Predicting customer buying behaviour

Project Description

Explore and prepare the dataset

First, spend some time exploring the dataset in the "Getting Started" Jupyter Notebook provided in the Resources section below to understand the different columns and some basic statistics of the dataset. Then, you should consider how to prepare the dataset for a predictive model. You should think about any new features you want to create in order to make your model even better. You can make use of the Resources provided to get you started with this task.

Train a machine learning model

When your data is ready for modelling, you should train a machine learning model to be able to predict the target outcome, which is a customer making a booking. For this task, you should use an algorithm that easily allows you to output information about how each variable within the model contributes to its predictive power. For example, a RandomForest is very good for this purpose.

Evaluate model and present findings

After training your model, you should evaluate how well it performed by conducting cross-validation and outputting appropriate evaluation metrics. Furthermore, you should create a visualisation to interpret how each variable contributed to the model. Finally, you should summarise your findings in a single slide to be sent to your manager. Use the "PowerPoint Template" provided in the Resources section below to create your summary and make use of the links provided to help with this task.

In [1]:
```
import numpy as np
from numpy import count_nonzero, median, mean
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import random

import sklearn

import statsmodels.api as sm

import datetime
from datetime import datetime, timedelta


import scipy.stats

import xgboost as xgb
from xgboost import XGBClassifier, XGBRegressor
from xgboost import to_graphviz, plot_importance, plot_tree

from sklearn.model_selection import cross_val_score, train_test_split, GridSearchCV, Ran
from sklearn.model_selection import cross_validate, KFold, RepeatedStratifiedKFold
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler, OneHotEnco
```

```
#from sklearn.pipeline import Pipeline
#from sklearn.feature_selection import RFE, RFECV, SelectKBest, f_classif, f_regression,
#from sklearn.inspection import permutation_importance

from sklearn.tree import export_graphviz, plot_tree
from sklearn.metrics import confusion_matrix, classification_report, mean_absolute_error
from sklearn.metrics import plot_confusion_matrix, plot_precision_recall_curve, plot_roc
from sklearn.metrics import auc, f1_score, precision_score, recall_score, roc_auc_score

#from sklearn.experimental import enable_hist_gradient_boosting
#from sklearn.linear_model import ElasticNet, Lasso, LinearRegression, LogisticRegressio
#from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, ExtraTreeClassi
#from sklearn.svm import SVC, SVR, LinearSVC, LinearSVR
#from sklearn.naive_bayes import GaussianNB, MultinomialNB
#from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor, ExtraTreesC
#from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor, His



%matplotlib inline
#sets the default autosave frequency in seconds
%autosave 60
sns.set_style('dark')
sns.set(font_scale=1.2)

plt.rc('axes', labelsize=14)
plt.rc('xtick', labelsize=12)
plt.rc('ytick', labelsize=12)


#from tpot import TPOTClassifier, TPOTRegressor
#from imblearn.under_sampling import RandomUnderSampler
#from imblearn.over_sampling import RandomOverSampler
#from imblearn.over_sampling import SMOTE

import warnings
warnings.filterwarnings('ignore')

# import pickle
# from pickle import dump, load

# Use Folium library to plot values on a map.
#import folium

# Use Feature-Engine library

#import feature_engine.missing_data_imputers as mdi
#from feature_engine.outlier_removers import Winsorizer
#from feature_engine import categorical_encoders as ce


#from pycaret.classification import *
#from pycaret.clustering import *
#from pycaret.regression import *

pd.set_option('display.max_columns',None)
#pd.set_option('display.max_rows',100)
pd.set_option('display.width', 1000)
pd.set_option('display.float_format','{:.2f}'.format)


random.seed(0)
np.random.seed(0)
np.set_printoptions(suppress=True)
```

```
Autosaving every 60 seconds
```

Exploratory Data Analysis

In [2]:
```
df = pd.read_csv("train.csv")
```

In [3]:
```
df
```

Out[3]:

	num_passengers	sales_channel	trip_type	purchase_lead	length_of_stay	flight_day	wants_extra_baggage
0	2	0	1	262	19	6	1
1	1	0	1	112	20	6	0
2	2	0	1	243	22	3	1
3	1	0	1	96	31	6	0
4	2	0	1	68	22	3	1
...	...	...	...	...	...	...	...
49995	2	0	1	27	6	6	1
49996	1	0	1	111	6	7	0
49997	1	0	1	24	6	6	0
49998	1	0	1	15	6	1	1
49999	1	0	1	19	6	4	0

