Project 3: Predicting the Fare Price of Ubers

Source: [https://www.kaggle.com/datasets/yasserh/uber-fares-dataset (https://www.kaggle.com/datasets/yasserh/uber-fares-dataset)](https://www.kaggle.com/datasets/yasserh/uber-fares-dataset)

Getting and Exploring the Data

In [1]:
```
import pandas as pd
import numpy as np
ubers = pd.read_csv("uber.csv")
ubers.head(5)
```

Out[1]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	c
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	

In [2]:
```
ubers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
Unnamed: 0           200000 non-null int64
key                  200000 non-null object
fare_amount          200000 non-null float64
pickup_datetime      200000 non-null object
pickup_longitude     200000 non-null float64
pickup_latitude      200000 non-null float64
dropoff_longitude    199999 non-null float64
dropoff_latitude     199999 non-null float64
passenger_count      200000 non-null int64
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

It looks like there are 200,000 entries of Ubers in this data set. Also, it appears that dropoff_longitude and dropoff_latitude are the only two factors that have a null entry in one of their rows (they only have 199,999 non-null entries)

In [3]:
```
# Let's get rid of the 'Unnamed: 0' and 'key' columns and drop the NA values
ubers = ubers.drop(columns=['Unnamed: 0', 'key'])
ubers = ubers.dropna()
```

In [4]:
```
max(ubers['pickup_datetime'])
```

Out[4]:
```
'2015-06-30 23:40:39 UTC'
```

In [5]:
```
min(ubers['pickup_datetime'])
```

Out[5]:
```
'2009-01-01 01:15:22 UTC'
```

The data set contains Uber rides from the start of 2009 to the end of June of 2015

In [6]:
```
print(min(ubers['fare_amount']), " to ", max(ubers['fare_amount']))
```

