In [1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# get current working directory
cwd = os.getcwd()

df = pd.read_csv(cwd + "/filtered_customer_booking.csv", index_col=0)
```

In [3]:
```
df = df.reset_index(drop=True)
```

In [4]:
```
df
```

Out[4]:

	num_passengers	sales_channel	trip_type	purchase_lead	length_of_stay	flight_hour	flight_day	route	booking_origin	wants_extra_baggage	wants
0	2	Internet	RoundTrip	262	19	7	6	AKLDEL	New Zealand	1	
1	1	Internet	RoundTrip	112	20	3	6	AKLDEL	New Zealand	0	
2	2	Internet	RoundTrip	243	22	17	3	AKLDEL	India	1	
3	1	Internet	RoundTrip	96	31	4	6	AKLDEL	New Zealand	0	
4	2	Internet	RoundTrip	68	22	15	3	AKLDEL	India	1	
...	...	...	...	...	...	...	...	...	...	...	...
49977	2	Internet	RoundTrip	27	6	9	6	PERPNH	Australia	1	
49978	1	Internet	RoundTrip	111	6	4	7	PERPNH	Australia	0	
49979	1	Internet	RoundTrip	24	6	22	6	PERPNH	Australia	0	
49980	1	Internet	RoundTrip	15	6	11	1	PERPNH	Australia	1	
49981	1	Internet	RoundTrip	19	6	10	4	PERPNH	Australia	0	

49982 rows × 14 columns

In [5]:
```
df_final = df
```

In [6]:
```
from sklearn.preprocessing import OneHotEncoder


#create instance of one hot encoder
encoder = OneHotEncoder(handle_unknown='ignore')

#one hot encode Sales Channel
encoder_df = pd.DataFrame(encoder.fit_transform(df[["sales_channel"]]).toarray())
encoder_df = encoder_df.rename(columns={0:'Internet', 1:'Mobile'})
df_final = df_final.join(encoder_df)

#one hot encode trip type
encoder_df = pd.DataFrame(encoder.fit_transform(df[["trip_type"]]).toarray())
encoder_df = encoder_df.rename(columns={0:'RoundTRip', 1:'OneWayTrip',2:'CircleTrip'})
df_final = df_final.join(encoder_df)
```

In [8]:
```
#drop categorical columns now
df_final.drop(['sales_channel', 'trip_type','booking_origin', 'route'], axis=1, inplace = True)
```

In [9]:
```
#store the label for supervised learning
label = df['booking_complete']
```

In [10]:
```
df_final = df_final.drop('booking_complete', axis=1)
```

In [11]:
```
df_final
```

Out[11]:

	num_passengers	purchase_lead	length_of_stay	flight_hour	flight_day	wants_extra_baggage	wants_preferred_seat	wants_in_flight_meals	flight_duratio
0	2	262	19	7	6	1	0	0	5.5
1	1	112	20	3	6	0	0	0	5.5
2	2	243	22	17	3	1	1	0	5.5
3	1	96	31	4	6	0	0	1	5.5
4	2	68	22	15	3	1	0	1	5.5
...	...	...	...	...	...	...	...	...	.
49977	2	27	6	9	6	1	0	1	5.6
49978	1	111	6	4	7	0	0	0	5.6
49979	1	24	6	22	6	0	0	1	5.6
49980	1	15	6	11	1	1	0	1	5.6
49981	1	19	6	10	4	0	1	0	5.6

49982 rows × 14 columns

Normalizaing the values

In [12]:
```
from sklearn.preprocessing import StandardScaler

#create a standard scaler object
scaler = StandardScaler()

