2023

Predicting Flight Ticket Prices

CA1 PROJECT | MACHINE LEARNING |CCT COLLEGE DUBLIN

EDUARDO J. MATOS ROMERO
SBA23298

Index

Project Motivation (56 words)

Ireland is one of the leading countries in the aviation industry, with companies such as Ryanair or Air Lingus being examples of hugely successful companies across Europe, so to base a project on any of the examples of domains in this industry as part of a portfolio of projects seems to me to be hugely valuable.

Problem Domain (81 words)

The topic that has been chosen to be analysed is the price of flight ticket in India for domestic flights, considering for the analysis a data collection from 12 national airlines, 5 sources and 6 destinations.
To analyse and making a prediction of the price of tickets it has been analysed 10 features.

- Features
  - Airline
  - Date of journey
  - Source
  - Destination
  - Route
  - Departure time
  - Arrival time
  - Duration
  - Total stops
  - Additional information

Methods (195 words)

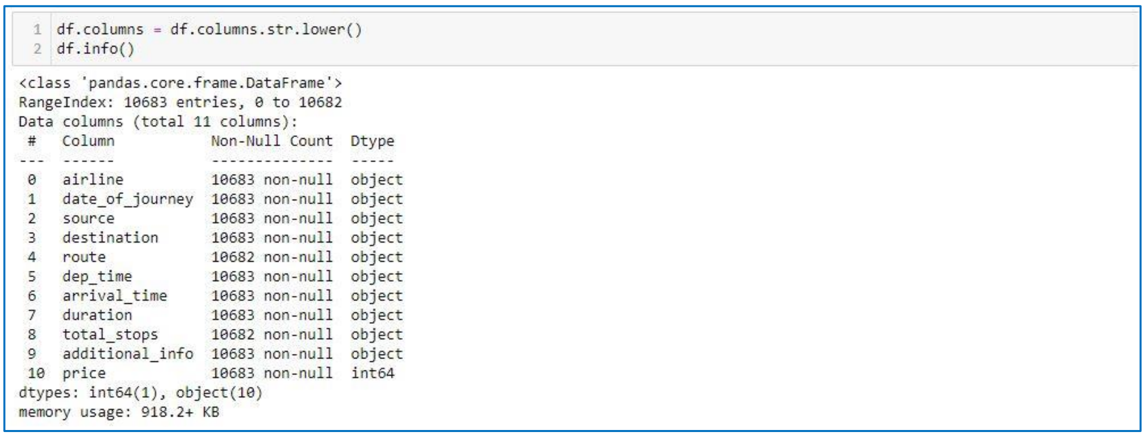
Since the objective of our project is to develop a predictive model using machine learning and Python, the method that has been decided to be applied is a regression.

1. To develop a regression model, 7 different regression algorithms were tested.
- Linear Regression
- Ridge Regression
- Lasso Regression
- **Decision Tree Regressor**
- **Random Forest Regressor**
- Support Vector Regressor
- K-Nearest Neighbours Regressor
2. The regression models were tested using 3 different approaches of data train-test splitting, with the intention of choosing the train-test splitting that develops the best performance of the algorithms.
3. The 2 regression models with the best predictive performance were hyperparameter tuned to adapt them to the particular task and conditions on which they were chosen to work.
- Decision Tree hyperparameters tuned:
- **Random Forest hyperparameters tuned:**
4. A final algorithm was chosen to develop the regression machine learning algorithm**.**
- **Random Forest hyperparameters tuned.**

Project Description (807 words)

Exploratory Data analysis

Most of the features in our dataset are in a categorical nominal format.

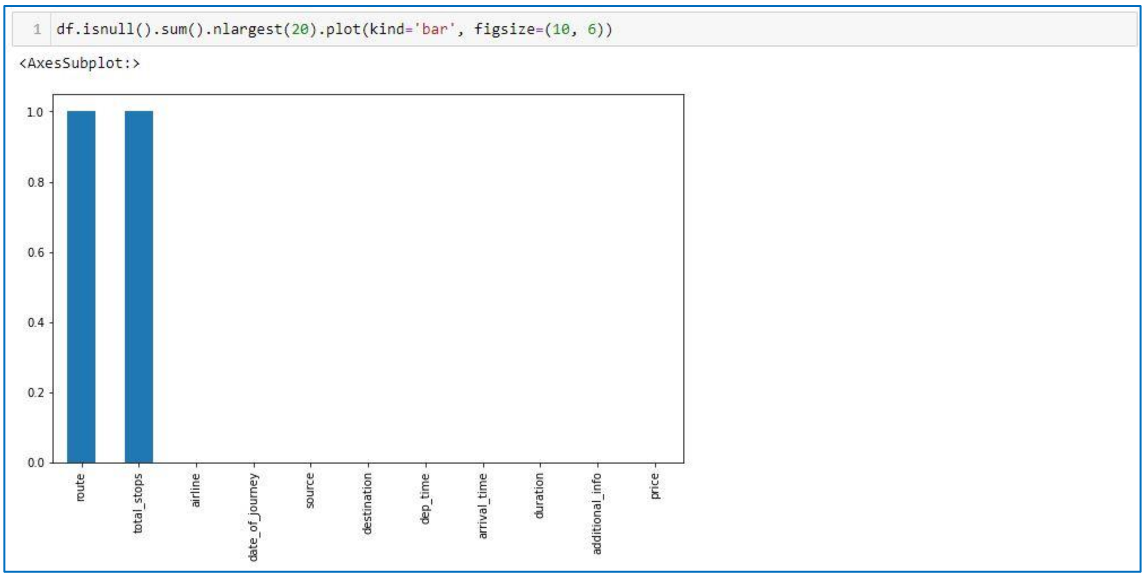


```
df.columns = df.columns.str.lower()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   airline          10683 non-null  object
 1   date_of_journey  10683 non-null  object
 2   source           10683 non-null  object
 3   destination      10683 non-null  object
 4   route            10682 non-null  object
 5   dep_time         10683 non-null  object
 6   arrival_time     10683 non-null  object
 7   duration         10683 non-null  object
 8   total_stops      10682 non-null  object
 9   additional_info  10683 non-null  object
 10  price            10683 non-null  int64
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

Illustration 1. Datatypes.

There are certain features in our dataset that have a considerably limited number of entries.



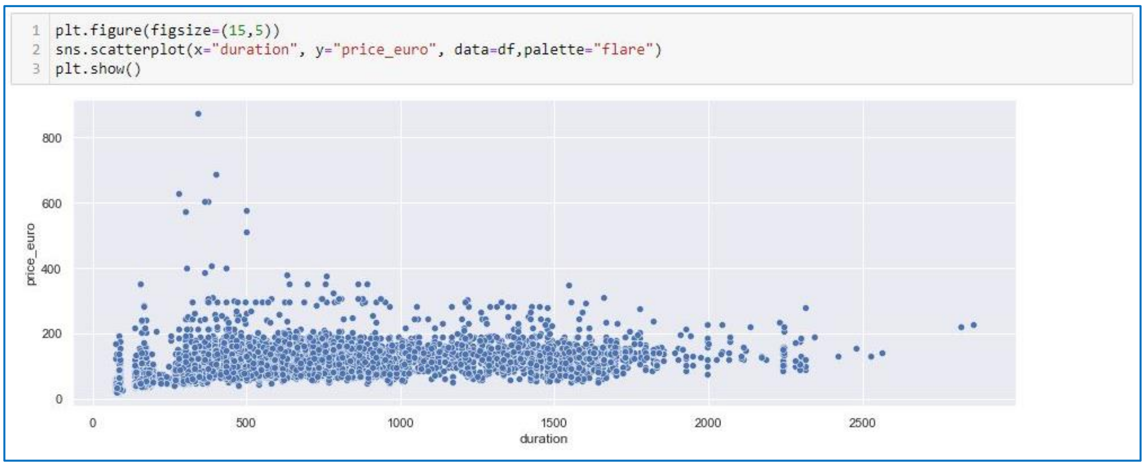
```
df.isnull().sum().nlargest(20).plot(kind='bar', figsize=(10, 6))
```

