AI Lab Report: Machine Learning

Table of contents

Table of contents
Part 1: Data Preprocessing
1.1. Understand dataset
1.2. Handle inappropriate data
1.3. Gain insight relationship between variables
Part 2: Explore Data Analysis (EDA)
Graph 1: Distribution of purchase leads
Graph 2: Top 10 countries with most booking was made
Graph 3: Percentage of complete booking
Graph 4: Percentage of people interested in 3 different trip types
Graph 5: Top 20 destinations of average flight duration
Graph 6: Booking status in each day
Graph 7: Percentage of completed bookings through Mobile and Internet
Part 3: Machine Learning Modeling
3.1. Algorithms
3.2. Data Preparation
3.3. Create Machine Learning Models
Part 4: Evaluation
4.1. Logistic Regression
4.2. Decision Tree
4.3. Random_Forrest
4.4. KNN
4.5. SVC
4.6. Evaluation table
References

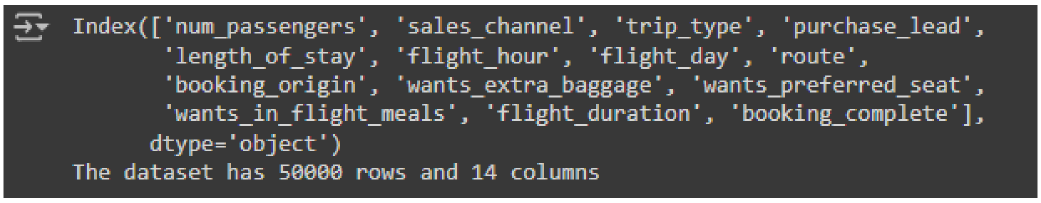
Part 1: Data Preprocessing

1.1. Understand dataset

In the data preprocessing phase, we first examined the shape and all the columns of data to understand its dimension and context

- Python code for checking

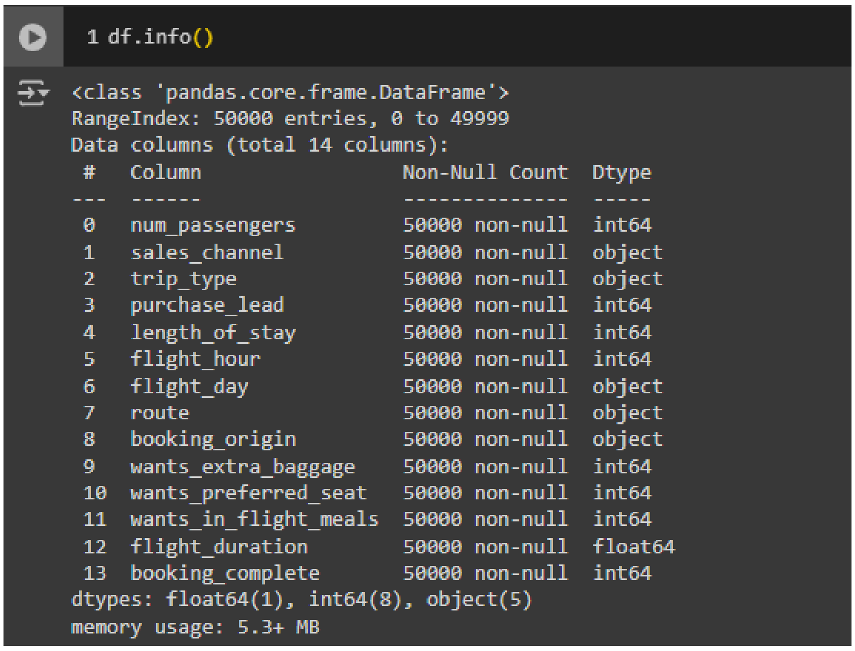
```
# Check data columns and shape
print(df.columns)
print(f'The dataset has {df.shape[0]} rows and {df.shape[1]} columns')
```



```
Index(['num_passengers', 'sales_channel', 'trip_type', 'purchase_lead',
       'length_of_stay', 'flight_hour', 'flight_day', 'route',
       'booking_origin', 'wants_extra_baggage', 'wants_preferred_seat',
       'wants_in_flight_meals', 'flight_duration', 'booking_complete'],
      dtype='object')
The dataset has 50000 rows and 14 columns
```

Then, we have to check for more information of attributes

- Python code for checking



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   num_passengers         50000 non-null  int64
 1   sales_channel          50000 non-null  object
 2   trip_type              50000 non-null  object
 3   purchase_lead          50000 non-null  int64
 4   length_of_stay         50000 non-null  int64
 5   flight_hour            50000 non-null  int64
 6   flight_day             50000 non-null  object
 7   route                  50000 non-null  object
 8   booking_origin         50000 non-null  object
 9   wants_extra_baggage    50000 non-null  int64
 10  wants_preferred_seat   50000 non-null  int64
 11  wants_in_flight_meals  50000 non-null  int64
 12  flight_duration        50000 non-null  float64
 13  booking_complete       50000 non-null  int64
dtypes: float64(1), int64(8), object(5)
memory usage: 5.3+ MB
```

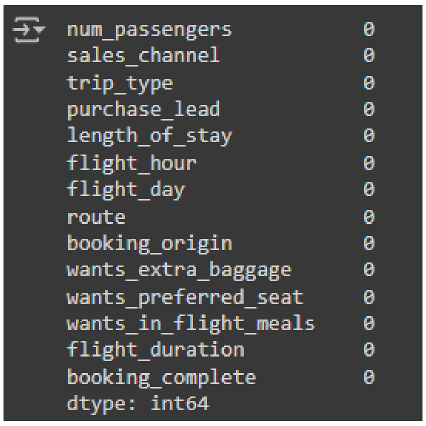
About the dataset:

- `num_passengers` = number of passengers travelling
- `sales_channel` = sales channel booking was made on
- `trip_type` = trip Type (Round Trip, One Way, Circle Trip)
- `purchase_lead` = number of days between travel date and booking date
- `length_of_stay` = number of days spent at destination
- `flight_hour` = hour of flight departure
- `flight_day` = day of week of flight departure
- `route` = origin -> destination flight route
- `booking_origin` = country from where booking was made
- `wants_extra_baggage` = if the customer wanted extra baggage in the booking
- `wants_preferred_seat` = if the customer wanted a preferred seat in the booking
- `wants_in_flight_meals` = if the customer wanted in-flight meals in the booking
- `flight_duration` = total duration of flight (in hours)
- `booking_complete` = flag indicating if the customer completed the booking

1.2. Handle inappropriate data

We checked for null values and removed them if any were found

```
# Check for null values
df.isnull().sum()
```



```
num_passengers           0
sales_channel            0
trip_type                0
purchase_lead            0
length_of_stay           0
flight_hour              0
flight_day               0
route                    0
booking_origin           0
wants_extra_baggage      0
wants_preferred_seat     0
wants_in_flight_meals    0
flight_duration          0
booking_complete         0
dtype: int64
```

Next, we checked for duplicate values and dropped them to ensure data integrity