#fit and transform the data
scaled_df = scaler.fit_transform(df_final)
```

In [13]:
```
#create a dataframe of scled data
scaled_df = pd.DataFrame(scaled_df, columns = df_final.columns)
```

In [14]:
```
# add the labels back to the dataframe
scaled_df['label']  = label
```

In [15]:
```
scaled_df
```

Out[15]:

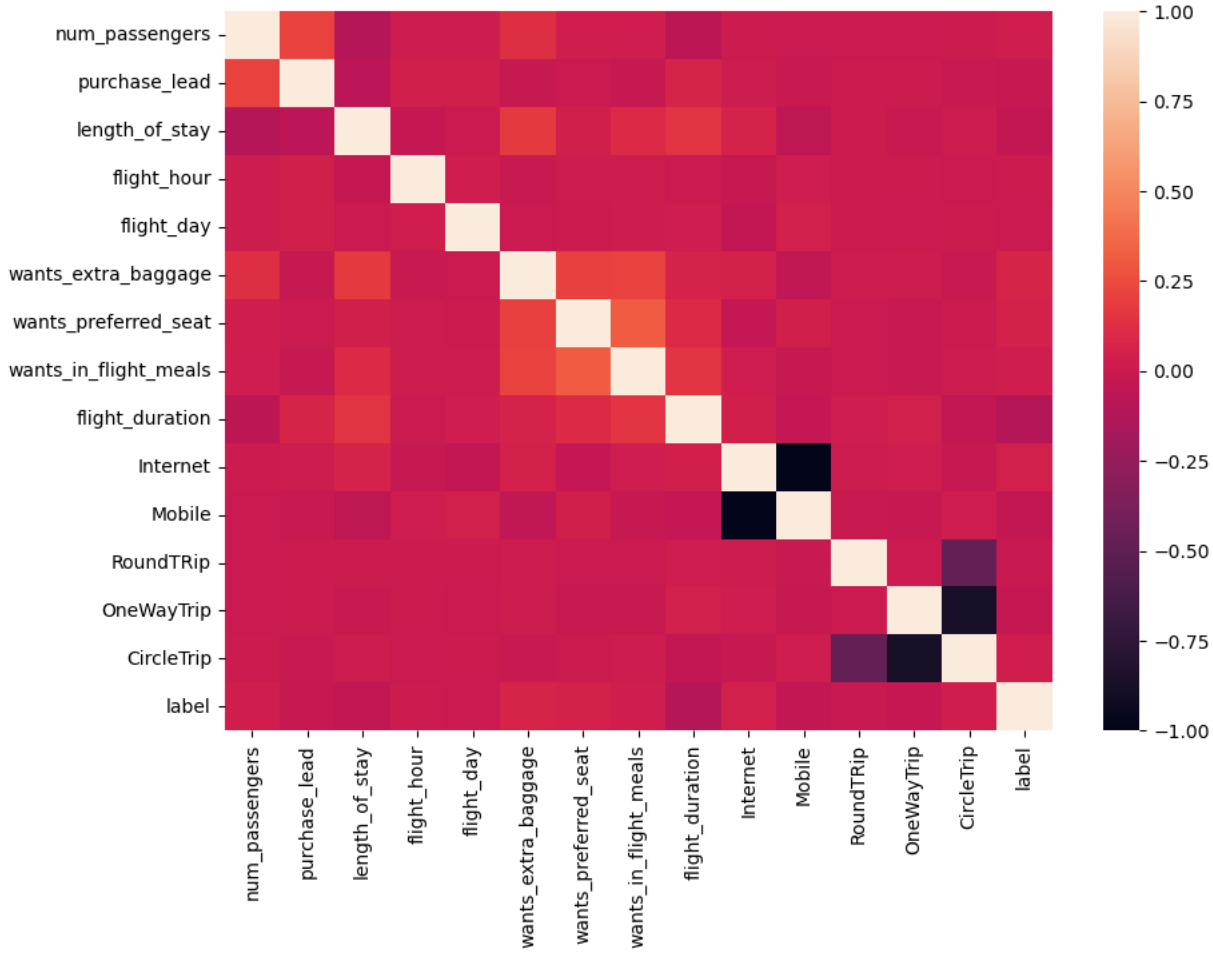
	num_passengers	purchase_lead	length_of_stay	flight_hour	flight_day	wants_extra_baggage	wants_preferred_seat	wants_in_flight_meals	flight_duratio
0	0.400769	1.971093	-0.119401	-0.381588	1.096876	0.703587	-0.650054	-0.863557	-1.17404
1	-0.579424	0.302987	-0.089895	-1.120618	1.096876	-1.421288	-0.650054	-0.863557	-1.17404
2	0.400769	1.759799	-0.030885	1.465988	-0.408618	0.703587	1.538334	-0.863557	-1.17404
3	-0.579424	0.125056	0.234662	-0.935861	1.096876	-1.421288	-0.650054	1.158002	-1.17404
4	0.400769	-0.186323	-0.030885	1.096473	-0.408618	0.703587	-0.650054	1.158002	-1.17404
...	...	...	...	...	...	...	...	...	.
49977	0.400769	-0.642272	-0.502969	-0.012073	1.096876	0.703587	-0.650054	1.158002	-1.10724
49978	-0.579424	0.291867	-0.502969	-0.935861	1.598707	-1.421288	-0.650054	-0.863557	-1.10724
49979	-0.579424	-0.675634	-0.502969	2.389776	1.096876	-1.421288	-0.650054	1.158002	-1.10724
49980	-0.579424	-0.775721	-0.502969	0.357443	-1.412280	0.703587	-0.650054	1.158002	-1.10724
49981	-0.579424	-0.731238	-0.502969	0.172685	0.093214	-1.421288	1.538334	-0.863557	-1.10724

49982 rows × 15 columns

Correlation matrix

In [16]:
```
corr = scaled_df.corr()

