American Airlines

Zowie, Parik, ZhangXiang, Siddarth

Hypothesis

"Low prices, low delay rates with quality service will lead to higher customer satisfaction which in turn, generates higher revenue."

Table of contents

01
Zowie
“Fare Per Mile is most affected by time of flight”

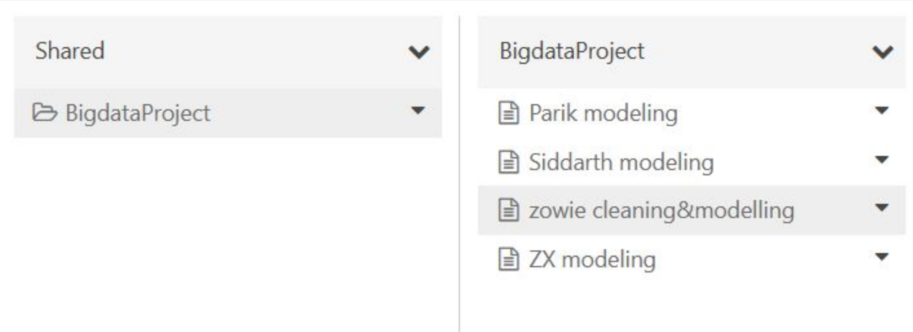
02
ZhangXiang
“The quality of a flight experience can affect passengers choice of flight”

03
Parik
“Customer service are the primary contributing factor to availability of passengers”

04
Siddarth
“Weather conditions are the primary contributing factor to airline delay”

Integration

- Folder structure



- Merging of data files

```
[{'Airline': 'AA', 'Year': 2017, 'Quarter': 1, 'len': 272},
 {'Airline': 'AA', 'Year': 2017, 'Quarter': 2, 'len': 1443},
 {'Airline': 'AA', 'Year': 2017, 'Quarter': 3, 'len': 3052},
 {'Airline': 'AA', 'Year': 2017, 'Quarter': 4, 'len': 2215},
 {'Airline': 'AA', 'Year': 2018, 'Quarter': 1, 'len': 2679},
 {'Airline': 'AA', 'Year': 2018, 'Quarter': 2, 'len': 2813},
 {'Airline': 'AA', 'Year': 2018, 'Quarter': 3, 'len': 2855},
 {'Airline': 'AA', 'Year': 2018, 'Quarter': 4, 'len': 2218},
 {'Airline': 'AA', 'Year': 2019, 'Quarter': 1, 'len': 2001},
```

```
zowie_df.shape
```

```
(67968, 42)
```

```
# Iterate over each dictionary and remove rows randomly until the desired length is achieved
for attr in result:
    filtered_df = zowie_df[(zowie_df['Airline'] == attr['Airline']) & (zowie_df['Year'] == attr['Year']) & (zowie_df['Quarter'] == attr['Quarter'])]
    if len(filtered_df) > attr['len']:
        remove_indices = random.sample(list(filtered_df.index), len(filtered_df) - attr['len'])
        zowie_df.drop(remove_indices, inplace=True)
```

01

Zowie

“Fare Per Mile is most affected by time of flight”

Overview

Goals

- Investigate the extent of impact of time of flight on fare per mile, to prove the hypothesis true or false
- Uncover the top factors that affect fare per mile

Target Column

- Fare Per Mile

Data Cleaning feedback

Handling of null values

- Initially, when removing all rows with null values without any other imputations, **24,038 rows**, 27.5% of rows were removed
- This is a fairly large percentage and may be removing valuable data point as well

- Hence, I impute null values in FoodandBeverage and In-flightEntertainment with 0
- I then removed rows that contained null values in other columns. Imputing values in other columns would disrupt the distribution, and affect the resulting analysis to be skewed/biased
- Therefore, a **lower percentage of rows were removed**.

Handle null values

```
# fill null values with 0 in numerical columns
df = df.na.fill(value=0,subset=['FoodandBeverage',
'In-flightEntertainment'])

# drop the row if there are null values in any other
df = df.na.drop()
```

```
Number of rows removed:  17806
Percentage of rows removed:  20.811369931859886 %
```

Modeling

- Data is split into 70% training and 30% test data

- Model comparison is done between 3 models
  i. Linear regression
  ii. Random forest - ensemble model
  iii. Gradient boosting regressor - ensemble model

- Used **mean absolute error (MAE)** as the main performance metric used to evaluate model performance

- **Gradient boosting regressor** model performed the best with a MAE of 0.156

```
1 print(x_train.shape)
2 print(x_test.shape)
```

```
(46820, 10)
(20067, 10)
```

	Name	MAE	RMSE	R2
2	GB	0.155819	0.254494	-0.342162
1	RF	0.177492	0.288551	-0.124596
0	LR	0.193467	0.299968	-2.629684

Modeling - kfold cross validation

- After performing k-fold cross validation on the model, the resulting MAE is **0.157**, which is not significantly better than the initial results

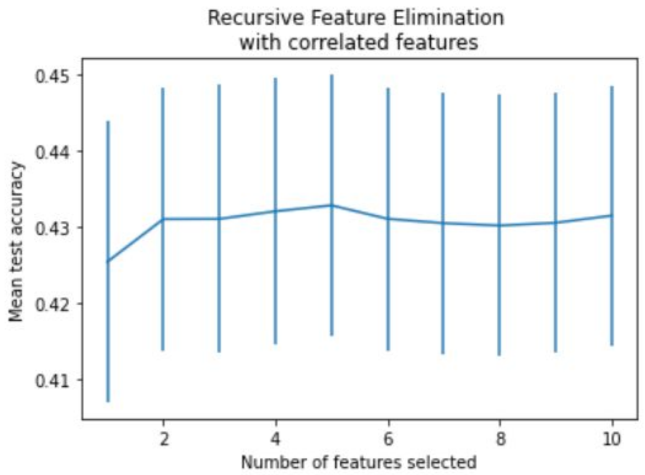
```python
for name, model in models:
    if name in list(gmodels['Name']):
        kfold = KFold(n_splits=10, random_state=123, shuffle=True)
        score = cross_val_score(model, x, y, cv=kfold,
        scoring='neg_mean_absolute_error').mean()
        names.append(name)
        scores.append(score*-1)

kf_cross_val = pd.DataFrame({'Name': names, 'Score':scores})
kf_cross_val.sort_values(by='Score', ascending=False)
```

	Name	Score
0	GB	0.156863

Feature selection

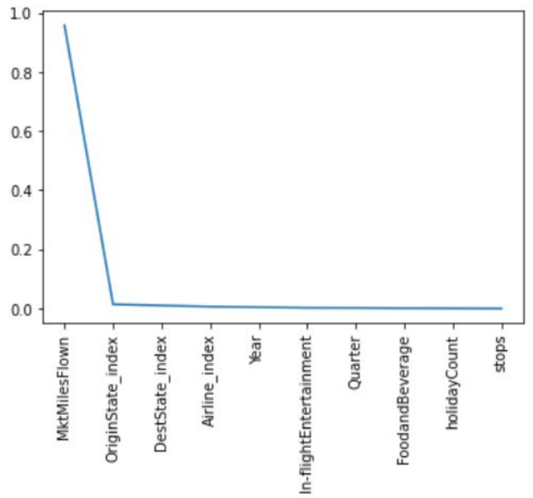
- Through RFECV, the optimal number of features is 5
- Thus, by looking at the feature importances from the model, I used the top 5 features to perform modeling again.

```
Optimal number of features: 5
```



feature importance of all variables

```
cols, vals = zip(*sortFeatures)

# plot line chart
sns.lineplot(x=cols, y=vals)
plt.xticks(rotation=90)
plt.show()
```



Feature selection

- **MAE decreased** when kfold cross validation is performed on the new model using the top 5 features.

k-fold cross validation

Cmd 98

Python

```python
# caluculate accuracy score with new set of features
features = list(cols)

initial_score = cross_val_score(model, x, y, cv=kfold,
scoring='neg_mean_absolute_error').mean()
print('initial MAE:', '%.5f' %(initial_score*-1))

fe_score = cross_val_score(model, x[features], y, cv=kfold,
scoring='neg_mean_absolute_error').mean()
print('MAE after Feature Selection', '%.5f' %(fe_score*-1))
```

```
initial MAE: 0.15687
MAE after Feature Selection 0.15680
```

Command took 1.26 minutes -- by zowieongzo@gmail.com at 07/08/2023, 7:05:28 pm on kk (cl

Feature selection

- In a normal prediction, **MAE and RMSE increased very slightly**, meaning that there is more error between actual and predicted values after feature selection.

- However, **R^2 score increased significantly,** from negative to a positive value. This indicates that the model fits the data better after feature selection. (line of best fit)

```
print('MAE Score before feature selection:','%.3f' % gmodels['MAE'][0])
print('R2 Score before feature selection:','%.3f' % gmodels['R2'][0])
print('RMSE Score before feature selection:','%.3f' % gmodels['RMSE'][0])
```

```
MAE Score before feature selection: 0.156
R2 Score before feature selection: -0.342
RMSE Score before feature selection: 0.254
```

Command took 0.09 seconds -- by zowieongzo@gmail.com at 07/08/2023, 7:05:29 pm on kk (clone)

d 104

```
model = GradientBoostingRegressor(n_estimators=10)
model.fit(x_train,y_train)

y_pred = model.predict(x_test)
print('MAE Score after feature selection:','%.3f' % mean_absolute_error
(y_test, y_pred))
print('R2 Score after feature selection:','%.3f' % r2_score(y_test,
y_pred))
print('RMSE Score after feature selection:','%.3f' % np.sqrt
(mean_squared_error(y_test, y_pred)))
```

```
MAE Score after feature selection: 0.169
R2 Score after feature selection: 0.360
RMSE Score after feature selection: 0.268
```

Command took 0.39 seconds -- by zowieongzo@gmail.com at 07/08/2023, 7:05:29 pm on kk (clone)

Hyperparameter tuning

- Looped through a range of 5 to 14 for number of parameters.
- Too many estimators will be too resource consuming for a simple task like this use case. Hence, i chose 14 as a limit, to prevent overfitting as well.

	Estimators	MAE	RMSE	R2
1	14	0.16325659228672337	0.2618404899712328	0.38853046807169045
2	13	0.16424933389330318	0.26294023661109484	0.3833832599086924
3	12	0.1654742917130737	0.26423675806185265	0.37728736644225325
4	11	0.1668917571769623	0.265838217603209	0.3697163448379147
5	10	0.16857102023258425	0.26781823698107954	0.36029240685851116
6	9	0.1706689140644128	0.27028432171000655	0.3484572433283256
7	8	0.17311844713762123	0.27323395045476423	0.3341589988714826

Metrics before tuning, number of estimators is 10

	Name	MAE	RMSE	R2
0	GB	0.168571	0.267818	0.360292

Metrics after tuning, number of estimators is 14

	Name	MAE	RMSE	R2
0	GB	0.163257	0.26184	0.38853

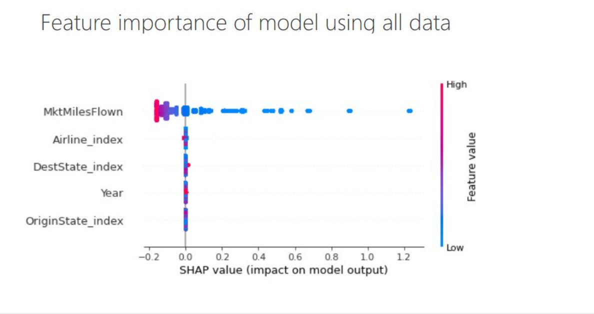
- Initial no. of estimators = 10, the model with 14 base estimators performed best, where the metrics improved slightly.
- This is also due to the increased number of base estimators.