```
-52.0  to  499.0
```

The fares range from -52 to 499. A few things seem a little weird with those values. First, a negative fare value. Second, a fare value that high. Will need to look into this more

In [7]:
```
np.where(ubers['fare_amount'] < 0)
```

Out[7]:
```
(array([ 63395,  71246,  79903,  89321,  92062,  98874, 104079, 111588,
        139271, 148802, 150300, 151680, 157411, 164055, 179110, 180443,
        190924], dtype=int64),)
```

In [8]:
```
ubers.iloc[63395]
```

Out[8]:
```
fare_amount                               -5
pickup_datetime      2015-03-03 23:07:41 UTC
pickup_longitude                    -73.9922
pickup_latitude                      40.7489
dropoff_longitude                   -73.9885
dropoff_latitude                     40.7482
passenger_count                            1
Name: 63395, dtype: object
```

In [9]:
```
ubers = ubers[ubers.fare_amount >= 0]
```

Looking at a few of the negative fare amounts, the data doesn't seem to make much sense as to why it would be negative...let's drop these

In [10]:
```
np.where(ubers['fare_amount'] == 0)
```

Out[10]:
```
(array([ 20744,  22182,  87464, 156725, 197154], dtype=int64),)
```

In [11]:
```
ubers.iloc[22182]
```

Out[11]:
```
fare_amount                            0
pickup_datetime      2010-03-20 02:59:51 UTC
pickup_longitude                 -73.9944
pickup_latitude                   40.7554
dropoff_longitude                -73.9987
dropoff_latitude                  40.8549
passenger_count                        2
Name: 22182, dtype: object
```

The same goes for when the fare is 0...let's drop these

In [12]:
```
ubers = ubers[ubers.fare_amount > 0]
```

In [13]:
```
np.where(ubers['fare_amount'] > 200)
```

Out[13]:
```
(array([  4292,  23680,  29259,  71711, 170062, 185304, 197470],
       dtype=int64),)
```

In [14]:
```
ubers.iloc[71715]
```

Out[14]:
```
fare_amount                          4.5
pickup_datetime      2015-05-28 23:26:17 UTC
pickup_longitude                 -73.9705
pickup_latitude                   40.7967
dropoff_longitude                -73.9757
dropoff_latitude                  40.7956
passenger_count                        1
Name: 71719, dtype: object
```

The same goes for when the fare is pricey (just looked at above 200)...let's drop these and then look at trips priced above 100

In [15]:
```
ubers = ubers[ubers.fare_amount < 200]
```

In [16]:
```
np.where(ubers['fare_amount'] > 100)
```

Out[16]:
```
(array([  2053,   5967,   6612,   9059,  11300,  15361,  15460,  15934,
         16386,  18414,  19508,  22559,  23145,  23229,  28805,  31701,
         33906,  34105,  36328,  39711,  43707,  44368,  45089,  45782,
         51012,  53012,  53991,  58039,  72774,  73583,  79305,  87145,
         90806,  92383,  93030,  95324,  99969, 105791, 110485, 112822,
        114266, 114924, 117349, 120134, 122665, 123980, 127198, 129025,
        131077, 131407, 131856, 134508, 138887, 140405, 141106, 144154,
        145232, 147782, 149003, 150961, 151580, 155042, 158098, 159879,
        163199, 166538, 166862, 173943, 174234, 178426, 183917, 184875,
        188207, 190740, 194426, 196588], dtype=int64),)
```

In [17]:
```
ubers.iloc[5967]
```

Out[17]:
```
fare_amount                              105
pickup_datetime      2011-05-06 00:40:00 UTC
pickup_longitude                    -73.7523
pickup_latitude                      40.9233
dropoff_longitude                   -73.7523
dropoff_latitude                     40.9233
passenger_count                            1
Name: 5968, dtype: object
```

In [18]:
```
ubers.iloc[9059]
```

Out[18]:
```
fare_amount                            126.1
pickup_datetime      2011-06-13 15:46:00 UTC
pickup_longitude                    -73.7887
pickup_latitude                      40.6406
dropoff_longitude                   -74.0014
dropoff_latitude                      41.048
passenger_count                            1
Name: 9060, dtype: object
```

Some of these are logical, but others aren't. For now going to keep them in, but will return to this later...

In [19]:
```
timeZones = ubers['pickup_datetime'].str[-3:]
timeZones.unique()
```

Out[19]:
```
array(['UTC'], dtype=object)
```

Nice, just working with data in UTC time zone

In [20]:
```
ubers['passenger_count'].unique()
```

Out[20]:
```
array([  1,   3,   5,   2,   4,   6,   0, 208], dtype=int64)
```

In [21]:
```
ubers['passenger_count'].value_counts()
```

Out[21]:
```
1      138408
2       29423
5       14005
3        8877
4        4276
6        4271
0         708
208         1
Name: passenger_count, dtype: int64
```

Kinda surprising that there are so many rides of 0 passengers. Does this mean that the ride was cancelled? Might need to investigate this scenario further...

Also, going to drop the one ride of 208 passengers

In [22]:
```
ubers = ubers[ubers.passenger_count != 208]
```

Let's take a look at all the numerical features after cleaning it up a little bit

In [23]:
```
ubers.describe()
```

Out[23]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passeng
count	199968.000000	199968.000000	199968.000000	199968.000000	199968.000000	19996
mean	11.351885	-72.528891	39.936571	-72.527284	39.924984	
std	9.733257	11.434090	7.719085	13.112397	6.791829	
min	0.010000	-1340.648410	-74.015515	-3356.666300	-881.985513	
25%	6.000000	-73.992065	40.734797	-73.991407	40.733828	
50%	8.500000	-73.981823	40.752592	-73.980093	40.753042	
75%	12.500000	-73.967157	40.767158	-73.963662	40.768002	
max	196.000000	57.418457	1644.421482	1153.572603	872.697628	

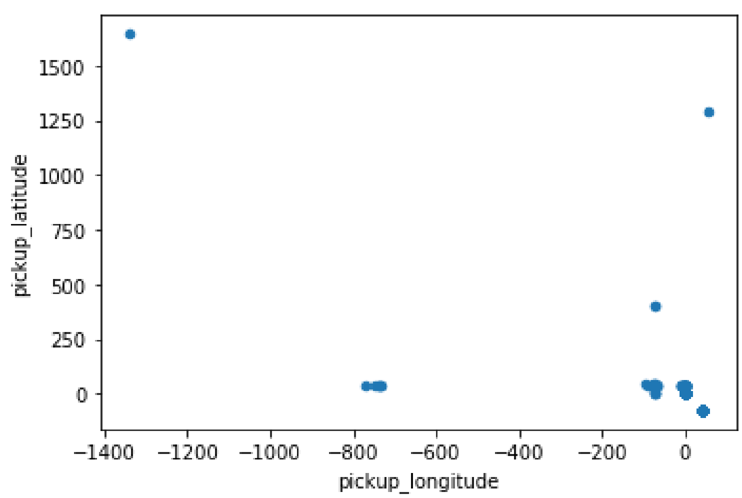
It also looks like the longitudes and latitudes could have some iffy data (not entirely sure if those values are realistic)

In [24]:
```
import matplotlib.pyplot as plt
plt = ubers.hist(bins=50, figsize=(20,15))
```

Besides information listed above, only new takeaways are the amount of fares negative and the grouping of longitude and latitudes so close to each other. Let's take a deeper dive into these longitudes and latitudes...

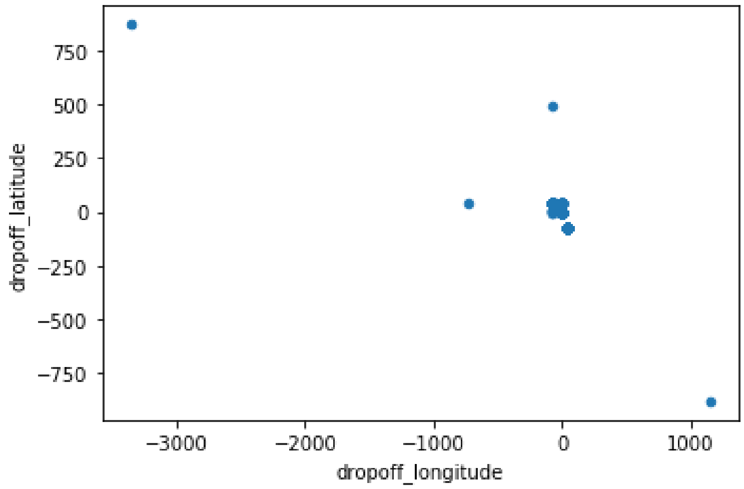
In [25]:
```
ubers.plot(kind="scatter", x="pickup_longitude", y="pickup_latitude")
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1908a481a88>