plt.figure(figsize=(10,7))

#plot the heatmap
sns.heatmap(corr)
```

Out[16]: <AxesSubplot:>



Splitting Train and Test Data

In [17]:
```
from sklearn.model_selection import train_test_split

X = scaled_df.iloc[:,:-1]
y = scaled_df['label']

X_train, X_test, y_train, y_test = train_test_split(X.to_numpy(), y.to_numpy(), test_size=0.20, random_state=42)
```

In [20]:
```
!pip install yellowbrick
```

```
Collecting yellowbrick
  Downloading yellowbrick-1.5-py3-none-any.whl (282 kB)
     ------------------------------------ 282.6/282.6 kB 2.2 MB/s eta 0:00:00
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in d:\anaconda3\lib\site-packages (from yellowbrick) (3.5.2)
Requirement already satisfied: scipy>=1.0.0 in d:\anaconda3\lib\site-packages (from yellowbrick) (1.9.1)
Requirement already satisfied: scikit-learn>=1.0.0 in d:\anaconda3\lib\site-packages (from yellowbrick) (1.2.2)
Requirement already satisfied: numpy>=1.16.0 in d:\anaconda3\lib\site-packages (from yellowbrick) (1.21.5)
Requirement already satisfied: cycler>=0.10.0 in d:\anaconda3\lib\site-packages (from yellowbrick) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in d:\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbric
k) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in d:\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbric
k) (1.4.2)
Requirement already satisfied: packaging>=20.0 in d:\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick)
(21.3)
Requirement already satisfied: pillow>=6.2.0 in d:\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick)
(9.2.0)
Requirement already satisfied: pyparsing>=2.2.1 in d:\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick)
(3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in d:\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbr
ick) (2.8.2)
Requirement already satisfied: joblib>=1.1.1 in d:\anaconda3\lib\site-packages (from scikit-learn>=1.0.0->yellowbrick) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\anaconda3\lib\site-packages (from scikit-learn>=1.0.0->yellowbrick)
(2.2.0)
Requirement already satisfied: six>=1.5 in d:\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2
->yellowbrick) (1.16.0)
Installing collected packages: yellowbrick
Successfully installed yellowbrick-1.5

WARNING: Ignoring invalid distribution -cikit-learn (d:\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -cikit-learn (d:\anaconda3\lib\site-packages)
```

In [21]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.inspection import permutation_importance

from yellowbrick.classifier import ConfusionMatrix
from sklearn.model_selection import GridSearchCV,  RepeatedStratifiedKFold
```

In [22]:
```
"""
    Create functions to fit and predict the values of whether customer would complete the booking.
    Also functions with metrics to evaluate the model prediction.

"""

def model_fit_predict(model, X, y, X_predict):
    model.fit(X, y)
    return model.predict(X_predict)

def acc_score(y_true, y_pred):
    return accuracy_score(y_true, y_pred)

def pre_score(y_true, y_pred):
    return precision_score(y_true, y_pred)

def f_score(y_true, y_pred):
    return f1_score(y_true, y_pred)
```