50000 rows × 11 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   num_passengers         50000 non-null  int64  
 1   sales_channel          50000 non-null  int64  
 2   trip_type              50000 non-null  int64  
 3   purchase_lead          50000 non-null  int64  
 4   length_of_stay         50000 non-null  int64  
 5   flight_day             50000 non-null  int64  
 6   wants_extra_baggage    50000 non-null  int64  
 7   wants_preferred_seat   50000 non-null  int64  
 8   wants_in_flight_meals  50000 non-null  int64  
 9   flight_duration        50000 non-null  float64
 10  booking_complete       50000 non-null  int64  
dtypes: float64(1), int64(10)
memory usage: 4.2 MB
```

In [5]:
```
df.describe(include='all')
```

Out[5]:

	num_passengers	sales_channel	trip_type	purchase_lead	length_of_stay	flight_day	wants_extra_baggage
count	50000.00	50000.00	50000.00	50000.00	50000.00	50000.00	50000.00
mean	1.59	0.11	1.01	84.94	23.04	3.81	0.67
std	1.02	0.32	0.13	90.45	33.89	1.99	0.47
min	1.00	0.00	1.00	0.00	0.00	1.00	0.00

25%	1.00	0.00	1.00	21.00	5.00	2.00	0.00
50%	1.00	0.00	1.00	51.00	17.00	4.00	1.00
75%	2.00	0.00	1.00	115.00	28.00	5.00	1.00
max	9.00	1.00	3.00	867.00	778.00	7.00	1.00

