# Activity_Course 6 Automatidata project lab

August 2, 2023

## 1 Automatidata project

**Course 6 - The Nuts and bolts of machine learning**

You are a data professional in a data analytics firm called Automatidata. Their client, the New York City Taxi & Limousine Commission (New York City TLC), was impressed with the work you have done and has requested that you build a machine learning model to predict if a customer will not leave a tip. They want to use the model in an app that will alert taxi drivers to customers who are unlikely to tip, since drivers depend on tips.

A notebook was structured and prepared to help you in this project. Please complete the following questions.

## 2 Course 6 End-of-course project: Build a machine learning model

In this activity, you will practice using tree-based modeling techniques to predict on a binary target class.

**The purpose** of this model is to find ways to generate more revenue for taxi cab drivers.

**The goal** of this model is to predict whether or not a customer is a generous tipper.

*This activity has three parts:*

**Part 1:** Ethical considerations * Consider the ethical implications of the request

- Should the objective of the model be adjusted?

**Part 2:** Feature engineering

- Perform feature selection, extraction, and transformation to prepare the data for modeling

**Part 3:** Modeling

- Build the models, evaluate them, and advise on next steps

Follow the instructions and answer the questions below to complete the activity. Then, complete an Executive Summary using the questions listed on the PACE Strategy Document.

Be sure to complete this activity before moving on. The next course item will provide you with a completed exemplar to compare to your own work.

# 3 Build a machine learning model

# 4 PACE stages

Throughout these project notebooks, you'll see references to the problem-solving framework PACE. The following notebook components are labeled with the respective PACE stage: Plan, Analyze, Construct, and Execute.

## 4.1 PACE: Plan

Consider the questions in your PACE Strategy Document to reflect on the Plan stage.

In this stage, consider the following questions:

1. What are you being asked to do?
2. What are the ethical implications of the model? What are the consequences of your model making errors?

- What is the likely effect of the model when it predicts a false negative (i.e., when the model says a customer will give a tip, but they actually won't)?
- What is the likely effect of the model when it predicts a false positive (i.e., when the model says a customer will not give a tip, but they actually will)?

3. Do the benefits of such a model outweigh the potential problems?
4. Would you proceed with the request to build this model? Why or why not?
5. Can the objective be modified to make it less problematic?

1. I am asked to develop a machine Learning model, whether a customer will leave a tip for the driver or not.
2. THe likely effect of the model when it's predict a false negative is that this will make upset the driver and driver will unlikely to get more rides for which model says the customer will tip more and more. The likely effect of model when it predicts a false positive is that the driver will not upset, but it will get happy. But such rides may not be considered by drivers at first, because the model is not offering tip. the rides may have to wait for a while in an area till a driver picks up those rides.
3. Yes the model may quickly identify those rides, because the customer will ready to pay for get ride instantly, and the driver will show him more professional.
4. Yes, I would proceed with the request to build this model.
5. Yes, the objective can be modified in a way that "you build a machine learning model to predict if a customer will leave a tip". This will help the app to recommend the rides first to the good drivers which will having tips. The driver is happy, and the customer is happy.

Suppose you were to modify the modeling objective so, instead of predicting people who won't tip at all, you predicted people who are particularly generous—those who will tip 20% or more? Consider the following questions:

1. What features do you need to make this prediction?
2. What would be the target variable?
3. What metric should you use to evaluate your model? Do you have enough information to decide this now?

We will need the customer's rating and his past rides information. Our target variable should be a binary variable demonstrating the probability of giving the tip. We should use Recall, because false negative is a problem.

***Complete the following steps to begin:***

### 4.1.1 Task 1. Imports and data loading

Import packages and libraries needed to build and evaluate random forest and XGBoost classification models.

[181]:
```
# Import packages and libraries
### YOUR CODE HERE ###
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier, plot_importance
```

[182]:
```
# RUN THIS CELL TO SEE ALL COLUMNS
# This lets us see all of the columns, preventing Juptyer from redacting them.
pd.set_option('display.max_columns', None)
```

Begin by reading in the data. There are two dataframes: one containing the original data, the other containing the mean durations, mean distances, and predicted fares from the previous course's project called nyc_preds_means.csv.

**Note:** `Pandas` reads in the dataset as `df0`, now inspect the first five rows. As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the .csv file, or provide more code, in order to access the dataset and proceed with this lab. Please continue with this activity by completing the following instructions.

[183]:
```
# RUN THE CELL BELOW TO IMPORT YOUR DATA.

# Load dataset into dataframe
```

```
df0 = pd.read_csv('2017_Yellow_Taxi_Trip_Data.csv')

# Import predicted fares and mean distance and duration from previous course
nyc_preds_means = pd.read_csv('nyc_preds_means.csv')
```

Inspect the first few rows of `df0`.

[184]:
```
# Inspect the first few rows of df0
### YOUR CODE HERE ###
df0.head()
```

[184]:
```
   Unnamed: 0  VendorID    tpep_pickup_datetime   tpep_dropoff_datetime  \
0    24870114         2   03/25/2017 8:55:43 AM   03/25/2017 9:09:47 AM
1    35634249         1   04/11/2017 2:53:28 PM   04/11/2017 3:19:58 PM
2   106203690         1   12/15/2017 7:26:56 AM   12/15/2017 7:34:08 AM
3    38942136         2   05/07/2017 1:17:59 PM   05/07/2017 1:48:14 PM
4    30841670         2  04/15/2017 11:32:20 PM  04/15/2017 11:49:03 PM

   passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
0                6           3.34           1                  N
1                1           1.80           1                  N
2                1           1.00           1                  N
3                1           3.70           1                  N
4                1           4.37           1                  N

   PULocationID  DOLocationID  payment_type  fare_amount  extra  mta_tax  \
0           100           231             1         13.0    0.0      0.5
1           186            43             1         16.0    0.0      0.5
2           262           236             1          6.5    0.0      0.5
3           188            97             1         20.5    0.0      0.5
4             4           112             2         16.5    0.5      0.5

   tip_amount  tolls_amount  improvement_surcharge  total_amount
0        2.76           0.0                    0.3         16.56
1        4.00           0.0                    0.3         20.80
2        1.45           0.0                    0.3          8.75
3        6.39           0.0                    0.3         27.69
4        0.00           0.0                    0.3         17.80
```

Inspect the first few rows of `nyc_preds_means`.

