across validation and test sets. This improvement indicates that the tuning process helped optimise the model's complexity, resulting in better generalisation. Models incorporating cyclical encoding also performed well, particularly for time-based features like hour and month, which benefit from capturing periodic patterns. The CatBoost model with cyclical encoding performed notably well on the test set, suggesting it could be a valuable alternative in future experimentation.

The model that was used in the end was the approach involving hyperparameter tuning on the XGBoost model using Hyperopt. The best parameters were:

- learning_rate: 0.156
- max_depth: 11
- n_estimators: 183
- subsample: 0.924

- colsample_bytree: 0.645

c. Student C: Paiynthalir Samyappan Nallamuthu

Data preparation:

- Split the training dataset into train and validation datasets based on time, to ensure there is no data leakage of future data and the temporal order is preserved.
- By maintaining the temporal order, any underlying trends or seasonal patterns can be properly captured and analyzed, enhancing the model's ability to make accurate forecasts.
- The following numerical features are transformed to preserve their cyclical nature by using the function that takes a Data Frame containing cyclical features related to time and returns a new Dataframe with their corresponding sine and cosine transformations. And included in the 'preprocessor' pipeline
 - 'departure_month'
 - 'departure_hour'
 - 'departure_minute'
- Later in one of the experiments, used Standard Scaler transformation on the same numerical features.
- The following categorical features are transformed using one hot encoding and included in the 'preprocessor' pipeline.
 - startingAirport
 - destinationAirport
 - departure_dayofweek
 - cabin_type
- And later used the pipeline to perform data processing on validation and test datasets. Finally saved the pipeline as a model for API / Streamlit APP consumption.

Experiment A: The first leg's cabin type is taken as the cabin type for the entire journey and multiple hops with different cabin types are included in the training. And the numerical feature's cyclical nature captured using cos and sin transformation

ML Algorithm	Rationale	Description
Sgd Regression	<p>Efficiency: Suitable for large datasets, making it computationally efficient.</p> <p>Flexibility: Customizable with various loss functions and regularization techniques.</p> <p>Online Learning: Allows incremental updates as new data becomes available.</p>	<p>Experiment A1: Penalty tuned manually.</p> <p>Finally chose penalty='elasticnet', random_state=42 as hyper parameters.</p> <p>Experiment A2: hyperparameter tuned using hyperopt and got the best parameters as below,</p> <p>'alpha': 0.003132789538185341,</p> <p>'eta0': 0.04074446362226317,</p> <p>'max_iter': 2000,</p> <p>'penalty': 'l1',</p> <p>'tol': 0.0009845702868676745</p>

XG Boost	<p>Performance: High predictive accuracy and speed.</p> <p>Regularization: Prevents overfitting and improves generalization.</p>	<p>Experiment A3:</p> <p>objective='reg:squarederror'</p> <p>This means that the model will minimize the squared error.</p> <p>random_state=42</p> <p>This parameter ensures reproducibility of the model results.</p> <p>Experiment A4:</p> <p>hyperparameter tuned using hyperopt and got the best parameters as below,</p> <p>'n_estimators': 4,</p> <p>'max_depth': 2,</p> <p>'learning_rate': 0.07237361273714017,</p> <p>'subsample': 0.9225985721539292,</p> <p>'colsample_bytree': 0.8385977926683998,</p>
----------	--	--

Experiment B: The first leg's cabin type is taken as the cabin type for the entire journey and multiple hops with different cabin types are included in the training. And the numerical features are transformed using standard scaler

ML Algorithm	Rationale	Description
--------------	-----------	-------------

XG Boost	<p>Performance: High predictive accuracy and speed.</p> <p>Regularization: Prevents overfitting and improves generalization.</p>	<p>Experiment B1:</p> <p>objective='reg:squarederror'</p> <p>This means that the model will minimize the squared error.</p> <p>random_state=42</p> <p>This parameter ensures reproducibility of the model results.</p>
----------	--	---

Experiment C: The records which have multiple hops with different cabin types are filtered from the dataset. And the numerical feature's cyclical nature captured using cos and sin transformation

ML Algorithm	Rationale	Description
XG Boost	<p>Performance: High predictive accuracy and speed.</p> <p>Regularization: Prevents overfitting and improves generalization.</p>	<p>Experiment C1:</p> <p>objective='reg:squarederror':</p> <p>This means that the model will minimize the squared error.</p> <p>random_state=42:</p> <p>This parameter ensures reproducibility of the model results.</p>
Ligh GBM	LightGBM carries out leaf-wise (vertical) growth that results in more loss reduction and, in turn, higher accuracy while being faster	<p>Experiment C2:</p> <p>objective='regression'</p> <p>This is to be used to set the type of ML algorithm chosen</p>