```
# Check for duplicates
dup_vals = df.duplicated().sum()
print(f'There are {dup_vals} duplicate values in the dataset')
```

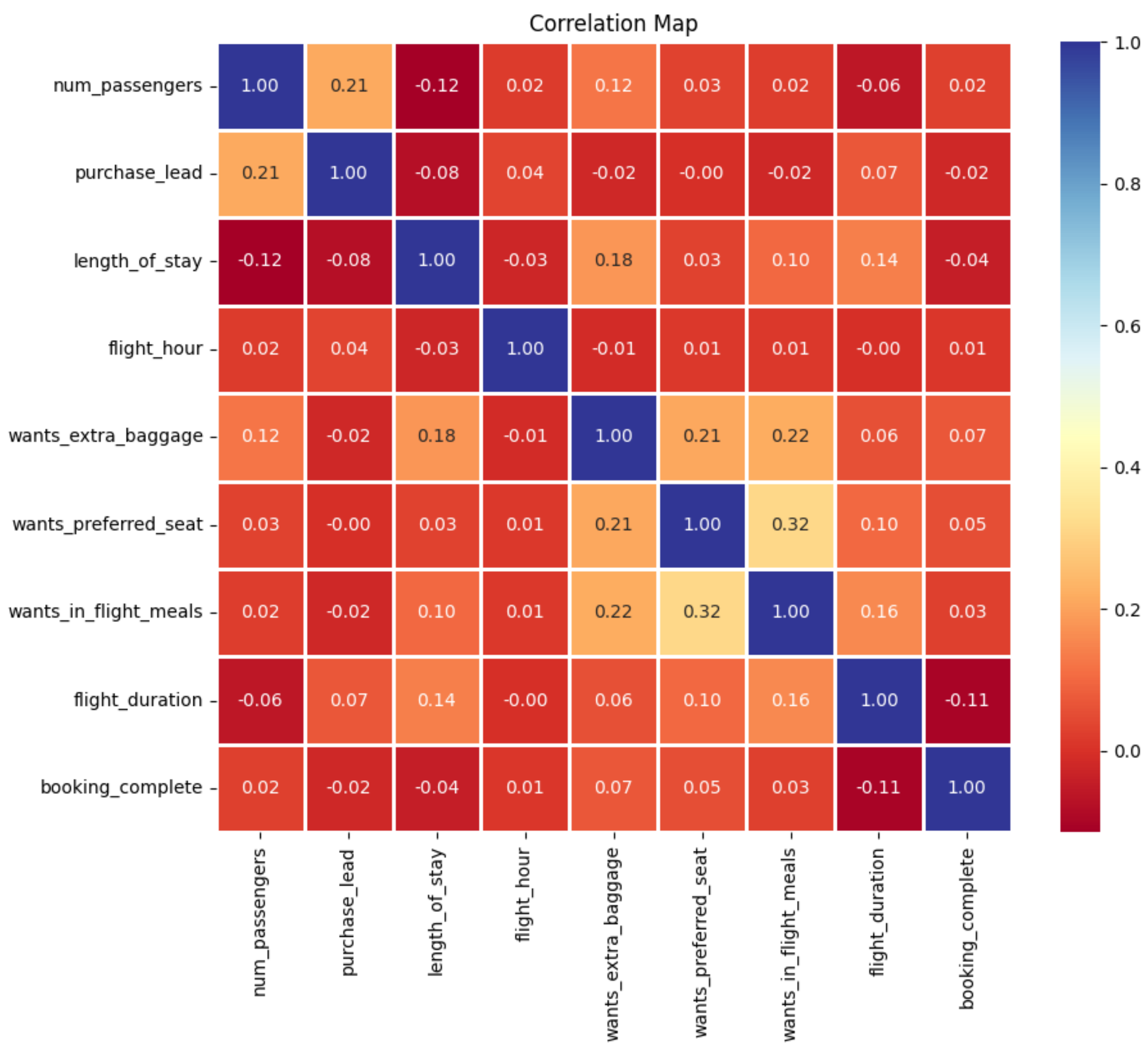
```
#Drop the duplicates from the dataset
df.drop_duplicates(inplace=True)
print(df.shape)
```

```
There are 719 duplicate values in the dataset
(49270, 14)
```

1.3. Gain insight relationship between variables

We visualized the correlation map to get this information since this visualization allowed us to identify patterns and dependencies among the features in the dataset, also helping us understand the interplay between various attribute