[185]:
```
# Inspect the first few rows of `nyc_preds_means`
### YOUR CODE HERE ###
nyc_preds_means.head()
```

[185]:
```
   mean_duration  mean_distance  predicted_fare
0      22.847222       3.521667       16.434245
1      24.470370       3.108889       16.052218
```

```
2     7.250000       0.881429       7.053706
3    30.250000       3.700000      18.731650
4    14.616667       4.435000      15.845642
```

**Join the two dataframes** Join the two dataframes using a method of your choice.

[186]:
```python
# Merge datasets
### YOUR CODE HERE ###
data = pd.concat([df0, nyc_preds_means], axis=1)
data.head()
```

[186]:
```
   Unnamed: 0  VendorID    tpep_pickup_datetime   tpep_dropoff_datetime  \
0    24870114         2   03/25/2017 8:55:43 AM   03/25/2017 9:09:47 AM
1    35634249         1   04/11/2017 2:53:28 PM   04/11/2017 3:19:58 PM
2   106203690         1   12/15/2017 7:26:56 AM   12/15/2017 7:34:08 AM
3    38942136         2   05/07/2017 1:17:59 PM   05/07/2017 1:48:14 PM
4    30841670         2  04/15/2017 11:32:20 PM  04/15/2017 11:49:03 PM

   passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
0                6           3.34           1                  N
1                1           1.80           1                  N
2                1           1.00           1                  N
3                1           3.70           1                  N
4                1           4.37           1                  N

   PULocationID  DOLocationID  payment_type  fare_amount  extra  mta_tax  \
0           100           231             1         13.0    0.0      0.5
1           186            43             1         16.0    0.0      0.5
2           262           236             1          6.5    0.0      0.5
3           188            97             1         20.5    0.0      0.5
4             4           112             2         16.5    0.5      0.5

   tip_amount  tolls_amount  improvement_surcharge  total_amount  \
0        2.76           0.0                    0.3         16.56
1        4.00           0.0                    0.3         20.80
2        1.45           0.0                    0.3          8.75
3        6.39           0.0                    0.3         27.69
4        0.00           0.0                    0.3         17.80

   mean_duration  mean_distance  predicted_fare
0      22.847222       3.521667       16.434245
1      24.470370       3.108889       16.052218
2       7.250000       0.881429        7.053706
3      30.250000       3.700000       18.731650
4      14.616667       4.435000       15.845642
```

### 4.2 PACE: Analyze

Consider the questions in your PACE Strategy Documentto reflect on the Analyze stage.

#### 4.2.1 Task 2. Feature engineering

You have already prepared much of this data and performed exploratory data analysis (EDA) in previous courses.

Call `info()` on the new combined dataframe.

[187]:
```
#==> ENTER YOUR CODE HERE
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             22699 non-null  int64
 1   VendorID               22699 non-null  int64
 2   tpep_pickup_datetime   22699 non-null  object
 3   tpep_dropoff_datetime  22699 non-null  object
 4   passenger_count        22699 non-null  int64
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64
 7   store_and_fwd_flag     22699 non-null  object
 8   PULocationID           22699 non-null  int64
 9   DOLocationID           22699 non-null  int64
 10  payment_type           22699 non-null  int64
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
 18  mean_duration          22699 non-null  float64
 19  mean_distance          22699 non-null  float64
 20  predicted_fare         22699 non-null  float64
dtypes: float64(11), int64(7), object(3)
memory usage: 3.6+ MB
```

You know from your EDA that customers who pay cash generally have a tip amount of $0. To meet the modeling objective, you'll need to sample the data to select only the customers who pay with credit card.

Copy `df0` and assign the result to a variable called `df1`. Then, use a Boolean mask to filter `df1` so it contains only customers who paid with credit card.

[188]:
```python
# Subset the data to isolate only customers who paid by credit card
#==> ENTER YOUR CODE HERE
df1 = data[data['payment_type'] == 1]
```

[189]:
```python
df1.head()
```

[189]:
```
   Unnamed: 0  VendorID   tpep_pickup_datetime  tpep_dropoff_datetime  \
0    24870114         2  03/25/2017 8:55:43 AM  03/25/2017 9:09:47 AM
1    35634249         1  04/11/2017 2:53:28 PM  04/11/2017 3:19:58 PM
2   106203690         1  12/15/2017 7:26:56 AM  12/15/2017 7:34:08 AM
3    38942136         2  05/07/2017 1:17:59 PM  05/07/2017 1:48:14 PM
5    23345809         2  03/25/2017 8:34:11 PM  03/25/2017 8:42:11 PM

   passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
0                6           3.34           1                  N
1                1           1.80           1                  N
2                1           1.00           1                  N
3                1           3.70           1                  N
5                6           2.30           1                  N

   PULocationID  DOLocationID  payment_type  fare_amount  extra  mta_tax  \
0           100           231             1         13.0    0.0      0.5
1           186            43             1         16.0    0.0      0.5
2           262           236             1          6.5    0.0      0.5
3           188            97             1         20.5    0.0      0.5
5           161           236             1          9.0    0.5      0.5

   tip_amount  tolls_amount  improvement_surcharge  total_amount  \
0        2.76           0.0                    0.3         16.56
1        4.00           0.0                    0.3         20.80
2        1.45           0.0                    0.3          8.75
3        6.39           0.0                    0.3         27.69
5        2.06           0.0                    0.3         12.36

   mean_duration  mean_distance  predicted_fare
0      22.847222       3.521667       16.434245
1      24.470370       3.108889       16.052218
2       7.250000       0.881429        7.053706
3      30.250000       3.700000       18.731650
5      11.855376       2.052258       10.441351
```

**Target** Notice that there isn't a column that indicates tip percent, which is what you need to create the target variable. You'll have to engineer it.

Add a `tip_percent` column to the dataframe by performing the following calculation:

$$tip\ percent = \frac{tip\ amount}{total\ amount - tip\ amount}$$

[190]:
```
# Create tip % col
#==> ENTER YOUR CODE HERE
df1['tip_percent'] = df1['tip_amount'] / (df1.total_amount - df1.tip_amount)
df1.tip_percent.head()
```

[190]:
```
0    0.200000
1    0.238095
2    0.198630
3    0.300000
5    0.200000
Name: tip_percent, dtype: float64
```

Now create another column called `generous`. This will be the target variable. The column should be a binary indicator of whether or not a customer tipped  20% (0=no, 1=yes).

1. Begin by making the `generous` column a copy of the `tip_percent` column.
2. Reassign the column by converting it to Boolean (True/False).
3. Reassign the column by converting Boolean to binary (1/0).

[191]:
```
# Create 'generous' col (target)
#==> ENTER YOUR CODE HERE
df1['generous'] = df1['tip_percent'] >= 0.2
df1.generous = df1.generous.astype(int)
df1['generous'].value_counts(normalize=True)
```

[191]:
```
0    0.651425
1    0.348575
Name: generous, dtype: float64
```

HINT

To convert from Boolean to binary, use `.astype(int)` on the column.

**Create day column** Next, you're going to be working with the pickup and dropoff columns.

Convert the `tpep_pickup_datetime` and `tpep_dropoff_datetime` columns to datetime.

[192]:
```
# Convert pickup and dropoff cols to datetime
#==> ENTER YOUR CODE HERE
df1.tpep_pickup_datetime = pd.to_datetime(df1.tpep_pickup_datetime)
df1.tpep_dropoff_datetime = pd.to_datetime(df1.tpep_dropoff_datetime)
```

Create a `day` column that contains only the day of the week when each passenger was picked up. Then, convert the values to lowercase.

[193]:
```python
# Create a 'day' col
#==> ENTER YOUR CODE HERE
df1['day'] = df1.tpep_pickup_datetime.dt.day_name().str.lower()
```

HINT

To convert to day name, use `dt.day_name()` on the column.

**Create time of day columns** Next, engineer four new columns that represent time of day bins. Each column should contain binary values (0=no, 1=yes) that indicate whether a trip began (picked up) during the following times:

`am_rush` = [06:00–10:00)
`daytime` = [10:00–16:00)
`pm_rush` = [16:00–20:00)
`nighttime` = [20:00–06:00)

To do this, first create the four columns. For now, each new column should be identical and contain the same information: the hour (only) from the `tpep_pickup_datetime` column.

[194]:
```python
# Create 'am_rush' col
#==> ENTER YOUR CODE HERE
df1['am_rush'] = df1.tpep_pickup_datetime.dt.hour

# Create 'daytime' col
#==> ENTER YOUR CODE HERE
df1['daytime'] = df1.tpep_pickup_datetime.dt.hour

# Create 'pm_rush' col
#==> ENTER YOUR CODE HERE
df1['pm_rush'] = df1.tpep_pickup_datetime.dt.hour

# Create 'nighttime' col
#==> ENTER YOUR CODE HERE
df1['nighttime'] = df1.tpep_pickup_datetime.dt.hour
```

You'll need to write four functions to convert each new column to binary (0/1). Begin with `am_rush`. Complete the function so if the hour is between [06:00–10:00), it returns 1, otherwise, it returns 0.

[195]:
```python
# Define 'am_rush()' conversion function [06:00-10:00)
#==> ENTER YOUR CODE HERE
def am_rush(hour):
    if 6 <= hour < 10:
        return 1
    return 0
```

Now, apply the `am_rush()` function to the `am_rush` series to perform the conversion. Print the first five values of the column to make sure it did what you expected it to do.

**Note:** Be careful! If you run this cell twice, the function will be reapplied and the values will all be changed to 0.

[196]:
```python
# Apply 'am_rush' function to the 'am_rush' series
#==> ENTER YOUR CODE HERE
df1.am_rush = df1.am_rush.apply(am_rush)
```

Write functions to convert the three remaining columns and apply them to their respective series.

[197]:
```python
# Define 'daytime()' conversion function [10:00-16:00)
#==> ENTER YOUR CODE HERE
def daytime(hour):
    if 10 <= hour < 16:
        return 1
    return 0
```

[198]:
```python
# Apply 'daytime()' function to the 'daytime' series
#==> ENTER YOUR CODE HERE
df1.daytime = df1.daytime.apply(daytime)
```

[199]:
```python
# Define 'pm_rush()' conversion function [16:00-20:00)
#==> ENTER YOUR CODE HERE
def pm_rush(hour):
    if 16 <= hour < 20:
        return 1
    return 0
```

[200]:
```python
# Apply 'pm_rush()' function to the 'pm_rush' series
#==> ENTER YOUR CODE HERE
df1.pm_rush = df1.daytime.apply(pm_rush)
```

[201]:
```python
# Define 'nighttime()' conversion function [20:00-06:00)
#==> ENTER YOUR CODE HERE
def nighttime(hour):
    if 20 <= hour < 24 or 0 <= hour < 6:
        return 1
    return 0
```

[202]:
```python
# Apply 'nighttime' function to the 'nighttime' series
#==> ENTER YOUR CODE HERE
df1.nighttime = df1.daytime.apply(nighttime)
```

**Create `month` column** Now, create a `month` column that contains only the abbreviated name of the month when each passenger was picked up, then convert the result to lowercase.

HINT

Refer to the strftime cheatsheet for help.

[203]:
```python
# Create 'month' col
#==> ENTER YOUR CODE HERE
df1['month'] = df1.tpep_pickup_datetime.dt.month_name().str.slice(stop=3)
```

Examine the first five rows of your dataframe.

[204]:
```python
#==> ENTER YOUR CODE HERE
df1.head()
```

[204]:
```
   Unnamed: 0  VendorID tpep_pickup_datetime tpep_dropoff_datetime  \
0    24870114         2  2017-03-25 08:55:43   2017-03-25 09:09:47
1    35634249         1  2017-04-11 14:53:28   2017-04-11 15:19:58
2   106203690         1  2017-12-15 07:26:56   2017-12-15 07:34:08
3    38942136         2  2017-05-07 13:17:59   2017-05-07 13:48:14
5    23345809         2  2017-03-25 20:34:11   2017-03-25 20:42:11

   passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
0                6           3.34           1                  N
1                1           1.80           1                  N
2                1           1.00           1                  N
3                1           3.70           1                  N
5                6           2.30           1                  N

   PULocationID  DOLocationID  payment_type  fare_amount  extra  mta_tax  \
0           100           231             1         13.0    0.0      0.5
1           186            43             1         16.0    0.0      0.5
2           262           236             1          6.5    0.0      0.5
3           188            97             1         20.5    0.0      0.5
5           161           236             1          9.0    0.5      0.5

   tip_amount  tolls_amount  improvement_surcharge  total_amount  \
0        2.76           0.0                    0.3         16.56
1        4.00           0.0                    0.3         20.80
2        1.45           0.0                    0.3          8.75
3        6.39           0.0                    0.3         27.69
5        2.06           0.0                    0.3         12.36

   mean_duration  mean_distance  predicted_fare  tip_percent  generous  \
0      22.847222       3.521667       16.434245     0.200000         1
1      24.470370       3.108889       16.052218     0.238095         1
2       7.250000       0.881429        7.053706     0.198630         0
3      30.250000       3.700000       18.731650     0.300000         1
5      11.855376       2.052258       10.441351     0.200000         1

        day  am_rush  daytime  pm_rush  nighttime month
0  saturday        1        0        0          1   Mar
1   tuesday        0        1        0          1   Apr
```

```
2    friday        1        0        0           1   Dec
3    sunday        0        1        0           1   May
5  saturday        0        0        0           1   Mar
```

**Drop columns** Drop redundant and irrelevant columns as well as those that would not be available when the model is deployed. This includes information like payment type, trip distance, tip amount, tip percentage, total amount, toll amount, etc. The target variable (`generous`) must remain in the data because it will get isolated as the `y` data for modeling.

[205]:
```
# Drop columns
#==> ENTER YOUR CODE HERE
df1 = df1.drop(columns=['Unnamed: 0', 'tpep_pickup_datetime',␣
 ↪'tpep_dropoff_datetime', 'trip_distance', 'store_and_fwd_flag',␣
 ↪'fare_amount', 'payment_type', 'extra', 'mta_tax', 'tip_amount',␣
 ↪'tolls_amount', 'improvement_surcharge', 'total_amount', 'tip_percent'])
```

**Variable encoding** Many of the columns are categorical and will need to be dummied (converted to binary). Some of these columns are numeric, but they actually encode categorical information, such as `RatecodeID` and the pickup and dropoff locations. To make these columns recognizable to the `get_dummies()` function as categorical variables, you'll first need to convert them to `type(str)`.

1. Define a variable called `cols_to_str`, which is a list of the numeric columns that contain categorical information and must be converted to string: `RatecodeID`, `PULocationID`, `DOLocationID`.
2. Write a for loop that converts each column in `cols_to_str` to string.

[206]:
```
# 1. Define list of cols to convert to string
#==> ENTER YOUR CODE HERE
cols_to_str = ['RatecodeID', 'PULocationID', 'DOLocationID']

# 2. Convert each column to string
#==> ENTER YOUR CODE HERE
for i in cols_to_str:
    df1[i] = df1[i].astype(str)
```

HINT

To convert to string, use `astype(str)` on the column.

Now convert all the categorical columns to binary.

1. Call `get_dummies()` on the dataframe and assign the results back to a new dataframe called `df2`.

[207]:
```
# Convert categoricals to binary
#==> ENTER YOUR CODE HERE
df2 = pd.get_dummies(df1)
```

**Evaluation metric** Before modeling, you must decide on an evaluation metric.

1. Examine the class balance of your target variable.

[208]:
```
# Get class balance of 'generous' col
#==> ENTER YOUR CODE HERE
df2['generous'].value_counts(normalize=True)
```

[208]:
```
0    0.651425
1    0.348575
Name: generous, dtype: float64
```

Approximately 1/3 of the customers in this dataset were "generous" (tipped 20%). The dataset is imbalanced, but not extremely so.

To determine a metric, consider the cost of both kinds of model error: * False positives (the model predicts a tip 20%, but the customer does not give one) * False negatives (the model predicts a tip < 20%, but the customer gives more)

False positives are worse for cab drivers, because they would pick up a customer expecting a good tip and then not receive one, frustrating the driver.

False negatives are worse for customers, because a cab driver would likely pick up a different customer who was predicted to tip more—even when the original customer would have tipped generously.

**The stakes are relatively even. You want to help taxi drivers make more money, but you don't want this to anger customers. Your metric should weigh both precision and recall equally. Which metric is this?**

That's F1 Score

### 4.3 PACE: Construct

Consider the questions in your PACE Strategy Document to reflect on the Construct stage.

#### 4.3.1 Task 3. Modeling

**Split the data** Now you're ready to model. The only remaining step is to split the data into features/target variable and training/testing data.

1. Define a variable `y` that isolates the target variable (`generous`).
2. Define a variable `X` that isolates the features.
3. Split the data into training and testing sets. Put 20% of the samples into the test set, stratify the data, and set the random state.

[209]:
```
# Isolate target variable (y)
#==> ENTER YOUR CODE HERE
y = df2.generous
```

```
# Isolate the features (X)
#==> ENTER YOUR CODE HERE
X = df2.drop(columns=['generous'])

# Split into train and test sets
#==> ENTER YOUR CODE HERE
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=123)
```

[210]:
```
len(X_train)
```

[210]: 12212

**Random forest** Begin with using `GridSearchCV` to tune a random forest model.

1. Instantiate the random forest classifier `rf` and set the random state.
2. Create a dictionary `cv_params` of any of the following hyperparameters and their corresponding values to tune. The more you tune, the better your model will fit the data, but the longer it will take.

- `max_depth`
- `max_features`
- `max_samples`
- `min_samples_leaf`
- `min_samples_split`
- `n_estimators`

3. Define a dictionary `scoring` of scoring metrics for GridSearch to capture (precision, recall, F1 score, and accuracy).
4. Instantiate the `GridSearchCV` object `rf1`. Pass to it as arguments:

- estimator=`rf`
- param_grid=`cv_params`
- scoring=`scoring`
- cv: define the number of you cross-validation folds you want (`cv=_`)
- refit: indicate which evaluation metric you want to use to select the model (`refit=_`)

**Note:** `refit` should be set to `'f1'`.

[211]:
```
# 1. Instantiate the random forest classifier
#==> ENTER YOUR CODE HERE
rf = RandomForestClassifier(random_state=42)

# 2. Create a dictionary of hyperparameters to tune
#==> ENTER YOUR CODE HERE
```

```python
cv_params = {'max_depth': [None],
             'max_features': [1.0],
             'max_samples': [0.7],
             'min_samples_leaf': [1],
             'min_samples_split': [2],
             'n_estimators': [300]
             }


# 3. Define a dictionary of scoring metrics to capture
#==> ENTER YOUR CODE HERE
scoring = {'accuracy', 'precision', 'recall', 'f1'}

# 4. Instantiate the GridSearchCV object
#==> ENTER YOUR CODE HERE
cv_rf = GridSearchCV(rf, cv_params, scoring = scoring, cv=4, refit='f1')
```

Now fit the model to the training data. Note that, depending on how many options you include in your search grid and the number of cross-validation folds you select, this could take a very long time—even hours. If you use 4-fold validation and include only one possible value for each hyperparameter and grow 300 trees to full depth, it should take about 5 minutes. If you add another value for GridSearch to check for, say, `min_samples_split` (so all hyperparameters now have 1 value except for `min_samples_split`, which has 2 possibilities), it would double the time to ~10 minutes. Each additional parameter would approximately double the time.

[212]:
```python
%%time
#==> ENTER YOUR CODE HERE
cv_rf.fit(X_train, y_train)
```

```
CPU times: user 3min 58s, sys: 140 ms, total: 3min 58s
Wall time: 3min 58s
```

[212]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False, random_state=42,
                                              verbose=0, warm_start=False),
```

```
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [None], 'max_features': [1.0],
                         'max_samples': [0.7], 'min_samples_leaf': [1],
                         'min_samples_split': [2], 'n_estimators': [300]},
             pre_dispatch='2*n_jobs', refit='f1', return_train_score=False,
             scoring={'recall', 'accuracy', 'f1', 'precision'}, verbose=0)
```

HINT

If you get a warning that a metric is 0 due to no predicted samples, think about how many features you're sampling with `max_features`. How many features are in the dataset? How many are likely predictive enough to give good predictions within the number of splits you've allowed (determined by the `max_depth` hyperparameter)? Consider increasing `max_features`.

If you want, use `pickle` to save your models and read them back in. This can be particularly helpful when performing a search over many possible hyperparameter values.

[213]:
```python
import pickle

# Define a path to the folder where you want to save the model
path = '/home/jovyan/work/'
```

[214]:
```python
def write_pickle(path, model_object, save_name:str):
    '''
    save_name is a string.
    '''
    with open(path + save_name + '.pickle', 'wb') as to_write:
        pickle.dump(model_object, to_write)
```

[215]:
```python
def read_pickle(path, saved_model_name:str):
    '''
    saved_model_name is a string.
    '''
    with open(path + saved_model_name + '.pickle', 'rb') as to_read:
        model = pickle.load(to_read)

        return model
```

Examine the best average score across all the validation folds.

[216]:
```python
# Examine best score
#==> ENTER YOUR CODE HERE
cv_rf.best_score_
```

[216]: 0.34662068603172513

Examine the best combination of hyperparameters.

[217]:
```python
cv_rf.best_params_
```

[217]:
```
{'max_depth': None,
 'max_features': 1.0,
 'max_samples': 0.7,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 300}
```

Use the `make_results()` function to output all of the scores of your model. Note that it accepts three arguments.

HINT

To learn more about how this function accesses the cross-validation results, refer to the `GridSearchCV` scikit-learn documentation for the `cv_results_` attribute.

[218]:
```python
def make_results(model_name:str, model_object, metric:str):
    '''
    Arguments:
    model_name (string): what you want the model to be called in the output␣
 ↪table
    model_object: a fit GridSearchCV object
    metric (string): precision, recall, f1, or accuracy

    Returns a pandas df with the F1, recall, precision, and accuracy scores
    for the model with the best mean 'metric' score across all validation folds.
    '''

    # Create dictionary that maps input metric to actual metric name in␣
 ↪GridSearchCV
    metric_dict = {'precision': 'mean_test_precision',
                   'recall': 'mean_test_recall',
                   'f1': 'mean_test_f1',
                   'accuracy': 'mean_test_accuracy',
                   }

    # Get all the results from the CV and put them in a df
    cv_results = pd.DataFrame(model_object.cv_results_)

    # Isolate the row of the df with the max(metric) score
    best_estimator_results = cv_results.iloc[cv_results[metric_dict[metric]].
 ↪idxmax(), :]

    # Extract Accuracy, precision, recall, and f1 score from that row
    f1 = best_estimator_results.mean_test_f1
    recall = best_estimator_results.mean_test_recall
    precision = best_estimator_results.mean_test_precision
    accuracy = best_estimator_results.mean_test_accuracy
```

```
    # Create table of results
    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy],
                          },
                         )

    return table
```

Call `make_results()` on the GridSearch object.

[219]:
```
#==> ENTER YOUR CODE HERE
rf_train_scores  = make_results("RandomForestClassifier tuned", cv_rf, 'f1')
rf_train_scores
```

[219]:

| | model | precision | recall | F1 | accuracy |
|---|---|---|---|---|---|
| 0 | RandomForestClassifier tuned | 0.46013 | 0.278066 | 0.346621 | 0.636014 |

A model with such low F1, precision, and recall scores is not good enough. Optional: try to improve the scores. Generally, unless your hyperparameter search space is completely off the mark, you won't get the degree of improvement you need to approve this model. However, it's worth trying, especially to practice searching over different hyperparameters.

HINT

For example, if the available values for `min_samples_split` were [2, 3, 4] and GridSearch identified the best value as 4, consider trying [4, 5, 6] this time.

Use your model to predict on the test data. Assign the results to a variable called `preds`.

HINT

You cannot call `predict()` on the GridSearchCV object directly. You must call it on the `best_estimator_`.

For this project, you will use several models to predict on the test data. Remember that this decision comes with a trade-off. What is the benefit of this? What is the drawback?

The benefit is that one model may be best suitable to our data than another. So, trying any that model on test data may be fruitful. The Drawback is that the test data is unseen, but by testing the model on unseen data, you would no have idea that how the model will be built on new data.

[232]:
```
# Get scores on test data
#==> ENTER YOUR CODE HERE
y_pred_rf = cv_rf.predict(X_test)
```

Use the below `get_test_scores()` function you will use to output the scores of the model on the test data.

[233]:
```
def get_test_scores(model_name:str, preds, y_test_data):
    '''
    Generate a table of test scores.

    In:
    model_name (string): Your choice: how the model will be named in the output␣
↪table
    preds: numpy array of test predictions
    y_test_data: numpy array of y_test data

    Out:
    table: a pandas df of precision, recall, f1, and accuracy scores for your␣
↪model
    '''
    accuracy = accuracy_score(y_test_data, preds)
    precision = precision_score(y_test_data, preds)
    recall = recall_score(y_test_data, preds)
    f1 = f1_score(y_test_data, preds)

    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy]
                          })

    return table
```

1. Use the `get_test_scores()` function to generate the scores on the test data. Assign the results to `rf_test_scores`.
2. Call `rf_test_scores` to output the results.

RF test results

[234]:
```
 # Get scores on test data
#==> ENTER YOUR CODE HERE
rf_test_scores = get_test_scores('RandomForest Test', y_pred_rf, y_test)
rf_test_scores
```

[234]:

| | model | precision | recall | F1 | accuracy |
|---|---|---|---|---|---|
| 0 | RandomForest Test | 0.478605 | 0.279371 | 0.352804 | 0.637078 |

**Question:** How do your test results compare to your validation results?

Scores decreased by ~ 0.02 to ~ 0.03

**XGBoost** Try to improve your scores using an XGBoost model.

1. Instantiate the XGBoost classifier `xgb` and set `objective='binary:logistic'`. Also set the random state.
2. Create a dictionary `cv_params` of the following hyperparameters and their corresponding values to tune:

- `max_depth`
- `min_child_weight`
- `learning_rate`
- `n_estimators`

3. Define a dictionary `scoring` of scoring metrics for grid search to capture (precision, recall, F1 score, and accuracy).
4. Instantiate the `GridSearchCV` object `xgb1`. Pass to it as arguments:

- estimator=`xgb`
- param_grid=`cv_params`
- scoring=`scoring`
- cv: define the number of cross-validation folds you want (`cv=_`)
- refit: indicate which evaluation metric you want to use to select the model (`refit='f1'`)

[223]:
```
# 1. Instantiate the XGBoost classifier
#==> ENTER YOUR CODE HERE
xgb = XGBClassifier(objective='binary:logistic')

# 2. Create a dictionary of hyperparameters to tune
#==> ENTER YOUR CODE HERE
cv_params = {
    'max_depth' : [8],
    'min_child_weight' : [2],
    'learning_rate' : [0.1],
    'n_estimators' : [500]
}

# 3. Define a dictionary of scoring metrics to capture
#==> ENTER YOUR CODE HERE
scoring = {'accuracy', 'precision', 'recall', 'f1'}

# 4. Instantiate the GridSearchCV object
#==> ENTER YOUR CODE HERE
cv_xgb = GridSearchCV(xgb, cv_params, scoring=scoring, cv=5, refit='f1')
```

Now fit the model to the `X_train` and `y_train` data.

[224]:
```
%%time
#==> ENTER YOUR CODE HERE
cv_xgb.fit(X_train, y_train)
```

```
CPU times: user 8min 49s, sys: 786 ms, total: 8min 49s
Wall time: 4min 25s
```

[224]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     gamma=None, gpu_id=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None, max…
                                     n_estimators=100, n_jobs=None,
                                     num_parallel_tree=None,
                                     objective='binary:logistic',
                                     predictor=None, random_state=None,
                                     reg_alpha=None, …),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.1], 'max_depth': [8],
                         'min_child_weight': [2], 'n_estimators': [500]},
             pre_dispatch='2*n_jobs', refit='f1', return_train_score=False,
             scoring={'recall', 'accuracy', 'f1', 'precision'}, verbose=0)
```

Get the best score from this model.

[225]:
```
# Examine best score
#==> ENTER YOUR CODE HERE
cv_xgb.best_score_
```

[225]:
```
0.35180331427668743
```

And the best parameters.

[226]:
```
# Examine best parameters
#==> ENTER YOUR CODE HERE
cv_xgb.best_params_
```

[226]:
```
{'learning_rate': 0.1,
 'max_depth': 8,
 'min_child_weight': 2,
 'n_estimators': 500}
```

XGB CV Results

Use the `make_results()` function to output all of the scores of your model. Note that it accepts three arguments.

[227]:
```
# Call 'make_results()' on the GridSearch object
#==> ENTER YOUR CODE HERE
xgb_train_scores = make_results('XGBClassifier Tuned', cv_xgb, 'f1')
```

```
xgb_train_scores
```

[227]:

| | model | precision | recall | F1 | accuracy |
|---|---|---|---|---|---|
| 0 | XGBClassifier Tuned | 0.4465 | 0.29033 | 0.351803 | 0.628562 |

Use your model to predict on the test data. Assign the results to a variable called `preds`.

HINT

You cannot call `predict()` on the GridSearchCV object directly. You must call it on the `best_estimator_`.

[228]:
```
# Get scores on test data
#==> ENTER YOUR CODE HERE
pred_y = cv_xgb.best_estimator_.predict(X_test)
```

XGB test results

1. Use the `get_test_scores()` function to generate the scores on the test data. Assign the results to `xgb_test_scores`.
2. Call `xgb_test_scores` to output the results.

[229]:
```
# Get scores on test data
#==> ENTER YOUR CODE HERE
xgb_test_scores = get_test_scores('XGBClassifier Test', pred_y, y_test)
test_scores = pd.concat([rf_test_scores, xgb_test_scores])
test_scores
```

[229]:

| | model | precision | recall | F1 | accuracy |
|---|---|---|---|---|---|
| 0 | RandomForest Test | 0.478605 | 0.279371 | 0.352804 | 0.637078 |
| 0 | XGBClassifier Test | 0.454135 | 0.279371 | 0.345934 | 0.625942 |

**Question:** Compare these scores to the random forest test scores. What do you notice? Which model would you choose?

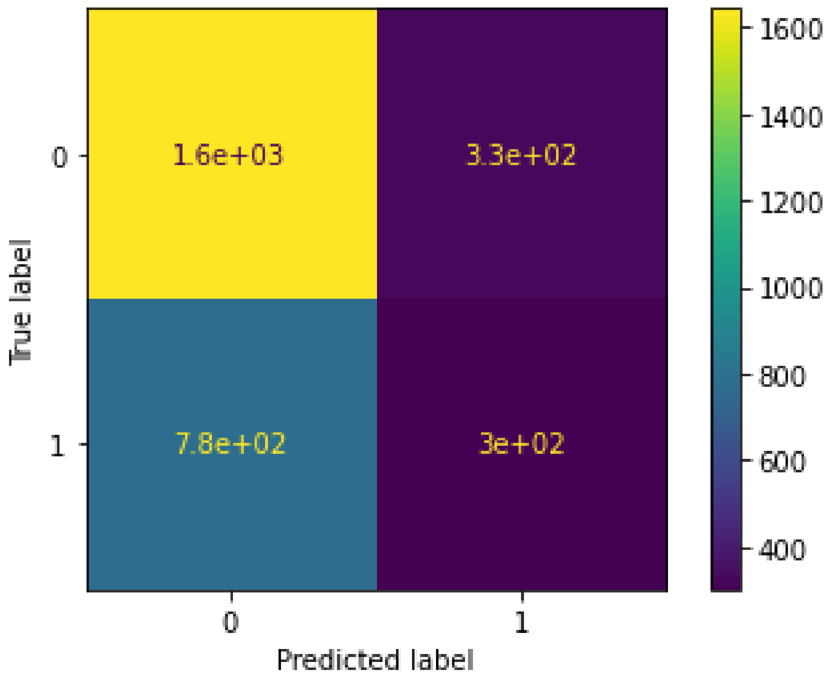
We would select to choose the RandomForest classifier since it's better in all metrics than XGB-Classifier, but both models are unsatisfactory.

Plot a confusion matrix of the model's predictions on the test data.

[237]:
```
# Generate array of values for confusion matrix
#==> ENTER YOUR CODE HERE
cm = confusion_matrix(y_test, y_pred_rf)

# Plot confusion matrix
#==> ENTER YOUR CODE HERE
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels = cv_rf.
 ↪classes_)
disp.plot()
```

[237]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f1ed0126950>



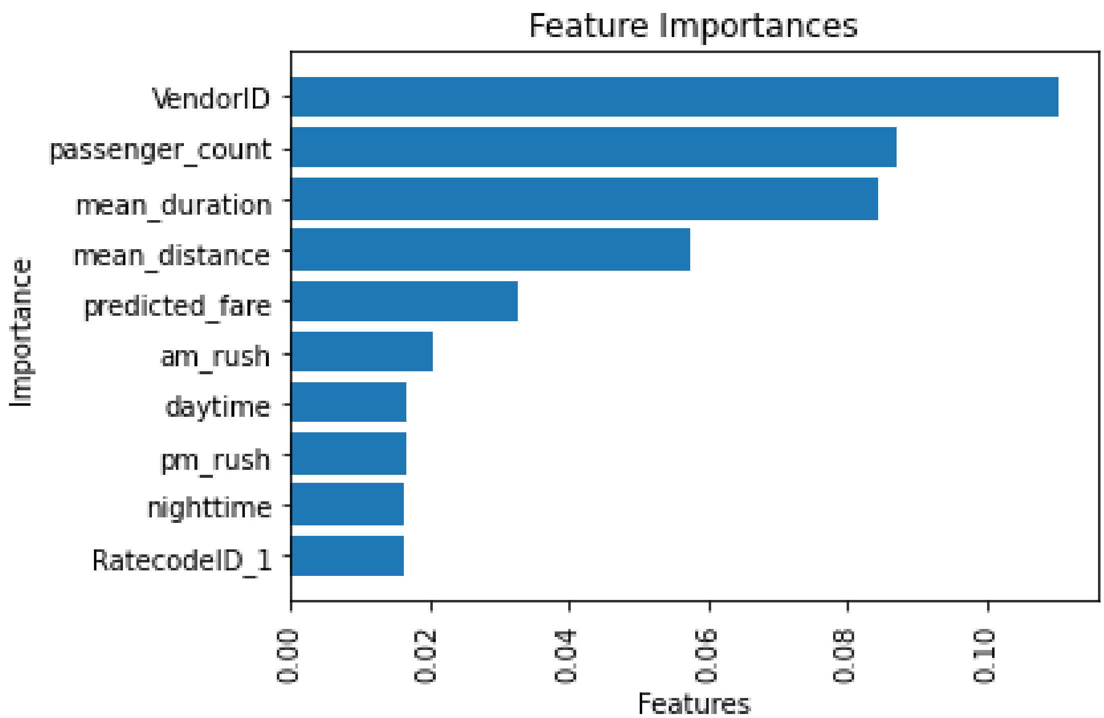

**Question:** What type of errors are more common for your model?

The models seems to predict False negative very more than the false positive, which is type II error. This means than the model will make more errors when the drivers weren't expecting at least 20% tip, but were surprised by the tip. it will make less errors when the drivers were expecting a tip atlest 20% percent but were they weren't tipped. This is good for our case. But the model needs to be further improved before deployment.

**Feature importance** Use the `plot_importance` function to inspect the top 10 most important features of your final model.

[289]:
```
#==> ENTER YOUR CODE HERE
# plot_importance(cv_rf.best_estimator_, max_num_features=10)
imp_features = sorted(cv_rf.best_estimator_.feature_importances_, reverse=True)
importances = dict(sorted(dict(zip(X.columns, imp_features)).items(), key=␣
↪lambda x : x[1], reverse=True)[:10])
features = list(importances.keys())[::-1]
values = list(importances.values())[::-1]
# sns.barplot(x=values, y=features)
plt.barh(features, values)
plt.xticks(rotation=90)
plt.xlabel('Features')
```

```
plt.ylabel('Importance')
plt.title('Feature Importances')
plt.tight_layout()
plt.show()
```




### 4.4 PACE: Execute

Consider the questions in your PACE Strategy Document to reflect on the Execute stage.

#### 4.4.1 Task 4. Conclusion

In this step, use the results of the models above to formulate a conclusion. Consider the following questions:

1. **Would you recommend using this model? Why or why not?**
   This is not a great model, but depending on how it's used it could still be useful. If the objective is only to help give taxi drivers a better idea of whether someone will leave a good tip, then it could be useful. It may be worthwhile to test it with a select group of taxi drivers to get feedback.
2. **What was your model doing? Can you explain how it was making predictions?**
   Unfortunately, RandomFOrest is not the most transparent machine learning algorithm. We

know that `Vendor_ID`, `passenger_count`, and `mean_duration` are the most important features, but we don't know how they influence tipping. This would require further exploration.

3. **Are there new features that you can engineer that might improve model performance?** In our case, we could try creating three new columns that indicate if the trip distance is short, medium, or far.

4. **What features would you want to have that would likely improve the performance of your model?** We could have features that showed the tip behaviour in past for loyal customers. We could include the tip_behavours for all the customers and added a engineering column based on past rides for each customer demontrating the level of loyality in 5 stars.

Remember, sometimes your data simply will not be predictive of your chosen target. This is common. Machine learning is a powerful tool, but it is not magic. If your data does not contain predictive signal, even the most complex algorithm will not be able to deliver consistent and accurate predictions. Do not be afraid to draw this conclusion. Even if you cannot use the model to make strong predictions, was the work done in vain? Consider any insights that you could report back to stakeholders.

**Congratulations!** You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.