Results and Analysis

Experiment	Training	Validation	Test
Experiment A1: SGD Regression	RMSE: 180.50 MAE: 127.56	RMSE: 185.98 MAE: 130.89	RMSE: 198.73 MAE: 146.27
Experiment A2: SGD Regression	RMSE: 180.63 MAE: 128.06	RMSE: 184.66 MAE: 131.84	RMSE: 185.56 MAE: 136.07
Experiment A3: XGBoost	RMSE: 147.99 MAE: 102.21	RMSE: 160.78 MAE: 110.27	RMSE: 167.57 MAE: 118.81
Experiment A4: XGBoost	RMSE: 213.70 MAE: 156.77	RMSE: 215.55 MAE: 158.65	Did not run since the validation score didn't improve
Experiment B1: XGBoost	RMSE: 149.29 MAE: 102.99	RMSE: 161.28 MAE: 109.81	RMSE: 169.29 MAE: 119.67
Experiment C1: XGBoost	RMSE: 143.13 MAE: 100.52	RMSE: 157.87 MAE: 108.36	RMSE Test: 164.58 MAE Test: 117.87
Experiment C2: Light GBM	RMSE: 153.31 MAE: 108.70	RMSE: 160.33 MAE: 111.32	RMSE Test: 163.60 MAE Test: 117.98

- XGBoost in Experiment C1 has the lowest RMSE and MAE across training, validation, and test datasets:
- C1 (XGBoost) outperforms the others significantly, especially in training and validation phases. C2 (Light GBM) performs reasonably well but is not as effective as C1 in training and validation.
- Conclusion:



- Experiment C1 with XGBoost is the best model for regression, based on the RMSE and MAE metrics across training, validation, and test datasets.
- Key Insights:
 - Cyclical transformation of time components aids better model performance compared to standard scaler numerical transformation
 - Removing the records that had different cabin types in different legs improved the performance
- In future,
 - hyper parameters can be tuned for existing models for better performance.
 - Try more advanced modelling like deep learning using neural networks.



6. Evaluation

a. Evaluation Metrics

- **Root Mean Square Error (RMSE):** Primary Metric
 - What It Is: RMSE measures how far off our predictions are from the actual sales figures, focusing on larger errors.
 - Why It Matters: This metric is crucial for understanding the accuracy of our sales forecasts. If our predictions are significantly wrong, it can lead to either too much stock (overstock) or not enough (stockouts), which can hurt profits.
- **Mean Squared Error (MSE):** Supporting metric
 - What It Is: MSE looks at the average of all the errors squared, giving a sense of how errors are spread out.
 - Why It Matters: MSE serves as an additional measure alongside RMSE. It helps us compare different models and understand overall prediction quality.

b. Business Impact and Benefits


The final models directly address key challenges in airfare prediction, and enables both **travellers** and **travel businesses** to make more informed decisions. The models' success is measured by their mean closeness to the true fares, which directly impacts its utility for travellers. The models show a clear improvement, as seen in RMSE/MAE throughout each experiment, reflecting significant gains over baseline estimates.

It is important to note that these models provide only an estimate, as airfare prices can still vary widely due to unpredictable factors, including:

- Last-minute demand spikes, in relation to sudden events
- Airline promotions, which may happen at irregular intervals
- Seasonal shifts
- Domestic economic changes, which can affect the whole aviation industry

c. Data Privacy and Ethical Concerns

Although the project does not include personal information, it still contains sensitive travel and location data that, if mishandled, could potentially affect user privacy or market competition. There is also a risk that businesses might use the fare predictions to increase prices, impacting



consumer welfare or misuse information about their competitors. It is essential to ensure the model does not favour business interests over travellers to avoid unfair pricing.

The dataset's limited three-month span does introduce biases, which could potentially affect the accuracy of predictions for specific times or groups. To address this, the app should clearly communicate its limitations, helping users understand the seasonal data constraints and the variability of airfare predictions. This transparency helps build trust and sets realistic expectations for users.



7. Deployment

a. Model Serving

All team members' models are served in the final application, which includes:

- Two XGBoost models
- One artificial neural network (ANN)

These are the best models from validation and testing, which are then exported to either the joblib (for XGBoost) or Keras (for ANN) format for consumption by the app. The steps for deploying the models are as follows:

- A separate Git repository is created to store the web app code and the deployed models. The structure of the source code is described in Figure 10.