In [6]:
```
df.shape
```

Out[6]:
```
(50000, 11)
```

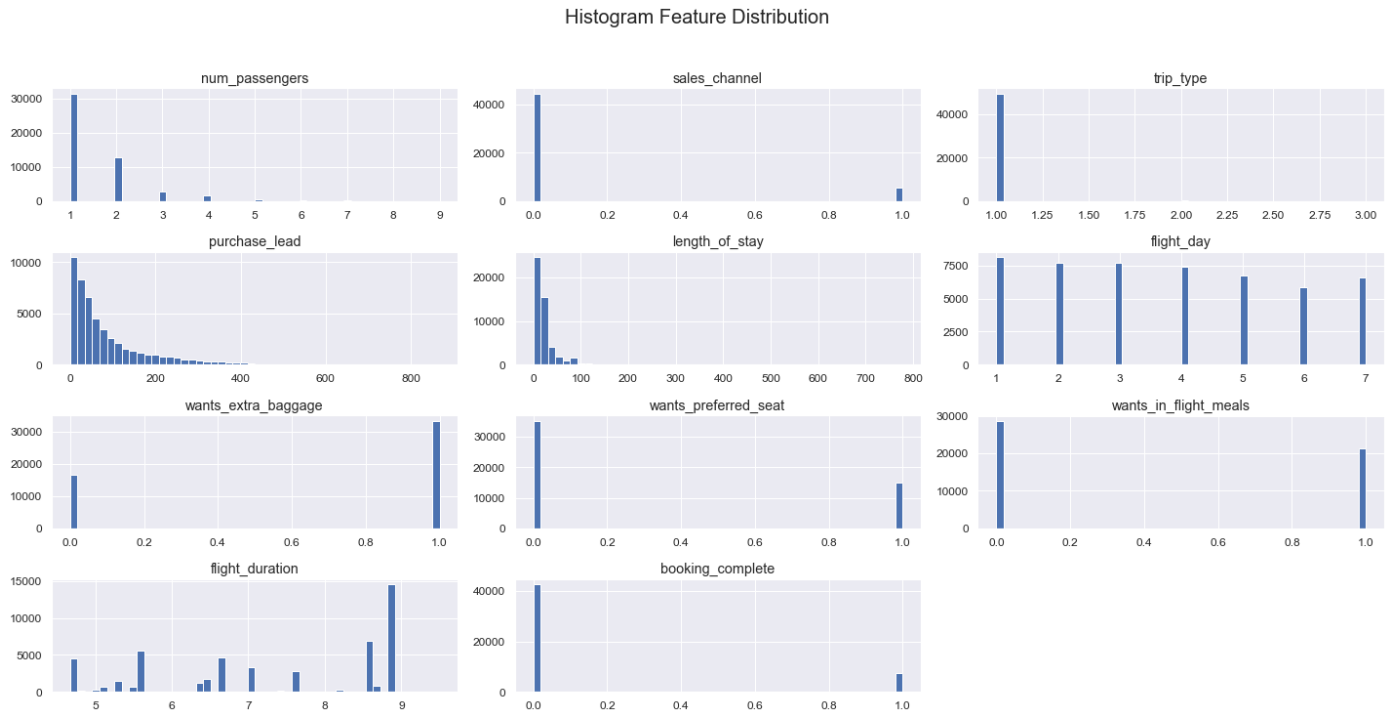
In [7]:
```
df.columns
```

Out[7]:
```
Index(['num_passengers', 'sales_channel', 'trip_type', 'purchase_lead', 'length_of_sta
y', 'flight_day', 'wants_extra_baggage', 'wants_preferred_seat', 'wants_in_flight_meal
s', 'flight_duration', 'booking_complete'], dtype='object')
```

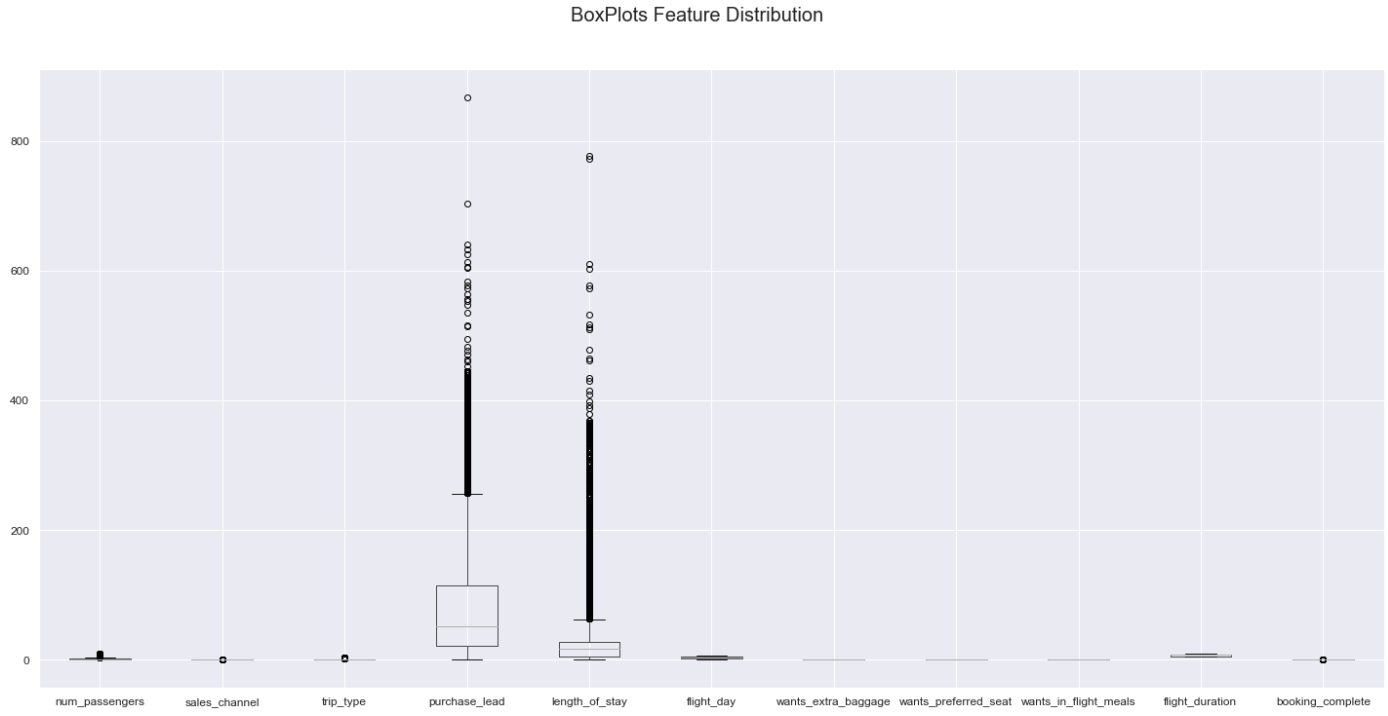
Data Visualization

Univariate Data Exploration

In [8]:
```
df.hist(bins=50, figsize=(20,10))
plt.suptitle('Histogram Feature Distribution', x=0.5, y=1.02, ha='center', fontsize=20)
plt.tight_layout()
plt.show()
```

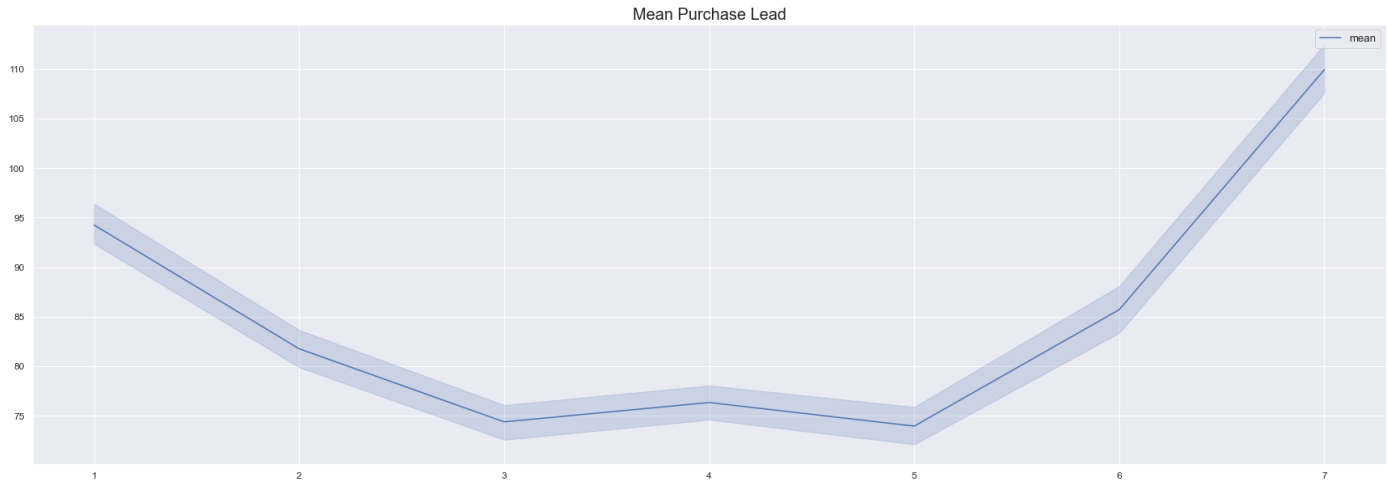


In [9]:
```
df.boxplot(figsize=(20,10))
plt.suptitle('BoxPlots Feature Distribution', x=0.5, y=1.02, ha='center', fontsize=20)
plt.tight_layout()
plt.show()
```

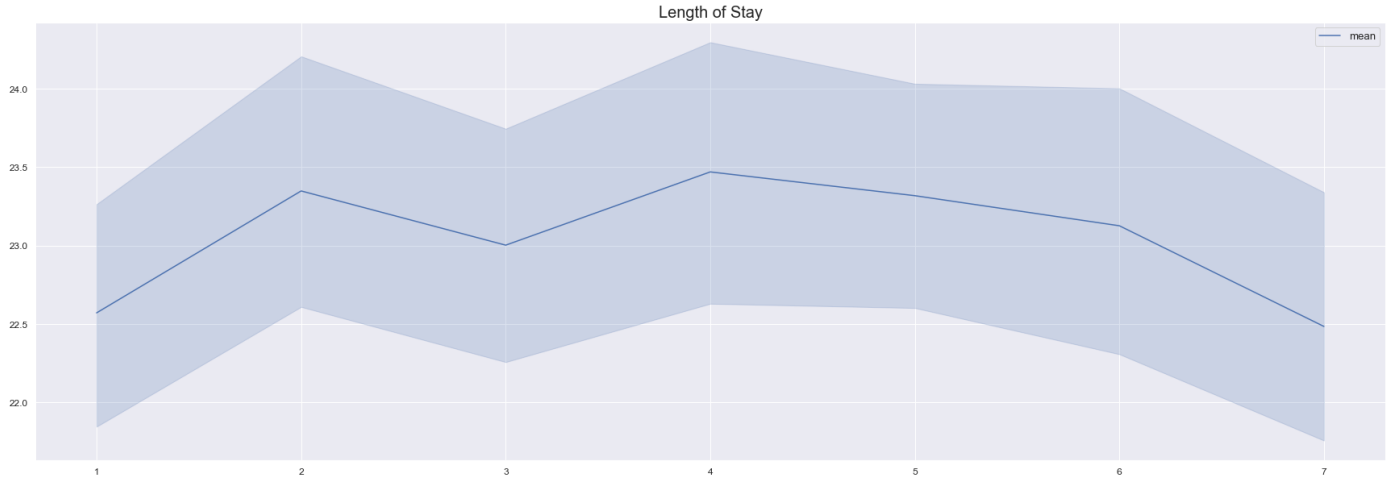


Time-Series Analysis

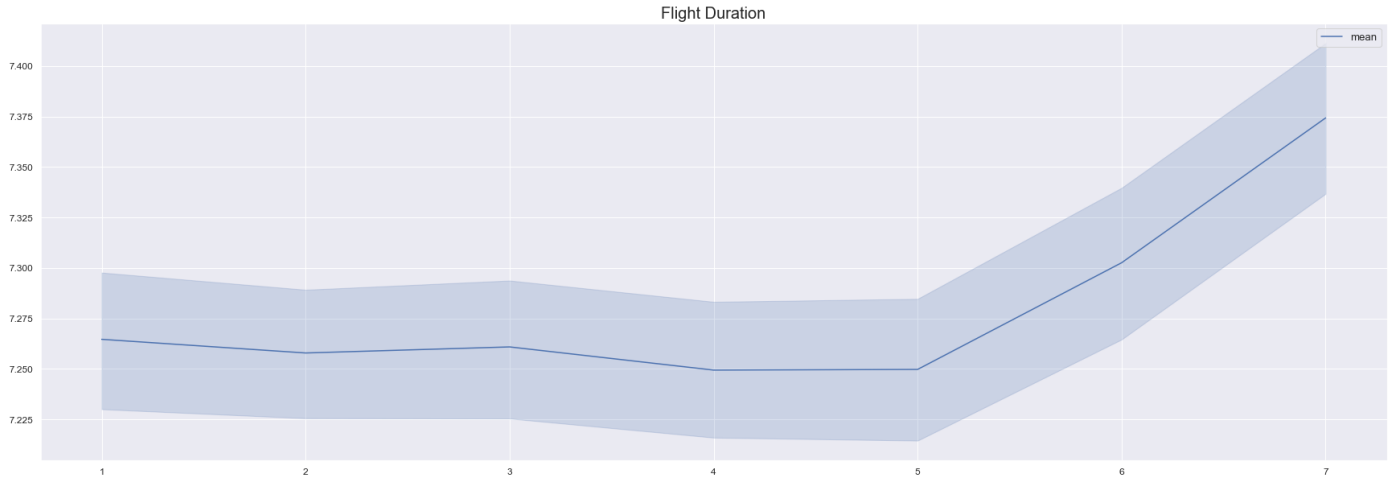
In [10]:
```
fig = plt.figure(figsize=(30,10))
sns.lineplot(x=df.flight_day,y=df.purchase_lead,data=df, estimator="mean")
plt.title("Mean Purchase Lead", fontsize=20)
plt.xlabel("", fontsize=20)
plt.ylabel("", fontsize=20)
plt.legend(['mean'])
plt.show()
```



