COS30049
Computing Technology Innovation Project

Assignment 2: AI-Based Project – Civil Aviation

Group: 15-01

Facilitator: Mr. Zehang Deng

Members:
Natalie Phan (103990764)
Ai Vi Tran (104209393)
Nolan Sia (102766524)

Wordcount: 2000 (excluding coverpage, table of contents, bibliography)

Table of Contents

Natalie Phan, Nolan Sia & Ai Vi Tran

Introduction

This machine learning project aims to address the financial and operational challenges caused by flight delays and pricing fluctuations, impacting both airlines and passengers. By leveraging past flight data and external factors like weather, the project seeks to build a predictive system to forecast delays and price changes. This system will assist users—including passengers, airlines, and travel agencies—in making informed travel decisions. The target audience includes frequent travellers and airline operators seeking to optimize flight management.

Problem Framing

Flight delays and fluctuating prices strain both passengers and airlines. Current systems rely on simplified models or historical averages, making them ineffective at adapting to real-time changes such as weather conditions, demand spikes, or operational disruptions.

Existing solutions suffer from limited predictive accuracy and fail to integrate various data sources to make reliable forecasts for delays or pricing. Additionally, they lack real-time adaptability and do not fully account for dynamic factors like last-minute cancellations or weather changes.

These platforms provide generic forecasts, overlooking the unique needs of user groups like travellers and airlines. Machine learning offers a more robust approach by analysing vast amounts of historical and current data to identify patterns and improve predictions. Regression and classification techniques can adjust predictions based on new inputs, offering more accurate forecasts and customized insights that enhance decision-making and operational efficiency.

Data Collection

1. Sources:

1.1 Kaggle: weatherAUS.csv

Kaggle is a popular platform for accessing publicly available machine learning and data science datasets, often provided by institutions or researchers.

The weatherAUS.csv dataset was used for this project because it provides thorough coverage of weather conditions in multiple places in Australia that are relevant for airfare analysis.
License: Apache 2.0 open source

1.2. Australian Government Data: flight_price.csv and otp_time_series_web.csv

Australian Government Data offers comprehensive, up-to-date information on Australian airports, airlines, and flight routes, including detailed pricing for domestic aviation. These two datasets were key to the project, providing a solid foundation for building predictive models that reflect real-world pricing and flight delay patterns in the Australian aviation market.

License:
flight_price.csv - Creative Commons Attribution 2.5 Australia
otp_time_series_web.csv - Creative Commons Attribution 3.0 Australia

Merging the three datasets enabled a more comprehensive analysis of the factors influencing flight pricing and delays, with a focus on the impact of weather conditions over time.

2. Challenges and solutions:

2.1 Inconsistent Dataset Timeline:

- Training a machine learning model requires a substantial amount of data, making it challenging to find two datasets with compatible timelines. Additional time was spent on data collection to ensure a high-quality dataset for the project.

2.2 Inconsistent Data Size:

- An additional dataset was needed to support the main one, but most lacked a matching sample size. More time was dedicated to data collection and cleaning to meet the requirements before merging the datasets.

Data Processing

The goal of data processing is to clean, standardize, and format the collected data to improve the performance of algorithms for analysis and prediction. The key steps involved include:

1. Handling Missing Values:

1.1 Checking the percentage of missing values:

```
# This for loop is to check the percentage of missing value which helps in considering features for training models
for col in weather_df.columns:
    missing_data = weather_df[col].isna().sum()
    missing_percent = missing_data/len(weather_df)*100
    print(f"Column {col} has {missing_percent:.2f}% missing values")

# This for loop is to check the percentage of missing value which helps in considering features for training models
for col in flight_price_df.columns:
    missing_data = flight_price_df[col].isna().sum()
    missing_percent = missing_data/len(flight_price_df)*100
    print(f"Column {col} has {missing_percent:.2f}% missing values")

# This for loop is to check the percentage of missing value which helps in considering features for training models
for col in delay_df.columns:
    missing_data = delay_df[col].isna().sum()
    missing_percent = missing_data/len(delay_df)*100
    print(f"Column {col} has {missing_percent:.2f}% missing values")
```

Figure 1: Percentage of missing value

A for loop was used to calculate the percentage of missing data and assess its reliability for further processing.