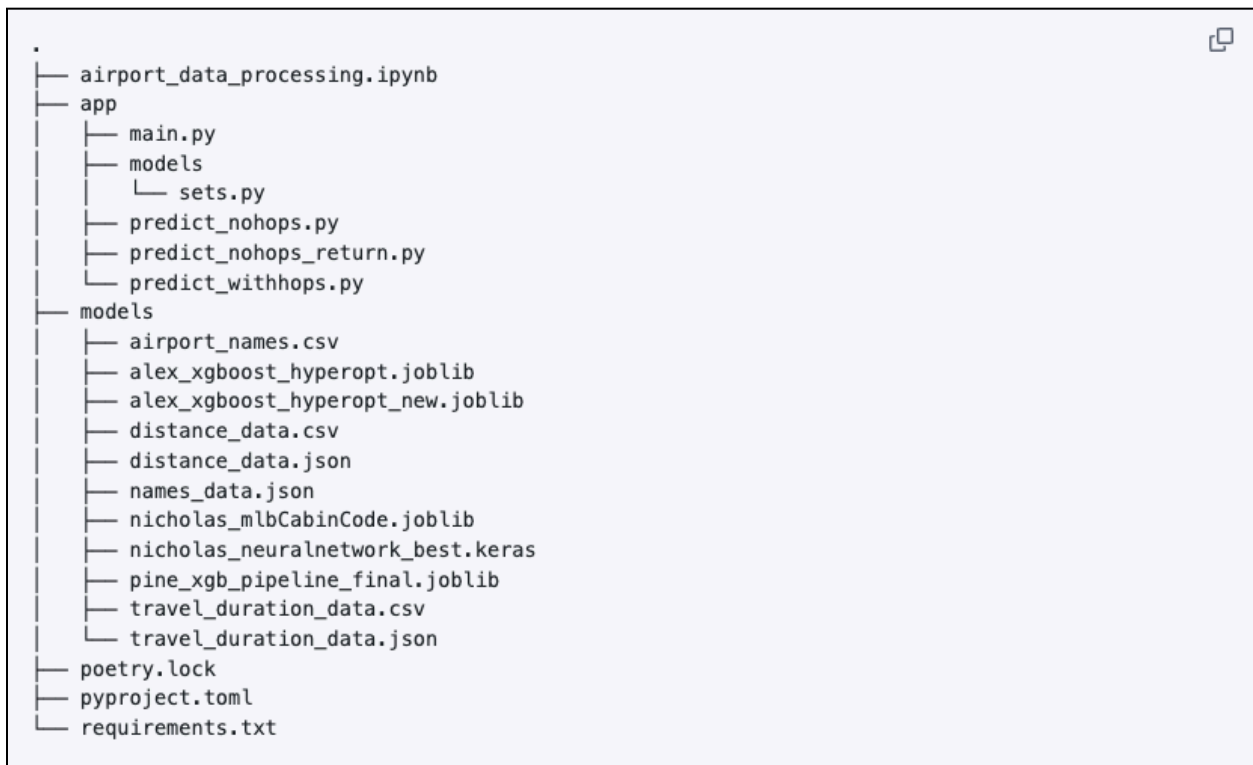


Figure 10: Project structure of the Streamlit web app.

- Configuration files are stored in the root directory: **poetry.lock**, **pyproject.toml**, **requirements.txt**

- The group's models and extra data are in **/models/**
- The **/app/** directory houses the main codebase, and the main script is called **main.py**. Extra scripts to call each team members' models are also stored here: **predict_nohops.py**, **predict_nohops_return.py**, **predict_withhops.py**
- There is an extra folder **/app/models/** that stores preprocessing codes for the ML models

There were some compatibility issues with versions when running models, since we did not agree on the versions of all dependencies from the beginning. This is a point of improvement for future deployment efforts.

b. Web App

The application aims to predict travel airfares for users within the US (Figure 11). Users supply information about the dates/times and origins/destinations for the trips they want to take, and other inputs e.g. cabin type, basic economy tickets. Users can choose between one-way, return, and multi-city flights. Instructions for installing and running the app are supplied in Appendix 2 of this report.

The main benefits of the app are:

- For customers: Provide airfare information across different types of ticket classes, which is essential for travellers to plan their journey and avoid price surprises when booked for, e.g. peak seasons.
- For providers: Provide the baseline for providers to price their tickets to maximise profit potentials. For example, airlines can use the information from this tool and start introducing markups for peak periods, and/or for travel dates that are very close to the search date.

Some improvement points for the next iterations include:

- A comparison of pricing across a range of time in a single view, which is a useful feature for customers who want to get a quick glance of airfares for a particular period.
- Current predictions are airline-agnostic. This may disregard the individual airline factor, which is considered the main driver for price. Including airline information is essential for customers to get the most value for their money while allowing them to fly with their favourite airlines.
- For multi-city trips, the origin for the first trip and destination for the last trip currently must be different. This is a major limitation and was an oversight during model training. Future iterations need to allow the model to accommodate circular trips, which is almost always the case for multi-city trips.



Your local estimator for all things airfare!

Have you ever wondered if you'd been overcharged airfare by the wealthy capitalist airlines? Do you desire knowing about the fairest possible fares before booking your tickets?

Look no further! Our airfare predictor will help you do exactly that, using the powers of our shiny in-house Machine Learning algorithms.

Go tell your airlines that you've been overcharged, and you would like a refund!

Go tell your friends and family about this app too!

Select your preferences below to get started.

One way Return Multi-city

Predict a one-way ticket

From	To
Atlanta Hartsfield International Airport (A...	Boston, Logan International Airport (BOS)
Date	Time
2024/11/07	10:00
<input type="checkbox"/> Basic economy only?	Cabin type
	Coach

Your journey summary is below:

From: Atlanta Hartsfield International Airport (ATL)
To: Boston, Logan International Airport (BOS)
Date: 2024-11-07
Time: 10:00:00
Basic economy: No
Cabin: Coach

Predict!

Figure 11: User interface (UI) of the web app.

8. Collaboration

a. Individual Contributions

Nicholas (Thanh Nhan) Le

I was responsible for:

- Gathering team members for the project.
- Developing 3 Machine Learning/neural network models, and deployed 1 as the final model.
- Setting up the Git repository for the app, and ideating/executing the entire logic of the Streamlit UI for team members to test and feedback.
- Creation of communication channels for team members to collaborate: MS Teams, WhatsApp.
- Some report formatting and reorganising.

Alexander Schou

I was responsible for:

- Writing the “Data Understanding” and “Business Understanding” parts of the report and creating the respective notebooks.
- Developing Machine learning models (LinearRegression, XGBoost and CatBoost) and deploy 1 as the final model.
- Writing about data privacy and ethical concerns.
- Report formatting and feedback on streamlit UI.

Paiynthalir Samyappan Nallamuthu

I was responsible for:

- Developing the first exploratory data analysis notebook that inspired the other team members
- Developing the first data preparation notebook that inspired the other team members
- Developing and testing machine learning models and deployed 1 as the final model.
- Report writing and formatting.



b. Group Dynamic

Group members assume equal responsibilities in model development and deployment. We communicated each week via Teams and WhatsApp. Since most of the model development process can be done individually and online, we learn from one another's notebooks through their GitHub pushes, and build off of their contributions to avoid wasting time on duplicated work. A highlight was that **Paiynthalir** started with exploratory data analysis (EDA) very early on, so other team members greatly benefited from her insights. **Nicholas** and **Alexander** also shared data insights that are built from Paiynthalir's analysis, which eventually informed their own modelling processes.

c. Ways of Working Together

We collaborated in an Agile manner, which means the work was iteratively built and instant feedback was given through each stage of analysis. This allowed us to push model prototypes very early on without having to wait for each other to finish coding.

We organise weekly discussions via WhatsApp and limited meetings to one when the project started and another when it was close to finishing. Progress tracking and modelling decision-making are almost entirely via GitHub pushes with discussions on Teams/WhatsApp along the way, which significantly speeds up the project timeline compared to the traditional one-meeting-a-week approach.

d. Issues Faced

During the course of the project, we encountered the following issues:

- For the first 2 weeks of the project, **Nicholas** was almost unavailable due to other subject commitments, so he was only able to kick off the project in the third week when **Paiynthalir** finished most of her EDA and moved on to modelling. This caused some inefficiency in version compatibility when the group put their models together. Fixing Python/package versions and re-running the notebooks resolved the issue.
- Some user inputs of the final UI translate to features that are unavailable to trained models, so a few models had to be re-trained using extra features.

To remedy these issues in future collaborations, we can clearly outline the expected outcomes for all features/models before starting the development, to avoid surprises as the project gets more complex.





9. Conclusion

This project provided an accurate and accessible airfare prediction tool that benefits both travellers and businesses. The tools were observed to achieve strong predictive performance despite going off of data within a limited timespan.