Random Forest Classifier

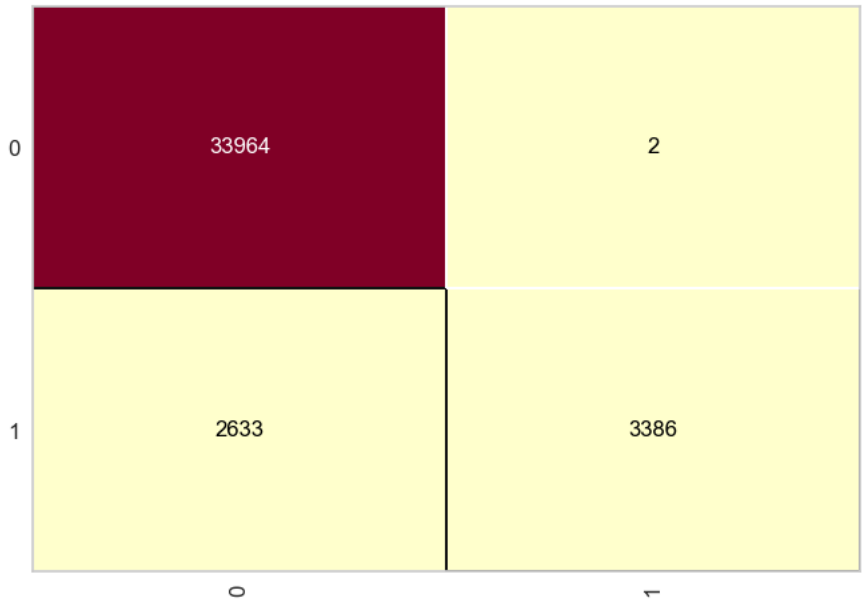
In [23]:
```
#create an instance of the classifier and fit the training data
clf_rf = RandomForestClassifier(max_depth =50 , min_samples_split=5,random_state=0)
```

In [24]:
```
y_pred_train = model_fit_predict(clf_rf, X_train, y_train, X_train)
set(y_pred_train)

#f1 score for training data
f1 = round(f1_score(y_train, y_pred_train),2)

#accuracy score for training data
acc = round(accuracy_score(y_train, y_pred_train),2)

#precision score for training data
pre = round(precision_score(y_train, y_pred_train),2)

print(f"Accuracy, precision and f1-score for training data are {acc}, {pre} and {f1} respectively")
```

```
Accuracy, precision and f1-score for training data are 0.93, 1.0 and 0.72 respectively
```

In [25]:
```
cm = ConfusionMatrix(clf_rf, classes=[0,1])
cm.fit(X_train, y_train)

cm.score(X_train, y_train)
```

Out[25]:
```
0.9341002876078529
```



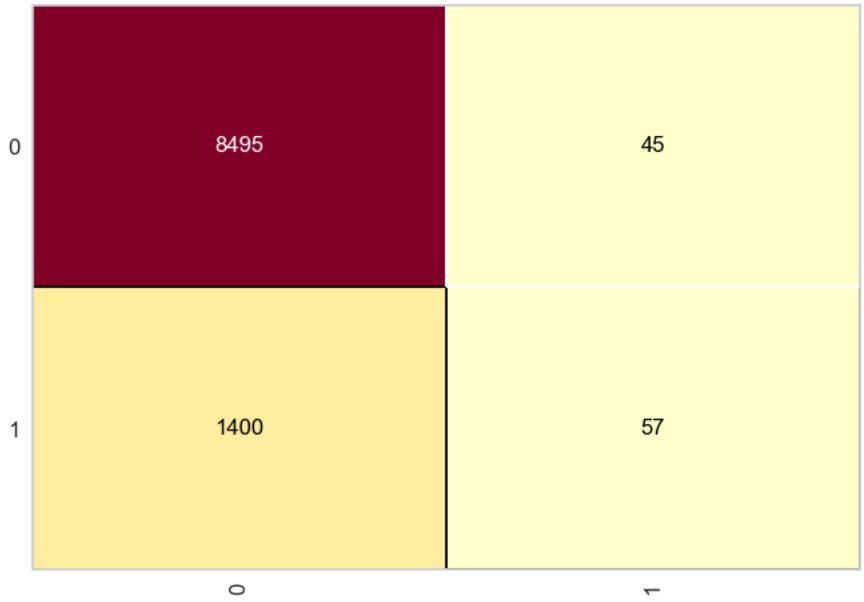
Checking Testing accuracy

In [26]:
```
#create an instance of the classifier and fit the training data
clf_rf = RandomForestClassifier(max_depth =50 , min_samples_split=5,random_state=0)

y_pred_test = model_fit_predict(clf_rf, X_train, y_train, X_test)

#f1 score for training data
f1 = round(f1_score(y_test, y_pred_test),2)

#accuracy score for training data
acc = round(accuracy_score(y_test, y_pred_test),2)

#precision score for training data
pre = round(precision_score(y_test, y_pred_test),2)

print(f"Accuracy, precision and f1-score for training data are {acc}, {pre} and {f1} respectively")
```

```
Accuracy, precision and f1-score for training data are 0.86, 0.56 and 0.07 respectively
```

