MACHINE LEARNING WEB APPLICATION REPORT

Project Title: Civil Aviation

SEPTEMBER 18, 2024

888 – PEAKY BLINDER
Xuan Tuan Minh Nguyen – Ba Viet Anh (Henry) Nguyen – Trong Dat Hoang
Tutor: Ningran Li

Word Count (Exclude cover page, table of content, bibliography and appendix): 2489

Contents

I. Introduction

In an era of increasingly complex air travel, understanding and predicting flight prices has become crucial for both consumers and airlines. The primary motivation behind this project is to empower travelers with insights that can help them make informed decisions about their airfare purchases, so they could potentially save money and reduce the stress associated with ticket booking.

Our intended users are primarily budget-conscious travelers who want to optimize their travel expenses. By providing them with a tool that can predict flight prices and price categories, we aim to help them plan their trips more effectively, choosing the most cost-effective options without compromising on their travel needs.

II. Problem Framing

The challenge of predicting flight prices is apparently multifaceted and complex. Airlines use dynamic pricing strategies that consider numerous variables, including demand, seasonality, and competitor pricing. This complexity makes it difficult for consumers to anticipate price fluctuations and make informed purchasing decisions.

Existing solutions, such as fare comparison websites, provide current prices but lack predictive capabilities. They don't offer insights into future price trends or the factors influencing these changes. This limitation often leads to suboptimal purchasing decisions, where travelers either buy too early at higher prices or wait too long and miss out on better deals.

A machine learning approach is particularly suitable for this problem due to its ability to:

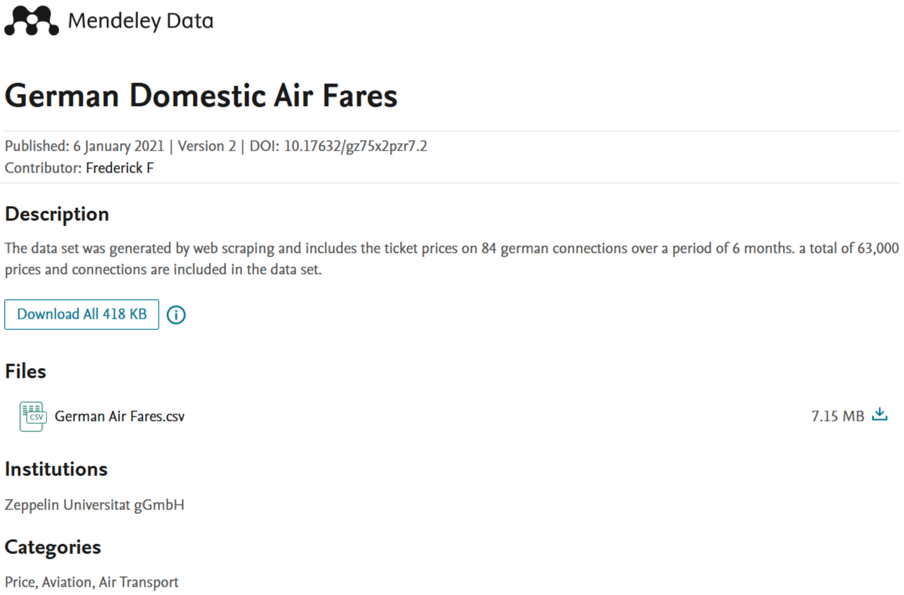
1. Handle large volumes of data with multiple variables
2. Identify complex patterns and relationships that may not be apparent through traditional analysis
3. Continuously learn and adapt to changing market conditions
4. Provide probabilistic predictions that can account for the inherent uncertainty in price fluctuations

Two questions (objectives) we want to help customers answer by using machine learning are:

1. “What is the price of the air fares in the future (with the same airlines, same destination or same seasonality)?”
2. “What will the price category of these flights belong to in the future?”

III. Data Collection

1. Source and Method



Mendeley Data

German Domestic Air Fares

Published: 6 January 2021 | Version 2 | DOI: 10.17632/gz75x2pzr7.2
Contributor: Frederick F

Description

The data set was generated by web scraping and includes the ticket prices on 84 german connections over a period of 6 months. a total of 63,000 prices and connections are included in the data set.

Files

German Air Fares.csv 7.15 MB

Institutions

Zeppelin Universitat gGmbH

Categories

Price, Aviation, Air Transport

The dataset “German Domestic Air Fares” was downloaded from Mendeley Data, published on January 6, 2021, by contributor Frederick F and associated with Zeppelin Universität. It is a prestigious university, so we decided to trust and use this dataset.

2. Data Collection Criteria

In selecting this dataset, we applied the following criteria to ensure its relevance and sufficiency for our analysis:

1. **Relevance to German civil aviation:** The dataset specifically focuses on domestic flights within Germany, aligning perfectly with our project scope.
2. **Comprehensiveness:** With 63,000 data points covering 84 routes, the dataset provides a broad and representative sample of the German domestic air travel market.
3. **Recency:** Published in 2021, the data reflects relatively recent pricing trends, crucial for developing a current predictive model.
4. **Feature richness:** The dataset includes key variables such as departure/arrival cities, dates, times, airlines, and prices, providing a solid foundation for our analysis.
5. **Temporal span:** Covering a 6-month period, the data allows for analysis of seasonal trends and temporal variations in pricing.

6. **Credibility:** Association with Zeppelin Universität, a recognized institution, lends academic credibility to the data.

3. Challenges and Solutions

We have faced challenges when collecting the data. We initially aimed to supplement the primary dataset with additional information such as jet fuel cost or number of travelers to Germany from 2019 to 2020. The data about jet fuel cost and traveler is not open source because it is company data. If we want it, we need to pay around $80. Solution: We focused on maximizing the value of our primary dataset through extensive feature engineering and creating derived variables to capture additional insights.

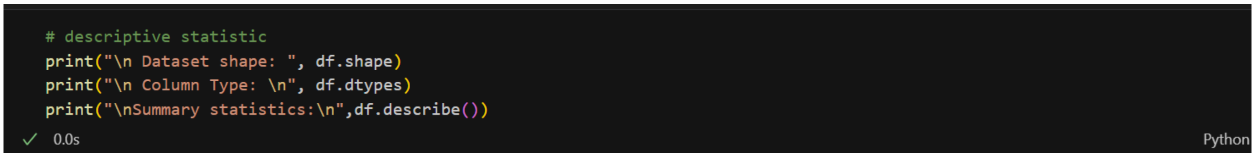
IV. Data Processing

Our data processing was designed to clean, transform, and prepare the raw data for machine learning algorithms. This section outlines the steps taken to ensure data quality, consistency, and relevance for our analysis:

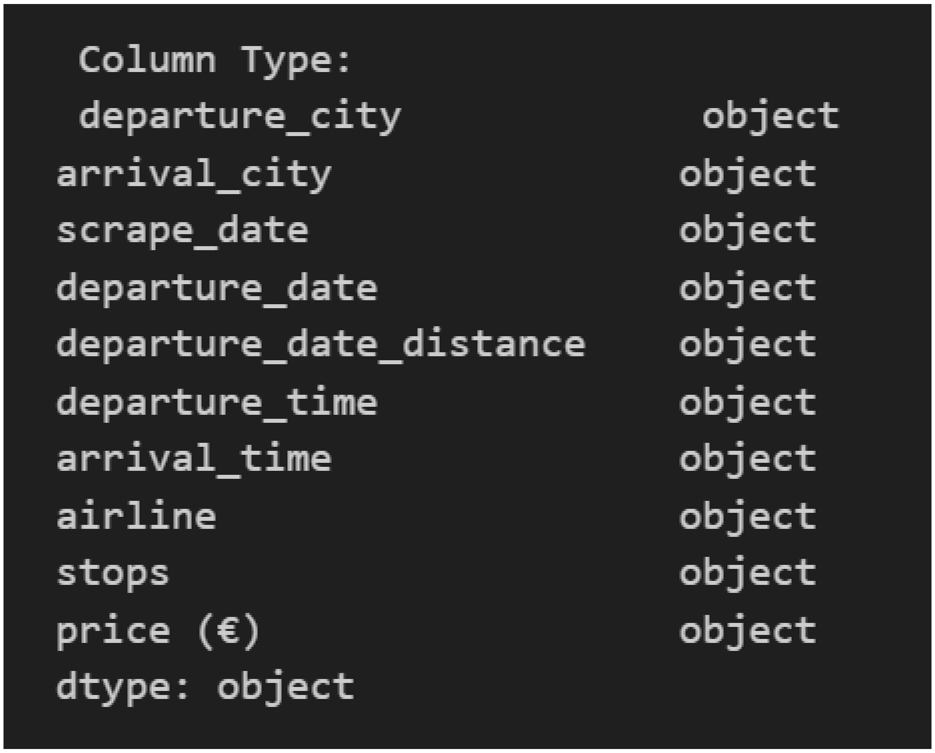
1. Exploratory Data Analysis

Before proceeding with data cleaning and preprocessing, we conducted a thorough Exploratory Data Analysis to understand the structure, quality, and characteristics of our dataset. This crucial step informed our subsequent data processing decisions.