1.2 Filling missing values with mean values and filling missing fields with mode values.

```
# WEATHER
# Filling missing value with mean
numeric_colums = weather_df_copy.select_dtypes(include=[np.number]).columns
weather_df_copy[numeric_colums] = weather_df_copy[numeric_colums].fillna(weather_df_copy[numeric_colums].mean())
# Filling missing fields with mode
categoric_colums = weather_df_copy.select_dtypes(include=[object]).columns
weather_df_copy[categoric_colums] = weather_df_copy[categoric_colums].apply(lambda x: x.fillna(x.mode()[0]))

# FLIGHT_PRICE
# Filling missing value with mean
numeric_colums = flight_price_df_copy.select_dtypes(include=[np.number]).columns
flight_price_df_copy[numeric_colums] = flight_price_df_copy[numeric_colums].fillna(flight_price_df_copy[numeric_colums].mean())
# Filling missing fields with mode
categoric_colums = flight_price_df_copy.select_dtypes(include=[object]).columns
flight_price_df_copy[categoric_colums] = flight_price_df_copy[categoric_colums].apply(lambda x: x.fillna(x.mode()[0]))

# DELAY
# Fill numerical columns with median values
numeric_colums = delay_df_copy.select_dtypes(include=[np.number]).columns
delay_df_copy[numeric_colums] = delay_df_copy[numeric_colums].fillna(delay_df_copy[numeric_colums].mean())
# Fill categorical columns with mode
categoric_colums = delay_df_copy.select_dtypes(include=[object]).columns
delay_df_copy[categoric_colums] = delay_df_copy[categoric_colums].apply(lambda x: x.fillna(x.mode()[0]))
```

Figure 2: Dealing with missing value

Replacing missing numerical values with the mean preserves the dataset's distribution and reduces bias. For categorical data, using the mode maintains consistency by keeping the most common value. Both methods improve data quality for modelling and analysis.

2. Removing unneeded columns.

```
# drop the unneeded columns in dataset which include more than 10% missing value
flight_price_df_copy = flight_price_df_copy.drop(columns=[col for col in flight_price_df_copy.columns if 'Unnamed' in col])
print(flight_price_df_copy)

weather_df_copy = weather_df_copy.drop(columns=['Evaporation', 'Sunshine', 'Cloud9am', 'Cloud3pm'])
print(weather_df_copy)

delay_df_copy = delay_df_copy.drop(columns=['Airline'])
```

Figure 3: Removing unneeded columns

Columns with high percentages of missing or non-meaningful values were removed to improve the dataset's quality.

3. Feature Scaling & Feature Engineering:

```
# It is used to convert categorical variables into a numerical format that machine learning algorithms can understand
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder

# Categorical columns
categorical_cols = ['Port1', 'Port2', 'Departing_Port', 'Arriving_Port']
# Numerical columns
numeric_cols = ['MinTemp', 'MaxTemp', 'WindGustSpeed', '$Value']
# Select features for modeling
X = merged_df[['Port1', 'Port2', 'MinTemp', 'MaxTemp', 'WindGustSpeed', '$Value', 'Departing_Port', 'Arriving_Port']]
# Select target for modeling - $Real - Released flight price on the website
y = merged_df['$Real']

ct = ColumnTransformer(
    transformers=[
        ('encode', OneHotEncoder(), categorical_cols),  # Encode categorical columns
        ('scale', StandardScaler(), numeric_cols)  # Scale numerical columns
    ]
)
```

Figure 4: Feature scaling & Feature Engineering

3.1 Feature scaling – StandardScaler:

Features were scaled to ensure equal contribution during training, particularly for models sensitive to input scale.