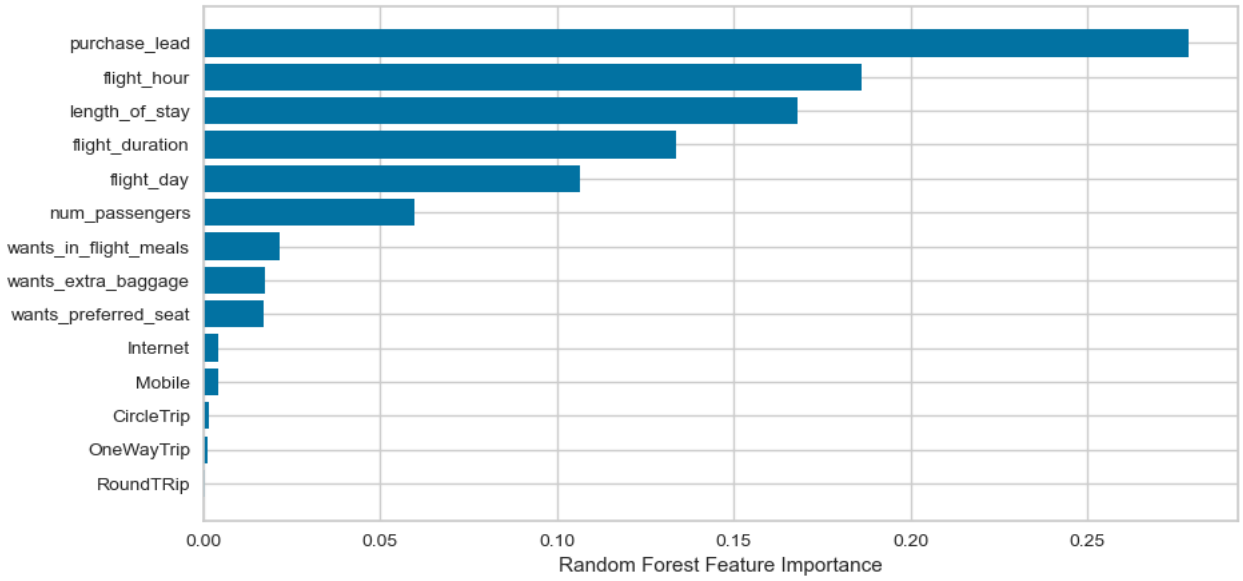
In [27]:
```
cm = ConfusionMatrix(clf_rf, classes=[0,1])
cm.fit(X_train, y_train)

cm.score(X_test, y_test)
```