In [26]:
```
ubers.plot(kind="scatter", x="dropoff_longitude", y="dropoff_latitude")
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x1908d1f4f88>



As expected, the longitude and latitudes are grouped tightly together (for the most part)

In [27]:
```
corr_matrix = ubers.corr()
```

In [28]:
```
# Let's see the correlation to the fare prices
corr_matrix["fare_amount"].sort_values(ascending=False)
```

Out[28]:
```
fare_amount          1.000000
passenger_count      0.011434
pickup_longitude     0.007987
dropoff_longitude    0.007055
pickup_latitude     -0.006466
dropoff_latitude    -0.008983
Name: fare_amount, dtype: float64
```

Doesn't appear to be much correlation at all. Let's add in the difference in latitude and longitude values and see if that shows better correlation with showing the distance between two points

Haversine formula for distance between two points: [https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/ (https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/)](https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/)

In [29]:
```
import math

# Python 3 program for the
# haversine formula
def haversine(lat1, lon1, lat2, lon2):

    # distance between latitudes
    # and longitudes
    dLat = (lat2 - lat1) * math.pi / 180.0
    dLon = (lon2 - lon1) * math.pi / 180.0

    # convert to radians
    lat1 = (lat1) * math.pi / 180.0
    lat2 = (lat2) * math.pi / 180.0

    # apply formulae
    a = (pow(math.sin(dLat / 2), 2) +
         pow(math.sin(dLon / 2), 2) *
             math.cos(lat1) * math.cos(lat2));
    rad = 6371
    c = 2 * math.asin(math.sqrt(a))
    return rad * c
```

In [30]:
```
distance = []
for i in range(len(ubers)):
    lat1 = ubers["pickup_latitude"].iloc[i]
    lon1 = ubers["pickup_longitude"].iloc[i]
    lat2 = ubers["dropoff_latitude"].iloc[i]
    lon2 = ubers["dropoff_longitude"].iloc[i]
    d = haversine(lat1, lon1, lat2, lon2)
    distance.append(d)
```

In [31]:
```
ubers['distance'] = distance
```

In [32]:
```
ubers.head(2)
```

Out[32]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitud
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.72321
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.75032

In [33]:
```
corr_matrix2 = ubers.corr()
```

In [34]:
```
# Let's see the correlation to the fare prices
corr_matrix2["fare_amount"].sort_values(ascending=False)
```

Out[34]:
```
fare_amount          1.000000
distance             0.026068
passenger_count      0.011434
pickup_longitude     0.007987
dropoff_longitude    0.007055
pickup_latitude     -0.006466
dropoff_latitude    -0.008983
Name: fare_amount, dtype: float64
```

The distance has the highest correlation to the fare amount, but still not a strong correlation

Let's see if we can find something to show a stronger correlation to the fare amount. Let's extract the data in the column "pickup_datetime" and create a column for the year, month, and hour...

In [36]:
```
# Get the year, month, and hour from the pickup_datetime column
# The hour is calculated by:
# 1) Get the hour
# 2) Add 1 to the hour if the minutes are >= 30
# 3) If adding 1, check to see if the hour is 24, which would mean it needs to be
pickup_year = []
pickup_month = []
pickup_hour = []