In [11]:
```
fig = plt.figure(figsize=(30,10))
sns.lineplot(x=df.flight_day,y=df.length_of_stay,data=df, estimator='mean')
plt.title("Length of Stay", fontsize=20)
plt.xlabel("", fontsize=20)
plt.ylabel("", fontsize=20)
plt.legend(['mean'])
plt.show()
```



In [12]:
```
fig = plt.figure(figsize=(30,10))
sns.lineplot(x=df.flight_day,y=df.flight_duration,data=df, estimator='mean')
plt.title("Flight Duration", fontsize=20)
plt.xlabel("", fontsize=20)
plt.ylabel("", fontsize=20)
plt.legend(['mean'])
plt.show()
```



Correlation

In [13]:
```
df.corr()
```

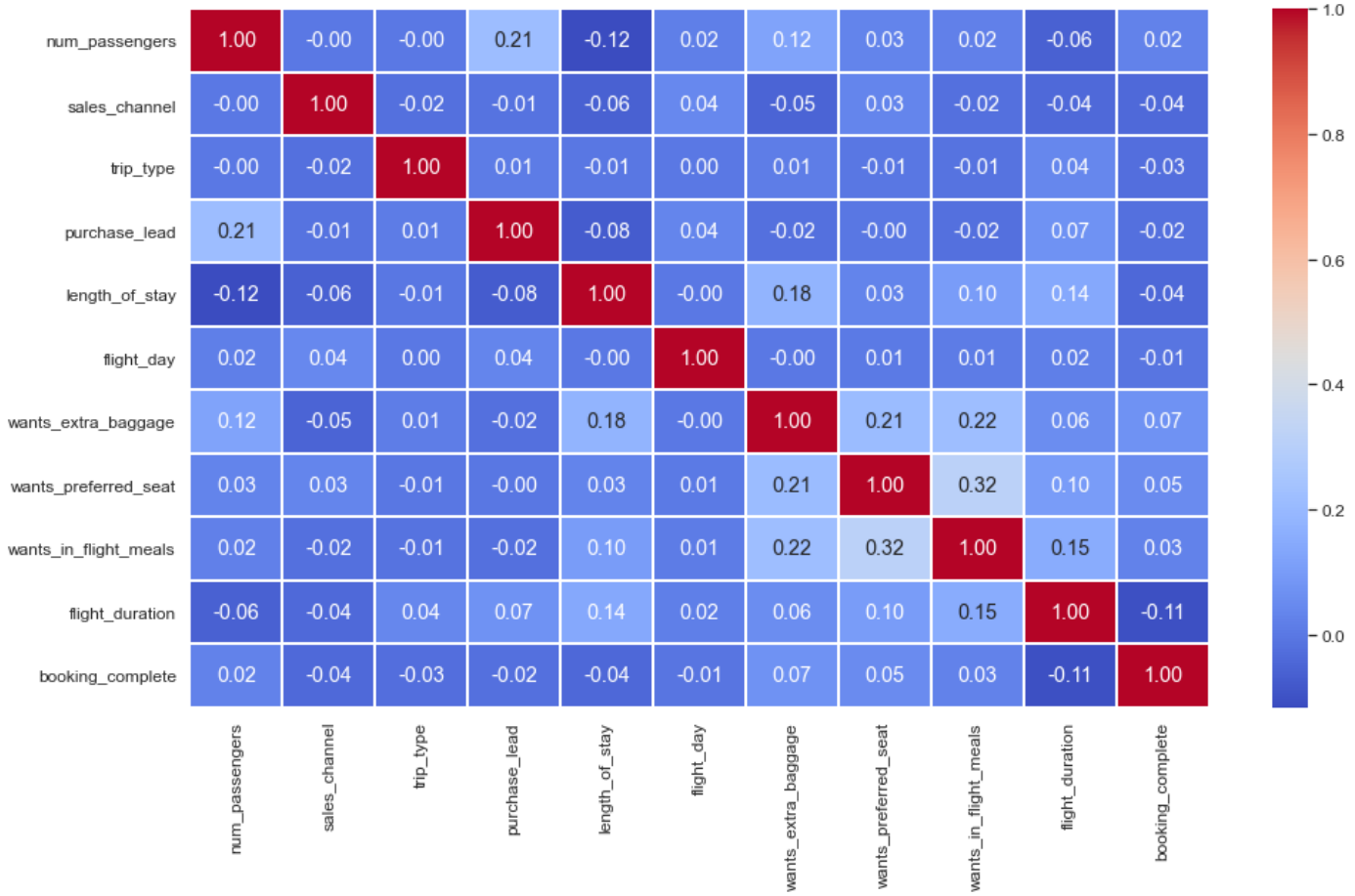
Out[13]:

	num_passengers	sales_channel	trip_type	purchase_lead	length_of_stay	flight_day	wants_e
num_passengers	1.00	-0.00	-0.00	0.21	-0.12	0.02	
sales_channel	-0.00	1.00	-0.02	-0.01	-0.06	0.04	
trip_type	-0.00	-0.02	1.00	0.01	-0.01	0.00	
purchase_lead	0.21	-0.01	0.01	1.00	-0.08	0.04	
length_of_stay	-0.12	-0.06	-0.01	-0.08	1.00	-0.00	
flight_day	0.02	0.04	0.00	0.04	-0.00	1.00	
wants_extra_baggage	0.12	-0.05	0.01	-0.02	0.18	-0.00	
wants_preferred_seat	0.03	0.03	-0.01	-0.00	0.03	0.01	
wants_in_flight_meals	0.02	-0.02	-0.01	-0.02	0.10	0.01	
flight_duration	-0.06	-0.04	0.04	0.07	0.14	0.02	