a) Initial Data Inspection:



```
# descriptive statistic
print("\n Dataset shape: ", df.shape)
print("\n Column Type: \n", df.dtypes)
print("\nSummary statistics:\n",df.describe())
```



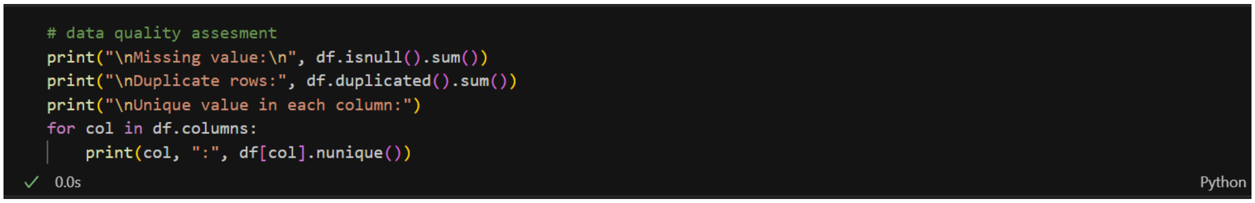
```
 Column Type:
 departure_city                object
arrival_city                  object
scrape_date                   object
departure_date                object
departure_date_distance       object
departure_time                object
arrival_time                  object
airline                       object
stops                         object
price (€)                     object
dtype: object
```

Key observations:

- All columns were initially read as 'object' type, indicating potential data type inconsistencies. We must convert this into the right type for analytics and train the model later.

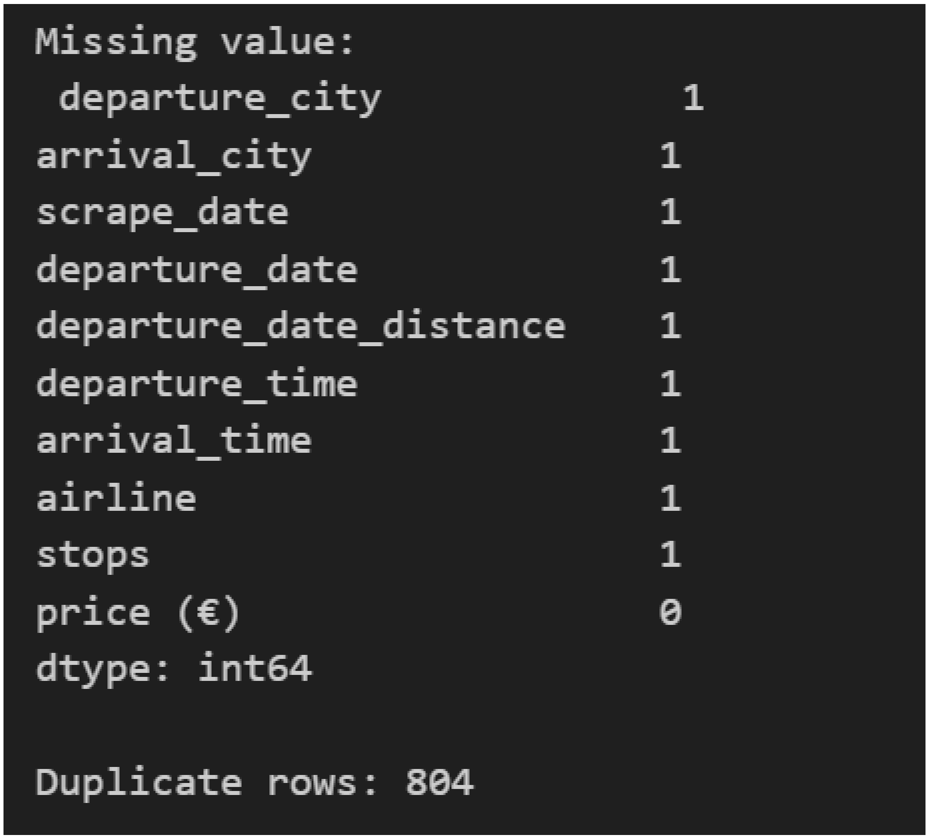
- Summary statistics revealed the presence of non-numeric data in what should be numeric columns.

b) Data quality assessment:



```python
# data quality assesment
print("\nMissing value:\n", df.isnull().sum())
print("\nDuplicate rows:", df.duplicated().sum())
print("\nUnique value in each column:")
for col in df.columns:
    print(col, ":", df[col].nunique())
```

The results:

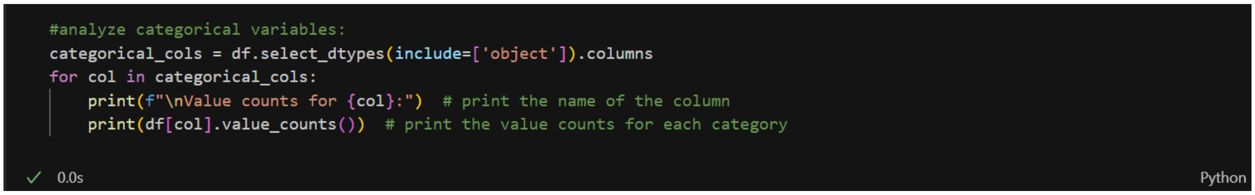


```
Missing value:
 departure_city             1
arrival_city               1
scrape_date                1
departure_date             1
departure_date_distance    1
departure_time             1
arrival_time               1
airline                    1
stops                      1
price (€)                  0
dtype: int64

Duplicate rows: 804
```

The rationale for addressing these issues:

- Missing values can skew analysis and model performance.
- Duplicate rows can lead to overfitting and bias in machine learning models.

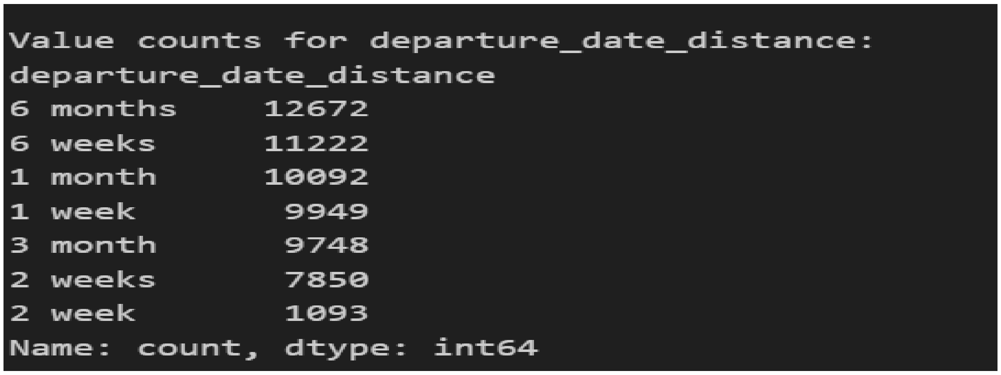
c) Categorical Variable Analysis



```
#analyze categorical variables:
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    print(f"\nValue counts for {col}:")  # print the name of the column
    print(df[col].value_counts())  # print the value counts for each category
```

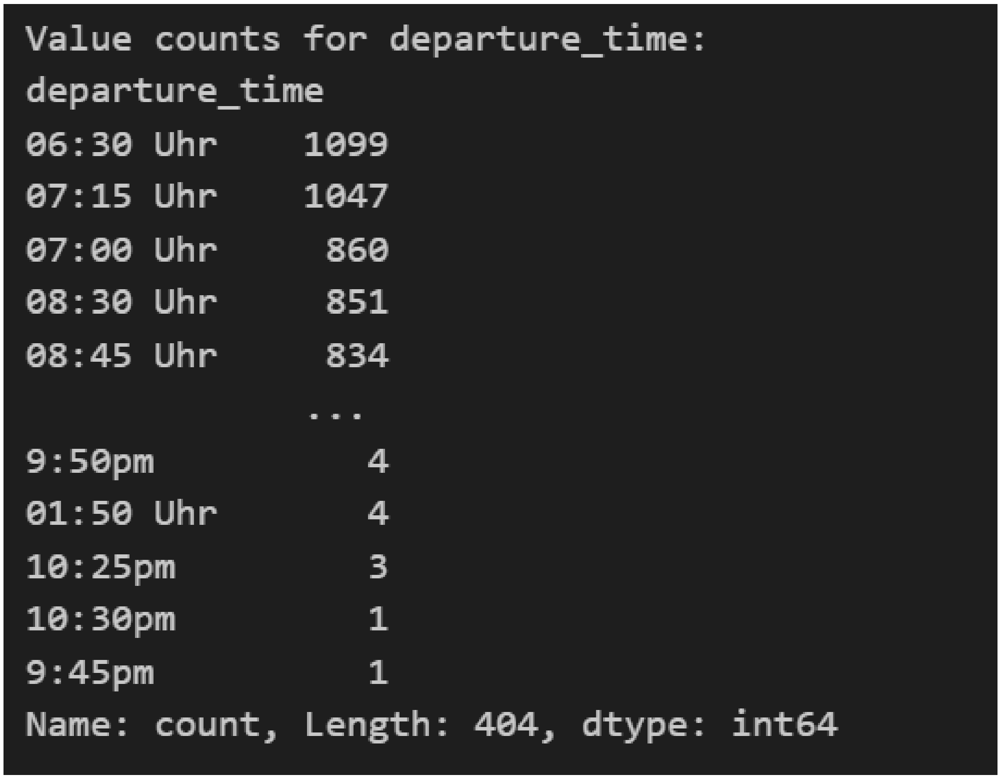
There are some inconsistencies in the data:

- **Departure_date_distance**: It used both months and weeks:



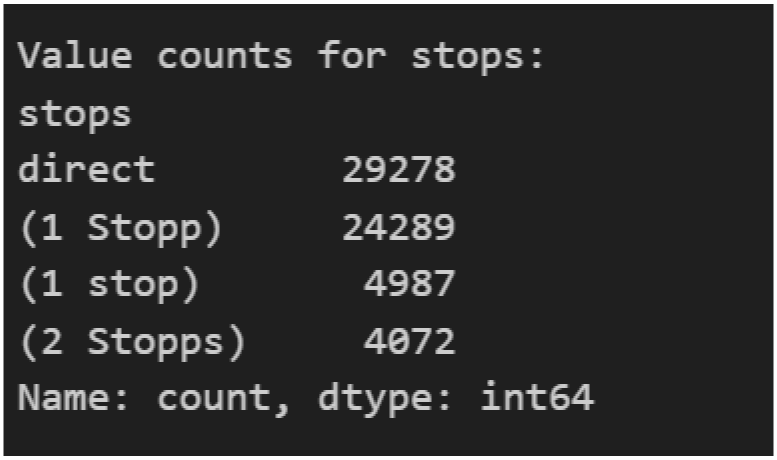
```
Value counts for departure_date_distance:
departure_date_distance
6 months     12672
6 weeks      11222
1 month      10092
1 week        9949
3 month       9748
2 weeks       7850
2 week        1093
Name: count, dtype: int64
```

- **Departure_time and arrival_time**: use both uhr, am and pm for the time



```
Value counts for departure_time:
departure_time
06:30 Uhr    1099
07:15 Uhr    1047
07:00 Uhr     860
08:30 Uhr     851
08:45 Uhr     834
             ...
9:50pm          4
01:50 Uhr       4
10:25pm         3
10:30pm         1
9:45pm          1
Name: count, Length: 404, dtype: int64
```

- **stops:**



```
Value counts for stops:
stops
direct          29278
(1 Stopp)       24289
(1 stop)         4987
(2 Stopps)       4072
Name: count, dtype: int64
```

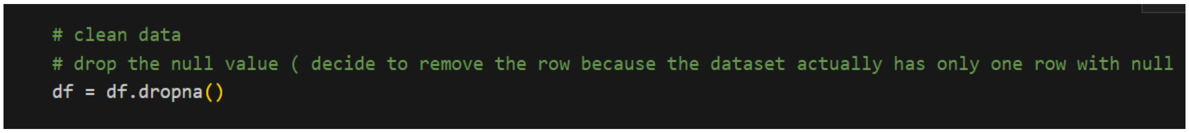
- **price**: some values have a comma, so we need to remove that then transform to the correct type.

2. Data Preprocessing

1. Data Cleaning**:**

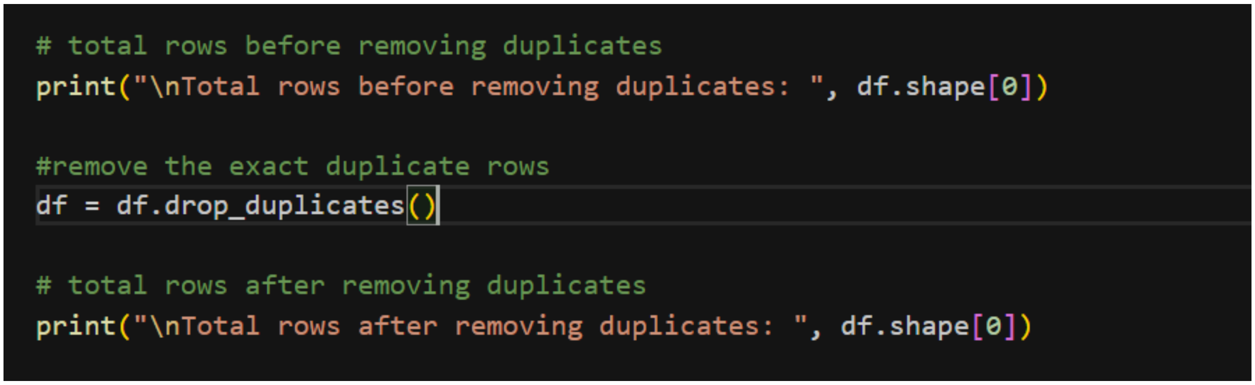
Thanks to the EDA, we have a general view of what we need to do in data cleaning. The method we used in data cleaning is following:

Handle missing value: We decided to drop the null value (entirely remove the row) because the dataset has only one row with null value and this row only has a price column, remaining columns are null.



```
# clean data
# drop the null value ( decide to remove the row because the dataset actually has only one row with null
df = df.dropna()
```

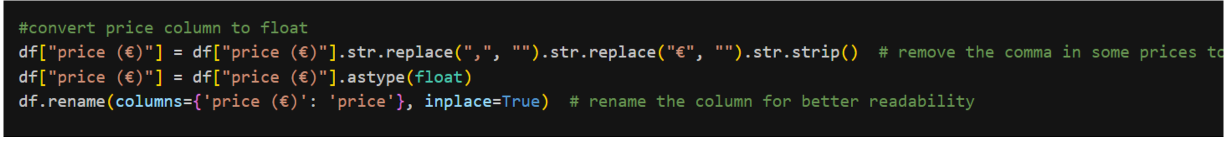
Removing duplicate: Decided to remove 804 exactly duplicated rows. Exact duplicates were removed to prevent bias in our analysis and ensure each data point represents a unique observation.



```
# total rows before removing duplicates
print("\nTotal rows before removing duplicates: ", df.shape[0])

#remove the exact duplicate rows
df = df.drop_duplicates()

# total rows after removing duplicates
print("\nTotal rows after removing duplicates: ", df.shape[0])
```

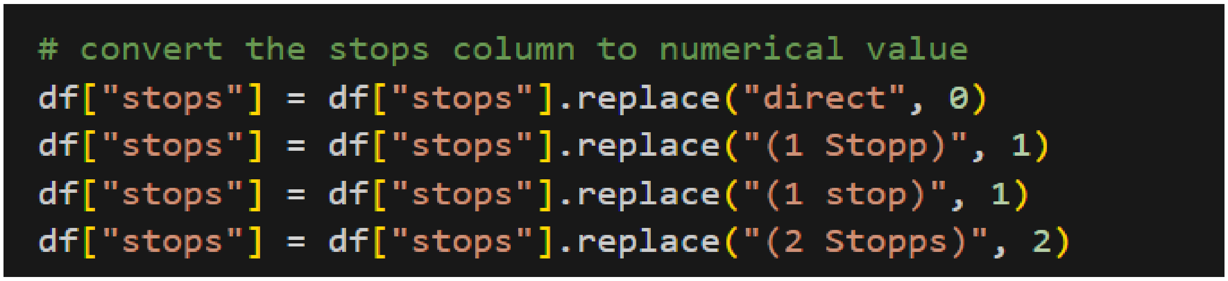
Handle inconsistencies:

Remove Commas in the 'price' Column: Eliminate any commas in the values of the 'price' column to ensure consistent numerical formatting and facilitate accurate numerical computations and analyses. The presence of commas in numerical data can cause errors during conversion to numeric types, which can lead to calculation or aggregation issues during data processing.



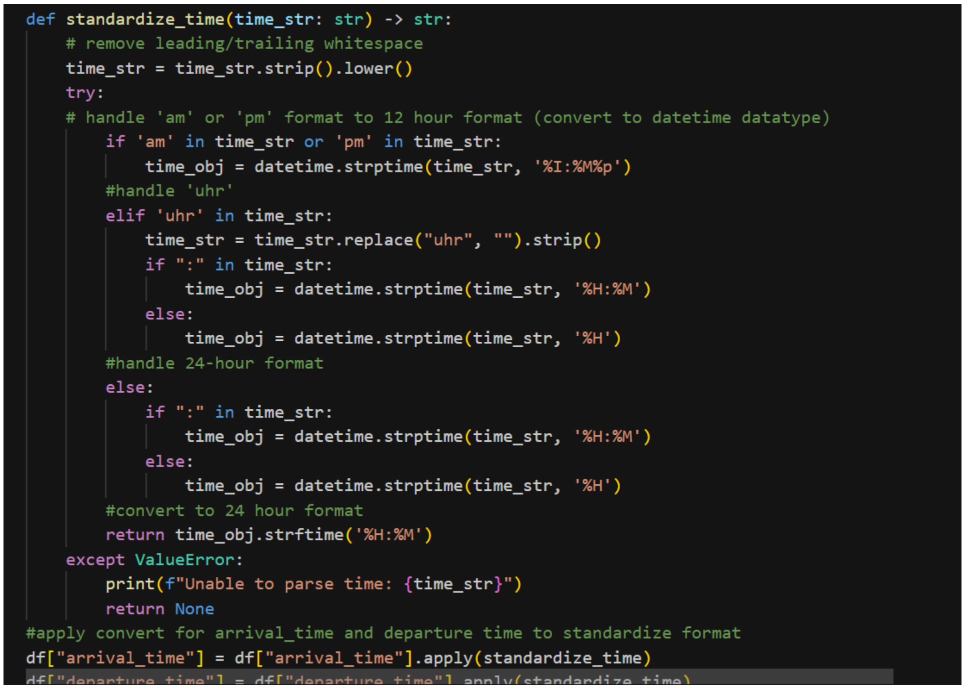
```
#convert price column to float
df["price (€)"] = df["price (€)"].str.replace(",", "").str.replace("€", "").str.strip()  # remove the comma in some prices tc
df["price (€)"] = df["price (€)"].astype(float)
df.rename(columns={'price (€)': 'price'}, inplace=True)  # rename the column for better readability
```

Standardize ‘stops’ columns: change ‘direct’ to 0; ‘1 Stopp’ and ‘1 Stop’ to 1 and (2 Stops) to 2. Standardizing categorical data in the 'stops' column to numerical values improves the usability of the data for machine learning models and allows for easier comparison, filtering, and interpretation.



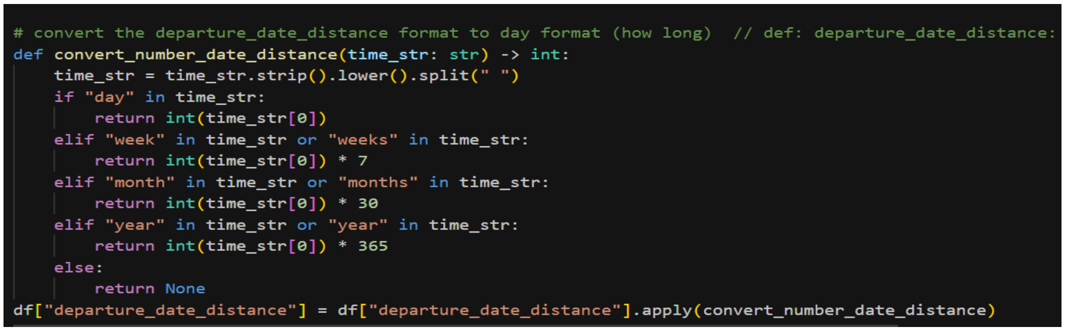
```
# convert the stops column to numerical value
df["stops"] = df["stops"].replace("direct", 0)
df["stops"] = df["stops"].replace("(1 Stopp)", 1)
df["stops"] = df["stops"].replace("(1 stop)", 1)
df["stops"] = df["stops"].replace("(2 Stopps)", 2)
```

Convert ‘departure_time’ and ‘arrival_time’ to 24-hour format. Converting time to a uniform 24-hour format avoids ambiguity and ensures consistency (before having a mix of am, pm and uhr), especially when performing time-based calculations or visualizations.



```
def standardize_time(time_str: str) -> str:
    # remove leading/trailing whitespace
    time_str = time_str.strip().lower()
    try:
    # handle 'am' or 'pm' format to 12 hour format (convert to datetime datatype)
        if 'am' in time_str or 'pm' in time_str:
            time_obj = datetime.strptime(time_str, '%I:%M%p')
        #handle 'uhr'
        elif 'uhr' in time_str:
            time_str = time_str.replace("uhr", "").strip()
            if ":" in time_str:
                time_obj = datetime.strptime(time_str, '%H:%M')
            else:
                time_obj = datetime.strptime(time_str, '%H')
        #handle 24-hour format
        else:
            if ":" in time_str:
                time_obj = datetime.strptime(time_str, '%H:%M')
            else:
                time_obj = datetime.strptime(time_str, '%H')
        #convert to 24 hour format
        return time_obj.strftime('%H:%M')
    except ValueError:
        print(f"Unable to parse time: {time_str}")
        return None
#apply convert for arrival_time and departure time to standardize format
df["arrival_time"] = df["arrival_time"].apply(standardize_time)
df["departure_time"] = df["departure_time"].apply(standardize_time)
```

In ‘departure_date_distance’, convert into a number of days instead of having a mix of weeks and months. This could help reduce the complexity and ensure uniformity.



```
# convert the departure_date_distance format to day format (how long)  // def: departure_date_distance:
def convert_number_date_distance(time_str: str) -> int:
    time_str = time_str.strip().lower().split(" ")
    if "day" in time_str:
        return int(time_str[0])
    elif "week" in time_str or "weeks" in time_str:
        return int(time_str[0]) * 7
    elif "month" in time_str or "months" in time_str:
        return int(time_str[0]) * 30
    elif "year" in time_str or "year" in time_str:
        return int(time_str[0]) * 365
    else:
        return None
df["departure_date_distance"] = df["departure_date_distance"].apply(convert_number_date_distance)
```

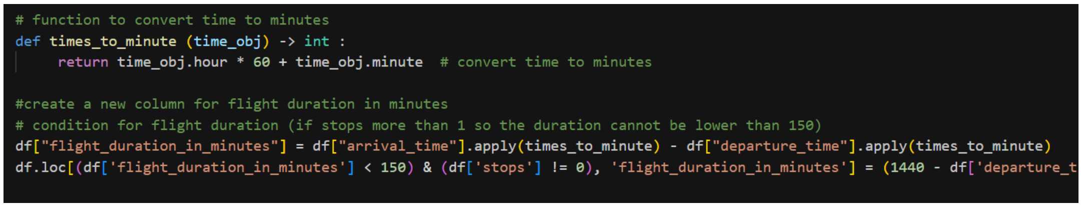
<u>Convert some column types to the type they should be:</u> ‘departure_date’, ‘scrape _ date’ to datetime datatype; ‘stops’ to integer; ‘arrival_time’ and ‘departure_time’ to datetime type.

2. Feature Extraction

After cleaning and preprocessing the data, we performed feature extraction to create new, potentially informative features which could affect flight prices and enhance our model's predictive power from the existing dataset. Here are the key features we extracted:

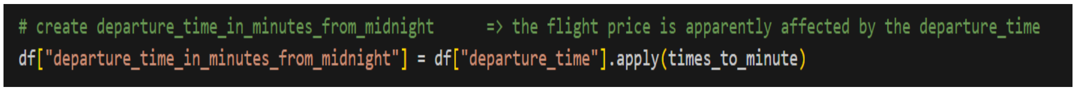
a) Flight Duration:

We created a new feature 'flight_duration_in_minutes' to capture the total flight time. We had faced some problems when we first calculated the flight duration, some flight durations are 5 minutes or just 10 minutes. We have researched and found that in Germany, the shortest duration for flight with more than 1 stop is 150 minutes. Hence, we added some conditions when creating this feature to handle the flights crossing midnight.



```
# function to convert time to minutes
def times_to_minute (time_obj) -> int :
    return time_obj.hour * 60 + time_obj.minute  # convert time to minutes

#create a new column for flight duration in minutes
# condition for flight duration (if stops more than 1 so the duration cannot be lower than 150)
df["flight_duration_in_minutes"] = df["arrival_time"].apply(times_to_minute) - df["departure_time"].apply(times_to_minute)
df.loc[(df['flight_duration_in_minutes'] < 150) & (df['stops'] != 0), 'flight_duration_in_minutes'] = (1440 - df['departure_t
```

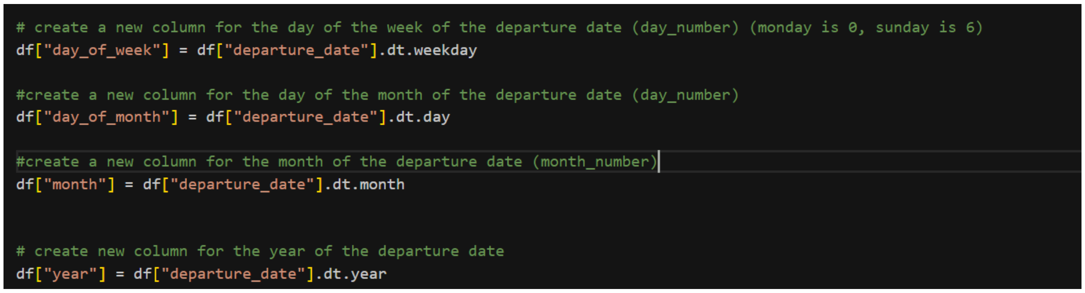
b) Departure Time In Minutes From Midnight:



```
# create departure_time_in_minutes_from_midnight      => the flight price is apparently affected by the departure_time
df["departure_time_in_minutes_from_midnight"] = df["departure_time"].apply(times_to_minute)
```

Departure time can affect flight prices, with peak hours potentially commanding higher prices.

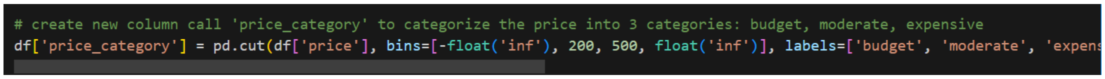
c) Day Of Week, Day of Month, Month and Year:



```
# create a new column for the day of the week of the departure date (day_number) (monday is 0, sunday is 6)
df["day_of_week"] = df["departure_date"].dt.weekday

#create a new column for the day of the month of the departure date (day_number)
df["day_of_month"] = df["departure_date"].dt.day

#create a new column for the month of the departure date (month_number)
df["month"] = df["departure_date"].dt.month


# create new column for the year of the departure date
df["year"] = df["departure_date"].dt.year
```

Flight prices often vary by day of the week, with weekends potentially being more expensive. Day of Month could capture any pricing patterns related to specific days of the month. Additionally, Seasonal variations in flight prices can be captured by the month of travel.

d) Price Category

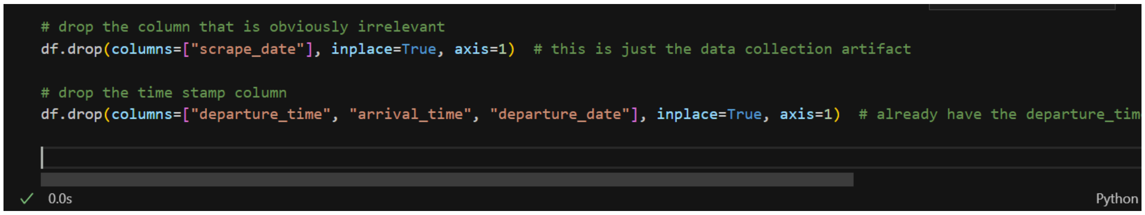


```
# create new column call 'price_category' to categorize the price into 3 categories: budget, moderate, expensive
df['price_category'] = pd.cut(df['price'], bins=[-float('inf'), 200, 500, float('inf')], labels=['budget', 'moderate', 'expens
```

Besides only predicting the flight prices, we also want to answer one more question "Which price category does this flight belong to". This categorization can be useful for classification. The customer can approximate the price range (each price category has already had the price range) for these flights in the future.

3.Drop unnecessary columns

We decided to drop some unnecessary columns: scrape_date which is just the data collection artifact, "departure_time", "arrival_time" and "departure_date" which already have the departure_time_in_minutes_from_midnight and flight_duration_in_minutes. Dropping these columns helps avoiding multicollinearity happening with the features extracted from these dropped columns.

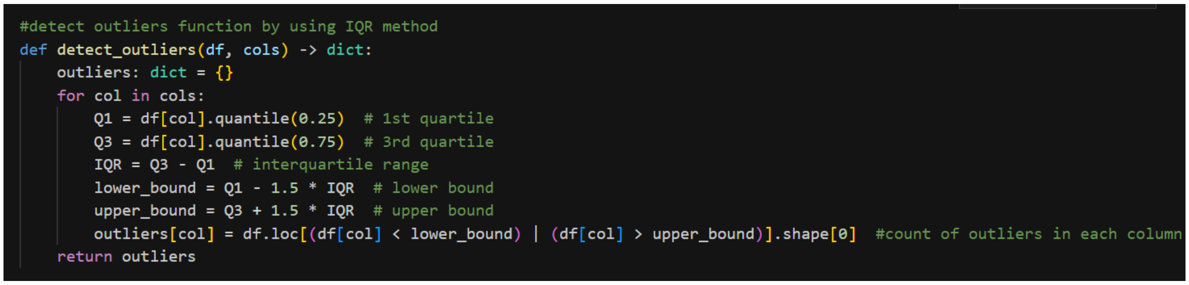


```
# drop the column that is obviously irrelevant
df.drop(columns=["scrape_date"], inplace=True, axis=1)  # this is just the data collection artifact

# drop the time stamp column
df.drop(columns=["departure_time", "arrival_time", "departure_date"], inplace=True, axis=1)  # already have the departure_tim
```

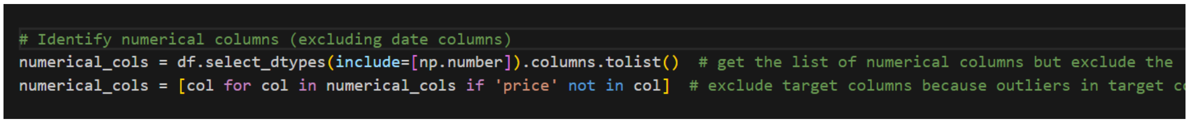
4.Handling Outliers

a) Detect Outliers:

We used the Interquartile Range (IQR) method to detect outliers, a robust technique that is less sensitive to extreme values compared to standard deviation-based methods (Leys et al., 2013). We applied this method to all numerical columns, excluding the target variable ('price') and any date-related columns.

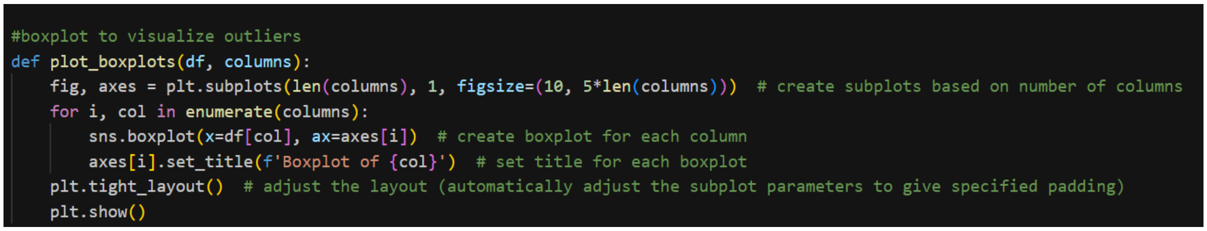


```
#detect outliers function by using IQR method
def detect_outliers(df, cols) -> dict:
    outliers: dict = {}
    for col in cols:
        Q1 = df[col].quantile(0.25)  # 1st quartile
        Q3 = df[col].quantile(0.75)  # 3rd quartile
        IQR = Q3 - Q1  # interquartile range
        lower_bound = Q1 - 1.5 * IQR  # lower bound
        upper_bound = Q3 + 1.5 * IQR  # upper bound
        outliers[col] = df.loc[(df[col] < lower_bound) | (df[col] > upper_bound)].shape[0]  #count of outliers in each column
    return outliers
```



```
# Identify numerical columns (excluding date columns)
numerical_cols = df.select_dtypes(include=[np.number]).columns.tolist()  # get the list of numerical columns but exclude the
numerical_cols = [col for col in numerical_cols if 'price' not in col]  # exclude target columns because outliers in target c
```

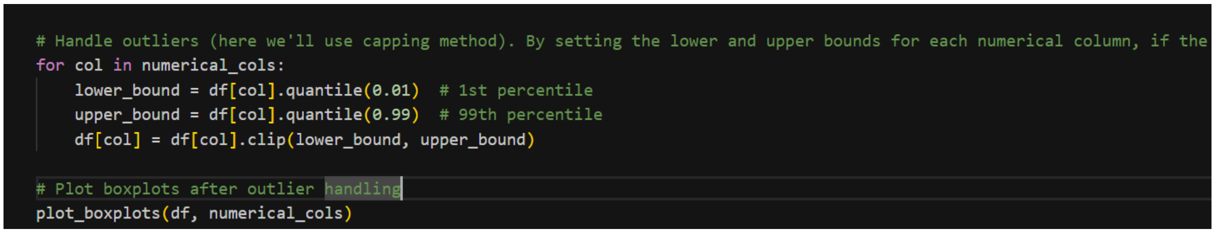
To better understand the distribution of our data and the presence of outliers, we created boxplots for each numerical column before and after outlier handling



```
#boxplot to visualize outliers
def plot_boxplots(df, columns):
    fig, axes = plt.subplots(len(columns), 1, figsize=(10, 5*len(columns)))  # create subplots based on number of columns
    for i, col in enumerate(columns):
        sns.boxplot(x=df[col], ax=axes[i])  # create boxplot for each column
        axes[i].set_title(f'Boxplot of {col}')  # set title for each boxplot
    plt.tight_layout()  # adjust the layout (automatically adjust the subplot parameters to give specified padding)
    plt.show()
```

b) Outliers Handling

For handling outliers, we employed a capping method, also known as winsorization (Ghosh and Vogt, 2012). In our case, we used the 1st and 99th percentiles as the lower and upper bounds respectively. This approach preserves the distribution of the data while mitigating the impact of extreme outliers (Kwak and Kim, 2017).



```
# Handle outliers (here we'll use capping method). By setting the lower and upper bounds for each numerical column, if the
for col in numerical_cols:
    lower_bound = df[col].quantile(0.01)  # 1st percentile
    upper_bound = df[col].quantile(0.99)  # 99th percentile
    df[col] = df[col].clip(lower_bound, upper_bound)

# Plot boxplots after outlier handling
plot_boxplots(df, numerical_cols)
```

5.Data Analysis and Visualization

We assumed that the dataset does not have a huge number of features (11 features), so we decided to analyze the influence of each feature on the target variable (price for regression and price category for classification).
Our data analysis aimed to uncover patterns and insights that would inform our model selection both price prediction (regression) and price category classification. We employed various visualization techniques to better understand the data and its underlying structures.

We had examined the distribution of flight prices using histograms, which revealed a right-skewed distribution so suggest the potential need for log transformation in our models. Box plots of prices by airline uncovered significant variations in pricing strategies across carriers. It indicated the importance of including an airline as a feature.

Scatter plots of price versus flight duration showed a positive correlation, but with considerable spread, hinting at the complex, non-linear nature of price determination. This insight suggested that non-linear models like Random Forests might be appropriate.

Time-based analyses, including line plots of average prices by month and day of week, revealed clear seasonal and weekly patterns. These cyclical trends pointed towards the potential effectiveness of time-series models or the need to engineer cyclical features for other model types.

Correlation heatmaps highlight strong relationships between certain features, which guide feature selection and indicate potential multicollinearity issues to address. The heatmap also suggested that ensemble methods might be effective in capturing complex feature interactions.

Violin plots of price distribution by number of stops showed distinct pricing patterns and the importance of numerical features. It suggested that tree-based models or linear regression might effectively capture these discrete influences on price.

We implemented some visualizations for the price category then we realized that which features affect the price will absolutely affect the price categories because price categories are created based on a specific range of the flight price.

These visualizations collectively informed our decision to explore both linear models (for interpretability) and more complex ensemble methods (to capture non-linear relationships and interactions) in our modeling phase.

6.Split the data, Scaling and Encoding:

After our data analysis and visualization phase, we proceeded with preparing our data for modeling in the following sequence: data splitting, feature scaling, and categorical encoding. This sequence is crucial for maintaining the integrity of our model evaluation and preventing data leakage.

Splitting the data before scaling and encoding prevents information from the test set from influencing the preprocessing of the training set, which could lead to overly optimistic model performance estimates (Hastie, Tibshirani and Friedman, 2009).

We applied MinMax scaling to our numerical features because our visualization of numerical features revealed non-normal distributions. It made MinMax scaling more appropriate than standardization. MinMax scaling is less affected by outliers and preserves zero values, which can be important in certain machine learning algorithms (Jayalakshmi and Santhakumaran, 2011).

We used One-Hot Encoding instead of Label Encoding for our categorical variables. As our observation, all the remaining categorical data is nominal so One-Hot Encoding is appropriate. This method creates binary columns for each category, allowing the model to treat each category independently (Potdar, S. Pardawala and Pai, 2017).

This sequence ensures that our test set remains truly unseen during the preprocessing steps and maintains the validity of our model evaluation.

7.Feature Selection

Based on the insights we gained from the Analysis and Visualization, we had already had the list of the features used to train the model. However, to make sure that our decision is correct or not, we implemented feature selection techniques to verify that. We used a stepwise regression method for regression and embedded method (random forest classifier importance) for classification. We have discussed more in X. Appendix (I. Feature Selection).

V. Machine learning model selection

The selection of appropriate machine learning models for our flight price prediction project was guided by several key criteria, tailored to the nature of our dataset and the specific requirements of our problem.

Selection Criteria

1. Ability to handle non-linear relationships
2. Robustness to outliers and noise
3. Capability to process high-dimensional data
4. Provision of feature importance rankings
5. Performance on similar time-series and pricing problems
6. Ability to capture complex interactions between features

Algorithms Considered

Based on these criteria, we considered the following algorithms:

1. Linear Regression
2. Support Vector Regression (SVR)
3. Random Forest Regressor
4. Gradient Boosting Regressor
5. Random Forest Classifier (for price categorization)

Rationale and Alignment with Problem Framing

After careful consideration, we selected Random Forest Regressor, Gradient Boosting Regressor, and Random Forest Classifier for our final models. The rationale behind these choices is as follows:

1. **Random Forest Regressor:** Our dataset exhibited complex, non-linear relationships between features and flight prices, as evidenced by the scatter plots in our exploratory data analysis. Random Forests excel at capturing such non-linear patterns (Breiman, 2001). Additionally, our feature space was high-dimensional after one-hot encoding categorical variables, and Random Forests handle high dimensionality effectively without overfitting (Biau and Scornet, 2016).
2. **Gradient Boosting Regressor:** This model was chosen for its typically high predictive performance on diverse datasets, including those with temporal components like our

flight booking dates (Zhang and Haghani, 2015). Gradient Boosting is particularly adept at capturing subtle interactions between features, which we suspected might be important in flight pricing based on our correlation analysis.

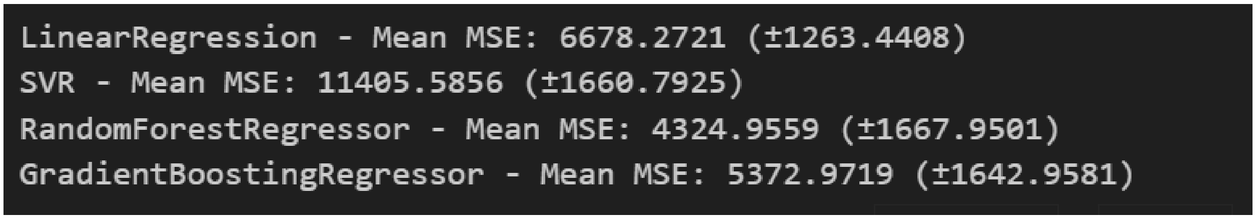
3. **Random Forest Classifier:** For our price categorization task, Random Forest Classifier was selected due to its ability to handle multi-class problems effectively. Our price categories (budget, moderate, expensive) were not ordinal, making tree-based methods more suitable than ordinal regression (Kotsiantis, 2007).

These choices align with our problem framing in several ways:

- They can capture the complex, possibly non-linear relationships in flight pricing.
- They provide feature importance rankings, crucial for understanding key price determinants.
- They are robust to outliers, which we observed in our price and flight duration data.

Preliminary Tests and Comparisons

To validate our choices, we conducted preliminary tests using 5-fold cross-validation. Results showed that Random Forest and Gradient Boosting consistently outperformed linear models and SVR on our dataset and confirmed our initial assessments. These preliminary results supported our model choices and demonstrated these models superior performance on our specific dataset and problem framing.



```
LinearRegression - Mean MSE: 6678.2721 (±1263.4408)
SVR - Mean MSE: 11405.5856 (±1660.7925)
RandomForestRegressor - Mean MSE: 4324.9559 (±1667.9501)
GradientBoostingRegressor - Mean MSE: 5372.9719 (±1642.9581)
```

VI. Implementation

1. Technical Implementation

Key libraries used:

- pandas: For data loading, cleaning, and preprocessing
- datetime: for datetime type
- NumPy: For numerical operations
- scikit-learn: For implementing machine learning models, preprocessing techniques, and model evaluation
- matplotlib and seaborn: For data visualization
- statsmodels: For statistical computations, particularly in feature selection

The project was challenging due to the need for extensive data wrangling, particularly in handling various time formats and categorizing price ranges. The high dimensionality of the feature space after one-hot encoding also required careful consideration during model training and evaluation.

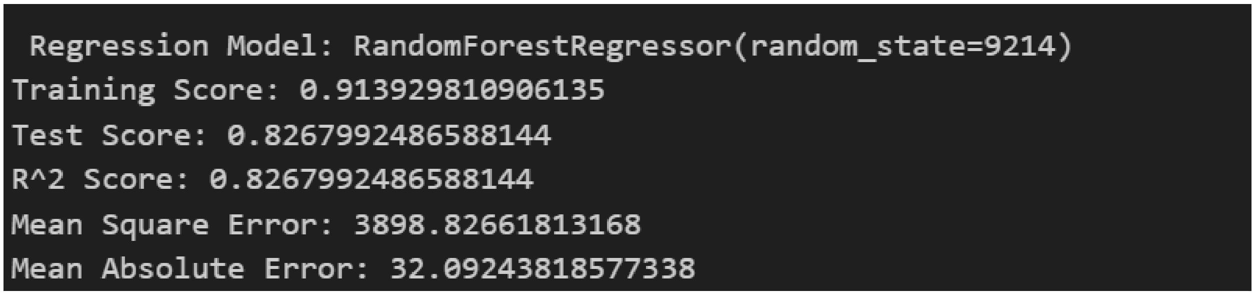
We referenced the scikit-learn documentation (Pedregosa et al., 2011) extensively for implementation details of machine learning algorithms and preprocessing techniques.

2.Implementation Evaluation

To evaluate the effectiveness of our implementation, we used various performance metrics (for regression and classification) and cross validation procedures to ensure a comprehensive assessment.

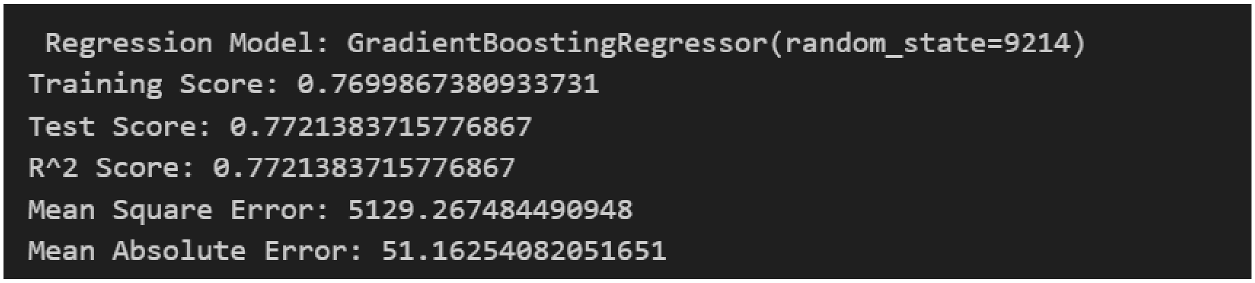
For the regression task:

1. Random Forest Regressor:



```
 Regression Model: RandomForestRegressor(random_state=9214)
Training Score: 0.913929810906135
Test Score: 0.8267992486588144
R^2 Score: 0.8267992486588144
Mean Square Error: 3898.82661813168
Mean Absolute Error: 32.09243818577338
```

2. Gradient Boosting Regressor:

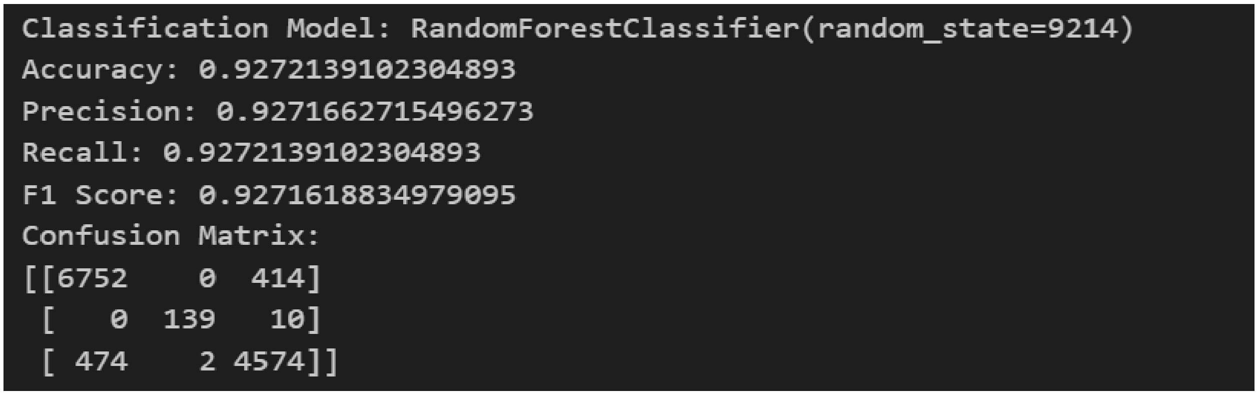


```
 Regression Model: GradientBoostingRegressor(random_state=9214)
Training Score: 0.7699867380933731
Test Score: 0.7721383715776867
R^2 Score: 0.7721383715776867
Mean Square Error: 5129.267484490948
Mean Absolute Error: 51.16254082051651
```

The Random Forest Regressor outperformed the Gradient Boosting Regressor, explaining approximately 82.68% of the variance in flight prices on the test set. This high R² score indicates that our model captures a significant portion of the factors influencing price variations. However, we observed some overfitting in the Random Forest model, as evidenced by the difference between training (0.9139) and test (0.8268) scores.

For the classification task:

3. Random Forest Classifier:



```
Classification Model: RandomForestClassifier(random_state=9214)
Accuracy: 0.9272139102304893
Precision: 0.9271662715496273
Recall: 0.9272139102304893
F1 Score: 0.9271618834979095
Confusion Matrix:
[[6752    0  414]
 [   0  139   10]
 [ 474    2 4574]]
```

The confusion matrix for the Random Forest Classifier: [[6752 0 414] [0 139 10] [474 2 4574]] and consistent scores across all metrics reveals that the model performs exceptionally well in identifying budget and expensive flights, with some misclassifications between moderate and expensive categories.

Our implementation effectively addresses the problem of flight price prediction and categorization in the German air travel market. The high accuracy of the classification model (92.72%) provides valuable insights for travelers looking to understand price ranges, while the regression model offers more precise estimates with a reasonable error margin (Mean Absolute Error of €32.09 for Random Forest Regressor).

To ensure robust performance estimates, we conducted 5-fold cross-validation. We will discuss more details about cross validation in X. Appendix (II. Cross-validation).

These results suggest that our models can be valuable tools for travelers seeking to understand and anticipate price fluctuations. However, the discrepancy between training and test scores in the regression models indicates room for improvement.

VII. Hyperparameter Tuning

We assume that there still is room for improvement of Random Forest Regressor and Gradient Boosting Regressor. We implemented some hyperparameter tuning techniques to improve these two models. We discussed further in the X. Appendix (III. Hyperparameter Tuning).

VIII. Conclusion

This project successfully developed machine learning models to predict and categorize flight prices in the German domestic air travel market, which effectively addresses the initial problem of providing travelers with accurate price forecasts and insights. Our Random Forest Regressor achieved a R^2 score of 0.8410 after hyperparameter tuning and explained a significant portion of price variability. The Random Forest Classifier demonstrated high accuracy (0.9272) in categorizing flights into price ranges to give valuable insights for budget-conscious travelers.

Key findings include the importance of temporal features such as duration and day of the week in predicting prices, as well as the significant impact of airlines on pricing.

Our solution met the objectives of providing both precise price predictions and broader price categorizations for different user needs. The models' performance, particularly after hyperparameter tuning, demonstrates their potential as practical tools for travelers and the aviation industry.

The implications of our results extend beyond individual travelers. Airlines and travel agencies could leverage similar models to optimize pricing strategies and improve customer satisfaction. The methodology developed here could be adapted to other markets or transportation sectors, contributing to broader applications in travel economics.

This project contributes to the field of travel analytics by demonstrating the effectiveness of machine learning in decoding complex pricing structures in air travel. It provides a foundation for more transparent and informed decision-making in flight bookings and leads to more efficient market dynamics and improved customer experiences in the aviation industry.

IX. Bibliography

1, Leys, C., Ley, C., Klein, O., Bernard, P. and Licata, L., 2013. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. Journal of Experimental Social Psychology, 49(4), pp.764-766.

2, Kwak, S.K. and Kim, J.H., 2017. Statistical data preparation: management of missing values and outliers. Korean journal of anesthesiology, 70(4), p.407.

3, Ghosh, D. and Vogt, A., 2012. Outliers: An evaluation of methodologies. In Joint statistical meetings (pp. 3455-3460). San Diego, CA: American Statistical Association.

4, Hastie, T., Tibshirani, R. and Friedman, J., 2009. The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.

5, Jayalakshmi, T. and Santhakumaran, A., 2011. Statistical normalization and back propagation for classification. International Journal of Computer Theory and Engineering, 3(1), pp.1793-8201.

6, Potdar, K., Pardawala, T.S. and Pai, C.D., 2017. A comparative study of categorical variable encoding techniques for neural network classifiers. International journal of computer applications, 175(4), pp.7-9.

7, Biau, G. and Scornet, E., 2016. A random forest guided tour. Test, 25(2), pp.197-227.

8, Breiman, L., 2001. Random forests. Machine learning, 45(1), pp.5-32.

9, Kotsiantis, S.B., 2007. Supervised machine learning: A review of classification techniques. Informatica, 31(3), pp.249-268.

10, Zhang, Y. and Haghani, A., 2015. A gradient boosting method to improve travel time prediction. Transportation Research Part C: Emerging Technologies, 58, pp.308-324.

11, Bergstra, J. and Bengio, Y., 2012. Random search for hyper-parameter optimization. Journal of machine learning research, 13(2).

12, Bergstra, J., Yamins, D. and Cox, D.D., 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In International conference on machine learning (pp. 115-123). PMLR.

13, Hyndman, R.J. and Athanasopoulos, G., 2018. Forecasting: principles and practice. OTexts.

14, McKinney, W., 2012. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. O'Reilly Media, Inc.

15, Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), pp.2825-2830.

16, VanderPlas, J., 2016. Python data science handbook: Essential tools for working with data. O'Reilly Media, Inc.

17, Chandrikasai, 2023. Data preprocessing: Splitting, Scaling and Encoding. Medium, https://medium.com/@chandrikasai9997/data-preprocessing-splitting-scaling-and-encoding-d3505a0d1892.

18, Nalcin, S., 2022. StandardScaler vs. MinMaxScaler vs. RobustScaler: Which one to use for your next ML project. Medium, https://medium.com/@onersarpnalcin/standardscaler-vs-minmaxscaler-vs-robustscaler-which-one-to-use-for-your-next-ml-project-ae5b44f571b9

19, Hayes, A., 2022. Stepwise Regression: Definition, Uses, Example, and Limitations. Investopedia, https://www.investopedia.com/terms/s/stepwise-regression.asp#:~:text=Stepwise%20regression%20is%20the%20step,statistical%20significance%20after%20each%20iteration

20, Friedman, J.H., 2001. Greedy function approximation: a gradient boosting machine. Annals of statistics, pp.1189-1232.

21, Guide To Data Cleaning: Definition, Benefits, Components, And How To Clean Your Data. Tableau, https://www.tableau.com/learn/articles/what-is-data-cleaning

X. Appendix

I. Feature selection

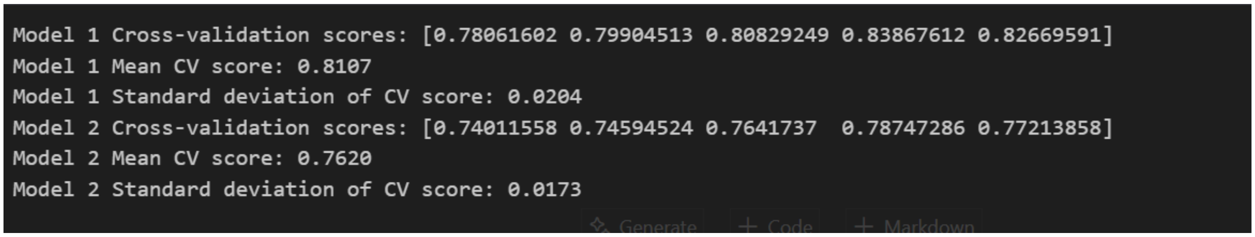
While our initial data analysis and visualization provided insights into potentially relevant features, we employed statistical methods to refine our feature selection. This step is crucial for improving model performance, reducing overfitting, and enhancing interpretability (Guyon and Elisseeff, 2003).
For our regression model, we implemented stepwise regression, a method that iteratively adds or removes features based on their statistical significance (James et al., 2013). This approach helped us identify the most influential predictors of flight prices.

For our classification model, we employed a Random Forest Importance technique, an embedded method that leverages the feature importance scores inherent to Random Forest algorithms (Breiman, 2001). To optimize feature selection, we modified our approach using the "elbow method", which identifies the point of diminishing returns in feature importance (Thorndike, 1953).

These methods allowed us to systematically select features that contribute most significantly to our predictive models, balancing model complexity with predictive power.

II. Cross-validation

Random Forest Regressor (model 1) and Gradient Boosting Regressor (model 2):



```
Model 1 Cross-validation scores: [0.78061602 0.79904513 0.80829249 0.83867612 0.82669591]
Model 1 Mean CV score: 0.8107
Model 1 Standard deviation of CV score: 0.0204
Model 2 Cross-validation scores: [0.74011558 0.74594524 0.7641737  0.78747286 0.77213858]
Model 2 Mean CV score: 0.7620
Model 2 Standard deviation of CV score: 0.0173
```

Random Forest Classifier:

- Mean CV Accuracy: 0.9213
- Mean CV Precision: 0.9212
- Mean CV Recall: 0.9213
- Mean CV F1 Score: 0.9212

The cross-validation results confirm the superior performance of the Random Forest models for both regression and classification tasks. The low standard deviations in the CV scores indicate consistent performance across different subsets of the data.

An unexpected outcome was the relatively high performance of the Random Forest Classifier compared to the regression models. This suggests that our price categorization task might be more tractable than precise price prediction, possibly due to the discrete nature of the categories smoothing out some of the price variability.

The 5-fold cross-validation results offer several valuable insights into our models' performance:

1. Model Ranking: The Random Forest models consistently outperform the Gradient Boosting Regressor. For regression, the Random Forest Regressor's mean CV score of 0.8107 is significantly higher than the Gradient Boosting Regressor 0.7620, indicating superior predictive power for flight price estimation.

2. Regression vs. Classification Performance: Interestingly, the Random Forest Classifier shows remarkably high performance across all metrics (accuracy, precision, recall, and F1 score all around 0.92). This suggests that categorizing flight prices into discrete ranges is more achievable than predicting exact prices, possibly due to the inherent variability in flight pricing being smoothed out in broader categories.

3. Model Stability: The low standard deviations in CV scores (0.0204 for Random Forest Regressor, 0.0173 for Gradient Boosting Regressor) indicate consistent performance across different data subsets. This suggests that both models, especially the Random Forest Regressor, are robust and likely to generalize well to unseen data.

4. Classification Consistency: The Random Forest Classifier's nearly identical scores across different metrics (all ~0.92) suggest balanced performance across different price categories. This indicates that the model is equally effective at identifying flights in all price ranges, without bias towards any category.

III. Hyperparameter tuning

To address the slight overfitting observed in our initial models, particularly the Random Forest Regressor, we implemented hyperparameter tuning using cross-validation techniques. This process aimed to optimize model performance and generalization (Bergstra and Bengio, 2012).

For the Random Forest Regressor, we employed RandomizedSearchCV, which efficiently samples from the parameter space (Bergstra et al., 2013). For the Gradient Boosting Regressor, we utilized GridSearchCV for a more exhaustive search.

The tuning process yielded improved results:

1. Tuned Random Forest Regressor:
   - R^2 Score: 0.8410 (improved from 0.8268)
   - Mean Squared Error: 3579.84 (reduced from 3898.83)

- Mean Absolute Error: 35.29 (slightly increased from 32.09)
- Training Score: 0.8581 (reduced from 0.9139, indicating less overfitting).

  Cross-validation results:

- Mean CV score: 0.8332 (improved from 0.8107)
- Standard deviation of CV score: 0.0199 (slightly reduced from 0.0204)

2. Tuned Gradient Boosting Regressor:
   - R^2 Score: 0.8026 (improved from 0.7721)
   - Mean Squared Error: 4443.10 (reduced from 5129.27)
   - Mean Absolute Error: 42.76 (reduced from 51.16)
   - Training Score: 0.8182 (improved from 0.7700, indicating better fit).

     Cross-validation results:

   - Mean CV score: 0.7925 (improved from 0.7620)
   - Standard deviation of CV score: 0.0170 (slightly reduced from 0.0173)

The tuning process successfully mitigated overfitting in the Random Forest Regressor, as evidenced by the reduced gap between training and test scores. Both models showed improved performance across multiple metrics, with the Random Forest Regressor maintaining its edge over the Gradient Boosting Regressor.

It's worth noting that due to computational constraints, we limited our parameter search space. We believe that with a more robust computer, we could probably run more complex tuning to gain better results.