```
# Plot the correlation map
numeric_df = df.select_dtypes(include=[float, int])
plt.figure(figsize=(10, 8))
sns.heatmap(numeric_df.corr(), annot=True, cmap="RdYlBu", fmt='.2f',
            annot_kws=None,
            linewidths=1,
            )
plt.title("Correlation Map")
plt.show()
```



	num_passengers	purchase_lead	length_of_stay	flight_hour	wants_extra_baggage	wants_preferred_seat	wants_in_flight_meals	flight_duration	booking_complete
num_passengers	1.00	0.21	-0.12	0.02	0.12	0.03	0.02	-0.06	0.02
purchase_lead	0.21	1.00	-0.08	0.04	-0.02	-0.00	-0.02	0.07	-0.02
length_of_stay	-0.12	-0.08	1.00	-0.03	0.18	0.03	0.10	0.14	-0.04
flight_hour	0.02	0.04	-0.03	1.00	-0.01	0.01	0.01	-0.00	0.01
wants_extra_baggage	0.12	-0.02	0.18	-0.01	1.00	0.21	0.22	0.06	0.07
wants_preferred_seat	0.03	-0.00	0.03	0.01	0.21	1.00	0.32	0.10	0.05
wants_in_flight_meals	0.02	-0.02	0.10	0.01	0.22	0.32	1.00	0.16	0.03
flight_duration	-0.06	0.07	0.14	-0.00	0.06	0.10	0.16	1.00	-0.11
booking_complete	0.02	-0.02	-0.04	0.01	0.07	0.05	0.03	-0.11	1.00

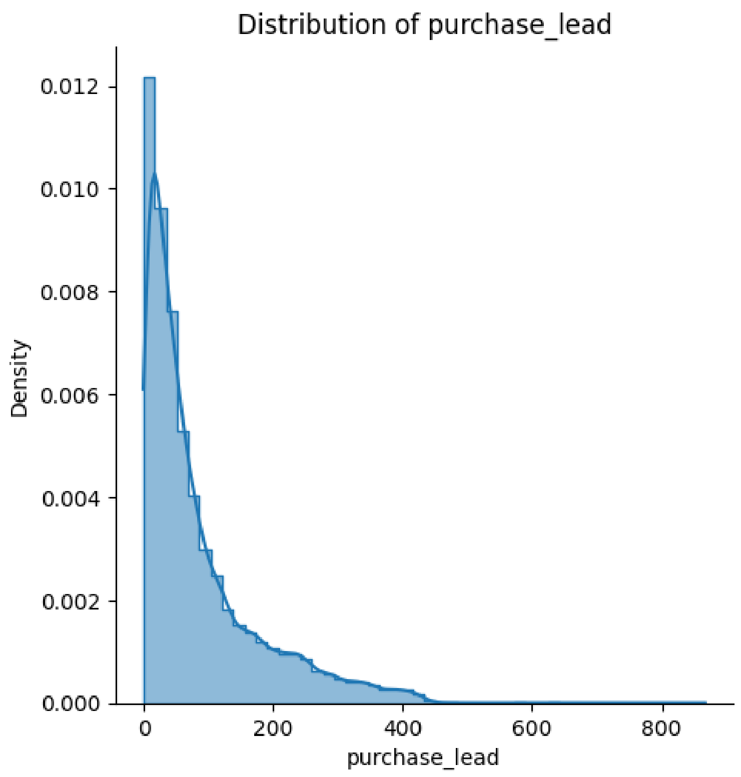
Dataset Summary:

- `Data Size`: The dataset has 49281 rows and 14 columns
- `Data Types`: There are features of type int64, float64, object and binary
- There are no missing values in the dataset
- `Unique Values`: There are varies but we can know the unique value in **flight_day** at the first glance to process.
- `Statistical Summary`: The statistical summary of the dataset shows that there are potential outliers or anomalies in the dataset.
- `Irrelavant Features`: The dataset has 14 features, but it seem like all the features are important and useful for the evaluation.

Part 2: Explore Data Analysis (EDA)

Graph 1: Distribution of purchase leads

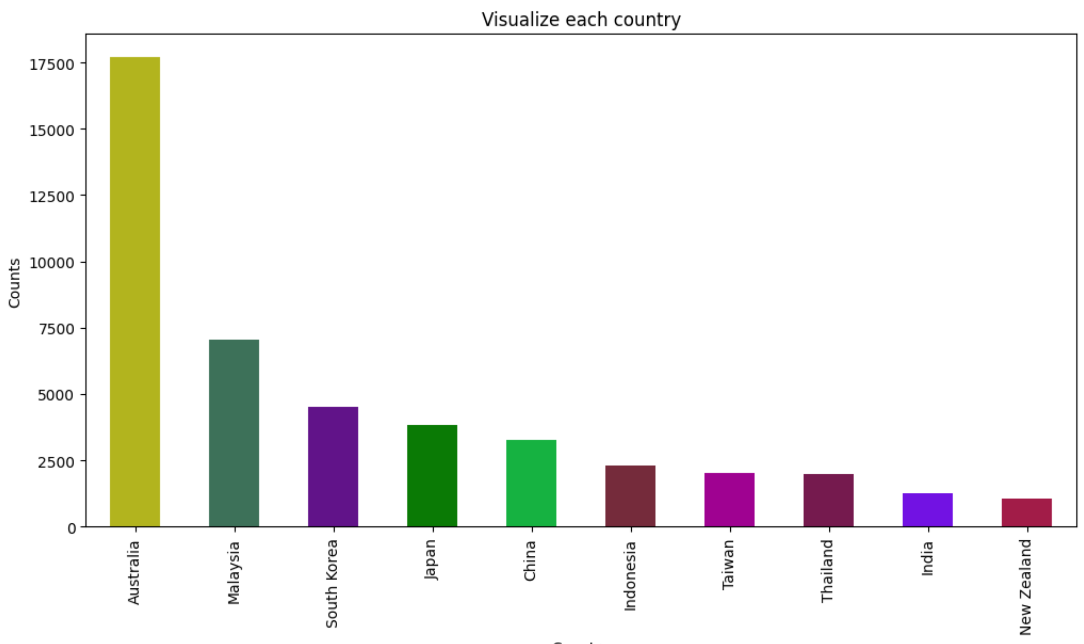
```
# Distribution of the number of purchase leads
plt.figure(figsize=(16, 8))
sns.displot(df['purchase_lead'], stat='density', kind='hist', kde=True, bins=50,
            element='step', fill=True)
plt.title("Distribution of purchase_lead")
plt.show()
```



This distribution plot revealed that the majority of purchase leads fall within the range of 200 to 400

Graph 2: Top 10 countries with most booking was made

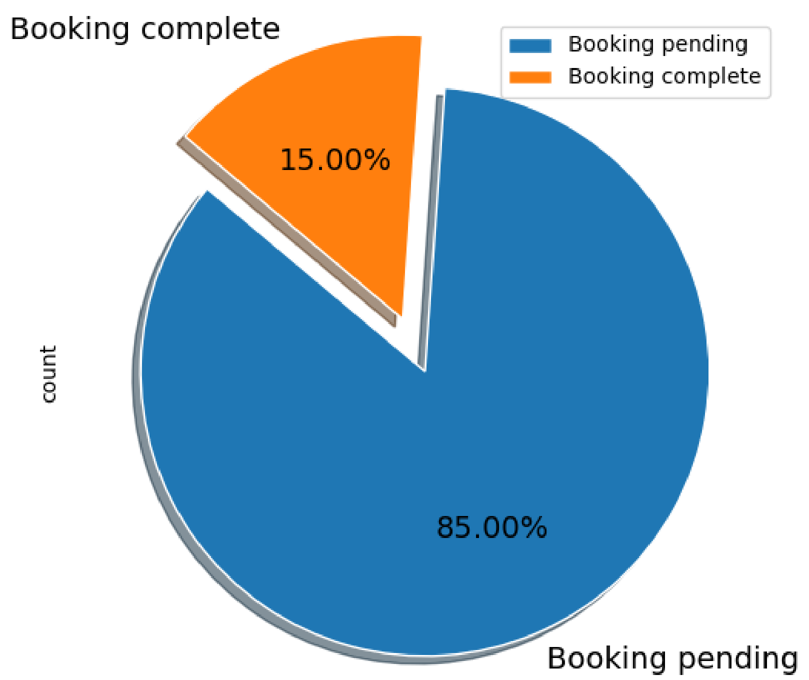
```
# Plot a bar chart to show the top 10 countries with most booking_origin
top_origins = df['booking_origin'].value_counts().nlargest(10)
top_origins.plot(kind='bar', figsize=(12, 6), color=generate_random_colors(len(t
plt.title("Visualize each country")
plt.xlabel("Country")
plt.xticks(rotation=90)
plt.ylabel("Counts")
plt.show()
```



From the graph, Australia is recorded the highest number of booking was made. followed by Malaysia in the 2nd place

Graph 3: Percentage of complete booking

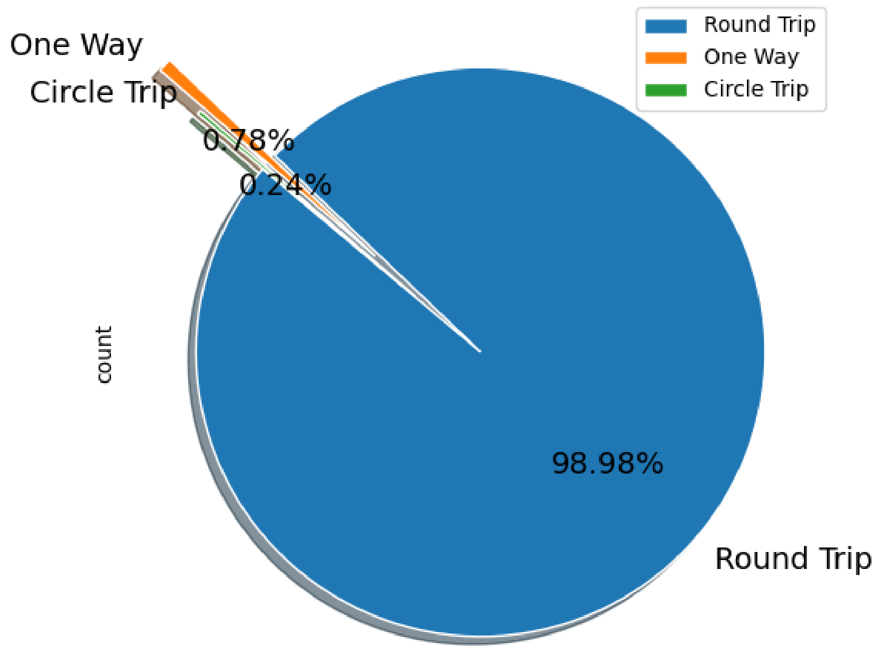
```
df['booking_complete'].value_counts().plot(kind='pie',
                                           explode=[0, 0.2],
                                           labels = ["Booking pending", "Booking
                                           colors=['#1f77b4', '#ff7f0e'],
                                           fontsize=14,
                                           wedgeprops={'linewidth': 1, 'edgecolo
                                           textprops={'fontsize': 14},
                                           legend=True,
                                           autopct='%1.2f%%',
                                           shadow=True,
                                           startangle=140,
                                           figsize=(12, 6))
plt.show()
```



From the pie chart, only 15% of the ticker bookings is completed, indicating that 85% is pending

Graph 4: Percentage of people interested in 3 different trip types

```
df['trip_type'].value_counts().plot(kind='pie',
                                    explode=[0, 0.5, 0.3],
                                    labels = ["Round Trip", "One Way", "C
                                    colors=['#1f77b4', '#ff7f0e', '#2ca02
                                    fontsize=14,
                                    wedgeprops={'linewidth': 1, 'edgecolo
                                    textprops={'fontsize': 14},
                                    legend=True,
                                    autopct='%1.2f%%',
                                    shadow=True,
                                    startangle=140,
                                    figsize=(12, 6))
plt.show()
```



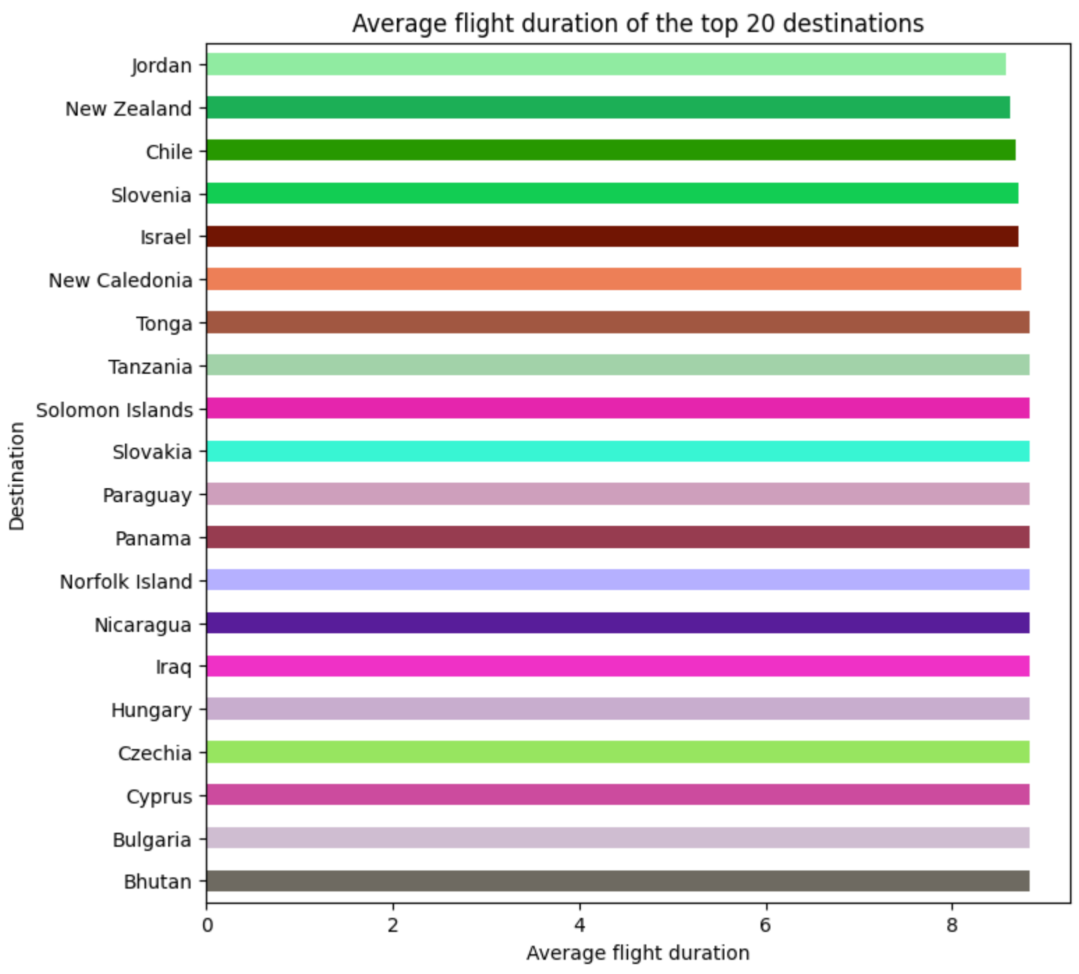
Since the Round Trip is significant high, we will create the dataframe for RoundTrip and find which day most of the ticket

```
roundtrip = df[df['trip_type'] == 'RoundTrip']
roundtrip.groupby('flight_day')['booking_complete'].value_counts().sort_values(a
.style.background_gradient(cmap='PuBuGn_r')
```

booking_complete	0	1
flight_day		
Fri	5647	971
Mon	6724	1191
Sat	4809	853
Sun	5465	911
Thu	6151	1104
Tue	6350	1114
Wed	6267	1222

Graph 5: Top 20 destinations of average flight duration

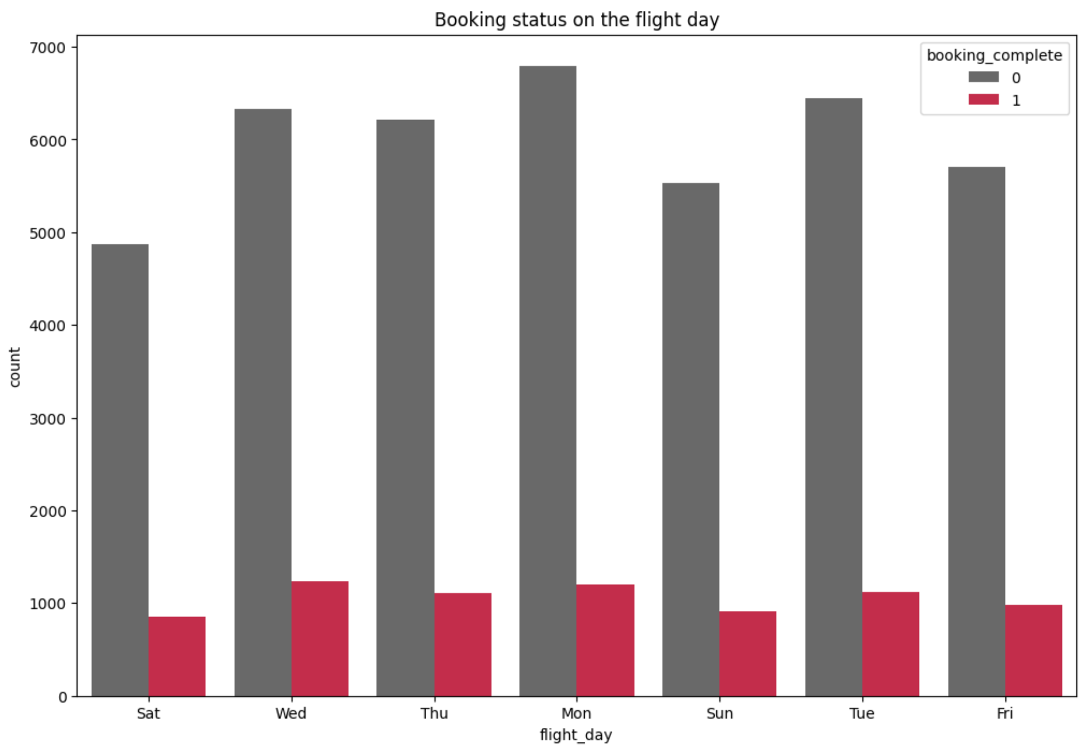
```
# Find the average flight duration of the top 20 destinations
avg_flight = df.groupby('booking_origin')['flight_duration'].mean().nlargest(20)
avg_flight_colors = generate_random_colors(len(avg_flight))
avg_flight.plot(kind='barh', figsize=(8, 8), color=avg_flight_colors)
plt.title("Average flight duration of the top 20 destinations")
plt.xlabel("Average flight duration")
plt.ylabel("Destination")
plt.show()
```



The plot indicates that countries like Jordan and New Zealand had a higher average flight duration compared to others

Graph 6: Booking status in each day

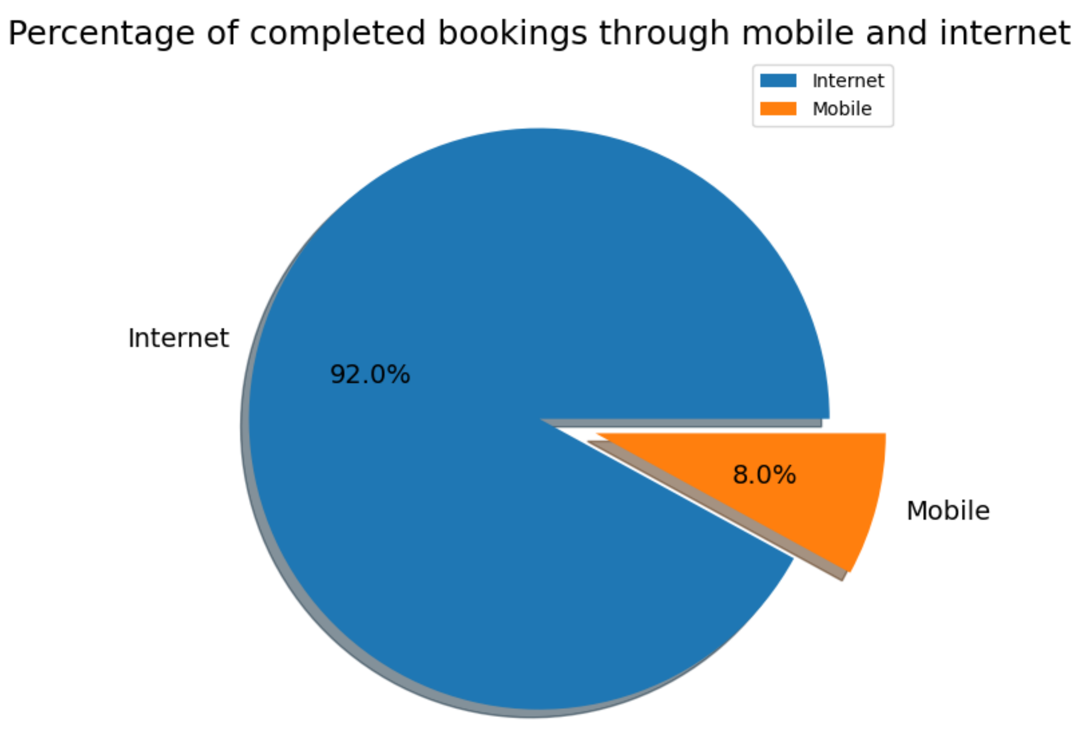
```
# Create countplot to understand the booking status on the flight day
plt.figure(figsize=(12, 8))
sns.countplot(data=df, x='flight_day', hue='booking_complete', palette=['dimgrey
plt.title("Booking status on the flight day")
plt.show()
```



Graph 7: Percentage of completed bookings through Mobile and Internet

```
# Compare % of completed bookings through mobile and internet
mobinet = df.groupby('sales_channel')['booking_complete'].sum()
plt.figure(figsize=(12, 6))
plt.pie(mobinet, labels=['Mobile', 'Internet'], colors=['#1f77b4', '#ff7f0e'],
        autopct='%1.1f%%',
        shadow=True,
        explode=[0, 0.2],
        textprops={'fontsize': 14})

plt.title("Percentage of completed bookings through mobile and internet", fontsi
plt.tight_layout()
plt.legend()
plt.show()
```



The pie chart shows that 92% of bookings were completed through Internet, while 8% were completed via mobile devices

Part 3: Machine Learning Modeling

3.1. Algorithms

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
```

3.2. Data Preparation

First, we will utilize the label encoder to convert categorical columns into numerical values, enabling us to work with these features in our machine learning models

```
# Convert categorical columns to numeric
for col in df.select_dtypes(include='object').columns:
    label_encoder = LabelEncoder()
    label_encoder.fit(df[col].unique())
    df[col] = label_encoder.transform(df[col])
```

Next, we will split the data into independent and dependent variables. To ensure uniformity in the data, we applied normalization techniques

```
# Divided the data into dependent and independent variables
X = df.drop(['booking_complete', 'purchase_lead', 'route'], axis=1)
y = df['booking_complete']
#Scaling data
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Subsequently, we split data into training and testing sets, reserving 20% of data for testing purposes, thus allowing us to evaluate the model’s performance on unseen data.

```
#Split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((39424, 11), (9857, 11), (39424,), (9857,))
```

3.3. Create Machine Learning Models

With this function, we could apply various classification algorithms to the data and compare their performance to determine the most suitable model

```
# Create a function for machine learning model
def model_building(model, X_train, X_test, y_train, y_test):
    # Model name and its accuracy
    print(f"Name of the model: {model.__class__.__name__}")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    print(f"Accuracy score: {score}")
    print('----' * 22)

     # Classification Report
    report = classification_report(y_test, y_pred, output_dict=True)
    report_df = pd.DataFrame(report).transpose()
    report_df = report_df[['precision', 'recall', 'f1-score', 'support']]
    styled_report = report_df.style.background_gradient(cmap='Blues').format(pre
    print('Classification report:')
    display(styled_report)
    print('----' * 22)

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()
```

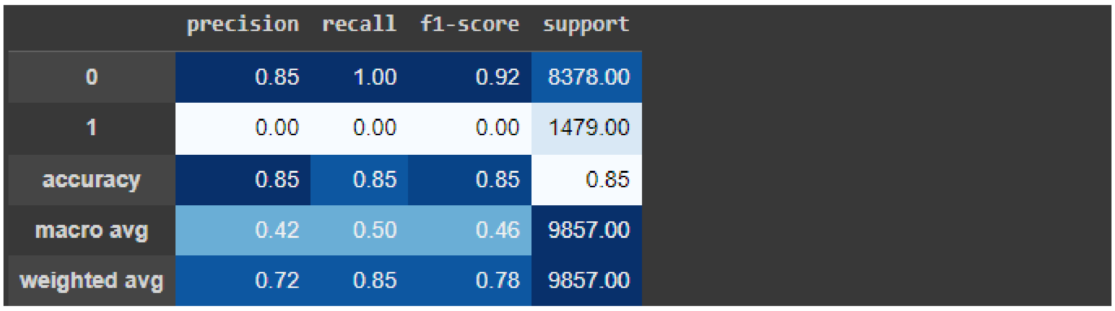
```
#Create all the models into dictionary
models = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'KNN': KNeighborsClassifier(),
    'SVM': SVC(),
    'XGBoost': XGBClassifier(),
    'CatBoost': CatBoostClassifier(iterations=1)
}
```

Part 4: Evaluation

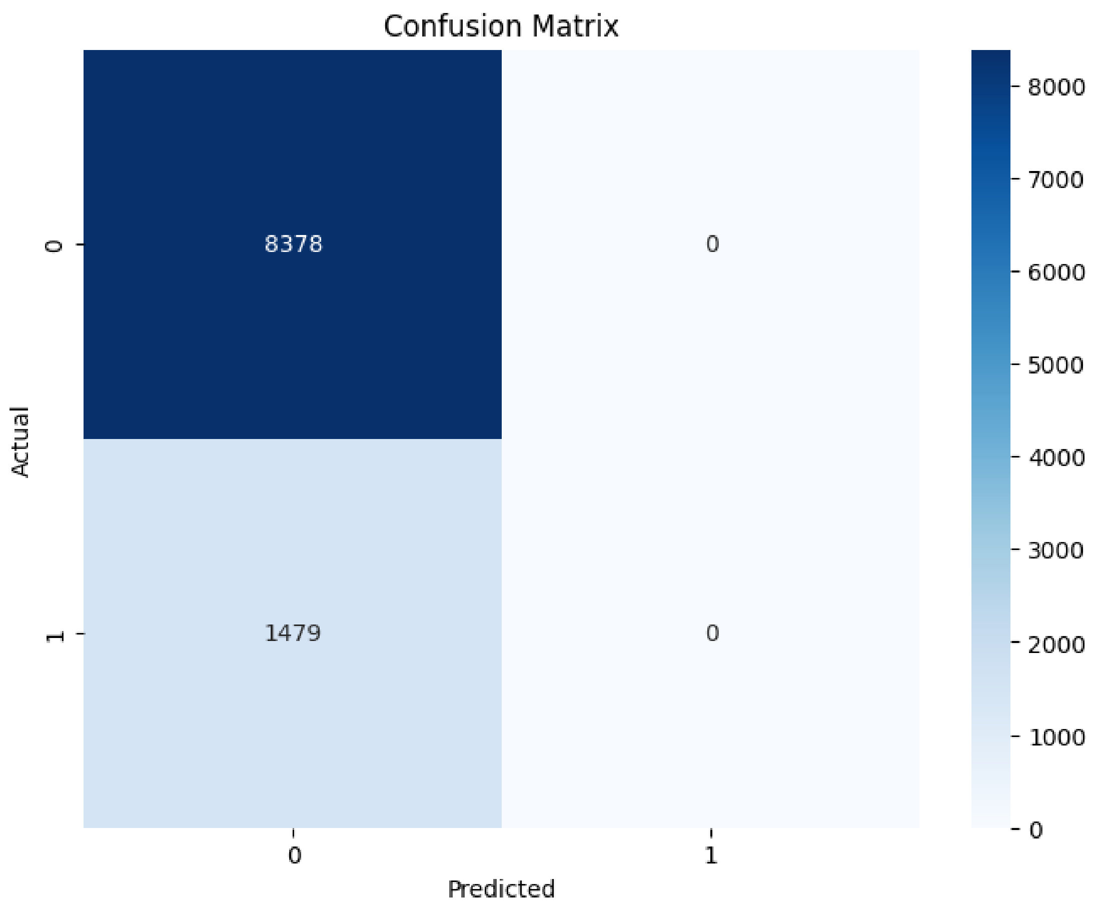
4.1. Logistic Regression

```
logisticR_model = list(models.values())[0]
logit = list(models.keys())[0]
model_building(logisticR_model, X_train, X_test, y_train, y_test)
```

```
Name of the model: LogisticRegression
Accuracy score: 0.8499543471644516
---------------------------------------------------------------------------
Classification report:
```



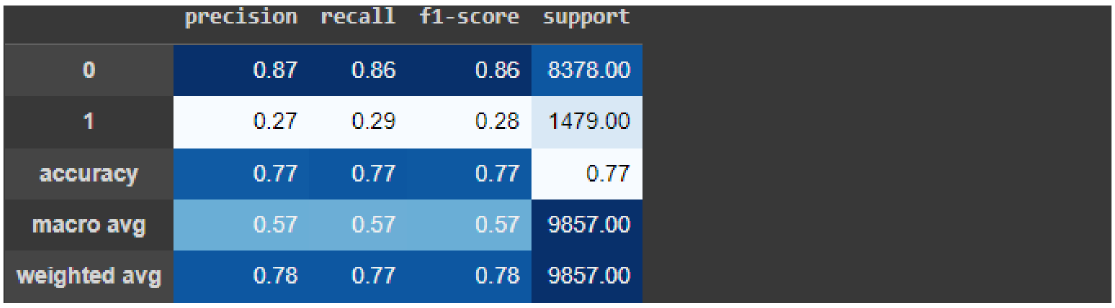
	precision	recall	f1-score	support
0	0.85	1.00	0.92	8378.00
1	0.00	0.00	0.00	1479.00
accuracy	0.85	0.85	0.85	0.85
macro avg	0.42	0.50	0.46	9857.00
weighted avg	0.72	0.85	0.78	9857.00



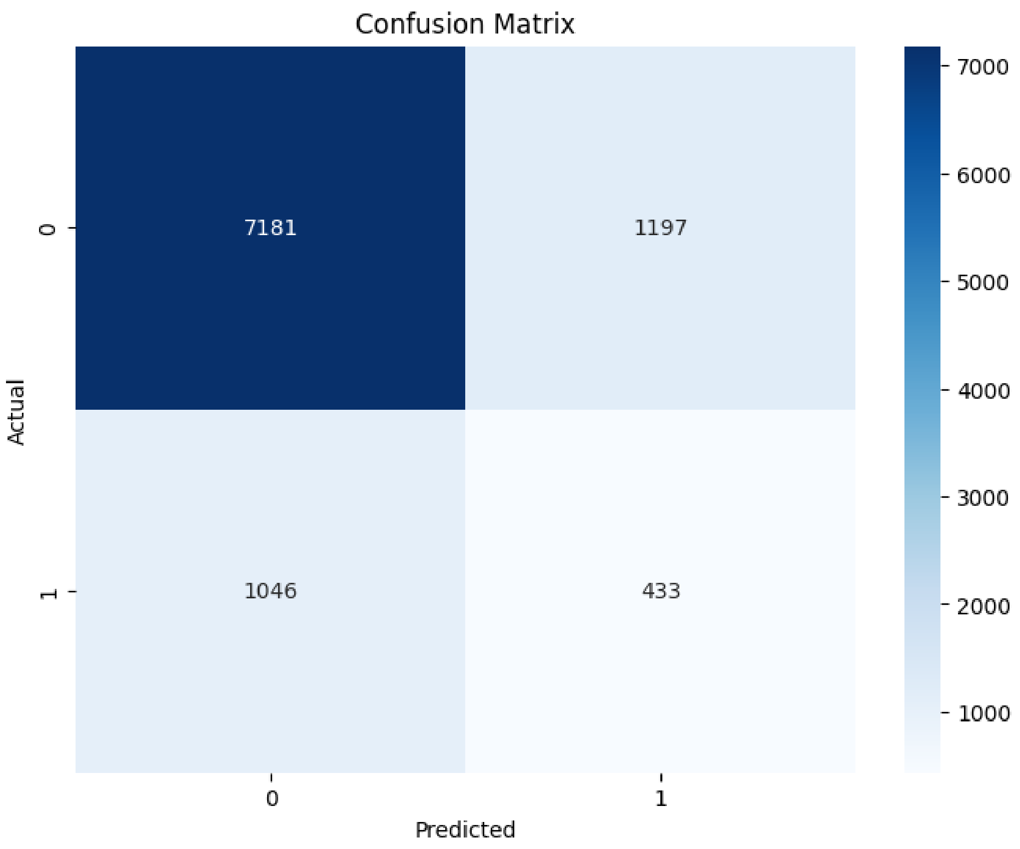
4.2. Decision Tree

```
tree_model = list(models.values())[1]
tree = list(models.keys())[1]
model_building(tree_model, X_train, X_test, y_train, y_test)
```

```
Name of the model: DecisionTreeClassifier
Accuracy score: 0.7724459774779344
-----------------------------------------------------------------------------
Classification report:
```



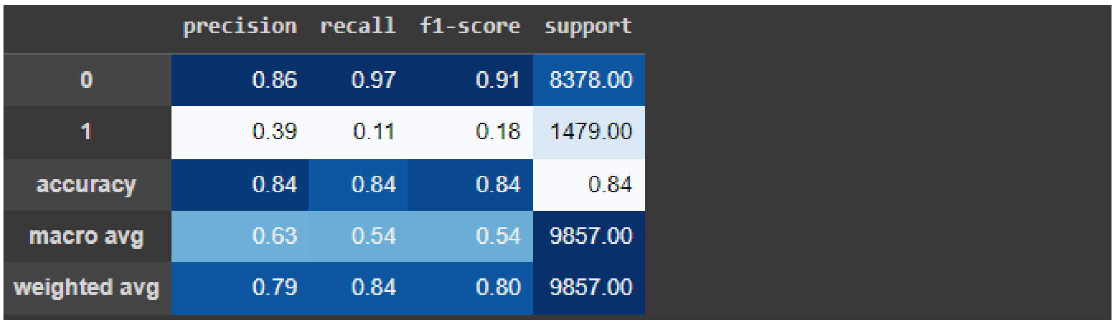
	precision	recall	f1-score	support
0	0.87	0.86	0.86	8378.00
1	0.27	0.29	0.28	1479.00
accuracy	0.77	0.77	0.77	0.77
macro avg	0.57	0.57	0.57	9857.00
weighted avg	0.78	0.77	0.78	9857.00



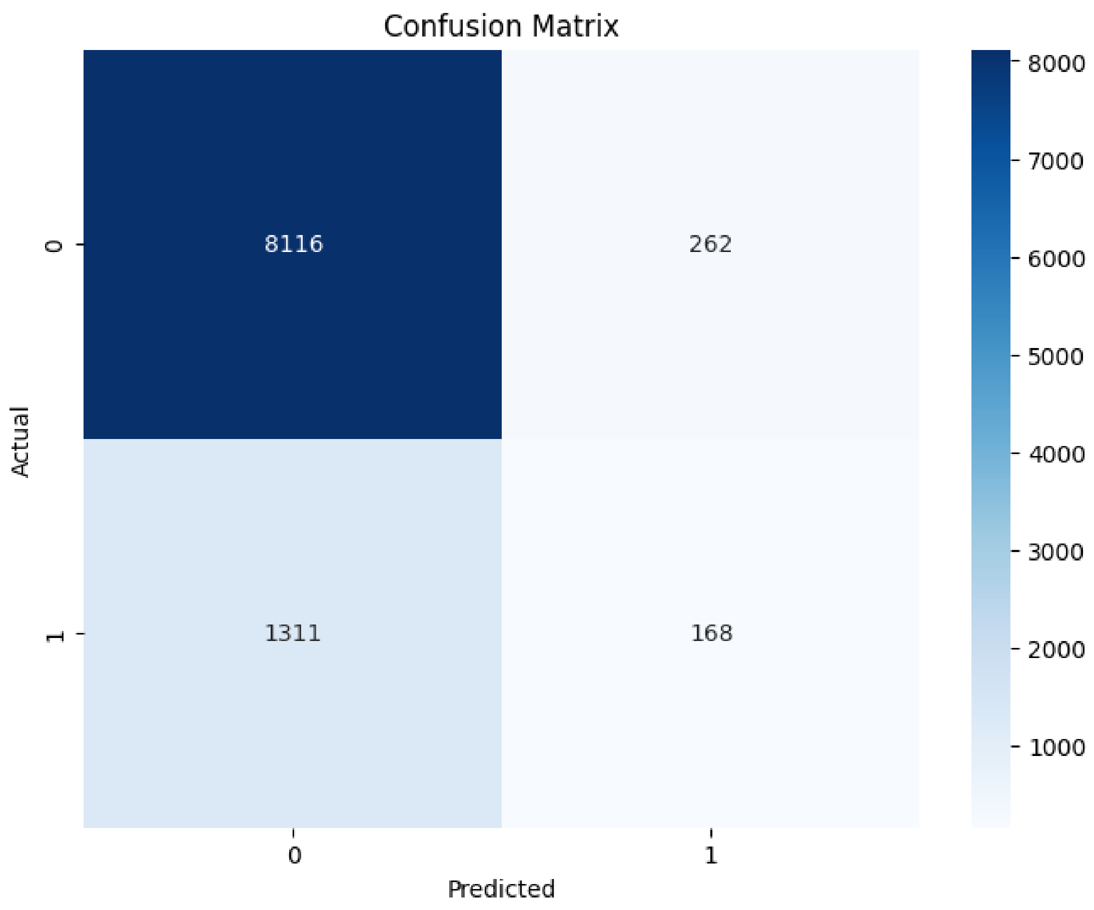
4.3. Random_Forrest

```
forest_model = list(models.values())[2]
forest = list(models.keys())[2]
model_building(forest_model, X_train, X_test, y_train, y_test)
```

```
Name of the model: RandomForestClassifier
Accuracy score: 0.8404179770721315
--------------------------------------------------------------------------------
Classification report:
```



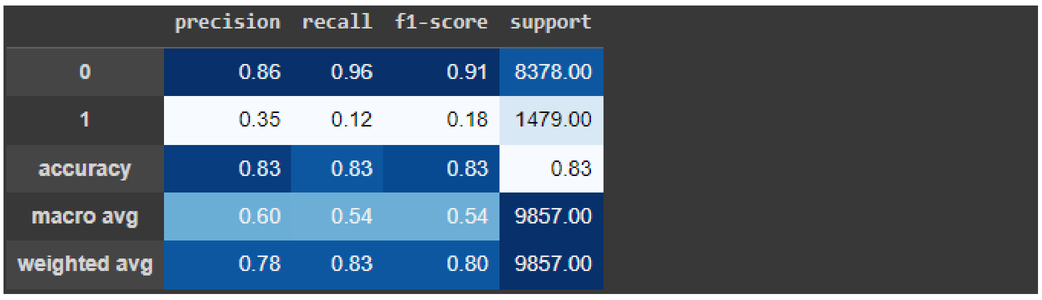
	precision	recall	f1-score	support
0	0.86	0.97	0.91	8378.00
1	0.39	0.11	0.18	1479.00
accuracy	0.84	0.84	0.84	0.84
macro avg	0.63	0.54	0.54	9857.00
weighted avg	0.79	0.84	0.80	9857.00



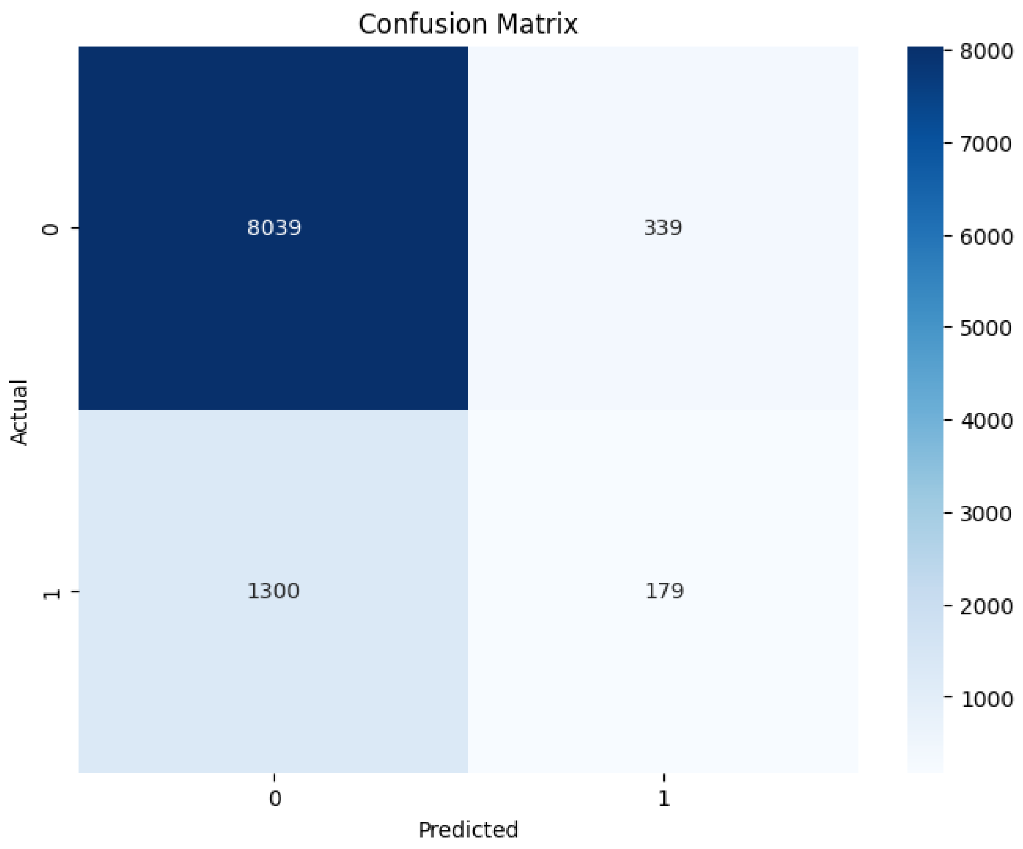
4.4. KNN

```
knn_model = list(models.values())[3]
knn = list(models.keys())[3]
model_building(knn_model, X_train, X_test, y_train, y_test)
```

```
Name of the model: KNeighborsClassifier
Accuracy score: 0.8337222278583748
--------------------------------------------------------------------------------
Classification report:
```



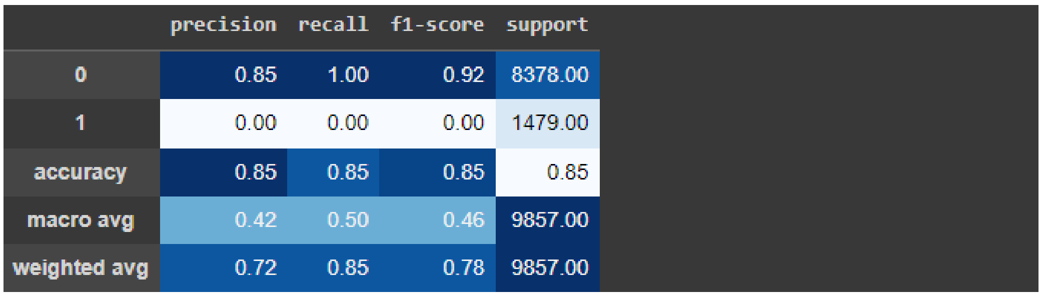
	precision	recall	f1-score	support
0	0.86	0.96	0.91	8378.00
1	0.35	0.12	0.18	1479.00
accuracy	0.83	0.83	0.83	0.83
macro avg	0.60	0.54	0.54	9857.00
weighted avg	0.78	0.83	0.80	9857.00



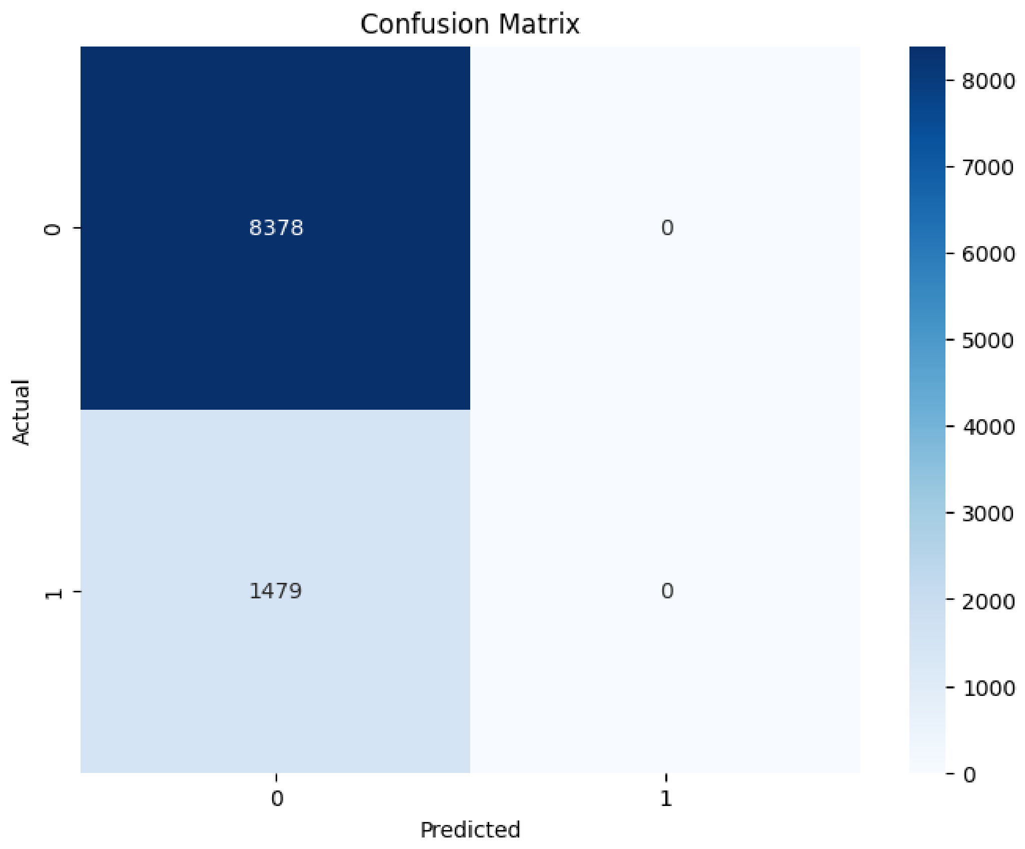
4.5. SVC

```
svc_model = list(models.values())[4]
svc = list(models.keys())[4]
model_building(svc_model, X_train, X_test, y_train, y_test)
```

```
Name of the model: SVC
Accuracy score: 0.8499543471644516
--------------------------------------------------------------------------------
Classification report:
```



	precision	recall	f1-score	support
0	0.85	1.00	0.92	8378.00
1	0.00	0.00	0.00	1479.00
accuracy	0.85	0.85	0.85	0.85
macro avg	0.42	0.50	0.46	9857.00
weighted avg	0.72	0.85	0.78	9857.00



4.6. Evaluation table

Metrics	SVC	KNN	Random_Forest	Decision_Tree	**Logistic_Regression**
Precision	85%	86%	86%	87%	85%
Recall	100%	96%	87%	86%	100%

References

1. *Projects - CS 188: Introduction to Artificial Intelligence, Spring 2021.* (n.d.). CS 188. https://inst.eecs.berkeley.edu/~cs188/sp21/projects/
2. *Russell, S. J., & Norvig, P. (2020). Artificial intelligence: A modern approach (4th ed.). Pearson*
3. *Poole, D. L., & Mackworth, A. K. (n.d.). Artificial intelligence: Foundations of computational agents.*