booking_complete	0.02	-0.04	-0.03	-0.02	-0.04	-0.01	

In [14]:
```
df.corr()["booking_complete"].sort_values()
```

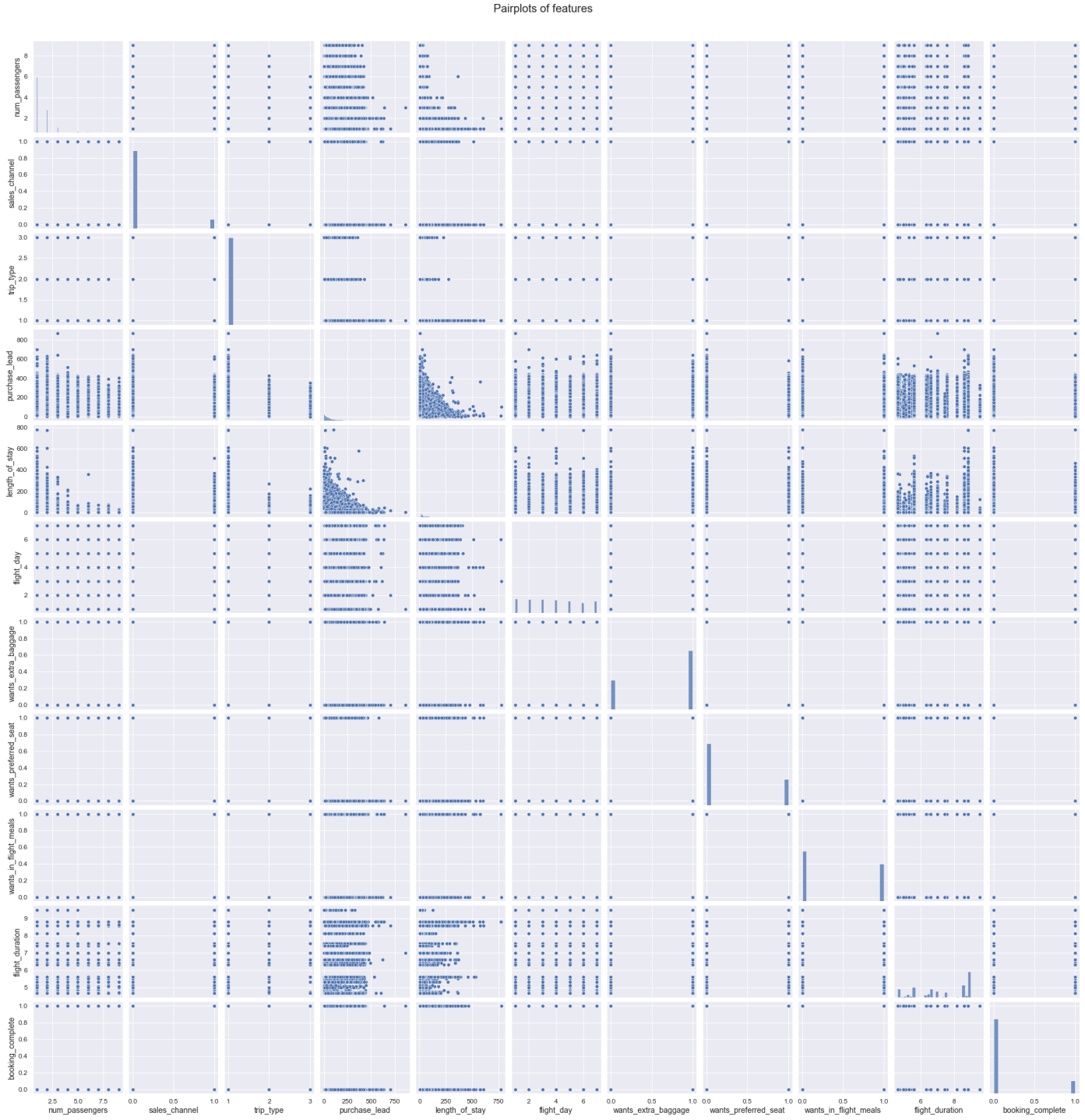
Out[14]:
```
flight_duration         -0.11
length_of_stay          -0.04
sales_channel           -0.04
trip_type               -0.03
purchase_lead           -0.02
flight_day              -0.01
num_passengers           0.02
wants_in_flight_meals    0.03
wants_preferred_seat     0.05
wants_extra_baggage      0.07
booking_complete         1.00
Name: booking_complete, dtype: float64
```

In [15]:
```
plt.figure(figsize=(16,9))
sns.heatmap(df.corr(),cmap="coolwarm",annot=True,fmt='.2f',linewidths=2)
plt.title("", fontsize=20)
plt.show()
```



Pairplots

In [16]:
```
sns.pairplot(df)
plt.suptitle('Pairplots of features', x=0.5, y=1.02, ha='center', fontsize=20)
plt.show()
```



Treat Missing Values

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
num_passengers          0
sales_channel           0
trip_type               0
purchase_lead           0
length_of_stay          0
flight_day              0
wants_extra_baggage     0
wants_preferred_seat    0
wants_in_flight_meals   0
flight_duration         0
booking_complete        0
dtype: int64
```