for i in range(len(ubers)):
    temp = ubers["pickup_datetime"].iloc[i]
    vals = temp.split(' ')
    date = vals[0]
    year = int(date[:4])
    month = int(date[5:7])
    time = vals[1]
    hour = int(time[:2])
    if minutes >= 30:
        hour += 1
        if hour == 24:
            hour = 0
    pickup_year.append(year)
    pickup_month.append(month)
    pickup_hour.append(hour)

ubers['pickup_year'] = pickup_year
ubers['pickup_month'] = pickup_month
ubers['pickup_hour'] = pickup_hour
```

In [37]:
```
ubers.head(1)
```

Out[37]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitud
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.72321

In [38]:
```
corr_matrix3 = ubers.corr()
```

In [39]:
```
# Let's see the correlation to the fare prices
corr_matrix3["fare_amount"].sort_values(ascending=False)
```

Out[39]:
```
fare_amount          1.000000
pickup_year          0.120005
distance             0.026068
pickup_month         0.023715
passenger_count      0.011434
pickup_longitude     0.007987
dropoff_longitude    0.007055
pickup_latitude     -0.006466
dropoff_latitude    -0.008983
pickup_hour         -0.024170
Name: fare_amount, dtype: float64
```

The year shows higher correlation, but still not the level of correlation that I was expecting to get

In [40]:
```
np.where(ubers['distance'] == 0)
```

Out[40]:
```
(array([     5,      7,     11, ..., 199885, 199900, 199931], dtype=int64),)
```

Back to the scenario of iffy Uber prices being so high, let's remove unrealistic data/outliers

1) remove the ones that have fares above 25, but a distance of 0

2) remove the ones that have fares above 50, but a distance < 5

3) remove the ones that have fares above 100, but a distance < 10

In [41]:
```
indexesToDrop = ubers[(ubers.fare_amount >= 25) & (ubers.distance == 0)].index
ubers.drop(indexesToDrop,inplace=True)

indexesToDrop = ubers[(ubers.fare_amount >= 50) & (ubers.distance <= 5)].index
ubers.drop(indexesToDrop,inplace=True)

indexesToDrop = ubers[(ubers.fare_amount >= 100) & (ubers.distance <= 10)].index
ubers.drop(indexesToDrop,inplace=True)
```

In [42]:
```
corr_matrix4 = ubers.corr()
# Let's see the correlation to the fare prices
corr_matrix4["fare_amount"].sort_values(ascending=False)
```

Out[42]:
```
fare_amount          1.000000
pickup_year          0.122956
distance             0.028033
pickup_month         0.024082
dropoff_latitude     0.016059
pickup_latitude      0.015607
passenger_count      0.014425
dropoff_longitude   -0.016718
pickup_longitude    -0.019414
pickup_hour         -0.022980
Name: fare_amount, dtype: float64
```

In [43]:
```
ubers.head(2)
```

Out[43]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.72
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.75

We don't need pickup_datetime anymore, let's drop that

In [44]:
```
ubers = ubers.drop(columns=['pickup_datetime'])
```

In [45]:
```
ubers.head(1)
```

Out[45]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_cou
0	7.5	-73.999817	40.738354	-73.999512	40.723217	

Stratified KFold Cross Validation

In [63]:
```
# Found this from https://stackoverflow.com/questions/40372030/pandas-round-to-the
def custom_round(x, base=10):
    return int(base * round(float(x)/base))
```

In [67]:

```
# Adapted from https://www.geeksforgeeks.org/stratified-k-fold-cross-validation/
from statistics import mean, stdev
from sklearn.model_selection import StratifiedKFold
from sklearn import linear_model

x = ubers.drop(columns=["fare_amount"]).values
y = pd.Series(ubers['fare_amount']).apply(lambda x: custom_round(x, base=10)).valu

# Create  classifier object.
lr = linear_model.LogisticRegression()

# Create StratifiedKFold object.
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
lst_accu_stratified = []

for train_index, test_index in skf.split(x, y):
    x_train_fold, x_test_fold = x[train_index], x[test_index]
    y_train_fold, y_test_fold = y[train_index], y[test_index]
    lr.fit(x_train_fold, y_train_fold)
    lst_accu_stratified.append(lr.score(x_test_fold, y_test_fold))
```

```
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668:
UserWarning: The least populated class in y has only 1 members, which is less th
an n_splits=10.
  % (min_groups, self.n_splits)), UserWarning)
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-l
earn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-l
earn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-l
earn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n)
```

```
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-l
earn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-l
earn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-l
earn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-l
earn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-l
earn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n)
```

```
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-l
earn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:765:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-l
earn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [71]:

```
# Print the output.
print('List of possible accuracy:', lst_accu_stratified)
print('\nMaximum Accuracy That can be obtained from this model is:',
      max(lst_accu_stratified)*100, '%')
print('\nMinimum Accuracy:',
      min(lst_accu_stratified)*100, '%')
print('\nOverall Accuracy:',
      mean(lst_accu_stratified)*100, '%')
print('\nStandard Deviation is:', stdev(lst_accu_stratified))
```

```
List of possible accuracy: [0.6643921529276002, 0.6643921529276002, 0.6643419798
30415, 0.664341979830415, 0.664341979830415, 0.664341979830415, 0.66434197983041
5, 0.6642918067332296, 0.664341979830415, 0.664341979830415]

Maximum Accuracy That can be obtained from this model is: 66.43921529276003 %

Minimum Accuracy: 66.42918067332296 %

Overall Accuracy: 66.43469971401335 %

Standard Deviation is: 2.8480568571448496e-05
```

OvO

In [61]:

```
# Found this from https://stackoverflow.com/questions/40372030/pandas-round-to-the
def custom_round(x, base=10):
    return int(base * round(float(x)/base))
```

In [62]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


from sklearn.multiclass import OneVsRestClassifier
from sklearn.multiclass import OneVsOneClassifier


from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from sklearn.model_selection import GridSearchCV

x = ubers.drop('fare_amount', axis=1)
y = pd.Series(ubers['fare_amount']).apply(lambda x: custom_round(x, base=10))


trainX, testX, trainY, testY = train_test_split(x, y, test_size = 0.2)

OvR_clf = OneVsRestClassifier(LogisticRegression())
OvR_clf.fit(trainX, trainY)

y_pred = OvR_clf.predict(testX)

print('Accuracy of OvR Classifier: {:.2f}'.format(accuracy_score(testY, y_pred)))
```

```
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
5: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit
-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\Kyle\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
5: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit
-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
```

OvO for the fare_amount rounded to the nearest $10 has an accuracy of 66%

Decision Tree Regressor

In [75]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

x = ubers.drop('fare_amount', axis=1)
y = ubers['fare_amount']

trainX, testX, trainY, testY = train_test_split(x, y, test_size = 0.2)

tree_reg = DecisionTreeRegressor(max_depth=5)
tree_reg.fit(trainX, trainY)

y_pred = tree_reg.predict(testX)

print('Mean Squared Error of Decision Tree Regressor: {:.2f}'.format(mean_squared_
```

```
Mean Squared Error of Decision Tree Regressor: 14.91
```

Grid Search Hyperparameter Tuning

In [76]:

```
# Source: https://www.malicksarr.com/hyperparameter-tuning-with-grid-search-in-pyt
from sklearn import datasets
from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import RandomForestRegressor

X = ubers.drop('fare_amount', axis=1)
y = ubers['fare_amount']

# Load the model parameters to be test
model_params = {
    'n_estimators': [50,100, 150],
    'max_features':  ['sqrt', 0.3, 0.6, 0.9 , 1.0],
    'min_samples_split': [0.1,0.3,0.6]
}

# create random forest regressor model
rf_model = RandomForestRegressor()

# set up Grid-Search meta-estimator
# this will train 100 models over 5 folds of cross validation (500 models total)
clf = GridSearchCV(rf_model, model_params, cv=5)

# train the random search meta-estimator to find the best model out of 100 candida
model = clf.fit(X, y)

print(model.best_estimator_.get_params())
```

```
{'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'mse', 'max_depth': None, 'ma
x_features': 0.9, 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_dec
rease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_spl
it': 0.1, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 150, 'n_jobs': None,
'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False}
```