Out[27]:
```
0.8554566369910973
```



In [28]:
```
plt.figure(figsize=(10,5))
sorted_idx = clf_rf.feature_importances_.argsort()
plt.barh(scaled_df.iloc[:,:-1].columns[sorted_idx], clf_rf.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

Out[28]:
```
Text(0.5, 0, 'Random Forest Feature Importance')
```



One major problem behind getting low F1 score is imbalanced dataset. We have higher entries that are classified 0 than 1. We could reduce the number of entries that are classified 0 to be equal around the number of entries that are classified as 1.

Balancing the dataset

In [29]:
```
scaled_df.label.value_counts()
```

Out[29]:
```
0    42506
1     7476
Name: label, dtype: int64
```

In [30]:
```
#create a dataframe having all labels 0 with 10000 samples
scaled_df_0 = scaled_df[scaled_df.label ==0].sample(n=8000)
```

In [31]:
```
#concatenate the two dataframee, one havng all labels 0 and other having all labels as 1
scaled_df_new = pd.concat([scaled_df[scaled_df.label==1], scaled_df_0], ignore_index=True)
```

In [32]:
```
#shuffle the dataframe rows
scaled_df_new = scaled_df_new.sample(frac = 1).reset_index(drop=True)
```

In [33]:
```
scaled_df_new
```

Out[33]:

	num_passengers	purchase_lead	length_of_stay	flight_hour	flight_day	wants_extra_baggage	wants_preferred_seat	wants_in_flight_meals	flight_duratio
0	-0.579424	-0.886928	1.886956	-0.935861	0.093214	0.703587	1.538334	1.158002	-0.18528
1	-0.579424	-0.586669	-0.561979	-1.305376	1.096876	-1.421288	1.538334	-0.863557	0.87029
2	-0.579424	1.048074	0.205157	0.542200	0.093214	0.703587	-0.650054	1.158002	1.03731
3	-0.579424	1.348333	-0.591484	0.542200	0.093214	-1.421288	-0.650054	1.158002	-1.10724
4	-0.579424	0.981350	-0.060390	-0.566346	-0.910449	-1.421288	-0.650054	1.158002	1.03731
...	...	...	...	...	...	...	...	...	...
15471	1.380962	2.037817	0.057631	1.281231	1.598707	0.703587	1.538334	1.158002	0.19552
15472	-0.579424	-0.842445	-0.532474	-0.751103	1.598707	-1.421288	-0.650054	-0.863557	-1.74192
15473	0.400769	-0.575548	-0.532474	-0.012073	1.096876	0.703587	1.538334	-0.863557	-0.43915
15474	0.400769	-0.408738	-0.561979	-0.196830	0.093214	0.703587	-0.650054	-0.863557	-0.18528
15475	-0.579424	-0.419858	-0.001380	0.542200	-0.408618	0.703587	-0.650054	1.158002	-1.74192

15476 rows × 15 columns

In [34]:
```
X = scaled_df_new.iloc[:,:-1]
y = scaled_df_new['label']

X_train, X_test, y_train, y_test = train_test_split(X.to_numpy(), y.to_numpy(), test_size=0.20, random_state=42)
```

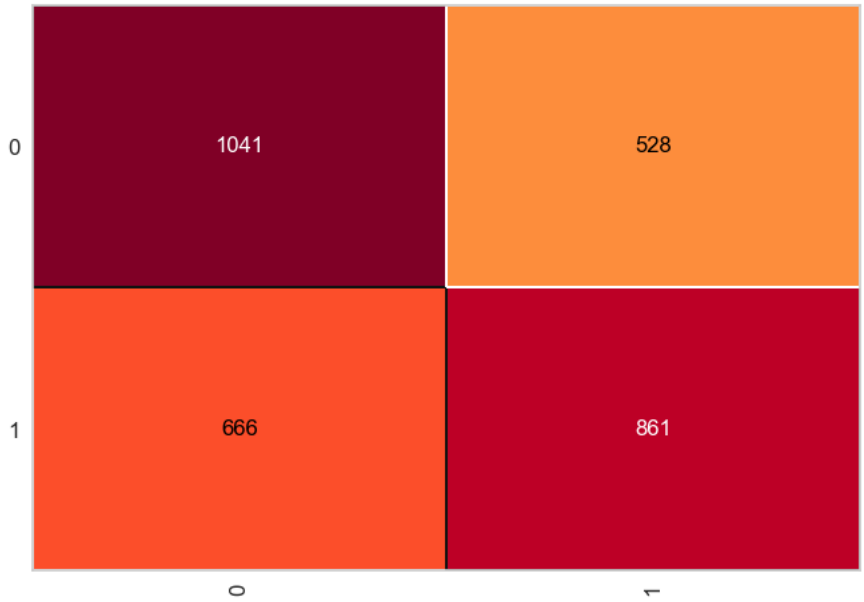
In [35]:
```
#create an instance of the classifier and fit the training data
clf_rf = RandomForestClassifier(n_estimators=50,max_depth =50 , min_samples_split=5,random_state=0)
```

In [36]:
```
ed_test = model_fit_predict(clf_rf, X_train, y_train, X_test)

score for training data
round(f1_score(y_test, y_pred_test),2)

uracy score for training data
= round(accuracy_score(y_test, y_pred_test),2)

cision score for training data
= round(precision_score(y_test, y_pred_test),2)

ll = round(recall_score(y_test, y_pred_test),2)

ificity = round(recall_score(y_test, y_pred_test, pos_label=0),2)

t(f"Accuracy, precision, recall and f1-score for training data are {acc}, {pre}, {recall}, {specificity} and {f1} respectively")
```

