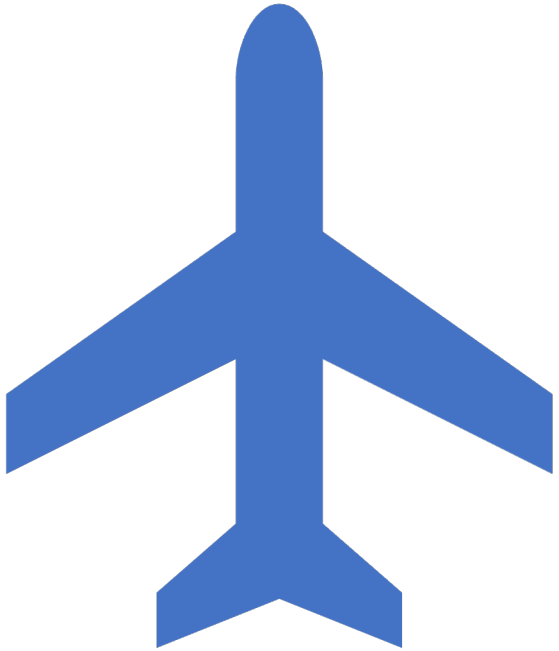
Dynamic Pricing Model for Airlines

- Team 2:
  - Abhishek Ramchandani
  - Devansh Thard

Introduction

- In the U.S. airline industry, dynamic pricing is a crucial strategy that leverages advanced machine learning algorithms to adjust ticket prices in real time based on demand, competition, and market trends. This approach integrates data from various sources like historical sales, search frequency, and passenger booking patterns. By using predictive models such as regression analysis and machine learning techniques like random forests, airlines can accurately forecast demand and optimize both load factors and profitability. This shift toward data-driven pricing reflects the industry's adaptation to the digital age, where quick analysis and action on large volumes of data are essential for competitive advantage.



Goals

- Our project aims to create a dynamic pricing model for the U.S. airline industry by analyzing a vast dataset. We seek to identify the key factors influencing flight pricing and build a predictive model for accurate ticket price forecasting. By leveraging machine learning algorithms, we aim to offer airlines actionable insights for real-time pricing adjustments, enhancing profitability and market agility. Our goal is to develop a model that drives revenue growth for airlines while ensuring fair and responsive pricing for consumers.

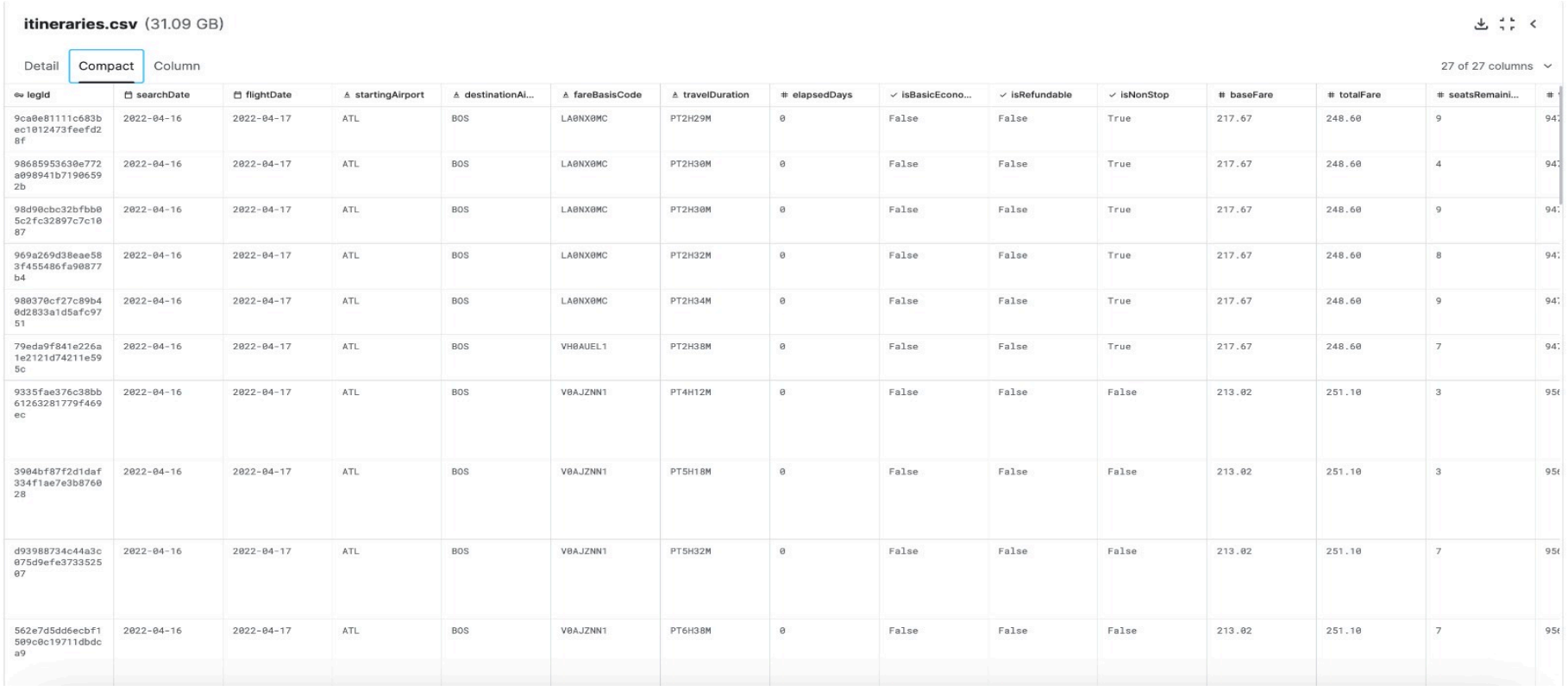
What are we trying to do?