Treat Duplicate Values

In [18]:
```
df.duplicated(keep='first').sum()
```

Out[18]:
```
1504
```

In [19]:
```
df[df.duplicated(keep=False)] #Check duplicate values
```

Out[19]:

	num_passengers	sales_channel	trip_type	purchase_lead	length_of_stay	flight_day	wants_extra_baggage
89	1	0	1	34	24	5	1
115	1	0	1	65	278	4	0
117	1	0	1	263	58	7	0
122	1	0	1	42	17	2	0
135	4	0	1	366	17	7	1
...	...	...	...	...	...	...	...
49934	1	0	1	2	6	4	0
49944	1	0	1	2	6	4	0
49958	4	1	1	108	6	1	1
49961	1	0	1	30	6	4	0
49972	1	0	1	33	6	3	1

2764 rows × 11 columns

Treat Data Types

In [20]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   num_passengers         50000 non-null  int64  
 1   sales_channel          50000 non-null  int64  
 2   trip_type              50000 non-null  int64  
 3   purchase_lead          50000 non-null  int64  
 4   length_of_stay         50000 non-null  int64  
 5   flight_day             50000 non-null  int64  
 6   wants_extra_baggage    50000 non-null  int64  
 7   wants_preferred_seat   50000 non-null  int64  
 8   wants_in_flight_meals  50000 non-null  int64  
 9   flight_duration        50000 non-null  float64
 10  booking_complete       50000 non-null  int64  
dtypes: float64(1), int64(10)
memory usage: 4.2 MB
```

Train Test Split

In [21]:
```
df = pd.read_csv("train.csv")
```

In [22]:
```
X = df.iloc[:,0:10]
y = df.iloc[:,10:]
```

Train Test Split Cont'd

In [23]:
```
X.values, y.values
```

Out[23]:
```
(array([[2.  , 0.  , 1.  , ..., 0.  , 0.  , 5.52],
        [1.  , 0.  , 1.  , ..., 0.  , 0.  , 5.52],
        [2.  , 0.  , 1.  , ..., 1.  , 0.  , 5.52],
        ...,
        [1.  , 0.  , 1.  , ..., 0.  , 1.  , 5.62],
        [1.  , 0.  , 1.  , ..., 0.  , 1.  , 5.62],
        [1.  , 0.  , 1.  , ..., 1.  , 0.  , 5.62]]),
 array([[0],
        [0],
        [0],
        ...,
        [0],
        [0],
        [0]], dtype=int64))
```

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [25]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[25]:
```
((40000, 10), (10000, 10), (40000, 1), (10000, 1))
```

Model Training

Using Regression or Classification Models

Using XGBoost (Scikit-Learn)

Using RandomSearchCV

In [26]:
```
model = XGBClassifier(random_state=0, n_estimators=100, objective='binary:logistic')
```

In [27]:
```
parameters = {'max_depth': np.arange(3,10,1),
              'eta': np.arange(0.05,0.3,0.05),
              'n_estimators':np.arange(100,1000,100),
              'min_child_weight': np.arange(1,4,1),
              'gamma':np.arange(0,10,2),
              'subsample':np.arange(0.5,0.9,0.1),
              'colsample_bytree':np.arange(0.5,0.9,0.1),
              'reg_alpha':np.arange(0,1,0.1),
              'reg_lambda':np.arange(0,1,0.1)
             }
```

In [28]:
```
randm = RandomizedSearchCV(estimator=model, param_distributions = parameters, cv = 5, n_
                           n_jobs=-1, scoring='accuracy')
```

In [29]:
```
#randm.fit(X, y)
```

In [30]:
```
#randm.best_estimator_
```

In [31]:
```
#randm.best_score_
```

In [32]:
```
#randm.best_params_
```

Final Model

In [33]:
```
xgbmodel = XGBClassifier(random_state=0, n_estimators=100, objective='binary:logistic')
```

In [34]:
```
xgbmodel.fit(X_train,y_train,eval_set=[(X_test,y_test)],eval_metric='error',early_stoppi
```

```
[0]     validation_0-error:0.15180
[1]     validation_0-error:0.15190
[2]     validation_0-error:0.15180
[3]     validation_0-error:0.15140
[4]     validation_0-error:0.15130
[5]     validation_0-error:0.15160
[6]     validation_0-error:0.15140
[7]     validation_0-error:0.15140
[8]     validation_0-error:0.15160
[9]     validation_0-error:0.15170
[10]    validation_0-error:0.15170
[11]    validation_0-error:0.15180
[12]    validation_0-error:0.15160
[13]    validation_0-error:0.15170
[14]    validation_0-error:0.15170
[15]    validation_0-error:0.15160
[16]    validation_0-error:0.15140
[17]    validation_0-error:0.15130
[18]    validation_0-error:0.15130
[19]    validation_0-error:0.15110
[20]    validation_0-error:0.15120
[21]    validation_0-error:0.15110
[22]    validation_0-error:0.15110
[23]    validation_0-error:0.15110
[24]    validation_0-error:0.15130
[25]    validation_0-error:0.15110
[26]    validation_0-error:0.15110
[27]    validation_0-error:0.15120
[28]    validation_0-error:0.15110
[29]    validation_0-error:0.15130
[30]    validation_0-error:0.15150
[31]    validation_0-error:0.15160
[32]    validation_0-error:0.15160
[33]    validation_0-error:0.15170
[34]    validation_0-error:0.15200
[35]    validation_0-error:0.15200
[36]    validation_0-error:0.15200
[37]    validation_0-error:0.15200
[38]    validation_0-error:0.15170
[39]    validation_0-error:0.15170
[40]    validation_0-error:0.15170
[41]    validation_0-error:0.15180
[42]    validation_0-error:0.15190
[43]    validation_0-error:0.15180
[44]    validation_0-error:0.15180
[45]    validation_0-error:0.15190
[46]    validation_0-error:0.15200
[47]    validation_0-error:0.15190
[48]    validation_0-error:0.15190
[49]    validation_0-error:0.15200
[50]    validation_0-error:0.15220
[51]    validation_0-error:0.15200
[52]    validation_0-error:0.15180
[53]    validation_0-error:0.15170
[54]    validation_0-error:0.15190
[55]    validation_0-error:0.15200
[56]    validation_0-error:0.15190
```

```
[57]    validation_0-error:0.15190
[58]    validation_0-error:0.15210
[59]    validation_0-error:0.15220
[60]    validation_0-error:0.15230
[61]    validation_0-error:0.15270
[62]    validation_0-error:0.15270
[63]    validation_0-error:0.15270
[64]    validation_0-error:0.15260
[65]    validation_0-error:0.15270
[66]    validation_0-error:0.15280
[67]    validation_0-error:0.15290
[68]    validation_0-error:0.15290
```

Out[34]:

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_bin=256, max_cat_threshold=64, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0, ...)
```

In [35]:

```
xgbmodel.fit(X_train,y_train,eval_set=[(X_test,y_test)],eval_metric='auc',early_stopping
```

```
[0]     validation_0-auc:0.66379
[1]     validation_0-auc:0.66446
[2]     validation_0-auc:0.67235
[3]     validation_0-auc:0.67466
[4]     validation_0-auc:0.67337
[5]     validation_0-auc:0.67848
[6]     validation_0-auc:0.67756
[7]     validation_0-auc:0.68194
[8]     validation_0-auc:0.68348
[9]     validation_0-auc:0.68303
[10]    validation_0-auc:0.68102
[11]    validation_0-auc:0.68239
[12]    validation_0-auc:0.68255
[13]    validation_0-auc:0.68201
[14]    validation_0-auc:0.68146
[15]    validation_0-auc:0.68338
[16]    validation_0-auc:0.68291
[17]    validation_0-auc:0.68260
[18]    validation_0-auc:0.68281
[19]    validation_0-auc:0.68160
[20]    validation_0-auc:0.68174
[21]    validation_0-auc:0.68124
[22]    validation_0-auc:0.68106
[23]    validation_0-auc:0.68098
[24]    validation_0-auc:0.68070
[25]    validation_0-auc:0.68013
[26]    validation_0-auc:0.68010
[27]    validation_0-auc:0.67975
[28]    validation_0-auc:0.67960
[29]    validation_0-auc:0.67955
[30]    validation_0-auc:0.67847
[31]    validation_0-auc:0.67817
[32]    validation_0-auc:0.67806
[33]    validation_0-auc:0.67756
[34]    validation_0-auc:0.67767
[35]    validation_0-auc:0.67742
[36]    validation_0-auc:0.67725
[37]    validation_0-auc:0.67747
[38]    validation_0-auc:0.67729
[39]    validation_0-auc:0.67684
[40]    validation_0-auc:0.67681
```

```
[41]    validation_0-auc:0.67710
[42]    validation_0-auc:0.67610
[43]    validation_0-auc:0.67589
[44]    validation_0-auc:0.67480
[45]    validation_0-auc:0.67420
[46]    validation_0-auc:0.67483
[47]    validation_0-auc:0.67488
[48]    validation_0-auc:0.67502
[49]    validation_0-auc:0.67485
[50]    validation_0-auc:0.67493
[51]    validation_0-auc:0.67428
[52]    validation_0-auc:0.67439
[53]    validation_0-auc:0.67336
[54]    validation_0-auc:0.67328
[55]    validation_0-auc:0.67341
[56]    validation_0-auc:0.67293
[57]    validation_0-auc:0.67299
```

Out[35]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_bin=256, max_cat_threshold=64, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0, ...)
```

In [36]:
```
y_pred = xgbmodel.predict(X_test)
```

In [37]:
```
y_pred
```

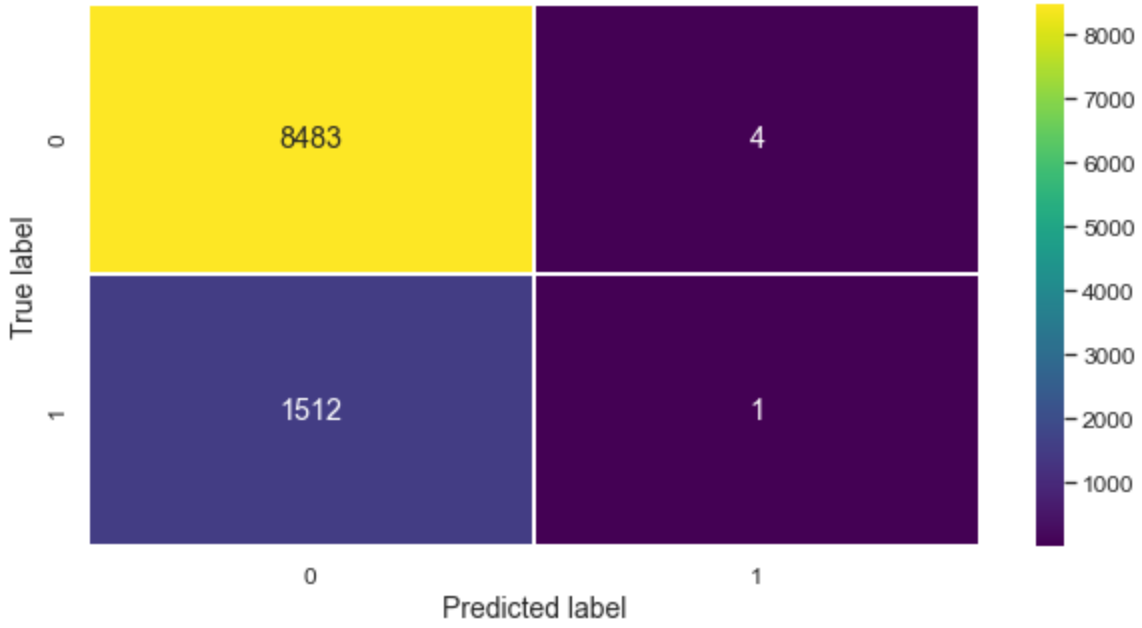
Out[37]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

Model Evaluation

In [38]:
```
cm = confusion_matrix(y_test,y_pred)
cm
```

Out[38]:
```
array([[8483,    4],
       [1512,    1]], dtype=int64)
```

In [39]:
```
fig , ax = plt.subplots(figsize=(10,5))
sns.heatmap(cm, annot=True,fmt='.4g',linewidths=2, cmap='viridis')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