```
Accuracy, precision, recall and f1-score for training data are 0.61, 0.62, 0.56, 0.66 and 0.59 respectively
```

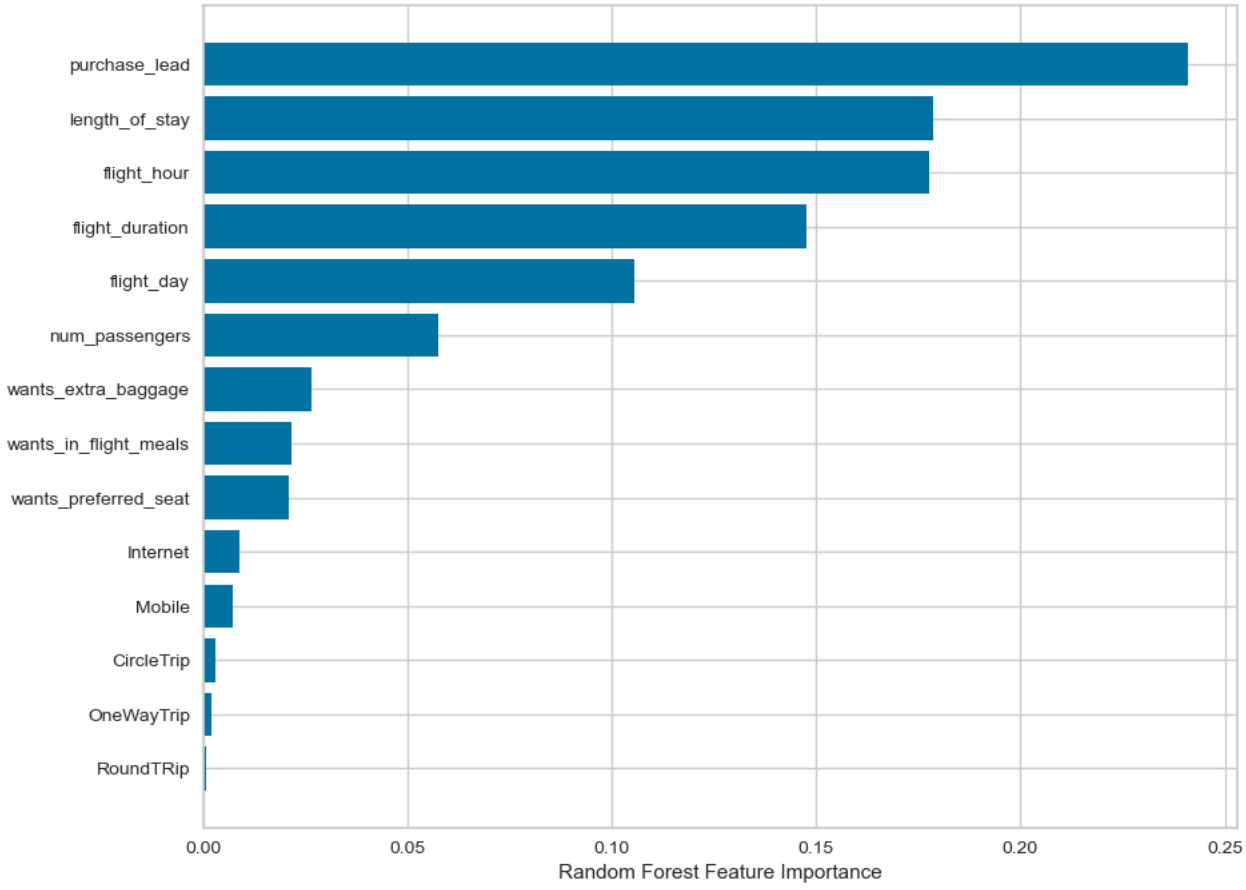
In [37]:
```
cm = ConfusionMatrix(clf_rf, classes=[0,1])
cm.fit(X_train, y_train)

cm.score(X_test, y_test)
```

Out[37]:
```
0.6143410852713178
```



In [38]:
```
plt.figure(figsize=(10,8))
sorted_idx = clf_rf.feature_importances_.argsort()
plt.barh(scaled_df.iloc[:,:-1].columns[sorted_idx], clf_rf.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

Out[38]:
```
Text(0.5, 0, 'Random Forest Feature Importance')
```



In [ ]: