Citibike Case Study By Mahwish Khalid

Background, Context, and Objective

Client: Mayor of New York City, Bill de Blasio Objective: Help the mayor get a better understanding of citibike ridership by creating an operating report for 2017. Ask: 1. Top 5 stations with the most starts (showing # of starts) 2. Trip duration by user type 3. Most popular trips based on start station and stop station) 4. Rider performance by Gender and Age based on avg trip distance (station to station), median speed (distance traveled / trip duration) 5. What is the busiest bike in NYC in 2017? How many times was it used? How many minutes was it in use? Note: A model that can predict how long a trip will take given a starting point and destination. Solution:- The dataset is massive load the dataset and import the libraries before loading te dataset.

QUESTION 1

Question 1: Top 5 Stations Let's check if there's any noise or cleanup which needs to be done before creating the chart. Any missing values? Mostly for Birth year and a few for User Type. We can ignore these for now and deal with them later. Let's get the data in the right format Trip Duration - Int Start Time - DateTime Stop Time - DateTime Start Station ID - Categorical Start Station Name - Categorical User Type - Categorical Birth Year - Ordinal Gender - Categorical Deal with trips which lasted less than 1.5 minute (90 seconds). If so, in the ideal world, we should not include this start, we may double count. If a bike is broken, a user will dock it again within a minute or two and pick-up another one.

Import Libraries

In [57]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas import *
import seaborn as sns
from matplotlib import rcParams
import datetime as dt
```

Import csv file

In [58]:
```
df=pd.read_csv("C:/Users/Mahwish/Desktop/mc master university/case study IBM/citibike.csv")
```

Basic Summary,dimension and Structure of Dataset

1)First Few Rows of Dataset

In [59]:
```
print(df.head())
```

```
   Trip Duration           Start Time            Stop Time  Start Station ID  \
0            680  2017-01-01 00:00:21  2017-01-01 00:11:41              3226   
1           1282  2017-01-01 00:00:45  2017-01-01 00:22:08              3263   
2            648  2017-01-01 00:00:57  2017-01-01 00:11:46              3143   
3            631  2017-01-01 00:01:10  2017-01-01 00:11:42              3143   
4            621  2017-01-01 00:01:25  2017-01-01 00:11:47              3143   

            Start Station Name  Start Station Latitude  \
0  W 82 St & Central Park West               40.782750   
1       Cooper Square & E 7 St               40.729236   
2              5 Ave & E 78 St               40.776829   
3              5 Ave & E 78 St               40.776829   
4              5 Ave & E 78 St               40.776829   

   Start Station Longitude  End Station ID             End Station Name  \
0               -73.971370            3165  Central Park West & W 72 St   
1               -73.990868             498         Broadway & W 32 St   
2               -73.963888            3152              3 Ave & E 71 St   
3               -73.963888            3152              3 Ave & E 71 St   
4               -73.963888            3152              3 Ave & E 71 St   

   End Station Latitude  End Station Longitude  Bike ID   User Type  \
0             40.775794             -73.976206    25542  Subscriber   
1             40.748549             -73.988084    21136  Subscriber   
2             40.768737             -73.961199    18147    Customer   
3             40.768737             -73.961199    21211    Customer   
4             40.768737             -73.961199    26819    Customer   

   Birth Year  Gender  
0      1965.0       2  
1      1987.0       2  
2         NaN       0  
3         NaN       0  
4         NaN       0  
```

2) Dimension of dataset

In [60]:
```
df.shape
```

Out[60]:
```
(726676, 15)
```

3)Data type of each column

In [61]:
```
print(df.dtypes)
```

```
Trip Duration                int64
Start Time                  object
Stop Time                   object
Start Station ID             int64
Start Station Name          object
Start Station Latitude     float64
Start Station Longitude    float64
End Station ID               int64
End Station Name            object
End Station Latitude       float64
End Station Longitude      float64
Bike ID                      int64
User Type                   object
Birth Year                 float64
Gender                       int64
dtype: object
```

So Dataset contains categorical Features

In [62]:
```
categorical = df.dtypes[df.dtypes == "object"].index
```

In [63]:
```
print(categorical)
```

```
Index(['Start Time', 'Stop Time', 'Start Station Name', 'End Station Name',
       'User Type'],
      dtype='object')
```

Categorical features must be transformed into numerical features to be useful in most types of analysis.

In [64]:
```
df["Start Time"] = pd.to_datetime(df["Start Time"] )
df["Stop Time"] = pd.to_datetime(df["Stop Time"] )
```

In [65]:
```
df["Start Station Name"] = df["Start Station Name"].astype('category')
df["End Station Name"] = df["End Station Name"].astype('category')
df["User Type"] = df["User Type"].astype('category')
```

4)Statistical Summary of data

In [66]:
```
print(df.describe())
```

```
       Trip Duration  Start Station ID  Start Station Latitude  \
count   7.266760e+05     726676.000000           726676.000000   
mean    7.778989e+02       1222.917630               40.737372   
std     1.124683e+04       1277.955252                0.072596   
min     6.100000e+01         72.000000                0.000000   
25%     3.310000e+02        358.000000               40.720874   
50%     5.260000e+02        482.000000               40.739355   
75%     8.600000e+02       3092.000000               40.755103   
max     5.325688e+06       3446.000000               40.804213   

       Start Station Longitude  End Station ID  End Station Latitude  \
count            726676.000000   726676.000000         726676.000000   
mean                -73.984795     1197.252902             40.737077   
std                   0.123776     1266.085070              0.072474   
min                 -74.031372       72.000000              0.000000   
25%                 -73.995299      356.000000             40.720828   
50%                 -73.987167      479.000000             40.739323   
75%                 -73.976682     3078.000000             40.755003   
max                   0.000000     3447.000000             40.804213   

       End Station Longitude        Bike ID     Birth Year         Gender  
count          726676.000000  726676.000000  697600.000000  726676.000000  
mean              -73.985133   21713.053902    1977.122481       1.166728  
std                 0.123782    4199.313576      11.925020       0.475971  
min               -74.033459   14529.000000    1885.000000       0.000000  
25%               -73.995960   17859.000000    1969.000000       1.000000  
50%               -73.987586   21295.000000    1979.000000       1.000000  
75%               -73.976806   25803.000000    1987.000000       1.000000  
max                 0.000000   27325.000000    2000.000000       2.000000  
```

Handling Missing Values ¶

1)Find Exact Number of Missing Values

In [67]:
```
pd.isnull(obj=df).values.ravel().sum()
```

Out[67]:
```
32269
```

2) Total Number of Missing values in Column

In [68]:
```
df.isnull().sum(axis=0)
```

Out[68]:
```
Trip Duration                0
Start Time                   0
Stop Time                    0
Start Station ID             0
Start Station Name           0
Start Station Latitude       0
Start Station Longitude      0
End Station ID               0
End Station Name             0
End Station Latitude         0
End Station Longitude        0
Bike ID                      0
User Type                 3193
Birth Year               29076
Gender                       0
dtype: int64
```

Mostly for Birth year and a few for User Type. So it safe to remove NA values of User Type and other unncessary data.

3) Deleting Unnecessary Data

In [69]:
```
df = df.dropna(subset=['User Type'])
```

In [70]:
```
df.isnull().sum(axis=0)
```

Out[70]:
```
Trip Duration                  0
Start Time                     0
Stop Time                      0
Start Station ID               0
Start Station Name             0
Start Station Latitude         0
Start Station Longitude        0
End Station ID                 0
End Station Name               0
End Station Latitude           0
End Station Longitude          0
Bike ID                        0
User Type                      0
Birth Year                 29071
Gender                         0
dtype: int64
```

Citi Bike riders often come across damage or broken bikes. Let’s drop any trips where a trip lasted less than 90 seconds ,As we can see from statistical summary minimum duration is 61sec.Also we drop double counts inStart Station Ltitude and End Station Latitude

In [71]:
```
df = df.drop(df.index[(df['Trip Duration'] < 90) &
                          (df['Start Station Latitude'] == df['End Station Latitude'])])
```

In [72]:
```
df.shape
```

Out[72]:
```
(722528, 15)
```

Question 1)Top 5 stations with the most starts (showing # of starts)

Data for Top 5 Stations visual

In [73]:
```
Top5_Stations=df['Start Station Name'].value_counts().head().index
Number_of_Start=df['Start Station Name'].value_counts().head().values
Top5_Stations = Top5_Stations.astype('object')
```

In [74]:
```
Top5_Stations
```

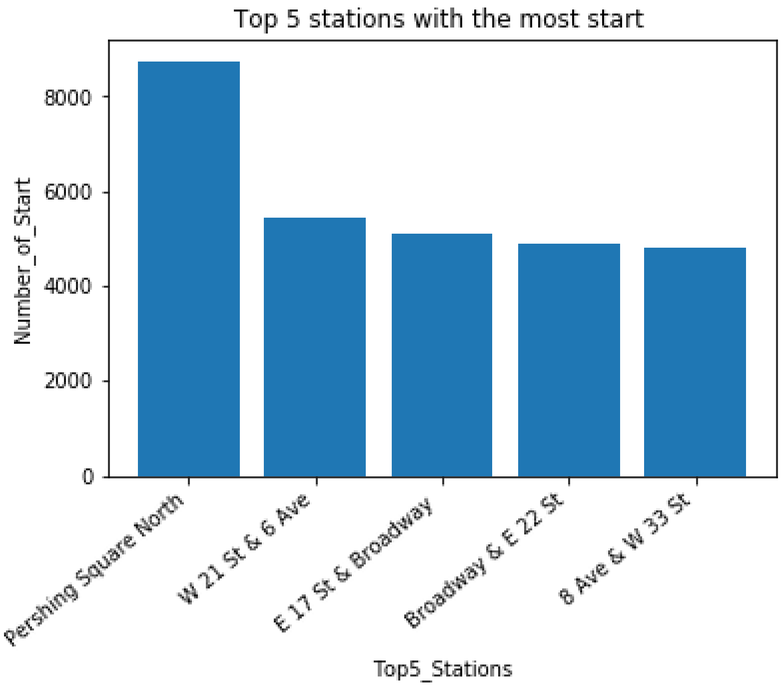
Out[74]:
```
Index(['Pershing Square North', 'W 21 St & 6 Ave', 'E 17 St & Broadway',
       'Broadway & E 22 St', '8 Ave & W 33 St'],
      dtype='object')
```

In [75]:
```
Number_of_Start
```

Out[75]:
```
array([8760, 5435, 5099, 4900, 4789], dtype=int64)
```

In [ ]:

In [76]:
```
import seaborn as sns
import matplotlib.pyplot as plt
% matplotlib inline

# Choose the names of the bars
Top5_Stations = ('Pershing Square North', 'W 21 St & 6 Ave', 'E 17 St & Broadway ','Broadway & E 22 St' ,'8 A
ve & W 33 St')
y_pos = np.arange(len(Top5_Stations))

# Create bars
ax=plt.bar(y_pos,Number_of_Start)

# Create names on the x-axis
plt.xticks(y_pos,Top5_Stations , color='black')
plt.yticks(color='black')


# Create names on the x-axis any Y axis
plt.xticks(y_pos, Top5_Stations)
Title = "Top 5 stations with the most start "
Xl = "Top5_Stations"
Yl ="Number_of_Start"
plt.title(Title)
plt.xlabel(Xl)
plt.ylabel(Yl)