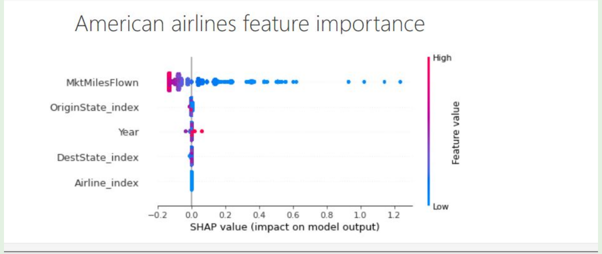
Model interpretation

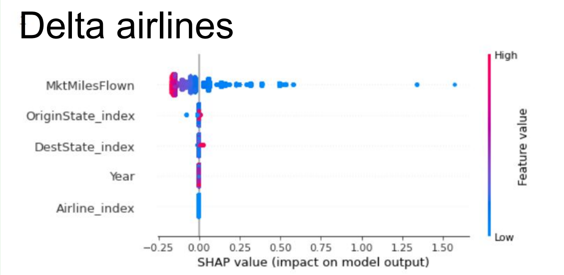
- Using SHAP values, I can see which features greatly affect the local predictions

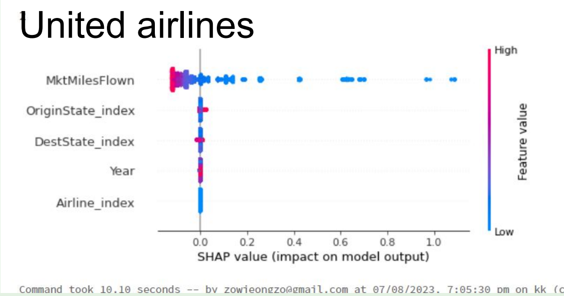
Competitor analysis

- Another 3 models were created for each airlines' individual data, to see the important factors for each airline.









Insights

- miles flown and origin state are the most important features for all models
- year has a higher impact on ticket prices for American Airlines
- destination state has a higher impact for Delta and United Airlines

This indicates that the research hypothesis should be **rejected**, as the time of flight does not affect ticket price to a large extent. Instead, miles flown and origin state has a more significant effect.

02

Zhang Xiang

“The quality of a flight experience can affect passengers choice of flight”

Overview

Goal: Improve Airline flight service by finding out what customer care and what Airline is lacking.

Build

- **Sentiment Classification (Numeric Rating/Text)**: Understanding customer satisfaction or dissatisfaction from reviews and comments. (Positive/Negative)
- **Topic Modelling**: Gain a deeper understanding of what customer are interested of through text, categorizing/clustering topics into different groups.
- **Aspect Based Sentiment Classification (With Transformer)**: Identifying sentiments associated with specific aspects or features experienced on the flight. (Positive/Neutral/Negative)

Target Column

- Sentiment

Data Preparation / Cleaning

General Cleaning

- Remove Outlier (Year Columns: Found 2500+)
- Remove Duplicates (From Web-Scraping)
- Remove Null/Missing Value (Drop rows with null)
- Remove Rows with Non-English Text ()
- Encode Sentiment
- Data Balancing (Random Undersampling)

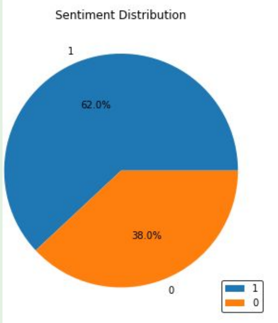
```
After undersampling:
0    24704
1    24704
Name: sentiment, dtype: int64
```

Drop Null

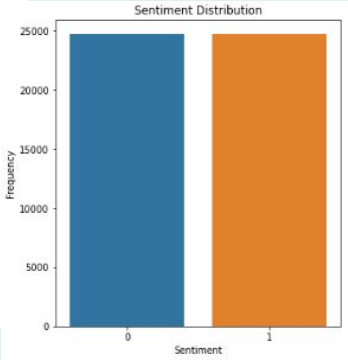
```
Before Removing Null:  (85559, 75)
After Removing Null:  (64956, 75)
```

Sentiment Encoded

Sentiment	Encoded Value
Negative (1, 2, 3)	0
Positive (4, 5)	1



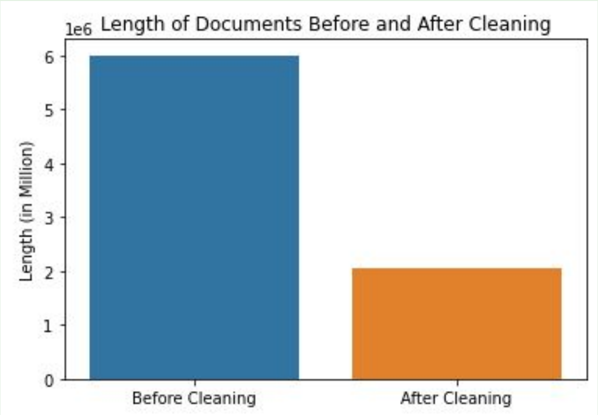
After Data Balancing



Data Preparation / Cleaning

Text Cleaning

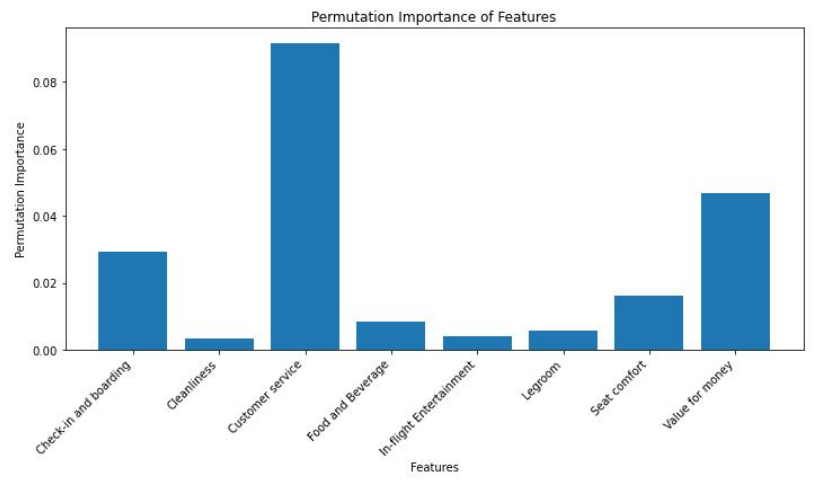
- Join review_header + reviews = combined_reviews
- Remove Emoji Icon😊
- Lowercasing Text (IDenTIfy => identify)
- Remove URLs, Email & Number
- Expand Contraction (I'm = I am)
- Remove Punctuation (!!!!????...)
- Remove Stop Words (i will spent => spent)
- Remove Spelling Error
- POS Tagging Lemmatization (ate(VERB) => eat)



remove_emoji_icon	lowercase	remove_url	remove_email	remove_text_number	expand_contraction	remove_punctuation	remove_stopwords	wrong_spelling	deep_spelling_cleansing	lemmatized	len_before_cleaning	len_after_cleaning
avoid american airline	avoid american ai	avoid american airlin	avoid american airlir	avoid american airlines it's	avoid american airlines	avoid american airlines it	['avoid', 'american', 'a	set()	['avoid', 'american', 'airlines',	avoid american airl	94	31
Good crew, bad seats,	good crew, bad se	good crew, bad seats	good crew, bad seat:	good crew, bad seats, agai	good crew, bad seats, a	good crew bad seats agai	['good', 'crew', 'bad', '	{'shoudl'}	['good', 'crew', 'bad', 'seats', 'f	good crew bad seat	64	19
Terrible experience...	terrible experien	terrible experience..	terrible experience.	terrible experience... very	terrible experience... ve	terrible experience very	['terrible', 'experienc	{'â€™'}	['terrible', 'experience', 'disa	terrible experience	293	98
I did not like it We sel	i did not like it we	i did not like it we se	i did not like it we se	i did not like it we selecte	i did not like it we selec	i did not like it we selecte	['not', 'selected', 'clas	{'tvs', 'stewardesse:	['not', 'selected', 'class', 'thou	not select class thir	118	36
WILL KICK YOU OFF TH	will kick you off t	will kick you off the p	will kick you off the	will kick you off the plane	will kick you off the pla	will kick you off the plane	['kick', 'plane', 'overb	{'bldg', 'overbookin	['kick', 'plane', 'told', 'passpor	kick plane tell pass	56	19
Great Xmas gift A few	great xmas gift a	great xmas gift a few	great xmas gift a few	great xmas gift a few years	great xmas gift a few ye	great xmas gift a few year	['great', 'xmas', 'gift',	{'nyc', 'curticy', 'jfk'}	['great', 'xmas', 'gift', 'years', '	great xmas gift yea	130	50

Sentiment Classification - Numeric Features

Features Used (Importance in Modelling)



Target: Predict Sentiment (Positive or Negative)

Sentiment Classification - Numeric Features

K-Fold Cross-Validation Model Comparison

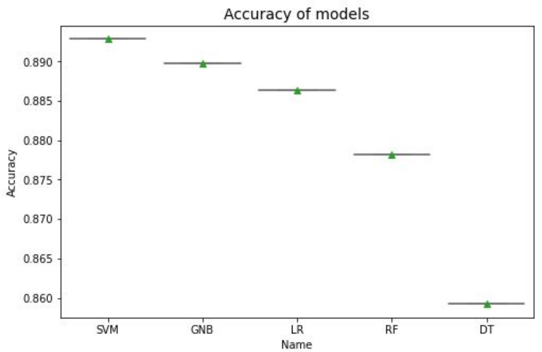
```
k_fold_model_comparison = k_fold_model_comparison()

k_fold_model_comparison
```

	Name	Accuracy	Precision	Recall	F1	Confusion Matrix	
0	SVM	0.892855	0.892876	0.892866	0.892852	[[4421, 520], [557, 4384]]	<sklearn.metrics._plot.confusion_m
1	GNB	0.889718	0.889737	0.889726	0.889714	[[4375, 566], [585, 4356]]	<sklearn.metrics._plot.confusion_m
2	LR	0.886328	0.886335	0.886338	0.886325	[[4329, 612], [550, 4391]]	<sklearn.metrics._plot.confusion_m
3	RF	0.878156	0.878172	0.878167	0.878153	[[4368, 573], [660, 4281]]	<sklearn.metrics._plot.confusion_m
4	DT	0.859207	0.859296	0.859218	0.859196	[[4177, 764], [821, 4120]]	<sklearn.metrics._plot.confusion_m

Select Top 2 Best Model for further Tuning

```
top_models = baseline_model_comparison.nlargest(2, 'Accuracy')
top_models_ranked = top_models.reset_index(drop=True)

print("Top 2 Models by Accuracy:")
print(top_models_ranked[['Name', 'Accuracy', 'F1']])
```

```
Top 2 Models by Accuracy:
  Name  Accuracy        F1
0  SVM  0.889091  0.889090
1   LR  0.884436  0.884435
```