```
<AxesSubplot:>
```

Illustration 2. Number of null values.

- Does price vary with the duration of the flight?

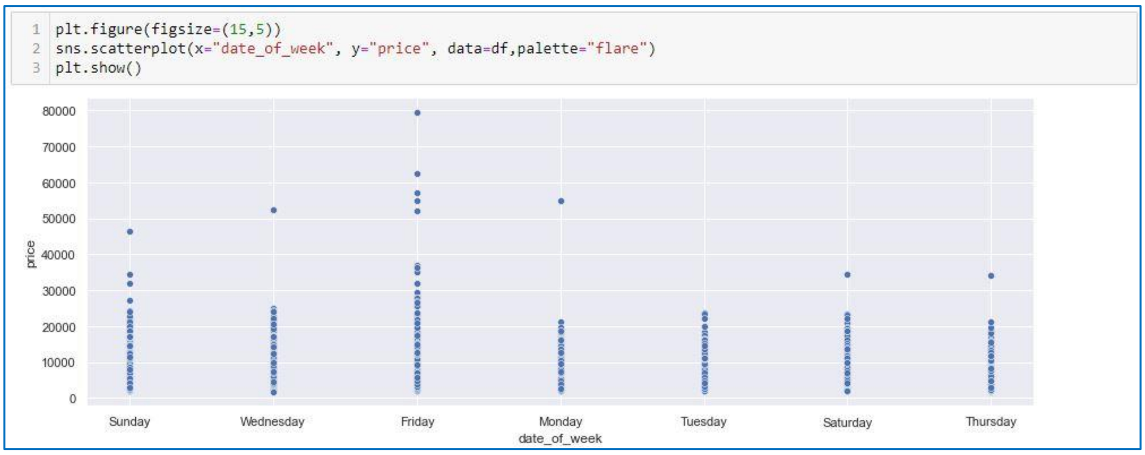
Our data shows that there is a correlation between the price and the duration of flights, where a ticket seems to be more expensive when the duration of the flight is shorter.



```python
plt.figure(figsize=(15,5))
sns.scatterplot(x="duration", y="price_euro", data=df,palette="flare")
plt.show()
```

Illustration 3. Correlation between price and duration.

- Does price vary with the date of week?
Our data shows that there is a correlation between the price and the day of the week the flights are departing, where a ticket seems to be more expensive when the are departing on Friday.

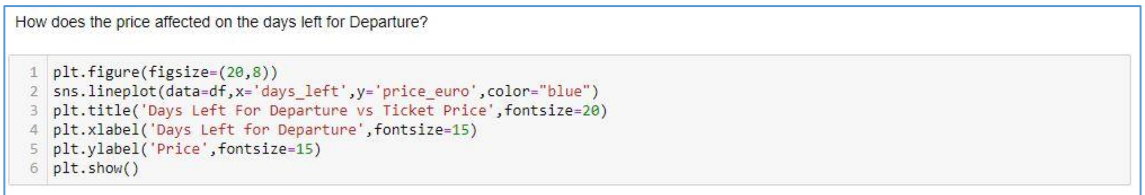


```python
plt.figure(figsize=(15,5))
sns.scatterplot(x="date_of_week", y="price", data=df,palette="flare")
plt.show()
```

Illustration 4. Correlation between price and day of departing.

- How is the price affected by the days of booking?

Prices seem to be more expensive when the flight day is closer and drastically cheaper 20 days before.


How does the price affected on the days left for Departure?

```
plt.figure(figsize=(20,8))
sns.lineplot(data=df,x='days_left',y='price_euro',color="blue")
plt.title('Days Left For Departure vs Ticket Price',fontsize=20)
plt.xlabel('Days Left for Departure',fontsize=15)
plt.ylabel('Price',fontsize=15)
plt.show()
```

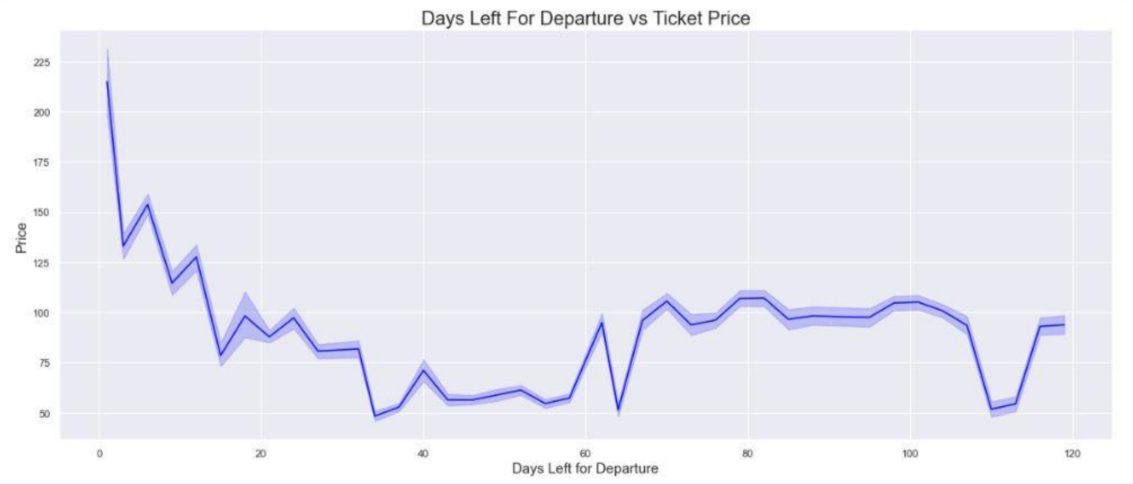


Illustration 5. Correlation between price and day of booking.

Data Normalization

Should we apply any special treatment for categorical data?

It was decided to use a label encoder approach instead of a hot encoder to avoid adding more columns to the data frame.

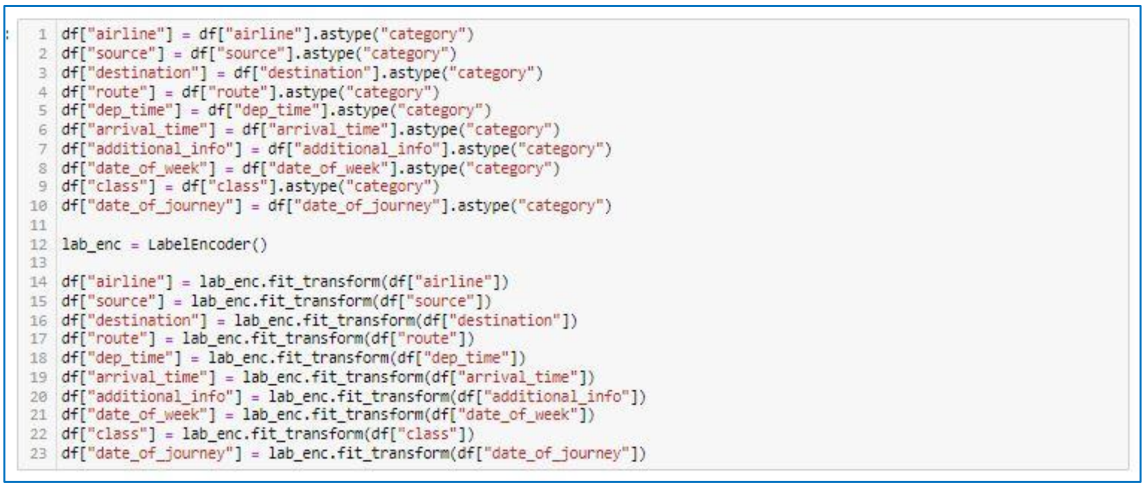

```
df["airline"] = df["airline"].astype("category")
df["source"] = df["source"].astype("category")
df["destination"] = df["destination"].astype("category")
df["route"] = df["route"].astype("category")
df["dep_time"] = df["dep_time"].astype("category")
df["arrival_time"] = df["arrival_time"].astype("category")
df["additional_info"] = df["additional_info"].astype("category")
df["date_of_week"] = df["date_of_week"].astype("category")
df["class"] = df["class"].astype("category")
df["date_of_journey"] = df["date_of_journey"].astype("category")

lab_enc = LabelEncoder()

df["airline"] = lab_enc.fit_transform(df["airline"])
df["source"] = lab_enc.fit_transform(df["source"])
df["destination"] = lab_enc.fit_transform(df["destination"])
df["route"] = lab_enc.fit_transform(df["route"])
df["dep_time"] = lab_enc.fit_transform(df["dep_time"])
df["arrival_time"] = lab_enc.fit_transform(df["arrival_time"])
df["additional_info"] = lab_enc.fit_transform(df["additional_info"])
df["date_of_week"] = lab_enc.fit_transform(df["date_of_week"])
df["class"] = lab_enc.fit_transform(df["class"])
df["date_of_journey"] = lab_enc.fit_transform(df["date_of_journey"])
```

Illustration 6. Features normalization (1).

Being aware of the problems that changing the data to a numerical ordinal format could cause, and considering that our data follows a normal distribution, a min_max_scl approach was performed after.

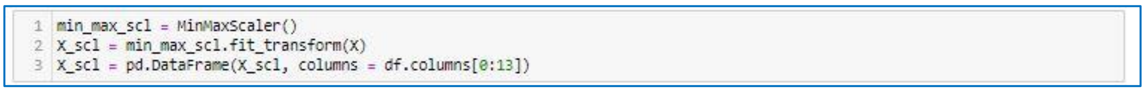
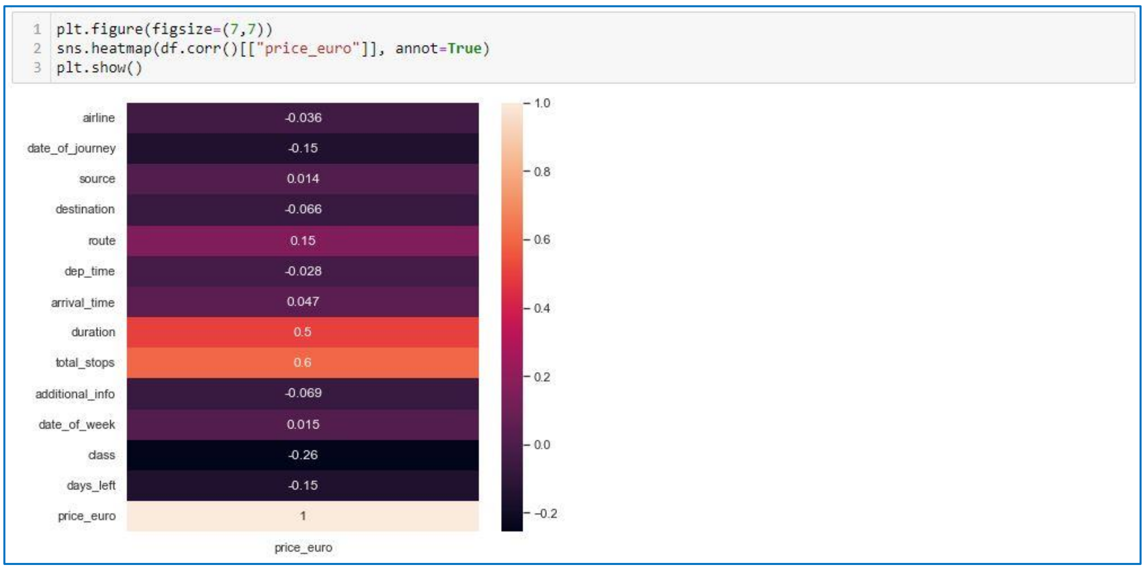

```
min_max_scl = MinMaxScaler()
X_scl = min_max_scl.fit_transform(X)
X_scl = pd.DataFrame(X_scl, columns = df.columns[0:13])
```

Illustration 7. Features normalization (2).

Data Cleaning

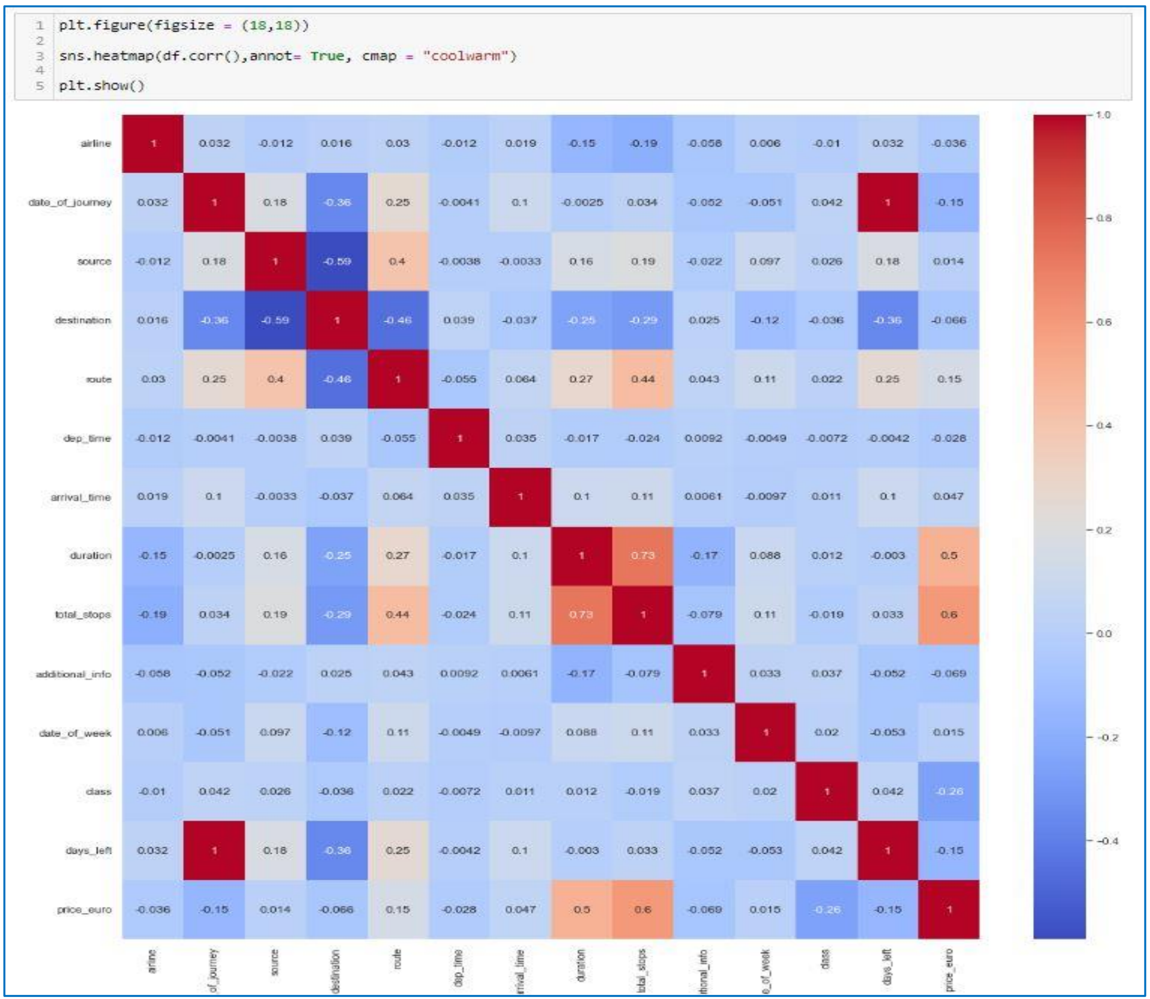
Should we delate any feature?

To analyse whether we are delating a column, it was decided to heatmap the correlation of features between them and the predictive feature, price.



```python
plt.figure(figsize=(7,7))
sns.heatmap(df.corr()[["price_euro"]], annot=True)
plt.show()
```

Illustration 8. Heatmap of the correlation of features between them and the predictive feature, price (1).



```python
plt.figure(figsize = (18,18))

sns.heatmap(df.corr(),annot= True, cmap = "coolwarm")

plt.show()
```

Illustration 9. Heatmap of the correlation of features between them and the predictive feature, price (2).

It was also decided to analyse the outliers and, after normalising the data, it turned out that the only column showing outliers was 'price_euro', which makes sense since almost all the other columns were categorical nominal values before normalisation and the outliers could correspond to the significantly lower number of business class tickets and the fact that they are significantly more expensive.

```
sns.set(style="whitegrid")

plt.figure(figsize=(12, 8))
sns.boxplot(data=df.select_dtypes(include=['int32', 'int64', 'float64']))
plt.title("Box Plot of Numerical Columns")
plt.xticks(rotation=15, ha='right')
plt.show()
```

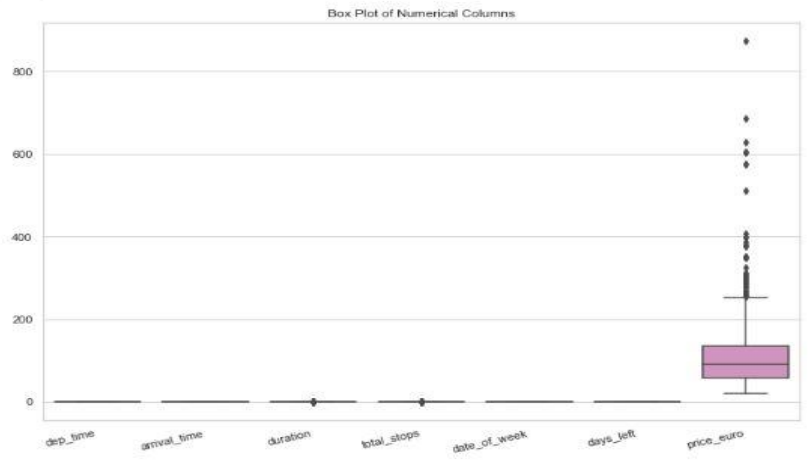
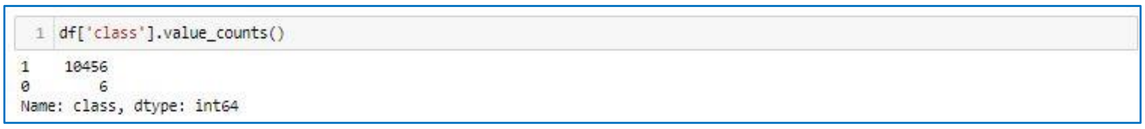


Illustration 10.Presence of outliers in features.



```
df['class'].value_counts()
```

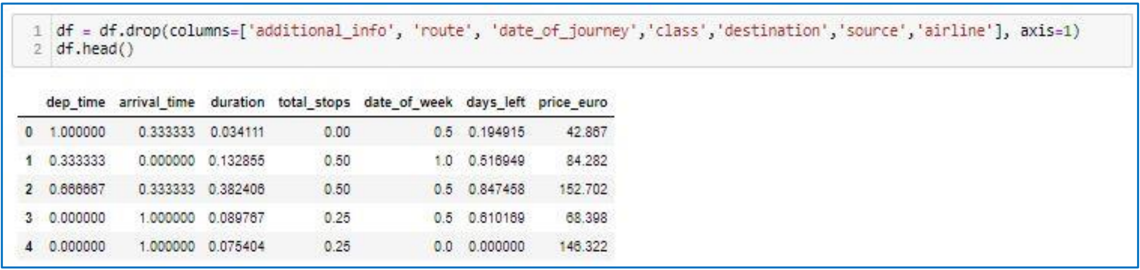
```
1    10456
0        6
Name: class, dtype: int64
```

Illustration 11. Number of business and economy tickets.
(0) Business Class | (1) Economy Class

Not further analysis have been decided to be done since we do want to keep those entries which correspond to business class tickets.

There seems to be a high correlation between duration and total stops with price, so we decide to keep these features long with departure time, arrival time, date of week and days remaining, as they may also be useful for further analysis.

Therefore, based on the analysis carried out, the following features were deleted: additional_info, route, date_of_journey, class, destination, source, and airline.



```
df = df.drop(columns=['additional_info', 'route', 'date_of_journey','class','destination','source','airline'], axis=1)
df.head()
```

	dep_time	arrival_time	duration	total_stops	date_of_week	days_left	price_euro
0	1.000000	0.333333	0.034111	0.00	0.5	0.194915	42.867
1	0.333333	0.000000	0.132855	0.50	1.0	0.516949	84.282
2	0.666667	0.333333	0.382406	0.50	0.5	0.847458	152.702
3	0.000000	1.000000	0.089767	0.25	0.5	0.610169	68.398
4	0.000000	1.000000	0.075404	0.25	0.0	0.000000	146.322

Illustration 12. Number of business and economy tickets.

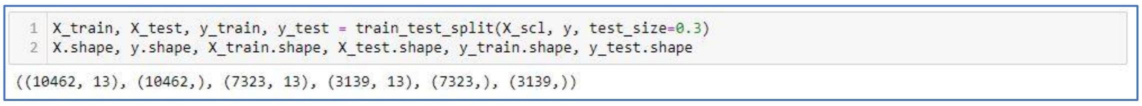
Modelling Testing

Make any difference in the algorithm performance the way the data is train-testing split?

As a way to perform the modelling testing:

- 3 different train-testing split approaches were considered, in order to see which one performs better:
  - 80-20%
  - 90-10%
  - 70-30%
- 7 different algorithms were analysed:
  - Linear Regression
  - Ridge Regression
  - Lasso Regression
  - Decision Tree Regressor
  - Random Forest Regressor
  - Support Vector Regressor
  - K-Nearest Neighbours Regressor

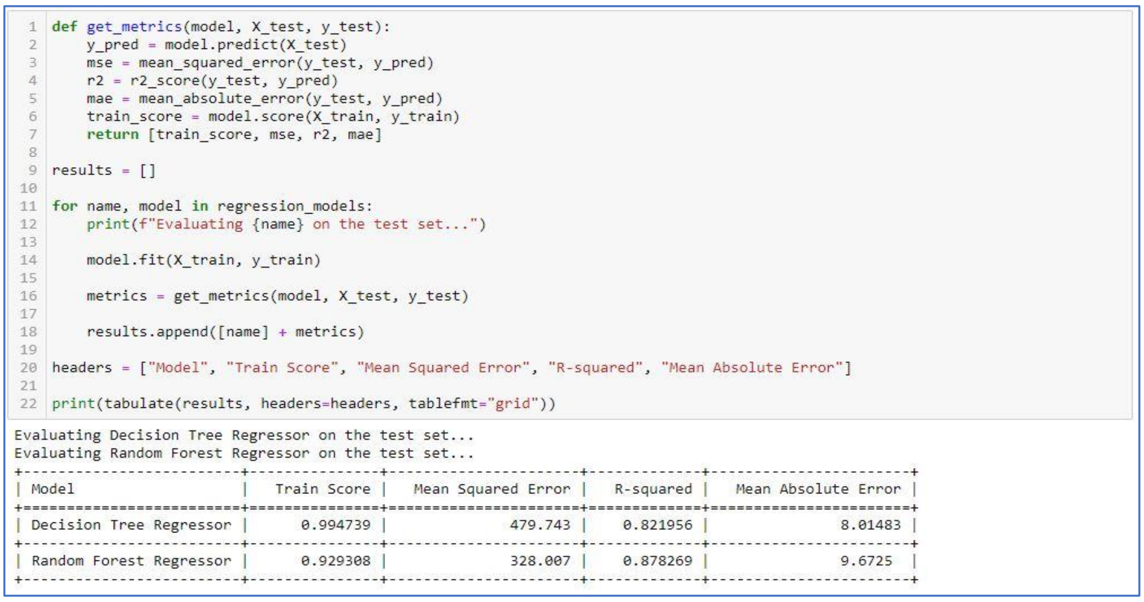
It turned out that the optimal models were Decision Tree Regressor and Random Forest Regressor, and the most effective train-testing split was 70% training and 30% testing.



```
X_train, X_test, y_train, y_test = train_test_split(X_scl, y, test_size=0.3)
X.shape, y.shape, X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((10462, 13), (10462,), (7323, 13), (3139, 13), (7323,), (3139,))
```

Illustration 13. 70-30% train-testing split.



```
def get_metrics(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    train_score = model.score(X_train, y_train)
    return [train_score, mse, r2, mae]

results = []

for name, model in regression_models:
    print(f"Evaluating {name} on the test set...")

    model.fit(X_train, y_train)

    metrics = get_metrics(model, X_test, y_test)

    results.append([name] + metrics)

headers = ["Model", "Train Score", "Mean Squared Error", "R-squared", "Mean Absolute Error"]

print(tabulate(results, headers=headers, tablefmt="grid"))
```

```
Evaluating Decision Tree Regressor on the test set...
Evaluating Random Forest Regressor on the test set...
+-------------------------+---------------+----------------------+-------------+-----------------------+
| Model                   |   Train Score |   Mean Squared Error |   R-squared |   Mean Absolute Error |
+=========================+===============+======================+=============+=======================+
| Decision Tree Regressor |      0.994739 |              479.743 |    0.821956 |               8.01483 |
+-------------------------+---------------+----------------------+-------------+-----------------------+
| Random Forest Regressor |      0.929308 |              328.007 |    0.878269 |               9.6725  |
+-------------------------+---------------+----------------------+-------------+-----------------------+
```

Illustration 14. Algorithms selection.

From the results we can see that:

- Decision tree:
  - Train score: It explain how the model generalized or fitted in the training data.
    - The model indicates that the algorithm performs exceptionally well on the training data, explaining approximately 99.5% of the variance.
  - Mean Squared Error: It explains the average squared difference between observed and predicted values.
    - The model highlights a reasonable score of 0.82, but there is still potential for improvement.

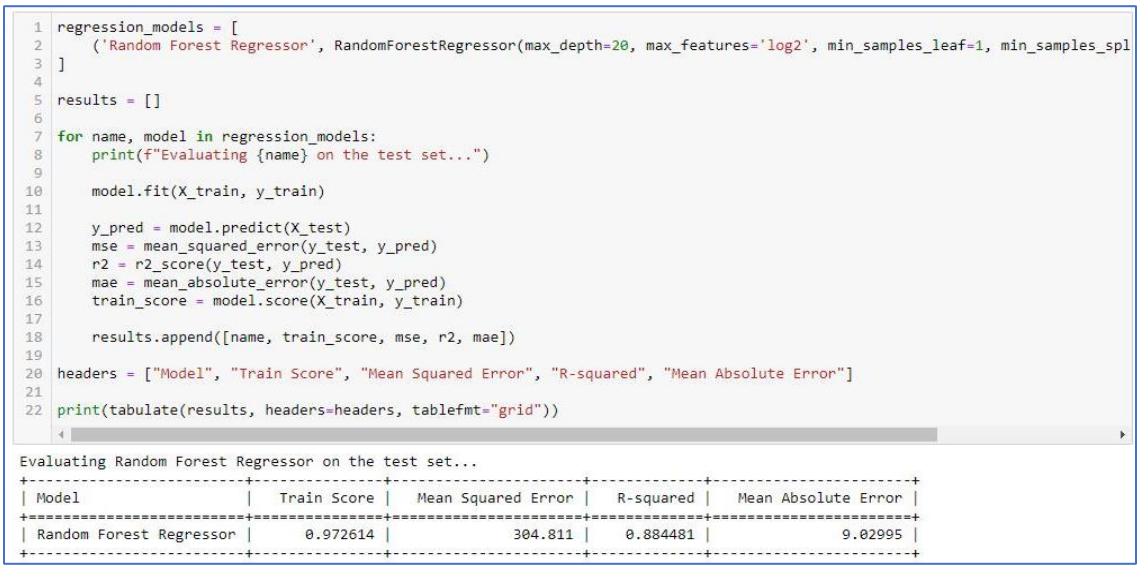
- R-squared: It shows how well the model predicts the outcome of the dependent variable.
  - In the model we can see that 0.82 is a reasonable score, but there is still scope for enhancement.
- Mean Absolute Error: It is the average variance between the significant values in the dataset and the projected values in it.
  - The analysis of the model indicates that, on average, its predictions deviate from the actual values by approximately 8 units.

- Random Forest:
  - Train score:
    - In the model, it is evident that the algorithm performs exceptionally well on the training data, accounting for approximately 92.9% of the variance.
  - Mean Squared Error:
    - Upon analysis, the model indicates that the Random Forest model's predictions are, on average, closer to the actual values than the Decision Tree, with a difference of 328.007.
  - R-squared:
    - The model shows a score of 0.88, indicating that the Random Forest model explains a higher percentage of the dependent variable's variance.
  - Mean Absolute Error
    - In the model we can see that 9.67 suggests that, on average, the Random Forest model's predictions have a slightly larger absolute difference from the actual values.

Hyperparameter Tuning

After analysing the 7 different algorithms from which the study began, it was determined that only the Decision Tree and Random Forest displayed superior performance.

Random Forest was then chosen to undergo a hyperparameter tuning analysis based on its better overall performance in previous analyses and as a prime example of an algorithm unlikely to over or underfit.



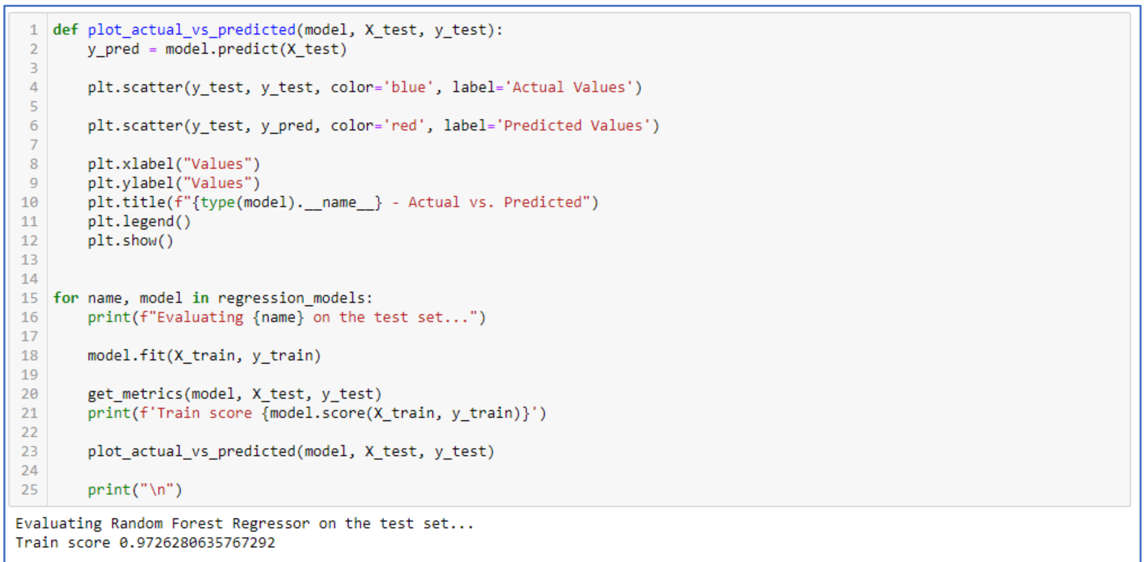
```python
regression_models = [
    ('Random Forest Regressor', RandomForestRegressor(max_depth=20, max_features='log2', min_samples_leaf=1, min_samples_spl
]

results = []

for name, model in regression_models:
    print(f"Evaluating {name} on the test set...")

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    train_score = model.score(X_train, y_train)

    results.append([name, train_score, mse, r2, mae])

headers = ["Model", "Train Score", "Mean Squared Error", "R-squared", "Mean Absolute Error"]

print(tabulate(results, headers=headers, tablefmt="grid"))
```

```
Evaluating Random Forest Regressor on the test set...
+-------------------------+---------------+----------------------+-------------+-----------------------+
| Model                   |   Train Score |   Mean Squared Error |   R-squared |   Mean Absolute Error |
+=========================+===============+======================+=============+=======================+
| Random Forest Regressor |      0.972614 |              304.811 |    0.884481 |               9.02995 |
+-------------------------+---------------+----------------------+-------------+-----------------------+
```

Illustration 15. Algorithm hyperparameters tuned performance.

After refining the hyperparameters, the Random Forest Regressor presented superior performance with an elevated R-squared (0.943) and decreased Mean Squared Error (236.418), showcasing improved predictive accuracy, and better fit to the test data relative to the prior configuration.

Upon analysing the model's performance, it is evident that it effectively predicts economic-class tickets and tickets with a low price. However, its performance is poor when dealing with business-class tickets or those priced higher than 500€. This can be attributed to the limited amount of business-class data, as previously discussed and proven.



```python
def plot_actual_vs_predicted(model, X_test, y_test):
    y_pred = model.predict(X_test)

    plt.scatter(y_test, y_test, color='blue', label='Actual Values')

    plt.scatter(y_test, y_pred, color='red', label='Predicted Values')

    plt.xlabel("Values")
    plt.ylabel("Values")
    plt.title(f"{type(model).__name__} - Actual vs. Predicted")
    plt.legend()
    plt.show()


for name, model in regression_models:
    print(f"Evaluating {name} on the test set...")

    model.fit(X_train, y_train)

    get_metrics(model, X_test, y_test)
    print(f'Train score {model.score(X_train, y_train)}')

    plot_actual_vs_predicted(model, X_test, y_test)

    print("\n")
```

```
Evaluating Random Forest Regressor on the test set...
Train score 0.9726280635767292
```

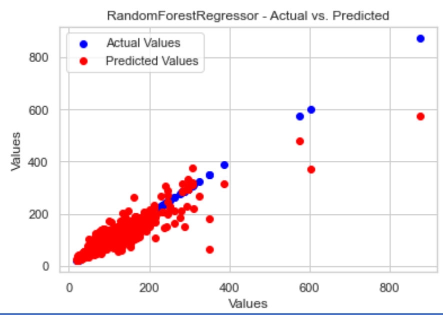
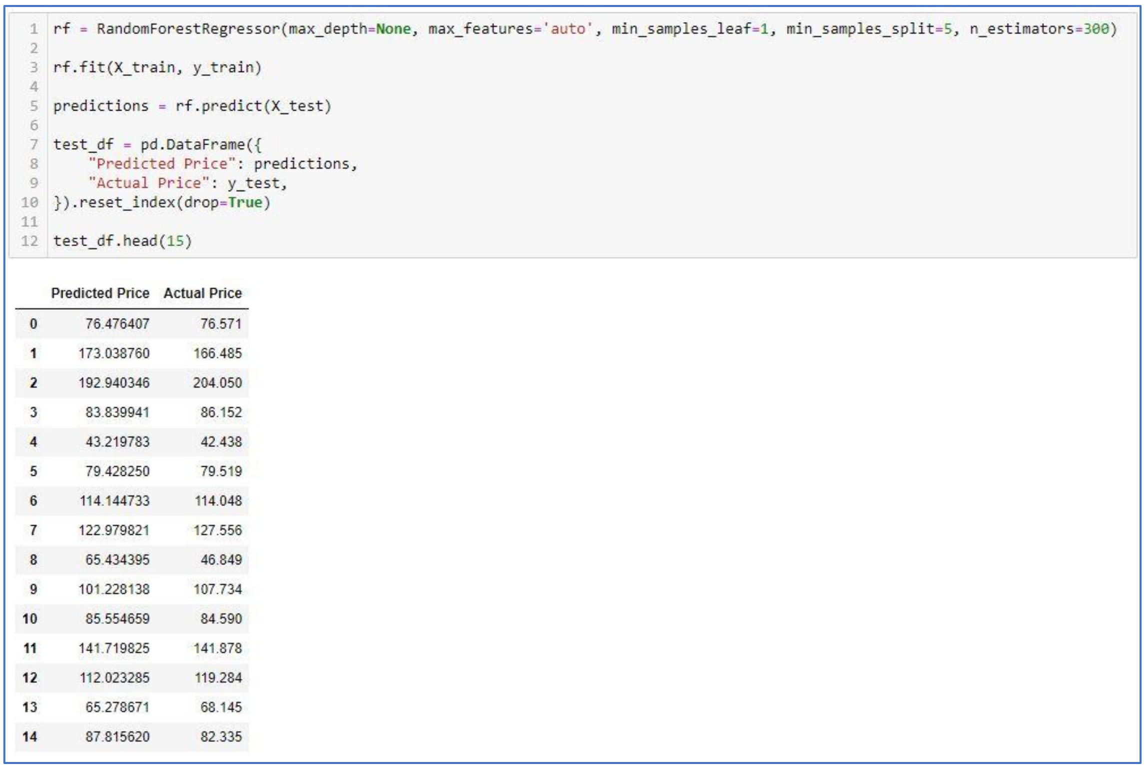


Illustration 16. Algorithm hyperparameters tuned prediction analysis.

Results

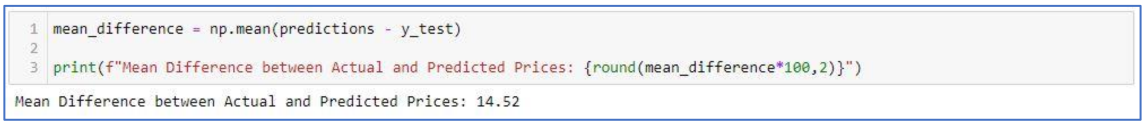
As seen in the outcome, the model achieved a flight ticket price forecast with a mere £14.52 discrepancy between the projected and factual prices.



```python
rf = RandomForestRegressor(max_depth=None, max_features='auto', min_samples_leaf=1, min_samples_split=5, n_estimators=300)

rf.fit(X_train, y_train)

predictions = rf.predict(X_test)

test_df = pd.DataFrame({
    "Predicted Price": predictions,
    "Actual Price": y_test,
}).reset_index(drop=True)

test_df.head(15)
```

	Predicted Price	Actual Price
0	76.476407	76.571
1	173.038760	166.485
2	192.940346	204.050
3	83.839941	86.152
4	43.219783	42.438
5	79.428250	79.519
6	114.144733	114.048
7	122.979821	127.556
8	65.434395	46.849
9	101.228138	107.734
10	85.554659	84.590
11	141.719825	141.878
12	112.023285	119.284
13	65.278671	68.145
14	87.815620	82.335

Illustration 17. Prediction example of 15 ticket prices.



```python
mean_difference = np.mean(predictions - y_test)

print(f"Mean Difference between Actual and Predicted Prices: {round(mean_difference*100,2)}")
```

```
Mean Difference between Actual and Predicted Prices: 14.52
```

Illustration 18.Mean Difference between Actual and Predicted Prices.

Reflective Journal (109 words)

In recent weeks, I dedicated substantial time to investigating the project's issue (30%), thoroughly scrutinising the data (45%), and constructing the model (40%). Further examination of the data highlighted an unforeseen correlation between flight duration and ticket costs, which urged me to conduct a deeper analysis. When visualising the time distribution, it became apparent that I balanced my focus well across different project phases, facilitating a comprehensive comprehension and proficient communication of discoveries. Collaborating with regression models and navigating hyperparameter tuning called for adaptability. Reflecting on this path, I have learnt to recognise the significance of planning, adaptability, and the insights gained, highlighting the iterative nature of data science.

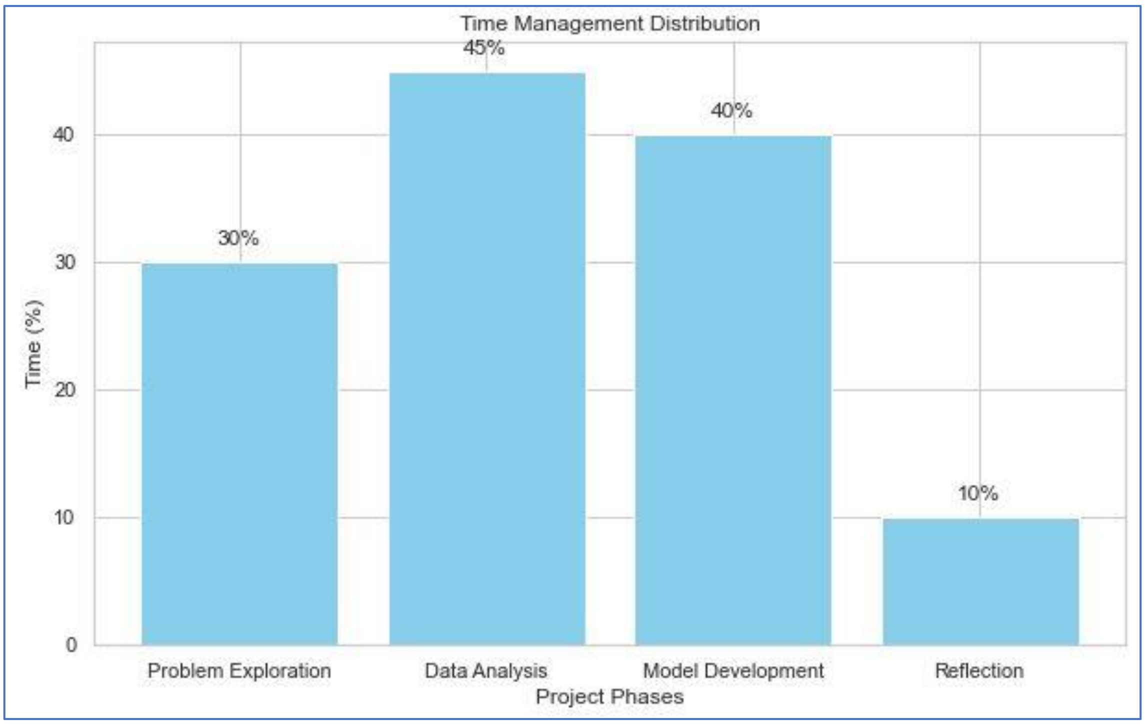


Illustration 19.Project phases | Time management distribution.

References

www.sciencedirect.com. (n.d.). Mean Absolute Error - an overview | ScienceDirect Topics. [online] Available at: https://www.sciencedirect.com/topics/engineering/mean-absolute-error.`

Jamal, T. (2021). Hyperparameter tuning in Python. [online] Medium. Available at: https://towardsdatascience.com/hyperparameter-tuning-in-python-21a76794a1f7.

Yadav, D. (2019). Categorical encoding using Label-Encoding and One-Hot-Encoder. [online] Medium. Available at: https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd.

Vincent Abba, I. (2023). What is R Squared? R2 Value Meaning and Definition. [online] freeCodeCamp.org. Available at: https://www.freecodecamp.org/news/what-is-r-squared-r2-value-meaning-and-definition/#:~:text=An%20R%2DSquared%20value%20shows.