- Datapoints we are dealing with
- We wish to correlate features with the price to see which features are strongly related to price variation.
    - Pearson's correlation coefficient for Linear Model.
    - Spearman's Rank correlation for the advanced models
- Build new features based on time-series related data.
- Find and execute suitable models that return optimal results for our testing and training data.

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

What do we have so far?

- A Feature Rich dataset that can perform and prove to be a good resource if refined further, as we intended to do.
  - We reduced dimensionality of the data to ensure model accuracy alongside encoding all data using label & frequency encoding and z-scale normalization.



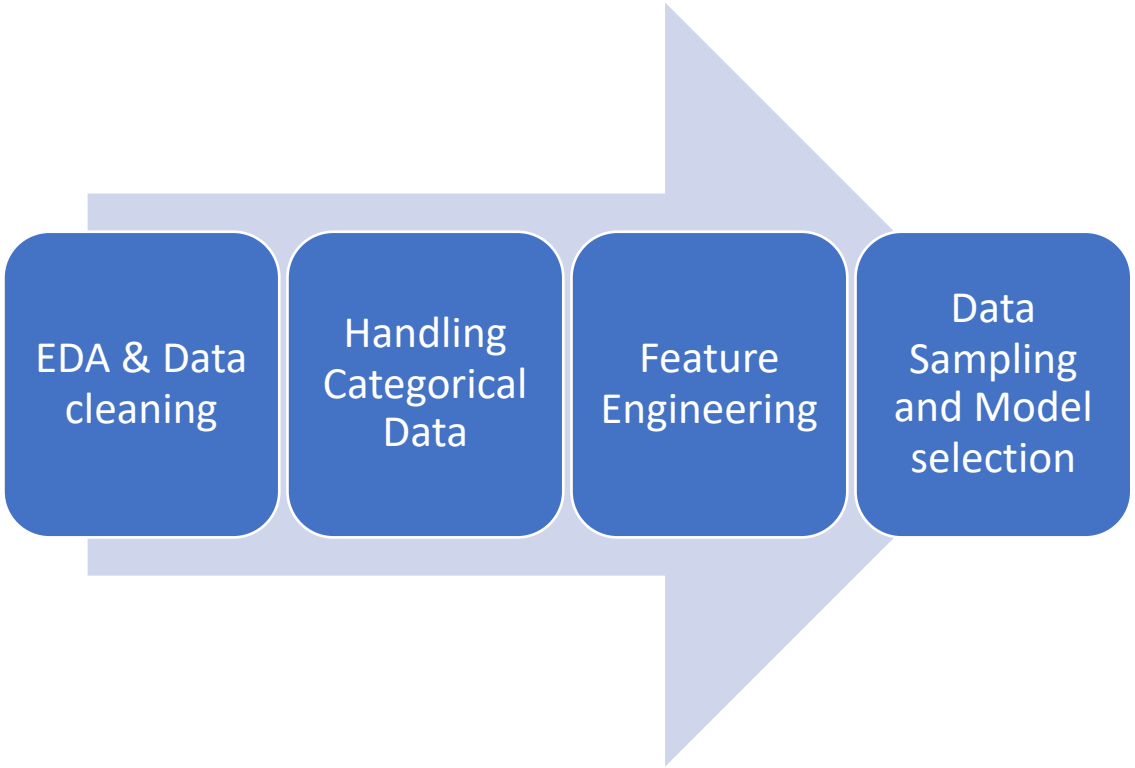
itineraries.csv (31.09 GB)

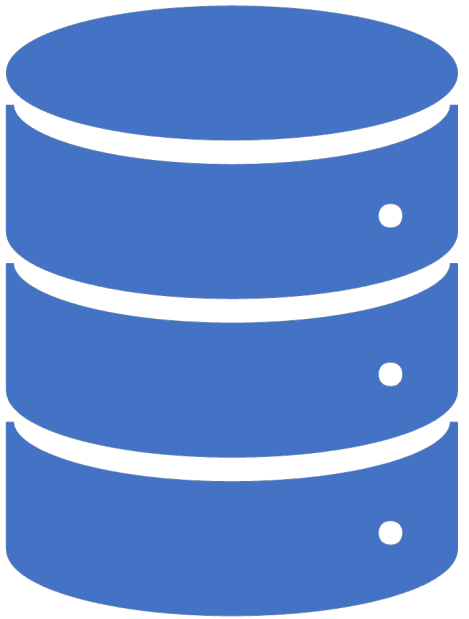
Detail | Compact | Column

27 of 27 columns

legId	searchDate	flightDate	startingAirport	destinationAi...	fareBasisCode	travelDuration	elapsedDays	isBasicEcono...	isRefundable	isNonStop	baseFare	totalFare	seatsRemaini...
9ca0e81111c683bec1012473feefd28f	2022-04-16	2022-04-17	ATL	BOS	LA0NX0MC	PT2H29M	0	False	False	True	217.67	248.60	9
98685953630e772a898941b71906592b	2022-04-16	2022-04-17	ATL	BOS	LA0NX0MC	PT2H30M	0	False	False	True	217.67	248.60	4
98d90cbc32bfbb05c2fc32897c7c1087	2022-04-16	2022-04-17	ATL	BOS	LA0NX0MC	PT2H30M	0	False	False	True	217.67	248.60	9
969a269d38eae583f455486fa90877b4	2022-04-16	2022-04-17	ATL	BOS	LA0NX0MC	PT2H32M	0	False	False	True	217.67	248.60	8
980370cf27c89b40d2833a1d5afc9751	2022-04-16	2022-04-17	ATL	BOS	LA0NX0MC	PT2H34M	0	False	False	True	217.67	248.60	9
79eda9f841e226a1e2121d74211e595c	2022-04-16	2022-04-17	ATL	BOS	VH0AUEL1	PT2H38M	0	False	False	True	217.67	248.60	7
9335fae376c38bb61263281779f469ec	2022-04-16	2022-04-17	ATL	BOS	V0AJZNN1	PT4H12M	0	False	False	False	213.02	251.10	3
3904bf87f2d1daf334f1ae7e3b876028	2022-04-16	2022-04-17	ATL	BOS	V0AJZNN1	PT5H18M	0	False	False	False	213.02	251.10	3
d93988734c44a3c075d9efe373352507	2022-04-16	2022-04-17	ATL	BOS	V0AJZNN1	PT5H32M	0	False	False	False	213.02	251.10	7
562e7d5dd6ecbf1509c0c19711dbdca9	2022-04-16	2022-04-17	ATL	BOS	V0AJZNN1	PT6H38M	0	False	False	False	213.02	251.10	7

How do we go on about it?





Data Cleaning Methods Used

Handling the missing values by :

Imputation: Filling in missing values using statistical methods.

Deletion: Removing rows or columns with missing values if they are deemed insignificant or cannot be reliably imputed.

Normalizing Fare Amounts

Z-score normalization: Standardizing fare values to have a mean of 0 and a standard deviation of 1.

Data Cleaning

- Data cleaning is the process of identifying and rectifying errors, inconsistencies, and inaccuracies within a dataset to ensure its integrity and usefulness for analysis. This involves handling missing or erroneous values, removing duplicate entries, standardizing data formats, and addressing outliers that could skew results.

```
egmentsCabinCode'].unique().compute
```

```
                                coach
                          coach||coach
                coach||coach||coach||coach
                        coach||coach||coach
                                first
                                  ...
               premium coach||first||coach
               premium coach||first||first
premium coach||coach||coach||first
               coach||premium coach||first
               first||premium coach||coach
segmentsCabinCode, Length: 71, dty
```

travelDuration

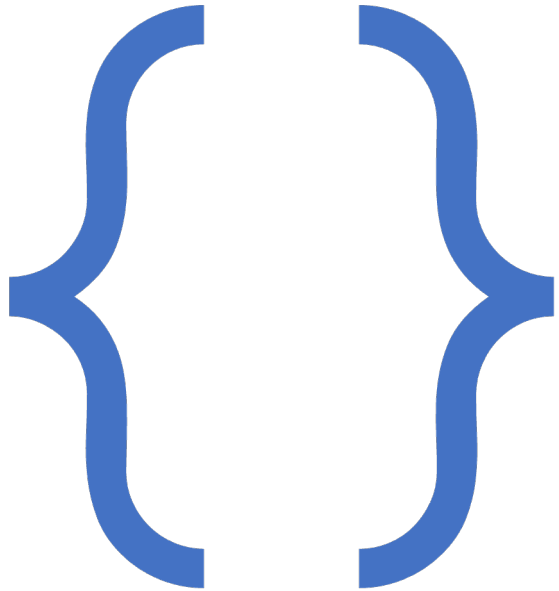
PT2H29M

```
_finalE['DepartureDate:month'] = df_finalE['DepartureDate'].dt.mon
_finalE['DepartureDate:day'] = df_finalE['DepartureDate'].dt.day
_finalE['DepartureDate:day_of_week'] = df_finalE['DepartureDate'].
_finalE[['DepartureDate:month','DepartureDate:day','DepartureDate:
```

DepartureDate:month	DepartureDate:day	DepartureDate:day_of_week
4	17	6
4	17	6
4	17	6
4	17	6
4	17	6

Handeling Categorical Variables

- Handling categorical variables involves converting qualitative data into a numerical format that can be utilized in statistical models and machine learning algorithms. This process, known as encoding, assigns a unique numerical value to each category within a variable. Common techniques include one-hot encoding, which creates binary columns for each category, and label encoding, which assigns integers to categories based on their frequency or some other criteria.



Handling Categorical Variables

Faced challenges encountered with one-hot encoding, leading to increased dimensionality.

Transitioned to label & frequency encoding technique for efficient resource management for categorical variables.

Made use of frequency encoding usage to make categorical data suitable for machine learning models by encoding them based on their frequency of occurrence.

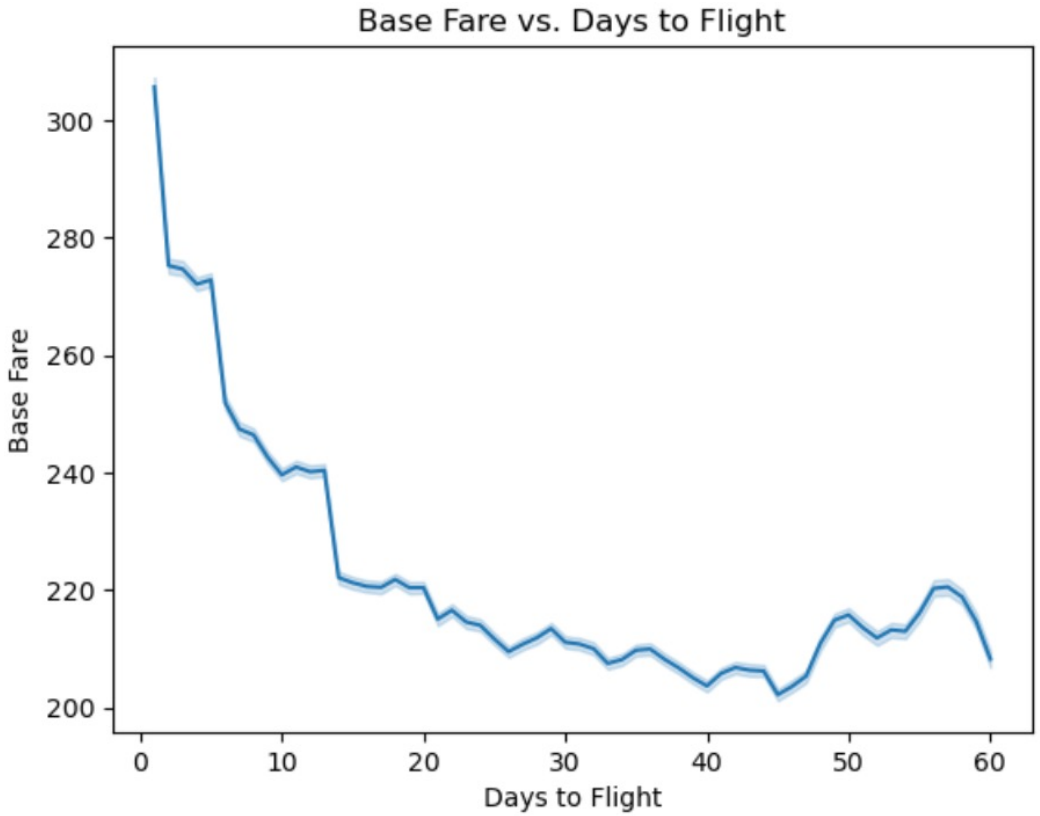
Feature Engineering

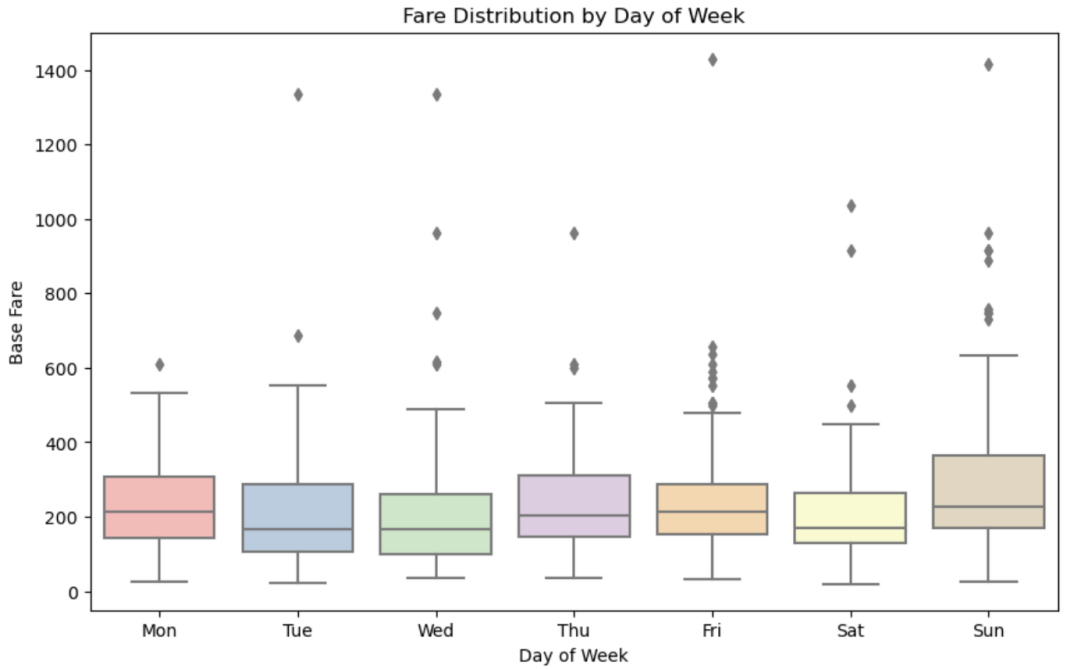
- Feature engineering is about selecting or creating the most relevant variables in a dataset to improve machine learning model performance. It involves identifying key predictors, crafting new features, and transforming variables for better prediction accuracy. Effective feature engineering is essential for extracting valuable patterns and relationships from data, leading to more accurate and interpretable models.

EDA:

As we can we see, the closer you are to the date of the flight the more expensive the flight price will be

Base Fare tends to be higher on Sundays, midweek and (surprisingly) Mondays indicating people like travelling during these days of the week

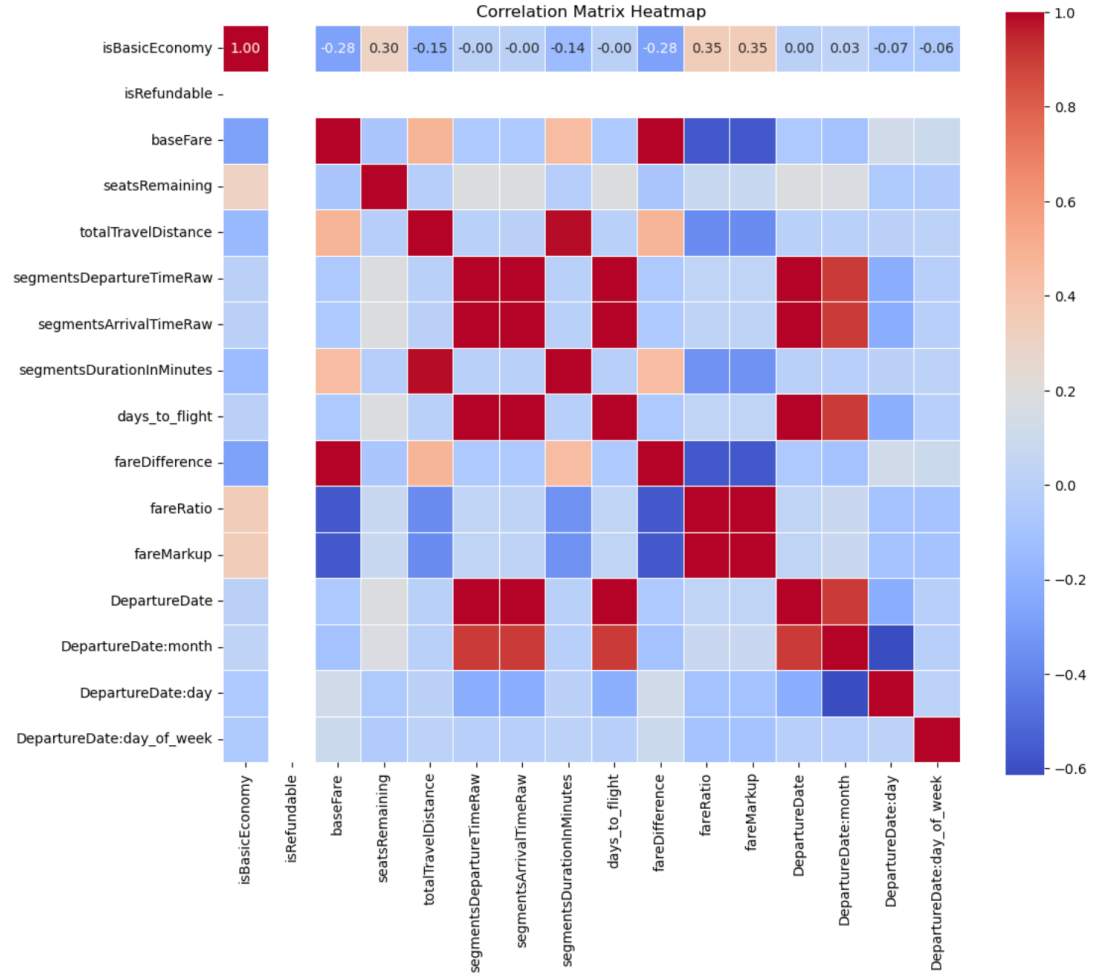




EDA:

As we can we see, the closer you are to the date of the flight the more expensive the flight price will be

Base Fare tends to be higher on Sundays, midweek and (surprisingly) Mondays indicating people like travelling during these days of the week



Feature Engineering

- Detailed description of newly engineered features:
  - daysToFlight: Capturing time duration between search date and departure date
  - fareDifference, fareRatio, fareMarkup: Calculating variations and markup in fare prices.
  - DepartureDate: month, day & day_of_week: Extracting temporal features for analysis and store datetime information.
  - Encoding categorical variables related to airport, airline, equipment, and cabin to numerical format.
  - Extracting hour and minute information for departure and arrival times.
  - These emphasis on the role of feature engineering in enhancing model performance and capturing additional nuances in pricing data.

Data Sampeling and Model Selection

- Data sampling involves selecting a representative subset from a larger dataset to train and evaluate machine learning models efficiently while mitigating biases. Model selection is the process of comparing and evaluating various algorithms' performance metrics, such as accuracy or error rates, to choose the most suitable one for a specific predictive task. These steps are essential for building robust machine learning models that generalize well to unseen data and effectively address the problem at hand.

Data Sampling and Model Selection

- The data sampling process involved selecting a representative subset of 5 million rows from the dataset to manage data size while preserving critical insights.
- Made use of native xgB,lgbm,lr,ridge,lasso libraries for initial model evaluation and selection.
- Model selection criteria and evaluation methods encompassed algorithms such as **linear regression, lasso & ridge regression, light gradient boost, and XGBoost**.
- Evaluating models on the entire dataset was essential to accurately assess performance in terms of precision and generalization capabilities, guiding further refinement and optimization efforts.

Algorithm	Train_Score	Test_Score	Train_Rmse	Test_Rmse	Train_Mae	Test_Mae
XGBRegressor_baseline	0.765303	0.763720	74.390983	74.705644	45.436795	45.515221
LGBMRegressor_baseline	0.723130	0.722395	80.798703	80.975469	51.427360	51.465795
Ridge_baseline	0.466980	0.466138	112.108294	112.293401	73.765442	73.779103
LinearRegression_baseline	0.466980	0.466138	112.108294	112.293401	73.765443	73.779104
Lasso_baseline	0.466571	0.465729	112.151347	112.336370	73.607327	73.620969

Results

- Overview to show how XGBoost outperforms standard Ridge and Linear Regression by almost 60%
- All models are probably being overtrained because of how rich in features the data is so we can see high accuracy in train and test scores

Linear Regression

- Linear regression helps us understand how different things like booking early or flight routes affect ticket prices by drawing a straight line on a graph.
- It's a simple way to see which factors are most important for predicting prices, making it a good starting point before using more complex models.

Linear Regression

- Output

	Train_Score	Test_Score	Train_Rmse	Test_Rmse	Train_Mae	Test_Mae
LinearRegression_baseline	0.466980	0.466138	112.108294	112.293401	73.765443	73.779104

Problems faced:

- Linear Regression struggles with non-straight relationships between features and the target.
- It can be thrown off by unusual data points and when features are too similar to each other.

XGBoost

- XGBoost is excellent at handling complex airline pricing data, finding patterns even in messy or incomplete information.
- It helps us build a smart model that predicts ticket prices accurately and quickly adapts to changes in the market, maximizing airline revenue while ensuring fair prices for customers.

XGBoost

- Hyperparameter Tuning:

n_estimators: The grid search evaluates the model performance with 100, 200, and 300 boosting rounds.

max_depth: The grid search considers trees with maximum depths of 3, 5, and 7.

learning_rate: The grid search tests learning rates of 0.1, 0.01, and 0.001.

```
param_grid_xgb = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001]
}
xgb_grid = GridSearchCV(xgb.XGBRegressor(random_state=42), param_grid_xgb, cv=5)
xgb_grid.fit(X_train, y_train)
best_params_xgb = xgb_grid.best_params_
print("Best parameters for XGBRegressor:", best_params_xgb)
```

XGBRegressor_baseline	0.765303	0.763720	74.390983	74.705644	45.436795	45.515221

LGBMBoost

- LightGBM is chosen for its ability to handle large datasets efficiently and identify complex patterns quickly.
- Its capability to learn from mistakes and improve predictions over time makes it ideal for real-time pricing adjustments in the dynamic airline industry.

LGBMBoost

- Hyperparameter Tuning:

n_estimators: LGBM will be tested with 100, 200, and 300 trees to determine which number works best.

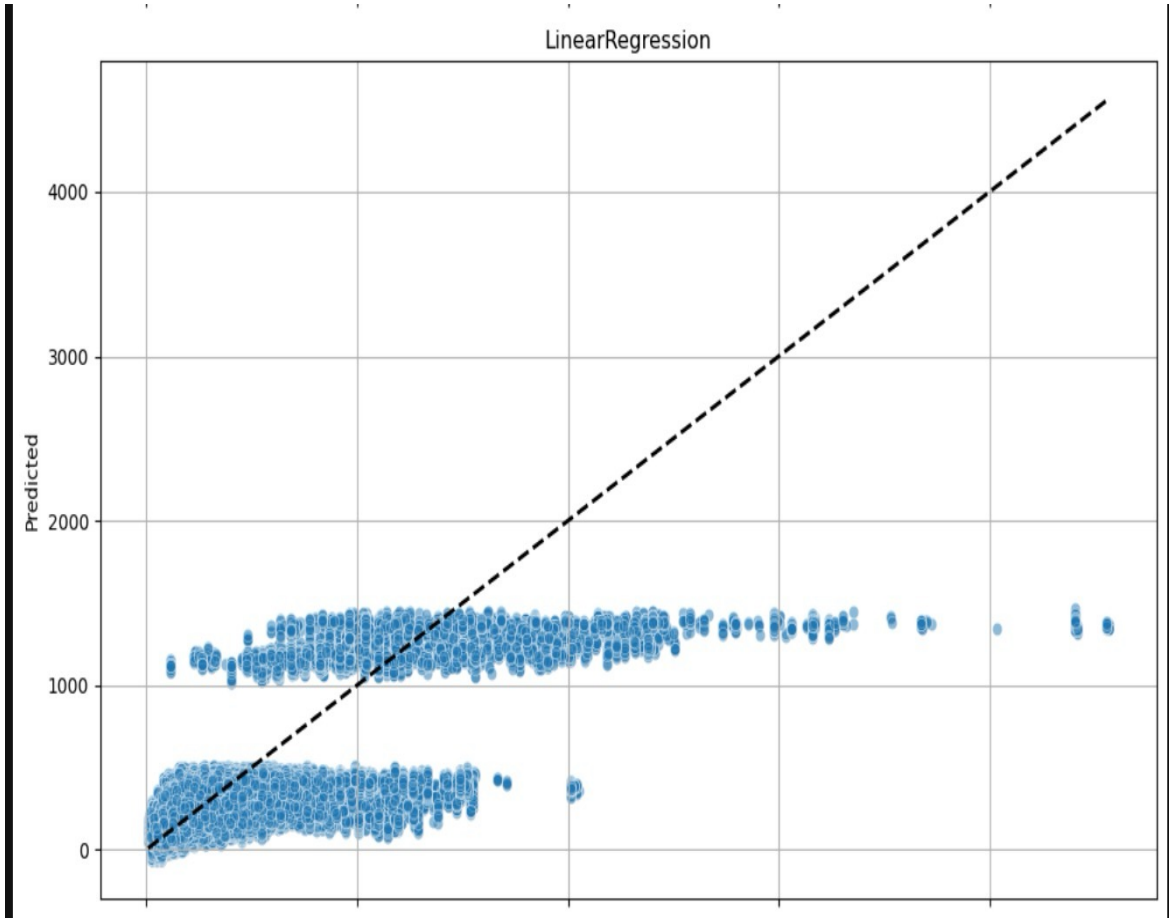
max_depth: It will be tested with trees of maximum depths 3, 5, and 7 to find the optimal depth.

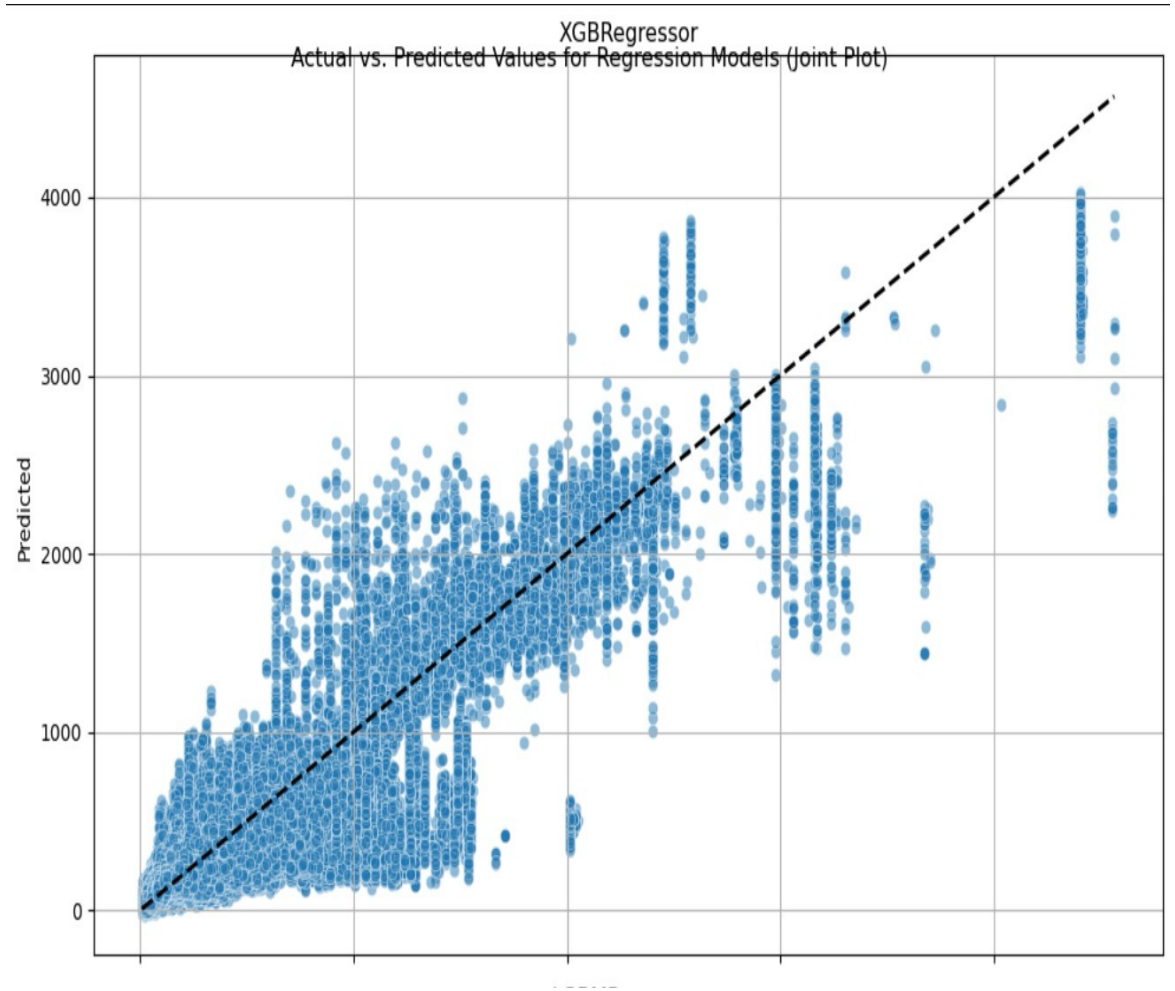
earning_rate: LGBM will be tested with learning rates of 0.1, 0.01, and 0.001 to determine which rate works best.

```
param_grid_lgbm = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001]
}
lgbm_grid = GridSearchCV(lightgbm.LGBMRegressor(random_state=42), param_grid_lgbm, cv=5)
lgbm_grid.fit(X_train, y_train)
best_params_lgbm = lgbm_grid.best_params_
print("Best parameters for LGBMRegressor:", best_params_lgbm)
```

LGBMRegressor_baseline	0.723130	0.722395	80.798703	80.975469	51.427360	51.465795

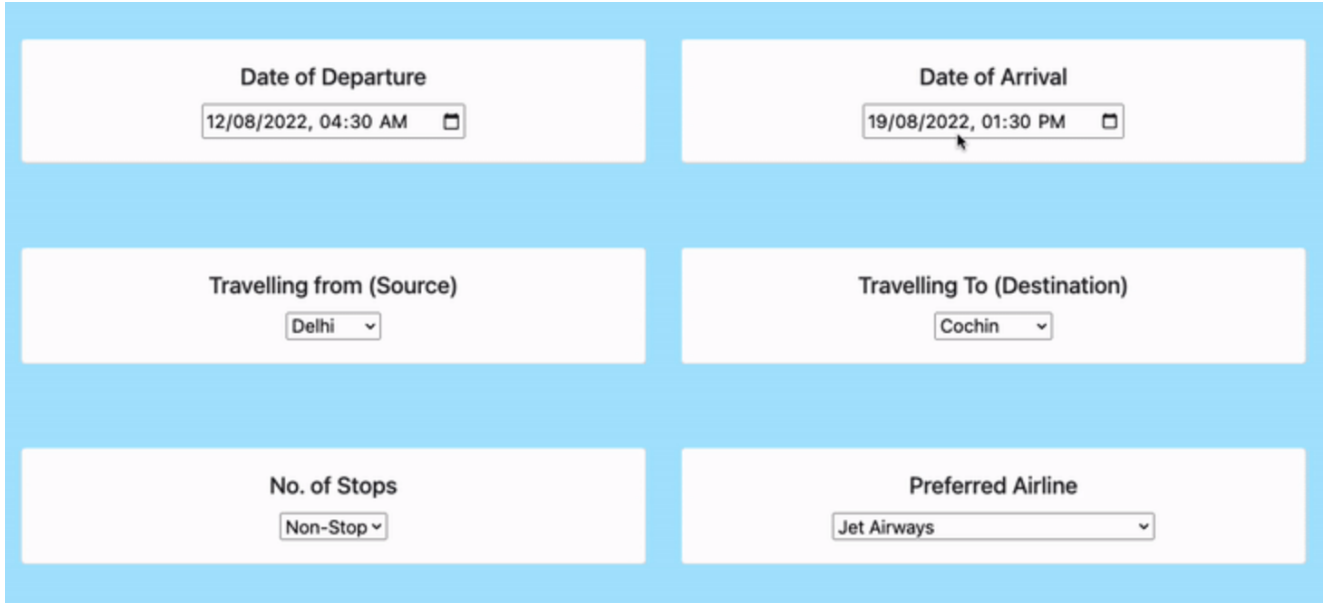
Results Visualization





What do we wish to do?

- If we get optimal results from the various models we wish to try:
  - Build a live Flask app that can take this model forward
  - Generate visualizations on PowerBI to understand patterns and further improvise on the system



Conclusion

- Our project developed a dynamic pricing model for U.S. airlines, using advanced analytics and machine learning. We identified key predictors, employed ARIMA and Random Forest Regression for accurate price forecasting, and utilized Expedia data for real-time market response. Our model enhances profitability and consumer satisfaction, offering actionable insights for data-driven pricing strategies, improving industry efficiency and competitiveness.

Thank you!