Sentiment Classification - Numeric Features

Hyperparameter Tuning for the top-2 Models

```
parameters = {'C': [0.1,0.2, 0.3,0.5, 1],
              'kernel': ['rbf', 'poly'],
              'gamma': ['auto', 'scale']}

# cv parameter can be used for number of folds to use for cross-validation.
grid_search = GridSearchCV(SVC(random_state=42), parameters, cv=5, n_jobs=-1,
grid_search.fit(X_train, y_train)

print('best parameters: ', grid_search.best_params_)
print("Best Model:", grid_search.best_estimator_)
print('best scrores: ', grid_search.best_score_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
best parameters:  {'C': 0.3, 'gamma': 'scale', 'kernel': 'rbf'}
Best Model: SVC(C=0.3, random_state=42)
best scrores:  0.8934371848345191
```

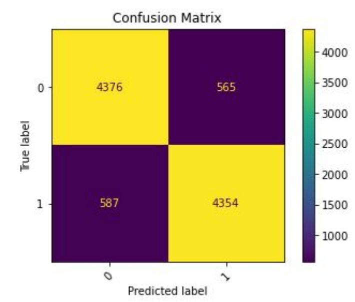
```
parameters = {'var_smoothing': np.logspace(0,-9, num=100)}

# cv parameter can be used for number of folds to use for cross-validation.
grid_search = GridSearchCV(GaussianNB(), parameters, cv=5, n_jobs=-1, verbose=True)
grid_search.fit(X_train, y_train)

print('best parameters: ', grid_search.best_params_)
print("Best Model:", grid_search.best_estimator_)
print('best scrores: ', grid_search.best_score_)
```

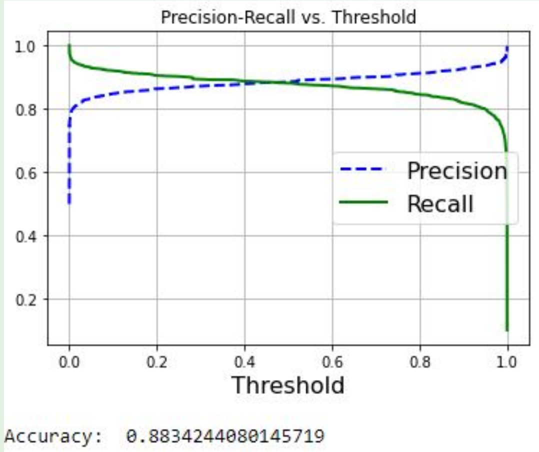
```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
best parameters:  {'var_smoothing': 0.0012328467394420659}
Best Model: GaussianNB(var_smoothing=0.0012328467394420659)
best scrores:  0.8899964558259101
```

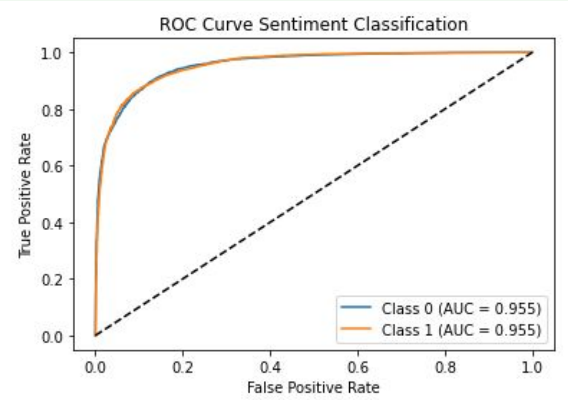
Sentiment Classification - Numeric Features

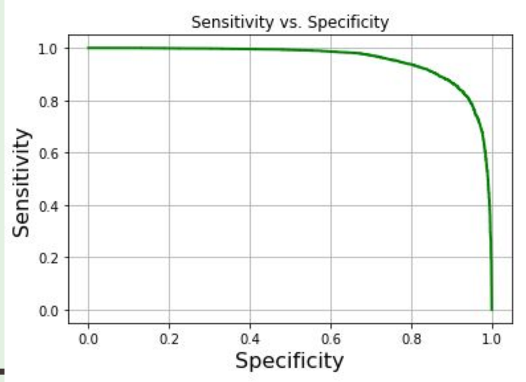
Selected Best Model: Gaussian Naive Bayes

```
GaussianNB(var_smoothing=0.004328761281083057)

              precision    recall  f1-score   support

           0       0.88      0.89      0.88      4941
           1       0.89      0.88      0.88      4941

    accuracy                           0.88      9882
   macro avg       0.88      0.88      0.88      9882
weighted avg       0.88      0.88      0.88      9882
```

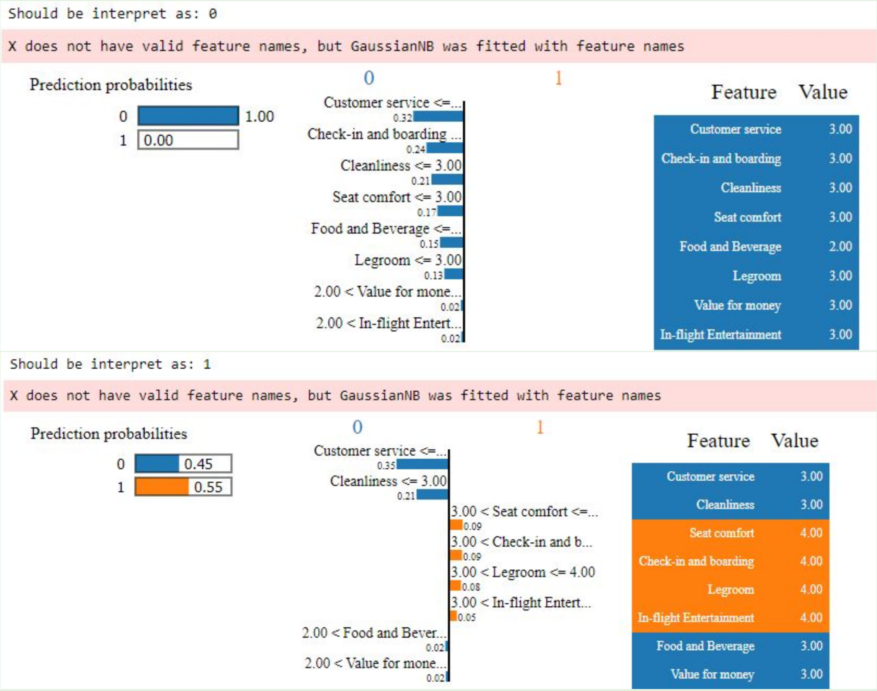








Sentiment Classification - Numeric Features



Sentiment Classification - Text

Vectorize Data with TFIDF (Unigram) - Best Performance

```python
# Sample Testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
                                                    random_state=42)
# Shape before transformation

print("The size of original dataset: ", X.shape)
print("The size of traning dataset: ", X_train.shape)
print("The size of testing dataset: ", X_test.shape)

max_features = None
# Transforming text into a format that the model can understand
vectorizer = TfidfVectorizer(ngram_range = (1,1), max_features = max_features)

X_train = vectorizer.fit_transform(X_train).toarray()
X_test = vectorizer.transform(X_test).toarray()
```

After several testing with models, Unigram and vectorizer of TFIDF perform better than Bigram and CountVectorizer.

Bigram & Trigram produces more than 60k columns of features, do not have enough RAM to support as well.

Features Used (Vectorized Text after cleaning)

zil	brazilian	breach	bread	breadstick	breadth	break	breakable	breakage	breakdown	breaker	breakfast	breaks	breaku
.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

17000+ Features Generated

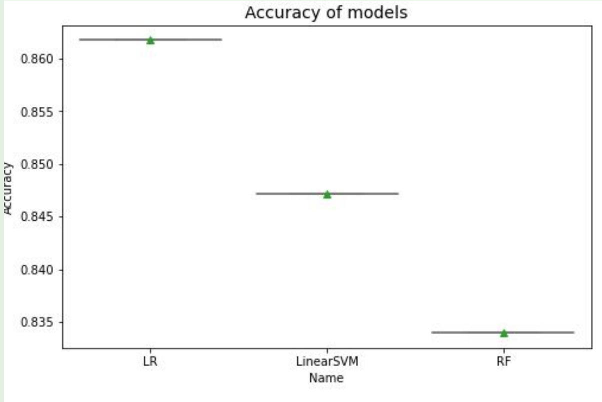
Sentiment Classification - Text

Model Comparison

```
baseline_model_comparison = model_comparison()

baseline_model_comparison
```

	Name	Accuracy	Precision	Recall	F1	Confusion Matrix	
0	LR	0.861769	0.861948	0.861769	0.861752	[[4313, 628], [738, 4203]]	<sklearn.metric
1	LinearSVM	0.847197	0.847232	0.847197	0.847193	[[4211, 730], [780, 4161]]	<sklearn.metric
2	RF	0.833940	0.834409	0.833940	0.833882	[[4213, 728], [913, 4028]]	<sklearn.metric

Select Top 2 Best Model for further Tuning



Sentiment Classification - Text

Feature Selection (Chi-Square)

A higher chi-squared value is generally considered better for feature selection purposes because it suggests a stronger association between a feature and the target variable

	Model	Accuracy	Precision Score	Recall	F1	Features
0	LogisticRegression()	0.862882	0.863072	0.862882	0.862864	17000
1	LogisticRegression()	0.864096	0.864308	0.864096	0.864077	16000
2	LogisticRegression()	0.864400	0.864587	0.864400	0.864382	15000
3	LogisticRegression()	0.863590	0.863784	0.863590	0.863572	14000
4	LogisticRegression()	0.862882	0.863100	0.862882	0.862861	13000
5	LogisticRegression()	0.863186	0.863407	0.863186	0.863165	12000
6	LogisticRegression()	0.862578	0.862800	0.862578	0.862557	11000
7	LogisticRegression()	0.862275	0.862478	0.862275	0.862256	10000
8	LogisticRegression()	0.862174	0.862353	0.862174	0.862157	9000
9	LogisticRegression()	0.860959	0.861180	0.860959	0.860938	8000
10	LogisticRegression()	0.860656	0.860909	0.860656	0.860631	7000
11	LinearSVC(random_state=42)	0.849524	0.849574	0.849524	0.849519	17000
12	LinearSVC(random_state=42)	0.849929	0.849964	0.849929	0.849925	16000
13	LinearSVC(random_state=42)	0.851447	0.851478	0.851447	0.851444	15000
14	LinearSVC(random_state=42)	0.852560	0.852595	0.852560	0.852557	14000
15	LinearSVC(random_state=42)	0.853167	0.853197	0.853167	0.853164	13000
16	LinearSVC(random_state=42)	0.853066	0.853108	0.853066	0.853062	12000
17	LinearSVC(random_state=42)	0.852358	0.852419	0.852358	0.852351	11000
18	LinearSVC(random_state=42)	0.852864	0.852939	0.852864	0.852856	10000
19	LinearSVC(random_state=42)	0.854786	0.854852	0.854786	0.854780	9000
20	LinearSVC(random_state=42)	0.855090	0.855161	0.855090	0.855083	8000
21	LinearSVC(random_state=42)	0.855394	0.855471	0.855394	0.855386	7000
22	LinearSVC(random_state=42)	0.854888	0.854972	0.854888	0.854879	6000
23	LinearSVC(random_state=42)	0.855798	0.855889	0.855798	0.855789	5000
24	LinearSVC(random_state=42)	0.857114	0.857170	0.857114	0.857108	4000
25	LinearSVC(random_state=42)	0.857316	0.857396	0.857316	0.857308	3000
26	LinearSVC(random_state=42)	0.857519	0.857579	0.857519	0.857513	2000

```python
def get_best_accuracy(data):
    models = set([d['Model'] for d in data])
    result = []
    for model in models:
        filtered_data = [d for d in data if d['Model'] == model]
        best_accuracy = max([d['Accuracy'] for d in filtered_data])
        best_model = [d for d in filtered_data if d['Accuracy'] == best_accuracy][0]
        result.append(best_model)
    return result

selected_features = pd.DataFrame(get_best_accuracy(models_performance)).sort_values(by='Accuracy',
selected_features
```

	Model	Accuracy	Precision Score	Recall	F1	Features
0	LogisticRegression()	0.864400	0.864587	0.864400	0.864382	15000
1	LinearSVC(random_state=42)	0.857519	0.857579	0.857519	0.857513	2000

- Logistic Regression Best Features: 15000
- LinearSVC Best Features: 2000

Sentiment Classification - Text

Hyperparameter Tuning for the top-2 Models

```
parameters = {'penalty': ['l1', 'l2'],
              'C': [1, 1.2, 1.5],
              'max_iter': [100, 150]
             }

# cv parameter can be used for number of folds to use for cross-validation.
grid_search = GridSearchCV(LogisticRegression(random_state=42), parameters, cv=5,
grid_search.fit(X_train, y_train)

print('best parameters: ', grid_search.best_params_)
print("best Model:", grid_search.best_estimator_)
print('best scores: ', grid_search.best_score_)
```

```
Accuracy:  0.8638939485934022
```

```
LogisticRegression
LogisticRegression(C=1.5, random_state=42)
```

```
parameters = {'C': [ 0.7, 1, 1.2],
              'loss': ['hinge', 'squared_hinge'],
              'penalty': ['l1', 'l2']}

# cv parameter can be used for number of folds to use for
grid_search = GridSearchCV(LinearSVC(random_state=42), pa
grid_search.fit(X_train, y_train)

print('best parameters: ', grid_search.best_params_)
print("best Model:", grid_search.best_estimator_)
print('best scrores: ', grid_search.best_score_)
```

```
Accuracy:  0.8573163327261688
```

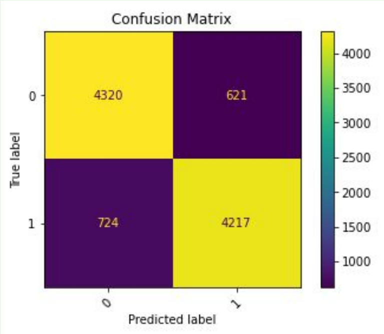
```
LinearSVC
LinearSVC(C=0.7, loss='hinge', random_state=42)
```

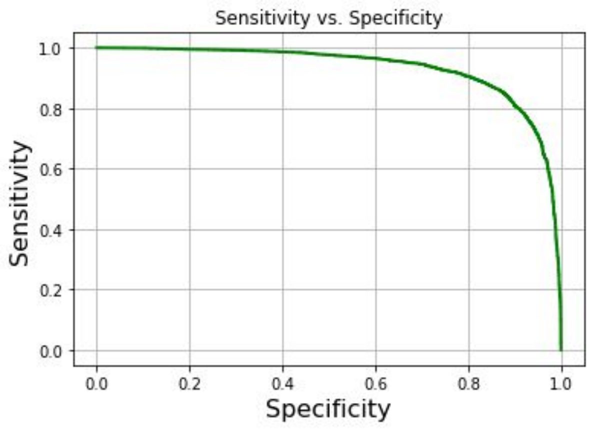
Sentiment Classification - Text

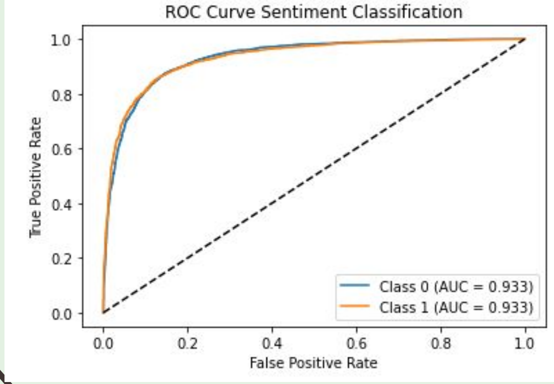
Selected Best Model: Logistic Regression

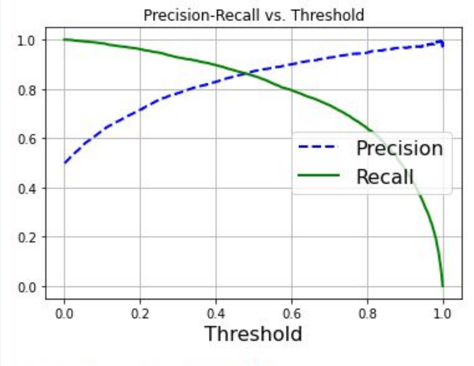
```
LogisticRegression(C=1.5, random_state=42)
```

	precision	recall	f1-score	support
0	0.86	0.87	0.87	4941
1	0.87	0.85	0.86	4941
accuracy			0.86	9882
macro avg	0.86	0.86	0.86	9882
weighted avg	0.86	0.86	0.86	9882










```
Accuracy:  0.8638939485934022
```

Topic Modelling

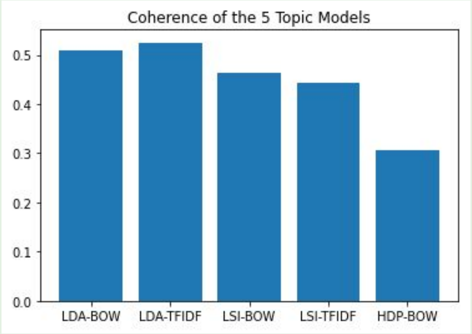
Performed all model below with BOW/TFIDF:

- Latent Dirichlet Allocation (LDA)
- Latent Semantic Indexing (LSI)
- Hierarchical Dirichlet Process (HDP)

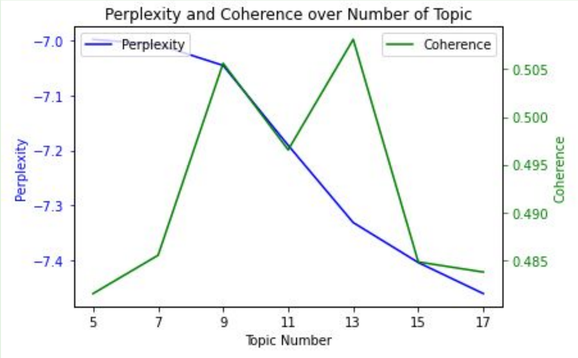
Identify best model through assessing Coherence and Perplexity

- **Coherence** measures how semantically similar the words are in a topic. A high coherence score indicates that the words in a topic are closely related and make sense together.
- **Perplexity** measures how well a topic model can predict new or unseen data. A low perplexity score indicates that the topic model is confident and accurate in its predictions.

Model Comparison through coherence



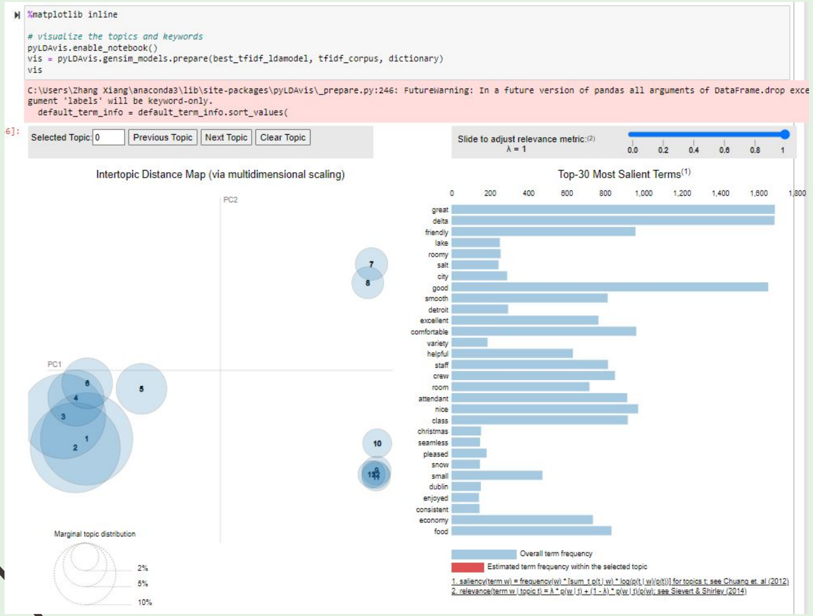
Best Topic Model at 13 Topics: LDA_BOW



Topic Modelling

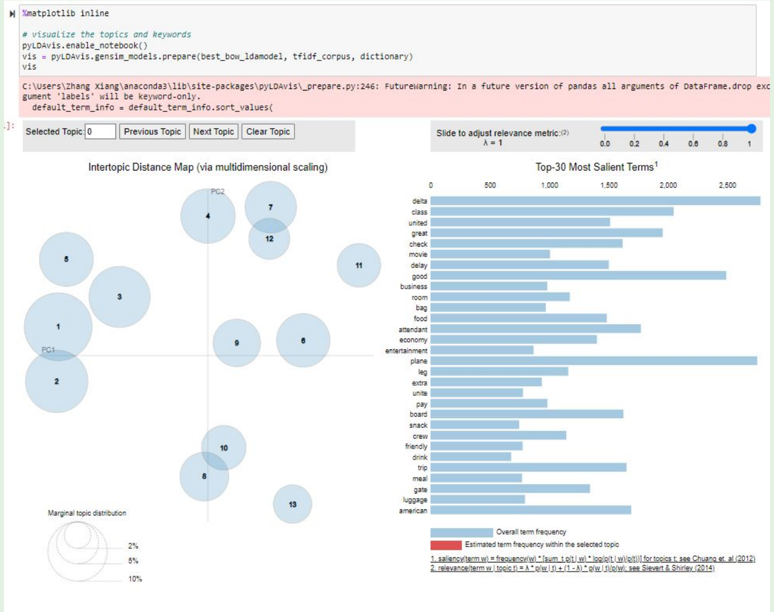
Topic Model Comparison through Visuals

LDA_TFIDF : Highest `Coherence Score:  0.5249280933183984`



Selected as Best: Able to identify distinct topics

LDA_BOW : Second Highest `Coherence Score:  0.5081221421959794`



Topic Modelling

Some topics Extracted from Best Model

```
Topic 3, has 9240 documents:
0.052*"delay" + 0.040*"hour" + 0.029*"plane" + 0.026*"minute" + 0.025*"gate"

Topic 7, has 4212 documents:
0.043*"attendant" + 0.031*"passenger" + 0.018*"people" + 0.017*"crew" + 0.014*"plane"

Topic 5, has 3961 documents:
0.073*"check" + 0.054*"bag" + 0.046*"board" + 0.034*"luggage" + 0.025*"carry"

Topic 0, has 3151 documents:
0.046*"food" + 0.041*"snack" + 0.039*"drink" + 0.038*"meal" + 0.029*"serve"

Topic 10, has 4207 documents:
0.024*"customer" + 0.023*"day" + 0.019*"agent" + 0.015*"book" + 0.015*"tell"

Topic 1, has 4442 documents:
0.057*"room" + 0.052*"leg" + 0.040*"plane" + 0.027*"small" + 0.025*"row"

Topic 6, has 1822 documents:
0.119*"united" + 0.056*"unite" + 0.039*"year" + 0.029*"american" + 0.029*"mile"

Topic 12, has 2239 documents:
0.040*"trip" + 0.032*"american" + 0.021*"san" + 0.019*"good" + 0.019*"direct"

Topic 9, has 1841 documents:
0.174*"class" + 0.083*"business" + 0.024*"food" + 0.022*"lounge" + 0.020*"economy"

Topic 11, has 8643 documents:
0.087*"good" + 0.063*"great" + 0.039*"friendly" + 0.032*"comfortable" + 0.032*"crew"

Topic 2, has 1603 documents:
0.096*"movie" + 0.064*"entertainment" + 0.041*"screen" + 0.039*"watch" + 0.036*"plane"

Topic 4, has 2372 documents:
0.053*"economy" + 0.050*"extra" + 0.050*"pay" + 0.028*"book" + 0.025*"price"

Topic 8, has 1675 documents:
0.302*"delta" + 0.051*"great" + 0.044*"atlanta" + 0.027*"trip" + 0.024*"comfort"
```