In [40]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      8487
           1       0.20      0.00      0.00      1513

    accuracy                           0.85     10000
   macro avg       0.52      0.50      0.46     10000
weighted avg       0.75      0.85      0.78     10000
```

In [41]:
```
plot_roc_curve(xgbmodel,X_test,y_test)
plt.show()
```

Plot Feature Importances

In [42]:
```
xgbmodel.feature_importances_
```

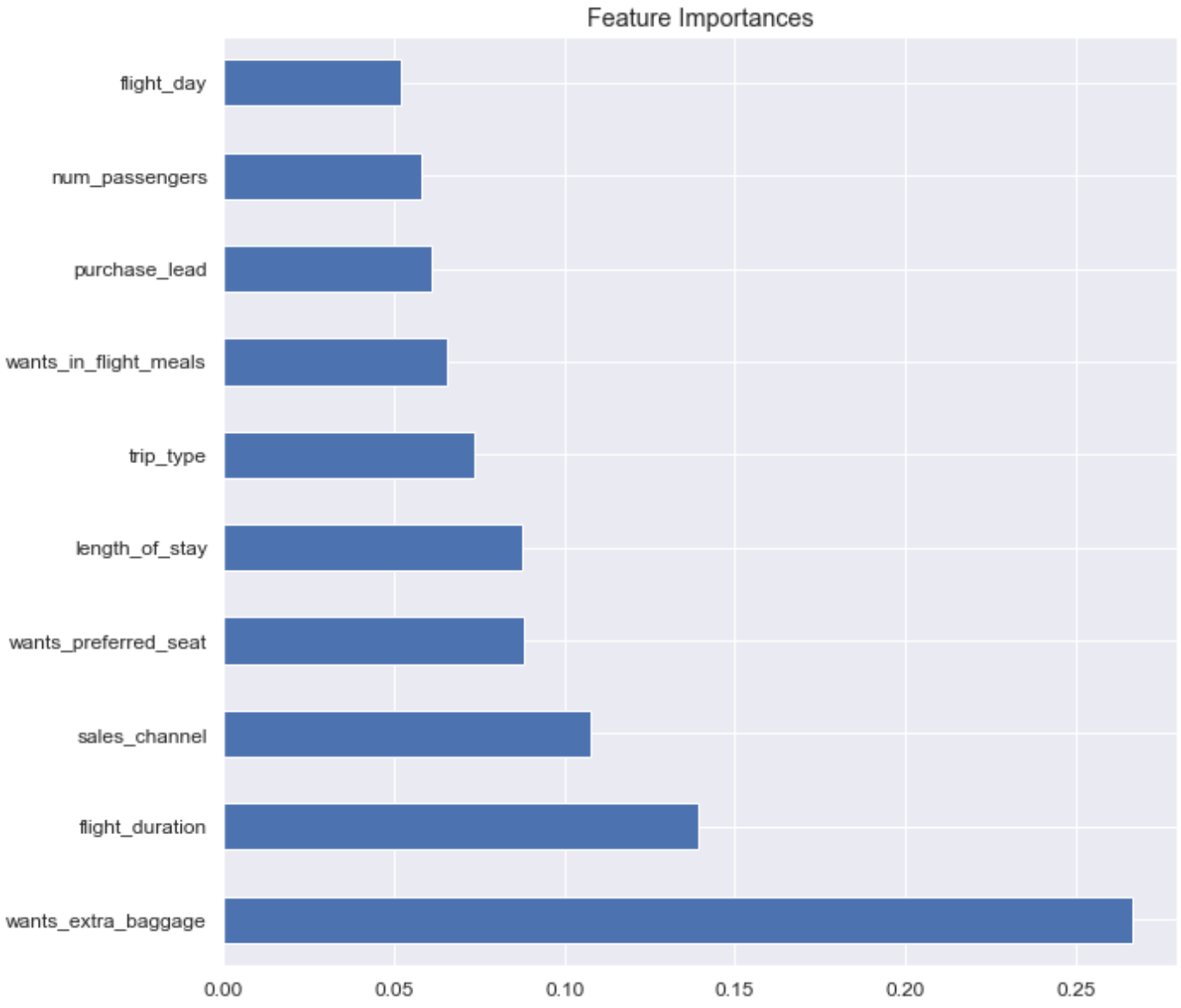
Out[42]:
```
array([0.05814559, 0.1075552 , 0.07374912, 0.06106425, 0.0877305 ,
       0.05178829, 0.2666444 , 0.08830693, 0.06558672, 0.13942891],
      dtype=float32)
```

In [43]:
```
feat_importances = pd.Series(xgbmodel.feature_importances_, index=X.columns)
```

In [44]:
```
feat_importances
```

Out[44]:
```
num_passengers          0.06
sales_channel           0.11
trip_type               0.07
purchase_lead           0.06
length_of_stay          0.09
flight_day              0.05
wants_extra_baggage     0.27
wants_preferred_seat    0.09
wants_in_flight_meals   0.07
flight_duration         0.14
dtype: float32
```

In [45]:
```
feat_importances.nlargest(10).plot(kind='barh', figsize=(10,10))
plt.title('Feature Importances')
plt.show()
```



Plot Tree

In [46]:
```
X.columns
```

Out[46]:
```
Index(['num_passengers', 'sales_channel', 'trip_type', 'purchase_lead', 'length_of_sta
y', 'flight_day', 'wants_extra_baggage', 'wants_preferred_seat', 'wants_in_flight_meal
s', 'flight_duration'], dtype='object')
```

In [ ]:

Cross-Validation

In [47]:
```
cv = cross_val_score(xgbmodel,X,y,cv=5,verbose=1,scoring='accuracy')
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.2s finished
```

In [48]:
```
cv.mean()
```

Out[48]:
```
0.61184
```

Table Formatted View

In [49]:
```
table = X_test.copy()
```

In [50]:
```
table["True Value"] = y_test.copy()
```

In [51]:
```
table["Predicted"] = np.round(y_pred,2)
```

In [52]:
```
table
```

Out[52]:

	num_passengers	sales_channel	trip_type	purchase_lead	length_of_stay	flight_day	wants_extra_baggage
11841	2	0	1	33	20	2	1
19602	1	0	1	115	137	5	1
45519	1	0	1	14	6	4	0
25747	1	0	1	1	2	1	1
42642	1	0	1	47	6	6	0
...	...	...	...	...	...	...	...
25091	1	0	1	31	20	7	1
27853	1	0	1	69	3	1	0
47278	2	0	1	94	6	4	0
37020	1	0	1	62	5	2	1
2217	1	0	1	61	105	4	1

10000 rows × 12 columns

Python code done by Dennis Lam

In [ ]:
```
```