This project notes the importance of cyclical transformations and categorical encoding to enable robust model learning and inference. Future work could integrate a broader data set to address seasonal trends and further enhance prediction accuracy.

Overall, this project not only addresses a critical gap in travel planning but also provides a scalable framework for continuous improvement, supporting informed decisions.





10. References

- [1] Bureau of Transportation Statistics, “Inter-Airport Distance,” *bts.gov*, 2017. <https://www.transtats.bts.gov/Distance.aspx> (accessed Nov. 01, 2024).
- [2] W. S. Sarle, “FAQ, Part 2 of 7: Learning,” *faqs.org*, Oct. 11, 2002. <http://www.faqs.org/faqs/ai-faq/neural-nets/part2/> (accessed Nov. 01, 2024).
- [3] Leonard's Guide Online, “United States Airport Codes : Three Letter US Airport Codes,” *leonardsguide.com*, 2024. <https://www.leonardsguide.com/us-airport-codes.shtml> (accessed Nov. 05, 2024).





Appendix 1: Data Dictionary

legId: An identifier for the flight.

searchDate: The date (YYYY-MM-DD) on which this entry was taken from Expedia.

flightDate: The date (YYYY-MM-DD) of the flight.

startingAirport: Three-character IATA airport code for the initial location.

destinationAirport: Three-character IATA airport code for the arrival location.

fareBasisCode: The fare basis code.

travelDuration: The travel duration in hours and minutes.

elapsedDays: The number of elapsed days (usually 0).

isBasicEconomy: Boolean for whether the ticket is for basic economy.

isRefundable: Boolean for whether the ticket is refundable.

isNonStop: Boolean for whether the flight is non-stop.

baseFare: The price of the ticket (in USD).

totalFare: The price of the ticket (in USD) including taxes and other fees.

seatsRemaining: Integer for the number of seats remaining.


totalTravelDistance: The total travel distance in miles. This data is sometimes missing.

segmentsDepartureTimeEpochSeconds: String containing the departure time (Unix time) for each leg of the trip. The entries for each of the legs are separated by '|l'.

segmentsDepartureTimeRaw: String containing the departure time (ISO 8601 format: YYYY-MM-DDThh:mm:ss.000±[hh]:00) for each leg of the trip. The entries for each of the legs are separated by '|l'.

segmentsArrivalTimeEpochSeconds: String containing the arrival time (Unix time) for each leg of the trip. The entries for each of the legs are separated by '|l'.

segmentsArrivalTimeRaw: String containing the arrival time (ISO 8601 format:



YYYY-MM-DDThh:mm:ss.000±[hh]:00) for each leg of the trip. The entries for each of the legs are separated by '|'.

segmentsArrivalAirportCode: String containing the IATA airport code for the arrival location for each leg of the trip. The entries for each of the legs are separated by '|'.

segmentsDepartureAirportCode: String containing the IATA airport code for the departure location for each leg of the trip. The entries for each of the legs are separated by '|'.

segmentsAirlineName: String containing the name of the airline that services each leg of the trip. The entries for each of the legs are separated by '|'.

segmentsAirlineCode: String containing the two-letter airline code that services each leg of the trip. The entries for each of the legs are separated by '|'.

segmentsEquipmentDescription: String containing the type of airplane used for each leg of the trip (e.g. "Airbus A321" or "Boeing 737-800"). The entries for each of the legs are separated by '|'.

segmentsDurationInSeconds: String containing the duration of the flight (in seconds) for each leg of the trip. The entries for each of the legs are separated by '|'.

segmentsDistance: String containing the distance traveled (in miles) for each leg of the trip. The entries for each of the legs are separated by '|'.

segmentsCabinCode: String containing the cabin for each leg of the trip (e.g. "coach"). The entries for each of the legs are separated by '|'.

Appendix 2: Installation and Running Instructions for Streamlit web app

1. Clone the repository:

```
git clone https://<username>@github.com/nicnl31/airfare-streamlit
```

where <username> is your GitHub username. When prompted for your password, you must use your personal access token (PAT). If you don't have one, you can create one following this guide: <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens#creating-a-personal-access-token-classic>

After the repository is cloned, go into it:

```
cd airfare-streamlit
```

2. Create and activate a virtual environment:

```
python -m venv .venv
```

```
source .venv/bin/activate
```

3. Inside your virtual environment, upgrade pip and install the project's dependencies:

```
pip install --upgrade pip
```

```
pip install -r requirements.txt
```

4. Run the app:

```
streamlit run app/main.py
```

5. Launch the app in your browser with localhost and port 8501:

```
http://localhost:8501
```