Aspect Based Sentiment Classification

Pre-Trained Transformer in Sentiment Classification

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from transformers import pipeline

model_name = "yangheng/deberta-v3-base-absa-v1.1"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

classifier = pipeline("text-classification", model=model, tokenizer=tokenizer)

aspects = ['legroom', 'food', 'service', 'Check-in and boarding', 'cleanliness', 'seat', 'value for money', 'entertainment']

#     print(aspect)
#     print(classifier(sentence, text_pair=aspect))
```

```
# Overall Prediction
sentiment_model_path = "cardiffnlp/twitter-xlm-roberta-base-sentiment"
sentiment_model = pipeline("sentiment-analysis", model=sentiment_model_path,
                           tokenizer=sentiment_model_path)
```

Aspect Based Sentiment Classification

Example of Testing

```
We had a great experience for the service at the flight, food was not too bad, but not enough space to stretch my leg.
However, overall I think is neutral
```

Main 3 aspects involved in the sentence.

```
service
Label negative: 0.004178052302449942
Label neutral: 0.0072022550739347935
Label positive: 0.9886196851730347
```

```
food
Label negative: 0.01878264918923378
Label neutral: 0.3369707465171814
Label positive: 0.6442466378211975
```

```
legroom
Label negative: 0.9900957345962524
Label neutral: 0.006372837815433741
Label positive: 0.0035313826519995928
```

Other aspects not in the sentence.

```
Check-in and boarding
Label negative: 0.029345011338591576
Label neutral: 0.6107067465782166
Label positive: 0.3599483072757721
```

```
cleanliness
Label negative: 0.038122426718473434
Label neutral: 0.3269740045070648
Label positive: 0.6349034905433655
```

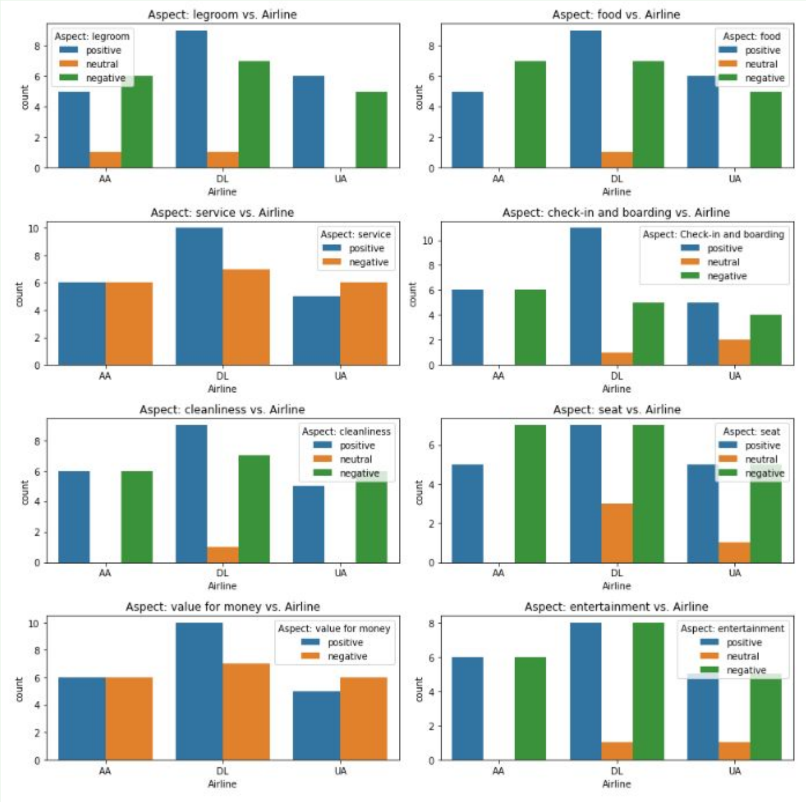
```
seat
Label negative: 0.21330423653125763
Label neutral: 0.523053765296936
Label positive: 0.26364201307296753
```

```
value for money
Label negative: 0.048669859766960144
Label neutral: 0.26902639865875244
Label positive: 0.6823037266731262
```

```
entertainment
Label negative: 0.04489463195204735
Label neutral: 0.3627760708332062
Label positive: 0.5923293232917786
```

```
Overall sentiment: positive with score 0.8118451833724976
Actual Overall Sentiment: 1
Predicted Overall Sentiment: positive
```

Aspect Based Sentiment Classification



03

Parikshit

“Quality of services are the primary contributing factor to availability of passengers”

Data Cleaning

```
Rows with more than or equals to 1 nulls: 20280
Rows with more than or equals to 2 nulls: 12518
Rows with more than or equals to 3 nulls: 11037
Rows with more than or equals to 4 nulls: 2904
Rows with more than or equals to 5 nulls: 1442
```

```
['Check-in and boarding',
 'Cleanliness',
 'Customer service',
 'Food and Beverage',
 'In-flight Entertainment',
 'Legroom',
 'Seat comfort',
 'Value for money']
```

```python
# Loop through each column and check the count of non-null values
for column in df.columns:
    non_null_count = df[column].count()
    if non_null_count < threshold_count:
        columns_with_less_than_82600.append(column)

# Extract data from columns to impute
data_to_impute = df[columns_with_less_than_82600].values

# Impute missing values using KNNImputer
knn_imputer = KNNImputer(n_neighbors=3)
imputed_data = knn_imputer.fit_transform(data_to_impute)

# Update the DataFrame with imputed values
df.loc[:, columns_with_less_than_82600] = imputed_data
```

Removed rows with more than or equals to 4 nulls so that it is more accurate

Only lost 5% of data so not a big information loss

The missing columns i used to input missing values with knn of neighbour =3.

Binning columns

```python
# Define the custom function to determine satisfaction category
def categorize_satisfaction(rating):
    if rating <= 3:
        return 'Neutral or Dissatisfied'
    else:
        return 'Satisfied'
```

```python
# Function to create time bins
def time_bin(time_value):
    try:
        time_value = int(time_value)
        if 0 <= time_value < 1100:
            return 'Morning'
        elif 1100 <= time_value < 1800:
            return 'Afternoon'
        else:
            return 'Evening'
```

Binned the satisfaction rating to satisfied and neutral or dissatisfied

Binned the dept_time and arr_time.

It will help simplify the patterns and relationships within the data.

Model performance

	Model	R-squared on Train	RMSE on Train	MAE on Train
0	LinearRegression	0.024704	7.867994	2.819075
1	Lasso	0.009871	7.927599	2.865253
2	Ridge	0.024684	7.868075	2.818683
3	ElasticNet	0.009895	7.927503	2.865233
4	DecisionTreeRegressor	1.000000	0.000000	0.000000
5	KNeighborsRegressor	0.219951	7.036501	2.486282
6	RandomForestRegressor	0.849171	3.094132	1.122694
7	ExtraTreesRegressor	1.000000	0.000000	0.000000
8	AdaBoostRegressor	-7.400254	23.090957	18.074577
9	XGBRegressor	0.535589	5.429337	2.345209
10	LGBMRegressor	0.233407	6.975546	2.606840

	Model	R-squared on Test	RMSE on Test	MAE on Test
0	LinearRegression	0.032884	6.751863	2.674447
1	Lasso	0.008857	6.835221	2.717076
2	Ridge	0.032816	6.752103	2.673206
3	ElasticNet	0.008878	6.835149	2.717126
4	DecisionTreeRegressor	1.000000	0.000000	0.000000
5	KNeighborsRegressor	0.219637	6.065027	2.371830
6	RandomForestRegressor	0.847812	2.678394	1.051063
7	ExtraTreesRegressor	1.000000	0.000000	0.000000
8	AdaBoostRegressor	-25.735845	35.500245	34.482529
9	XGBRegressor	0.731739	3.556010	1.831958
10	LGBMRegressor	0.342500	5.567141	2.333192

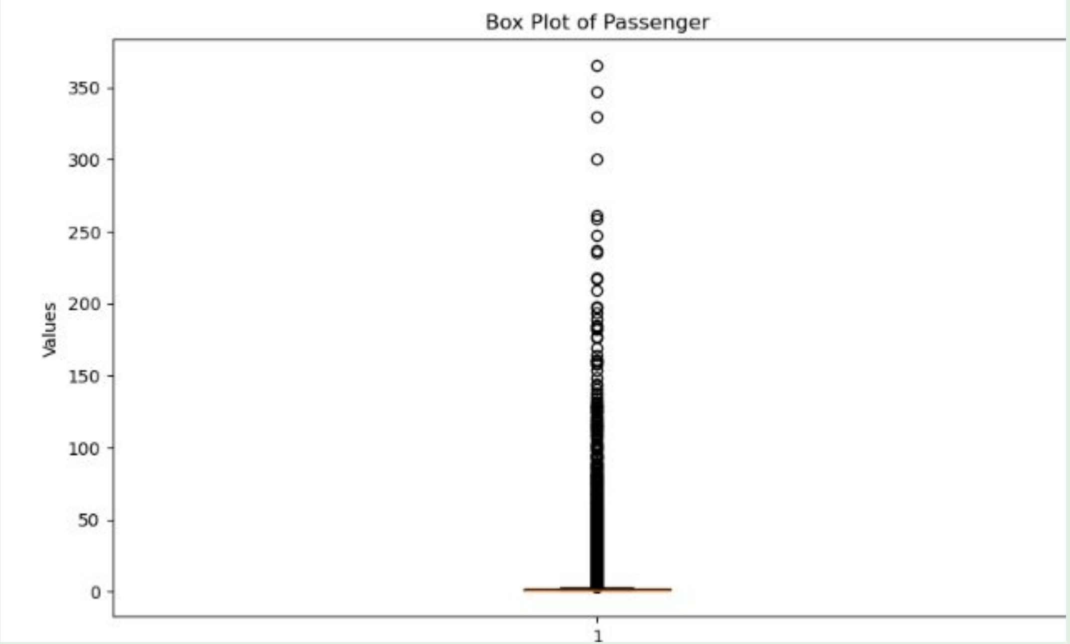
Generally all models are performing very bad by using R-squared

Most of the models have 0 or negative r-square which means there is little or no relation to the target variable(Passengers)

The only good model is Random forest Regressor.

We can also see that many of the models are overfitting.
Linear Regression,Ridge,XGBRegressor,LGBMRegessor

Outliers in target column



There are outliers in my target variable column as shown in the box plot.

Outliers might be causing the models be performing very bad as the r-squared is very bad.

Removed the outliers from passengers column for values above 200

Model performance after removing outliers

	Model	R-squared on Train	RMSE on Train	MAE on Train
0	LinearRegression	0.027976	6.995452	2.732455
1	Lasso	0.010562	7.057836	2.778221
2	Ridge	0.027954	6.995533	2.732112
3	ElasticNet	0.010577	7.057784	2.778254
4	DecisionTreeRegressor	1.000000	0.000000	0.000000
5	KNeighborsRegressor	0.215139	6.285990	2.414987
6	RandomForestRegressor	0.852464	2.725379	1.088082
7	ExtraTreesRegressor	1.000000	0.000000	0.000000
8	AdaBoostRegressor	-7.673464	20.896504	18.813099
9	XGBRegressor	0.439527	5.311959	2.314721
10	LGBMRegressor	0.200179	6.345617	2.532061

	Model	R-squared on Test	RMSE on Test	MAE on Test
0	LinearRegression	0.035570	6.641012	2.697169
1	Lasso	0.010536	6.726654	2.739582
2	Ridge	0.035442	6.641454	2.696338
3	ElasticNet	0.010657	6.726239	2.739355
4	DecisionTreeRegressor	1.000000	0.000000	0.000000
5	KNeighborsRegressor	0.241279	5.890337	2.358409
6	RandomForestRegressor	0.851113	2.609318	1.057479
7	ExtraTreesRegressor	1.000000	0.000000	0.000000
8	AdaBoostRegressor	-0.631111	8.636559	6.508548
9	XGBRegressor	0.730064	3.513415	1.833215
10	LGBMRegressor	0.362185	5.400654	2.315998

After removing the outliers, models are performing a bit better.

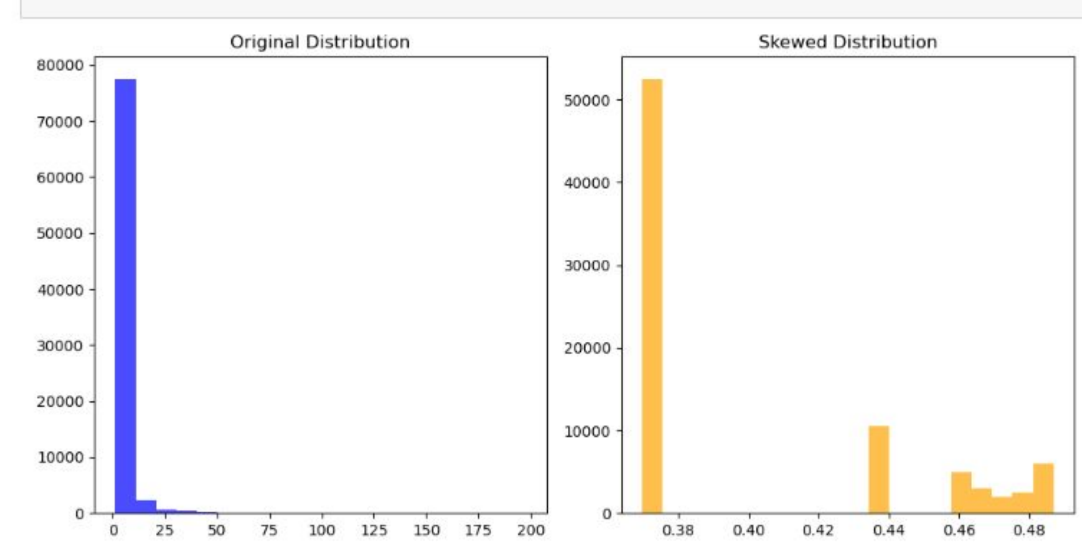
The mae and rmse have generally reduced compared to without removing outliers.

the model's(LR and Lasso) generalization performance has improved. It's better at making accurate predictions not only on the training data but also on new, unseen data.

There are still overfitting models like DT,extra trees.

Our best model,Random forest regressor, got better by 0.01 for R-squared.
The rmse on training reduced from 3.09 to 2.72. The rmse for testing has reduced from 2.67 to 2.60. The difference between the training and testing rmse has decreased which suggests that the model is overfitting less to the training data.

Passenger column skewed



Original distribution has unequal variance and non-constant spread of data points across the range

After doing the transformation,the distribution is still not as good but it is better

Need to transform using boxcox function so it is more spread out

can help stabilize the variance for different variables

improve the linearity and distribution of the data

Model performance after skewed

	Model	R-squared on Train	RMSE on Train	MAE on Train
0	LinearRegression	0.0698	0.0430	0.0387
1	Lasso	0.0197	0.0442	0.0407
2	Ridge	0.0698	0.0430	0.0387
3	ElasticNet	0.0263	0.0440	0.0405
4	DecisionTreeRegressor	1.0000	0.0000	0.0000
5	KNeighborsRegressor	0.2669	0.0382	0.0308
6	RandomForestRegressor	0.8719	0.0160	0.0137
7	ExtraTreesRegressor	1.0000	0.0000	0.0000
8	AdaBoostRegressor	0.0509	0.0435	0.0404
9	XGBRegressor	0.2225	0.0393	0.0343
10	LGBMRegressor	0.1521	0.0411	0.0364

	Model	R-squared on Test	RMSE on Test	MAE on Test
0	LinearRegression	0.0800	0.0429	0.0384
1	Lasso	0.0174	0.0444	0.0409
2	Ridge	0.0793	0.0430	0.0385
3	ElasticNet	0.0241	0.0442	0.0407
4	DecisionTreeRegressor	1.0000	0.0000	0.0000
5	KNeighborsRegressor	0.2423	0.0390	0.0318
6	RandomForestRegressor	0.8681	0.0163	0.0140
7	ExtraTreesRegressor	1.0000	0.0000	0.0000
8	AdaBoostRegressor	0.0393	0.0439	0.0410
9	XGBRegressor	0.3743	0.0354	0.0303
10	LGBMRegressor	0.2492	0.0388	0.0343

After skewing, models are performing way better. All the values have improved to positive for example AdaBoost.

KNN and Random Forest regressor are the best models. The rmse and mae has dropped alot after skewing.

Model performance comparison

Before skew

	Model	R-squared on Test	RMSE on Test	MAE on Test
0	LinearRegression	0.035570	6.641012	2.697169
1	Lasso	0.010536	6.726654	2.739582
2	Ridge	0.035442	6.641454	2.696338
3	ElasticNet	0.010657	6.726239	2.739355
4	DecisionTreeRegressor	1.000000	0.000000	0.000000
5	KNeighborsRegressor	0.241279	5.890337	2.358409
6	RandomForestRegressor	0.851113	2.609318	1.057479
7	ExtraTreesRegressor	1.000000	0.000000	0.000000
8	AdaBoostRegressor	-0.631111	8.636559	6.508548
9	XGBRegressor	0.730064	3.513415	1.833215
10	LGBMRegressor	0.362185	5.400654	2.315998

After skew

	Model	R-squared on Test	RMSE on Test	MAE on Test
0	LinearRegression	0.0800	0.0429	0.0384
1	Lasso	0.0174	0.0444	0.0409
2	Ridge	0.0793	0.0430	0.0385
3	ElasticNet	0.0241	0.0442	0.0407
4	DecisionTreeRegressor	1.0000	0.0000	0.0000
5	KNeighborsRegressor	0.2423	0.0390	0.0318
6	RandomForestRegressor	0.8681	0.0163	0.0140
7	ExtraTreesRegressor	1.0000	0.0000	0.0000
8	AdaBoostRegressor	0.0393	0.0439	0.0410
9	XGBRegressor	0.3743	0.0354	0.0303
10	LGBMRegressor	0.2492	0.0388	0.0343

We can see that after skewing our r-squared has increased which means it has helped the model better capture the relationships within the data and has improved the overall fit of the model.

Our rmse has reduced from 2.60 to 0.0163. It means the errors have become smaller. By transforming the target variable, it is more suitable for modelling and have helped the model better capture the data's distribution.

Feature Importance

	Feature	Importance
2	MktFare	0.1869
3	MktMilesFlown	0.1305
5	ARR_TIME	0.1070
4	DEP_TIME	0.1069
6	ARR_DELAY	0.0997
13	Food and Beverage	0.0349
14	In-flight Entertainment	0.0342
1	Quarter	0.0327
10	Check-in and boarding	0.0286
11	Cleanliness	0.0273
15	Legroom	0.0271
17	Value for money	0.0254
16	Seat comfort	0.0247
0	Year	0.0228
12	Customer service	0.0224
18	Satisfaction Rating	0.0203
9	LATE_AIRCRAFT_DELAY	0.0110
19	Airline_DL	0.0110
20	Airline_UA	0.0104
7	CARRIER_DELAY	0.0102
27	ARR_DELAY_bins_Small Delay	0.0040
25	Satisfaction Status_Satisfied	0.0037
26	ARR_DELAY_bins_No Delay	0.0028
24	DEPT_TIME_bin_Morning	0.0026
22	ARR_TIME_bin_Morning	0.0024
21	ARR_TIME_bin_Evening	0.0023
30	Arr_Delay_Status_Not Delayed	0.0022
23	DEPT_TIME_bin_Evening	0.0021
29	CARRIER_DELAY_bins_Small Delay	0.0019
28	CARRIER_DELAY_bins_No Delay	0.0016
8	SECURITY_DELAY	0.0001

We can see from the chart that from Random Forest Regressor,the columns with the highest importance is

MktFare,
MilesFlown,
arrival time,
departure time
arrival delay

For all the flight services like in-flight entertainment,cleanliness,legroom has lesser importance.

The binned columns also had the least importance.

K-fold validation

Before

```
Cross-Validation Mean MSE: 0.0018651758094722492
Cross-Validation Std Dev MSE: 1.0211400226066377e-05
```

After

```
Cross-Validation Mean MSE: 0.0018648735695060259
Cross-Validation Std Dev MSE: 1.0021178674432113e-05
```

By choosing the top 25 features,i used k-fold validation to check the mse. The mse increased very slightly. The standard deviation reduced suggests that the model's predictions are becoming more robust and reliable across different scenarios.

Optimization of models

```
Best Hyperparameters: {'n_estimators': 10, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': 10}
Test RMSE: 0.043263921090061455
```

The rmse got worse after tuning as i used GridSearchCV. I will not tune the model and keep the previous one with 0.000186 rmse.

Evaluation

There was no use of Binning the columns as they were the least important factors in the dataset.

Customer service such as cleanliness,in-flight entertainment were not the most important factors contributing to remaining availability of passengers.

Some of the most important factor is the arrival and departure time and also the arrival delay. We can see that passengers do not like having delayed flights. If the airline has a history of being late, Passengers are less likely to take the flight. The fare price also is the most important factor.

From here we can conclude that the hypothesis is proven wrong as the customer service is not the most important factor.

04

Siddarth

"Weather conditions are the primary contributing factor to airline delay"

Overview

Projected Outcome of Analysis:

- Analyse main leading factors that cause airline delays
- Identify if there are trends or patterns in flight delays based on time of day or day of week
- Discover if flights from certain airports are more prone to delays

Hypothesis Statement:

- Weather conditions are the primary contributing factors to airline delay

Target Column:

- Arrival Delay

Steps Taken

1. Modelling with imbalance dataset and outliers
2. K-Fold Cross Validation
3. Modelling Data with balanced dataset and no outliers
4. Feature importance and selection
5. Hyperparameter Tuning
6. Best Model Evaluation

Modelling with imbalance dataset and outliers

```
(('KNN', KNeighborsClassifier()))
(('LR', LogisticRegression(solver='liblinear')))
(('DT', DecisionTreeClassifier()))
(('GNB', GaussianNB()))
(('RF', RandomForestClassifier(n_estimators=10)))
(('GB', GradientBoostingClassifier()))
(('NN',MLPClassifier(max_iter=1000)))
```

```
names = []
scores = []
for name, model in models:
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    scores.append(accuracy_score(y_test, y_pred))
    names.append(name)

models_comparison = pd.DataFrame({'Name': names, 'Score': scores})
models_comparison
```

	Name	Score
0	KNN	0.871535
1	LR	0.994196
2	DT	0.996565
3	GNB	0.992656
4	RF	0.995203
5	GB	0.997868
6	NN	0.992597

K-Fold Cross Validation

```python
from sklearn.model_selection import KFold

names = []
scores = []
for name, model in models:
    kfold = KFold(n_splits=5, random_state=123, shuffle=True)
    score = cross_val_score(model, x_train, y_train, cv=kfold, scoring='accuracy').mean()
    names.append(name)
    scores.append(score)

kf_cross_val = pd.DataFrame({'Name': names, 'Score': scores})
kf_cross_val
```

	Name	Score
0	KNN	0.868944
1	LR	0.995662
2	DT	0.996846
3	GNB	0.993366
4	RF	0.995365
5	GB	0.997764
6	NN	0.993781

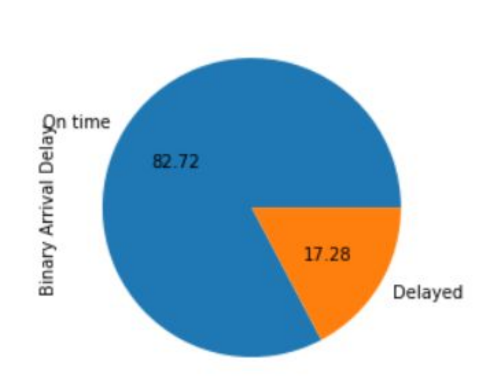
	precision	recall	f1-score	support
Delayed	0.99	1.00	0.99	2977
On time	1.00	1.00	1.00	13907
accuracy			1.00	16884
macro avg	0.99	1.00	1.00	16884
weighted avg	1.00	1.00	1.00	16884

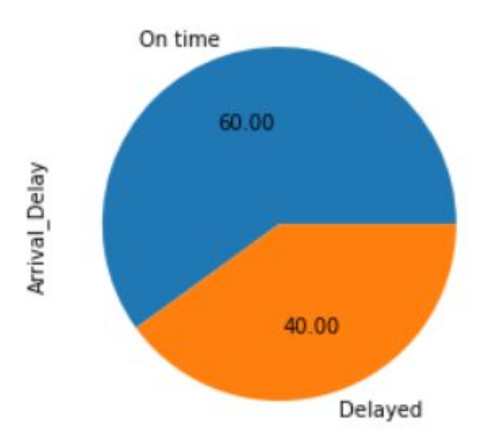
Accuracy Score: 0.997867803837953

Modelling with balanced dataset and no outliers

1. Balancing Training Dataset

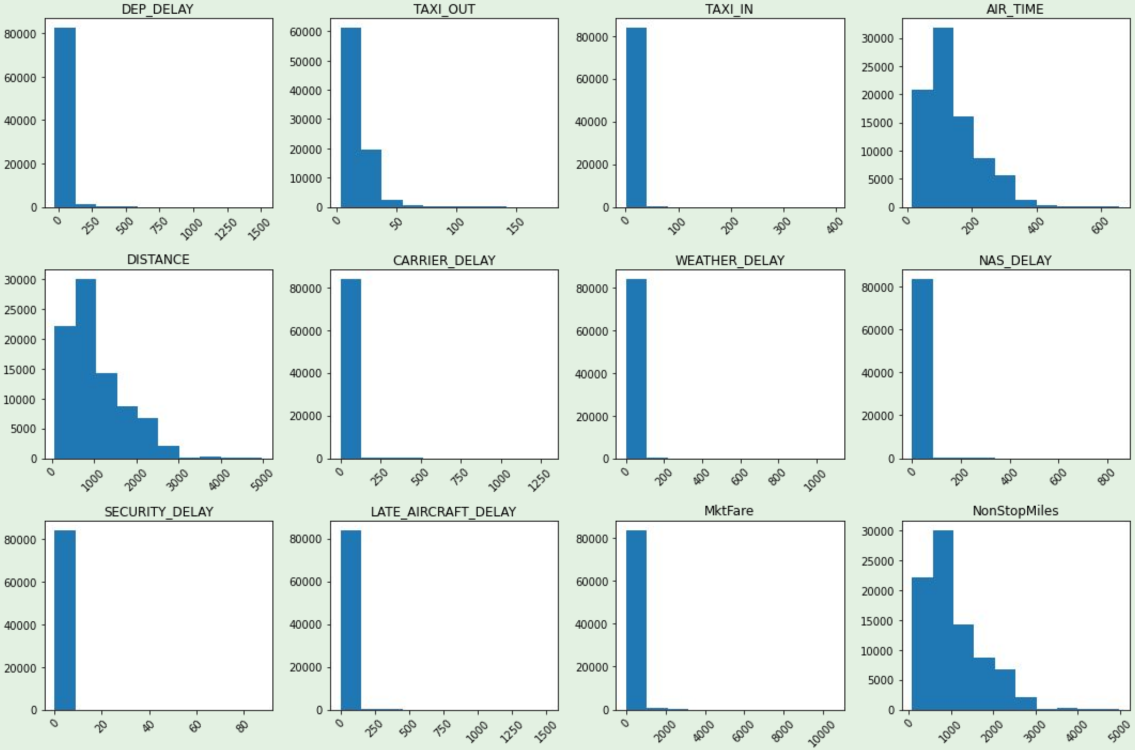
```
class_counts = Counter(y_train)
desired_class_ratio = {
    'Delayed': class_counts['Delayed'],
    'On time': int(class_counts['Delayed'] * (0.6 / 0.4))
}
rus = RandomUnderSampler(sampling_strategy=desired_class_ratio, random_state=123)

x_resampled, y_resampled = rus.fit_resample(x_train, y_train)
```





Modelling with balanced dataset and no outliers

2. Removing Outliers



```python
from sklearn.neighbors import NearestNeighbors
columns = ['DEP_DELAY', 'TAXI_OUT', 'TAXI_IN', 'AIR_TIME', 'DISTANCE',
           'CARRIER_DELAY', 'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY',
           'LATE_AIRCRAFT_DELAY', 'MktFare', 'NonStopMiles']

df_selected = df[columns]

df_normalized = (df_selected - df_selected.mean()) / df_selected.std()
```

```python
n_neighbors = 5

knn_model = NearestNeighbors(n_neighbors=n_neighbors)
knn_model.fit(df_normalized)


distances, _ = knn_model.kneighbors(df_normalized)

outlier_threshold = np.percentile(distances[:, -1], 95)

for col in columns:
    outlier_mask = distances[:, -1] > outlier_threshold
    median_value = df_selected[col].median()
    df_selected.loc[outlier_mask, col] = median_value


df_no_outliers = df.copy()
df_no_outliers[columns] = df_selected
```

Modelling with balanced dataset and no outliers

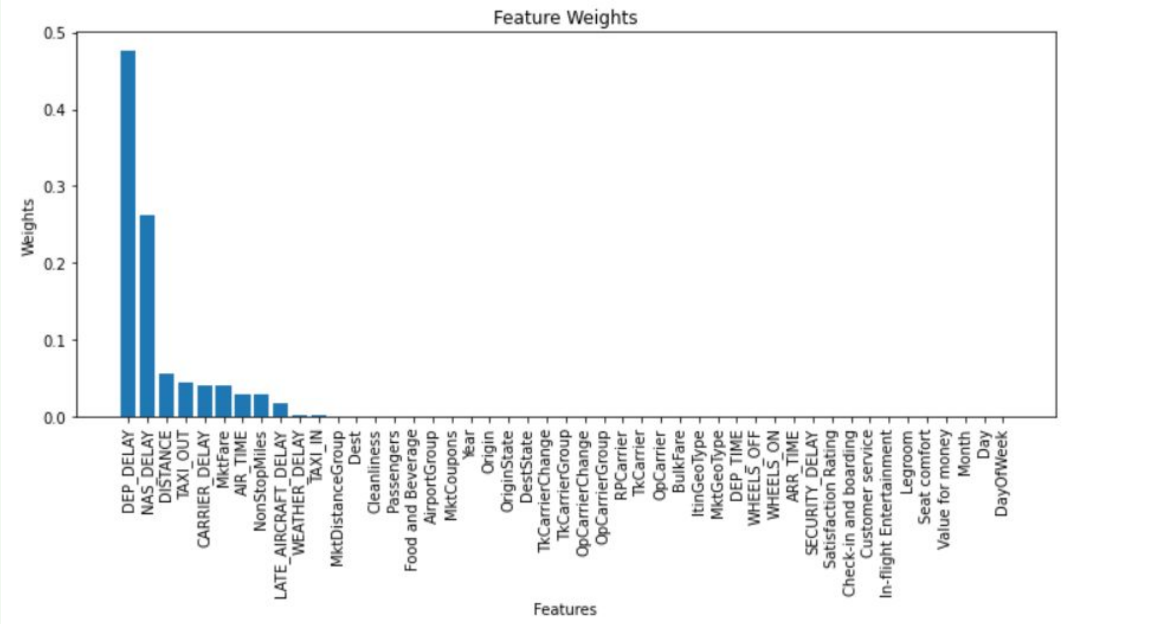
3. Modelling

```
names = []
scores = []
for name, model in models:
    model.fit(x_resampled, y_resampled)
    y_pred = model.predict(x_test)
    scores.append(accuracy_score(y_test, y_pred))
    names.append(name)

models_comparison = pd.DataFrame({'Name': names, 'Score': scores})
models_comparison
```

	Name	Score
0	KNN	0.852405
1	LR	0.956053
2	DT	0.991175
3	GNB	0.944918
4	RF	0.988806
5	GB	0.992656
6	NN	0.967721

```
              precision    recall  f1-score   support

     Delayed       0.96      1.00      0.98      2977
     On time       1.00      0.99      1.00     13907

    accuracy                           0.99     16884
   macro avg       0.98      1.00      0.99     16884
weighted avg       0.99      0.99      0.99     16884

Accuracy Score: 0.9926557687751718
```

Feature Importance and Selection




```python
# Select features of weight more than 0.002
selectedFeatures = []

for item in features_weight:
  if item[1] >0.002:
    selectedFeatures.append(item)

selectedFeatures
```

```
Out[71]: [('MktFare', 0.04061988310477336),
 ('NonStopMiles', 0.02877296667710131),
 ('DEP_DELAY', 0.4767140480632015),
 ('TAXI_OUT', 0.043925553579982464),
 ('AIR_TIME', 0.029490879253899625),
 ('DISTANCE', 0.05644150880011383),
 ('CARRIER_DELAY', 0.04099767863772121),
 ('WEATHER_DELAY', 0.002397074933395551),
 ('NAS_DELAY', 0.261509603994794),
 ('LATE_AIRCRAFT_DELAY', 0.017437647638456052)]
```

Hyperparameter Tuning

```
1    modelChosen.get_params()
```

```
Out[80]: {'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'deviance',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': None,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

```
parameters = {
    "learning_rate": [0.01, 0.1, 0.2, 0.3],
    "subsample": [0.8, 0.9, 1.0],
    "n_estimators": [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000],
    "max_depth": [4, 5, 6, 7, 8, 9, 10],
    "criterion": ["friedman_mse", "mse", "mae"],
    "loss": ["deviance", "exponential"],
    "max_features": ["auto", "sqrt", "log2", None]
}
```

```
tuned_randomforest = RandomizedSearchCV(estimator = modelChosen, param_distributions = parameters, n_iter = 50, cv = 3,random_state=42, n_jobs = -1)
# Fit the random search model
tuned_randomforest.fit(x_resampled, y_resampled)
```

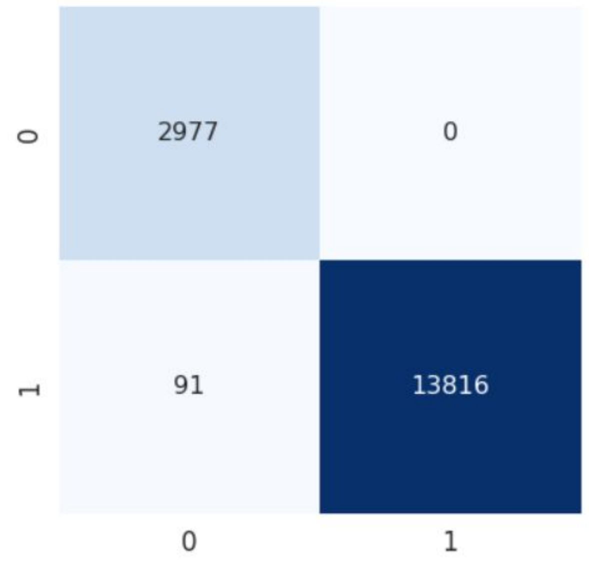
```
#get the model with best parameters
tuned_randomforest.best_params_
```

```
Out[75]:

{'subsample': 1.0,
 'n_estimators': 400,
 'max_features': 'auto',
 'max_depth': 4,
 'loss': 'deviance',
 'learning_rate': 0.3,
 'criterion': 'friedman_mse'}
```

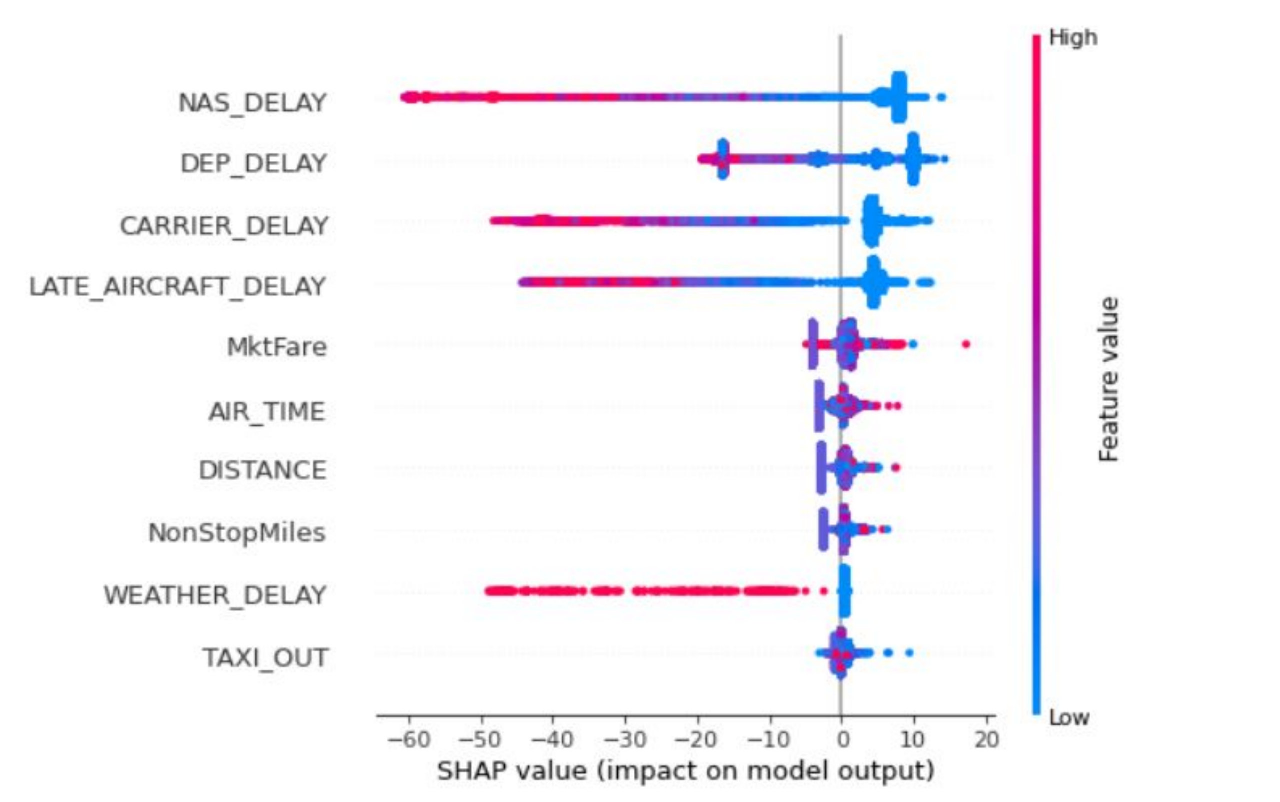
Best Model Evaluation

```python
# classification report
print(classification_report(y_test, modelChosen.predict(x_test)))

# accuracy score
y_pred = modelChosen.predict(x_test)
print("Accuracy Score: " + str(accuracy_score(y_test, y_pred)))
```

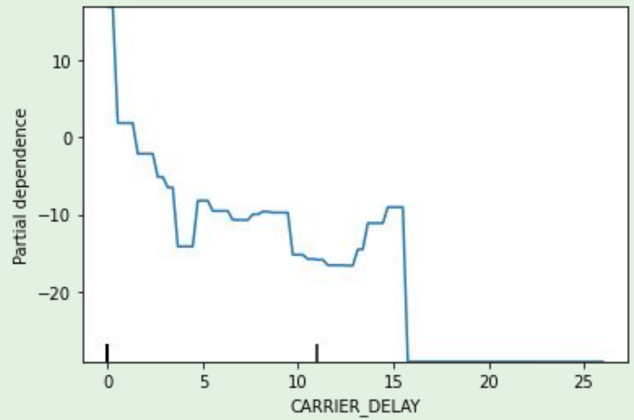
```
              precision    recall  f1-score   support

     Delayed       0.97      1.00      0.98      2977
     On time       1.00      0.99      1.00     13907

    accuracy                           0.99     16884
   macro avg       0.99      1.00      0.99     16884
weighted avg       0.99      0.99      0.99     16884

Accuracy Score: 0.9946102819237148
```

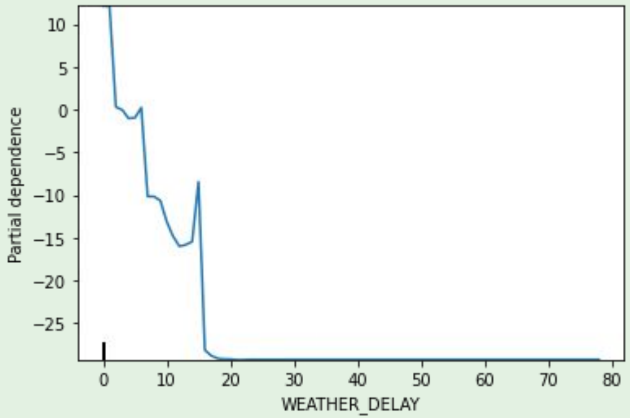


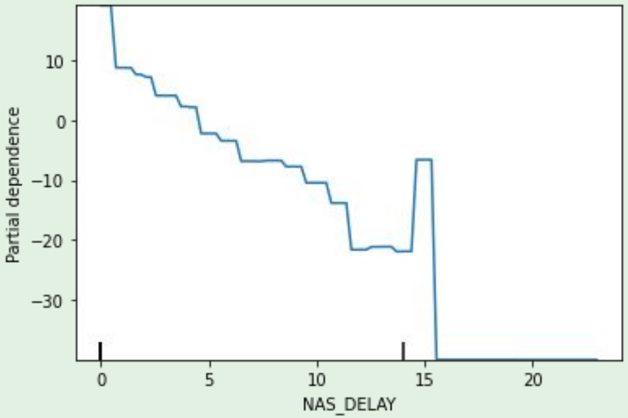
Best Model Evaluation

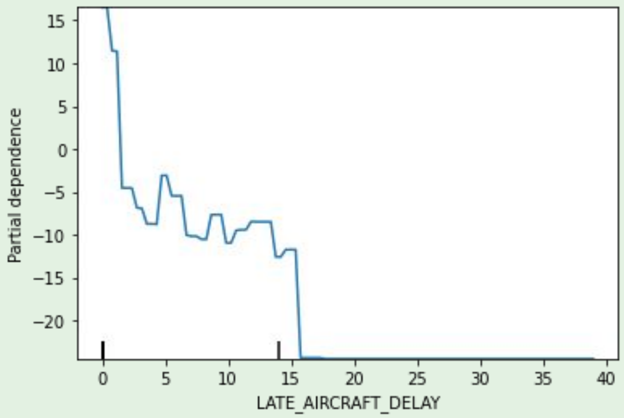


Best Model Evaluation



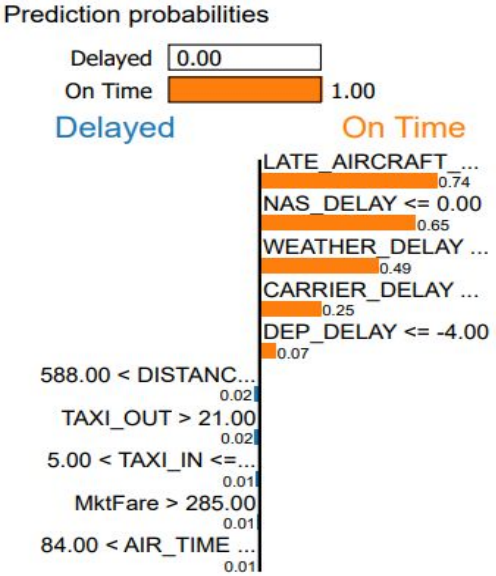






Best Model Evaluation

```
exp = explainer.explain_instance(
    data_row=x_test.iloc[3],
    predict_fn=modelChosen.predict_proba
)

exp.show_in_notebook(show_table=True)
```




```
print(x_test.iloc[3])
print(' ')
print(y_test.iloc[3])
```

```
MktFare                 518.0
NonStopMiles            733.0
DEP_DELAY                -6.0
TAXI_OUT                 22.0
TAXI_IN                   6.0
AIR_TIME                100.0
DISTANCE                733.0
CARRIER_DELAY             0.0
WEATHER_DELAY             0.0
NAS_DELAY                 0.0
LATE_AIRCRAFT_DELAY       0.0
Name: 1525, dtype: float64

On time
```

Conclusion

Hypothesis Recap:

- ‘Low prices, low delay rates with quality service will lead to higher satisfaction, generating higher revenue’.

Strategic Pricing Optimization:

- Implement dynamic pricing models that align with customer preferences, optimizing revenue without compromising customer satisfaction.

Service Excellence:

- Invest in comprehensive training for flight attendants to ensure courteous, attentive, and responsive service.

Elevated performance in flight time and reducing Delay time:

- Recognize the significance of departure delay and put more focus by being consistently on time

Operational Excellence:

- Enhance operational efficiency to minimize disruptions, improve on-time performance, and elevate passenger experience.