3.2 Feature Engineering - Encoding data:

Categorical columns were transformed into numerical representations to prevent bias and allow proper inclusion in the models.

4. Transforming data – Standardized data:

```
# Appending Date column in flight_price_df_copy to prepare for merging data (as there are only Month and Year in the dataset)
# We assume that the data is recorded on the first day of the month
flight_price_df_copy['Date'] = pd.to_datetime(flight_price_df_copy['Year'].astype(str) + '-' + flight_price_df_copy['Month'].astype(str) + '-0
weather_df_copy['Date'] = pd.to_datetime(weather_df_copy['Date'])
# Standardise data
weather_df_copy['Location'] = weather_df_copy['Location'].replace('PerthAirport', 'Perth')
```

Figure 5: Transforming data

Standardization was applied to bring numerical features into uniformity, preventing value magnitude differences from affecting the algorithm's learning process and improving model training.

5. Joining and merging datasets:

```
# We merge the flight_price file with the weatherAUS file
merged_df = pd.merge(
    flight_price_df_copy,
    weather_df_copy,
    left_on=['Date', 'Port1'],
    right_on=['Date', 'Location'],
    how='inner'
)
print(merged_df.head())
# We merge the above merged file with the delay file
merged_df = pd.merge(
    merged_df,
    delay_df,
    left_on=['Year', 'Month', 'Port1', 'Port2'],
    right_on=['Year', 'Month_Num', 'Departing_Port', 'Arriving_Port'],
    how='inner'
)
```

Figure 6: Joining and merging dataset

The flight pricing, weather, and delay data were merged using common fields (Year, Month, Port1, Port2) to create a unified dataset. This ensured consistency across sources and provided a comprehensive dataset for analysis and model building.

6. Outlier detection and removal:

```
# Calculate Q1, Q3, and IQR
Q1 = merged_df['$Real'].quantile(0.25)
Q3 = merged_df['$Real'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

print(f"Lower threshold value: {lower_bound}")
print(f"Upper threshold value: {upper_bound}")

# Check for outliers
outliers = merged_df[(merged_df['$Real'] < lower_bound) | (merged_df['$Real'] > upper_bound)]
print(f"Number of outliers before removal: {len(outliers)}")

# Filter and remove outliers
df_cleaned = merged_df[merged_df['$Real'] < upper_bound]  # Keep only values below the upper threshold
print(f"Total number of records initially: {len(merged_df)}")
print(f"Total number of records after removing outliers: {len(df_cleaned)}")

# Check the number of outliers in df_cleaned
outliers_after_cleaning = df_cleaned[(df_cleaned['$Real'] < lower_bound) | (df_cleaned['$Real'] > upper_bound)]
num_outliers_after_cleaning = len(outliers_after_cleaning)

print(f"Number of outliers in df_cleaned: {num_outliers_after_cleaning}")
```

Figure 7: Dealing with outliers

6.1 Outlier detection:

Interquartile Range (IQR) method was used to detect and remove outliers in the target feature ($Real).

6.2 Outlier removal:

Values above the top threshold were filtered out to ensure the model was trained on a more representative dataset, reducing their negative impact on performance.

7. Handling class imbalance:

```
# Step 3: Handle Imbalance using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)
```

Figure 8: Handling imbalance

Balancing target class (for classification): During training, minority classes were sufficiently represented due to the application of SMOTE, which addresses class imbalance in flight delays.

8. Export cleaned data:

```
merged_df.to_csv('merged_data.csv', index=False)
```

Figure 9: Export merged data

The cleaned dataset was saved as a new CSV file (merged_data.csv) to enable reuse without rerunning the preparation process, making it easier to train and experiment with models.

Data analysis

1. Bar plot:

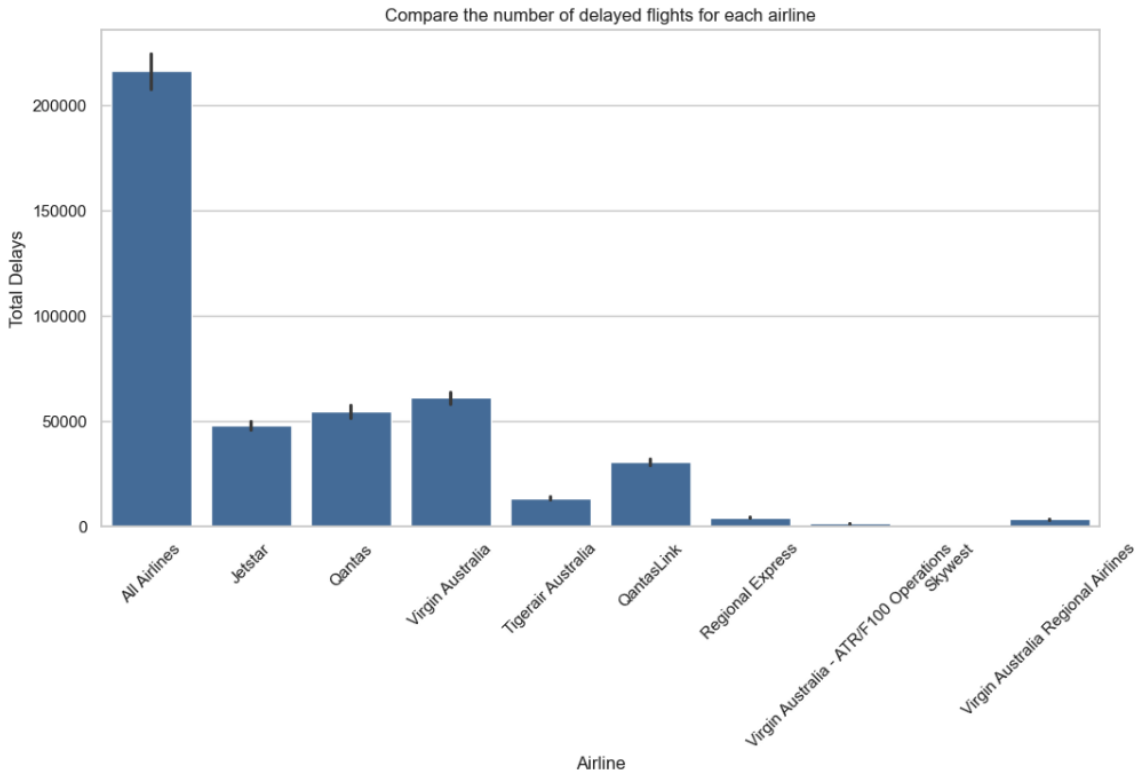

Figure 10: Bar plot

This tool compares delays across airlines, highlighting performance gaps and patterns. It provides a summary of the most frequently delayed airlines, aiding in analysis and feature selection for modelling.

2. Box plot

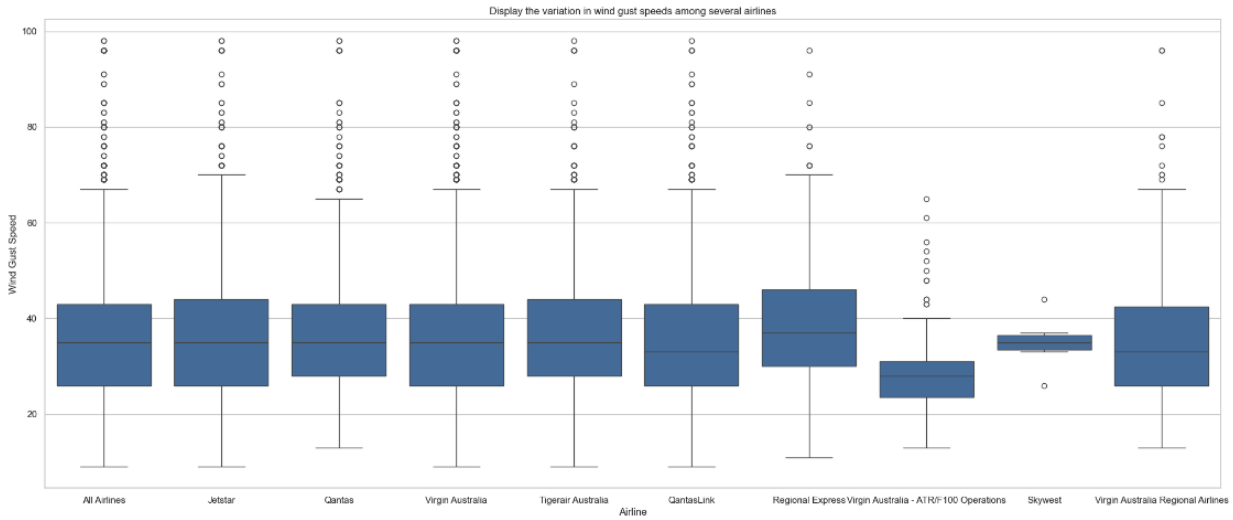

Figure 11: Box plot

This visualization shows wind condition variations by airline, including median, quartiles, and outliers. It offers valuable insights for modelling by revealing possible links between wind and flight delays.

3. Line plot

Figure 12: Line plot

This graph shows monthly average flight costs across years, highlighting annual variances and seasonal patterns. It helps identify price fluctuations, crucial for building accurate predictive models.

Machine Learning Model Selection

1. Train-Test Split:

```
from sklearn.model_selection import train_test_split
np.random.seed(42)
```

```
# Apply ColumnTransformer to encode and scale the data
X_processed = ct.fit_transform(X)
```

```
# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_processed, y, test_size=0.2, random_state=42)
```

```
print(merged_df['$Real'].describe())
```

Figure 13: Data splitting

After preprocessing, the dataset was divided into training and testing sets (80% and 20%). This division guaranteed the model's ability to generalise effectively to new data.

2. Selected models:

2.1 Random Forest regressor – Flight price prediction

a. Handling complex relationships: Unlike Linear Regression, Random Forest captures non-linear interactions between features, such as weather, route, and timeline, in determining flight prices.
b. Feature Importance: Random Forest provides insights into which features have the greatest impact on flight price.
c. Robust to overfitting: Its ensemble nature, combining multiple decision trees, helps reduce overfitting, especially with large datasets.
d. Performance – Model evaluation: Random Forest outperformed Linear Regression in preliminary testing and was chosen as one of the two main models.

```
Random Forest Test MSE: 12.080757425264007
Random Forest Test R-squared: 0.9996997823164813
```

Figure 14: Random Forest metrics

e. A Test MSE of 12.08 and Test R^2 of 0.9997 demonstrate the model's ability to efficiently handle complex data relationships.

2.2 Random Forest classifier – Flight delay prediction

a. Handling mixed data types: It effectively manages both numerical and categorical data, key features in the model.
b. Class imbalance: It addresses imbalanced data using SMOTE.

c. Performance – Model evaluation:

```
Accuracy: 0.9643683479870431
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.98      0.96      3207
           1       0.98      0.95      0.96      3276

    accuracy                           0.96      6483
   macro avg       0.96      0.96      0.96      6483
weighted avg       0.97      0.96      0.96      6483

Confusion Matrix:
[[3155   52]
 [ 179 3097]]
```

Figure 15: Classifier metrics

i. It showed strong performance in early testing, with high precision in predicting flight delays compared to basic classifiers.

ii. Preliminary testing achieved 96.4% accuracy with high precision and recall for 'Departures_Delayed,' which contributed to its selection as a main model.

3. Additional machine learning model:

3.1 Linear regression

a. Reason: Compared to Random Forest, Linear Regression is less computationally expensive and faster to train, making it ideal for preliminary modelling and data exploration without requiring significant processing power.

b. Baseline comparison: Starting with a simpler model like Linear Regression makes it easier to assess improvements when switching to more advanced models, such as Random Forest.

c. Performance – model evaluation:

```
{'Train MSE': 275.3203195736783,
 'Train R-squared': 0.9930942041647366,
 'Test MSE': 274.81169326601764,
 'Test R-squared': 0.9931706823461534,
 'Cross-Validation R-squared Mean': 0.9880261506060126,
 'Cross-Validation R-squared Std': 0.009030785807530254}
```

Figure 16: Linear metrics

d. With the R^2 achievement of 0.993 and relative accuracy, this is also the reason why this model was selected to be an alternative model if Random Forest is not employed.

4. Models that are not chosen:

4.1 Logistic regression:

This model is designed for binary or multiclass classification, but the target variable $Real is continuous, making Logistic Regression unsuitable.

4.2 Support Vector Machines (SVM):

While effective, SVM was not selected due to its need for extensive parameter tuning. Random Forest provided a better balance of performance and ease of use.

4.3 K-Means Clustering and DBSCAN:

These unsupervised learning methods were not appropriate since the dataset already had labelled categories.

Implementation

1. Technical Implementation

Several Python libraries were used for data preparation, modelling, and evaluation. Matplotlib and Seaborn handled data visualization, Pandas managed data manipulation, Scikit-learn was used for machine learning tasks, and imbalanced-learn addressed class imbalance with SMOTE.

Preprocessing included feature engineering, scaling, and handling missing values. Categorical columns were filled with the mode, numerical columns with the mean, and irrelevant columns were removed. Flight pricing, weather, and delay data were merged using common keys, with OneHotEncoding and StandardScaler applied for encoding and scaling.

Both Linear Regression and Random Forest models were tested for flight price prediction, with Random Forest chosen for its superior performance. SMOTE was used to address class imbalance, and Random Forest Classifier predicted flight delays. The dataset was split 80% for training and 20% for testing.

Challenges included handling missing data, merging datasets, and addressing class imbalance in delay predictions. Careful feature selection was required to identify the most impactful variables.

2. Implementation Evaluation

Random Forest outperformed other models for both flight price and delay predictions. Random Forest Regressor achieved an R-squared of 0.999, outperforming Linear Regression's 0.993, with an MSE of 11.88. For delay prediction, Random Forest Classifier reached 96.4% accuracy after using SMOTE to balance the dataset. The confusion matrix and classification report confirmed the model's reliability in predicting both delayed and on-time flights.

The significant influence of weather, particularly wind gusts, on flight prices and delays was a key finding. Adding real-time weather data could further improve performance. Random Forest outperformed simpler models like Linear Regression, particularly in handling non-linear relationships.

Conclusion

The project effectively used machine learning to predict flight prices and delays. Random Forest Regressor provided accurate price predictions, while the Classifier achieved 96.4% accuracy for delays. Weather, especially wind speeds, was a significant factor, and SMOTE improved delay predictions by addressing class imbalance.

Future work could integrate real-time weather and aircraft data for dynamic predictions and explore advanced models like neural networks or gradient boosting. A web application could provide real-time forecasts, enhancing the project's value for airlines and travellers.

References

- matplotlib.org. (n.d.). matplotlib.pyplot.scatter — Matplotlib 3.5.1 documentation. [online] Available at: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html.

- www.w3schools.com. (n.d.). Matplotlib Scatter. [online] Available at: https://www.w3schools.com/python/matplotlib_scatter.asp.

- Scikit-learn (2019). scikit-learn: machine learning in Python. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/.

- Imbalanced-learn.org. (2021). imbalanced-learn documentation — Version 0.8.1. [online] Available at: https://imbalanced-learn.org/stable/.

- Bureau of Infrastructure, T. and R.E. (2020). Aviation Statistics. [online] Bureau of Infrastructure, Transport and Regional Economics. Available at: https://www.bitre.gov.au/statistics/aviation.

- www.kaggle.com. (n.d.). Rain in Australia. [online] Available at: https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package.