# Rotation of the bars names
plt.xticks(y_pos,Top5_Stations , rotation=40, ha = 'right')
# Show graphic
plt.show()
```



According to their website

https://www.citibikenyc.com/pricing (https://www.citibikenyc.com/pricing)

Question 2)Trip duration by user type

According to their website https://www.citibikenyc.com/pricing For Annual Members pass the first 45min ride is free but if the user wants to keep,they need to pay $2.50/15minForvisitordaypass$12 for 24hours,the 30min of each ride included in this pass but if the user wants to keep,they need to pay $2.50/15minFor3dayspassis24/72hrs, the30minsofeachrideincludedinthispassbutiftheuserwanttokeep, theyneedtopay$4/15min. The reason for discussing all rates is here as very few people want to ride a bike more than 1hour or safer side 2hours because it not economical. Also if the bike is away for more than 2hours so it may be a chance of stolen. So it means there are lots of inconsistency in Trip duration so we are dividing our data in two portions one with abnormalities and other with normalities for visualizing the part of the case study. 1)First Half- with anomalies in dataset The graph under ax2 is a bargraph of average trip duration for each user type. It's helpful, but would be better to see a boxplot and get an idea of the distribution and see mintues instead of seconds. 2)Second graph is a basic Boxplot based with anomalies included. As we can see, there is too much noise for this to be useful. It'll be better to look at this without anomalies. Second Half - without anomalies in dataset Still not useful, let's add a column with minutes for trip Duration. Boxplot with minutes is much more useful. There are still some outliers, however, it is informative.

calculate Trip Duration

In [77]:
```
#This question is a bit unclear in terms of what to do with the anomalies/inconsistency, so I'll be
#making two graphs. One with anomalies, one without.
TD_user = pd.DataFrame()
TD_user['Avg. Trip Duration'] = round(df.groupby('User Type')['Trip Duration'].mean(),2)
TD_user = TD_user.reset_index()
TD_user['User Type'] = TD_user['User Type'].astype('object')
```

In [78]:
```
TD_user['Avg. Trip Duration']
```

Out[78]:
```
0    2525.06
1     718.81
Name: Avg. Trip Duration, dtype: float64
```

In [79]:
```
TD_user
```

Out[79]:

	User Type	Avg. Trip Duration
0	Customer	2525.06
1	Subscriber	718.81

In [80]:
```
TD_user['User Type']
```

Out[80]:
```
0      Customer
1    Subscriber
Name: User Type, dtype: object
```

In [81]:
```
df.groupby('User Type')['Trip Duration']
```

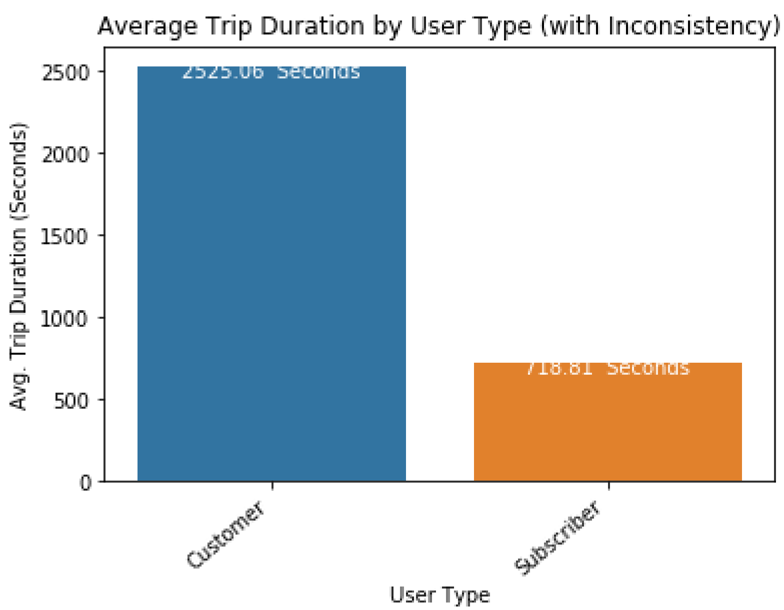
Out[81]:
```
<pandas.core.groupby.SeriesGroupBy object at 0x000001EB5BC89898>
```

In [82]:
```
df.groupby('User Type')['Trip Duration'].mean()
```

Out[82]:
```
User Type
Customer      2525.061078
Subscriber     718.807176
Name: Trip Duration, dtype: float64
```

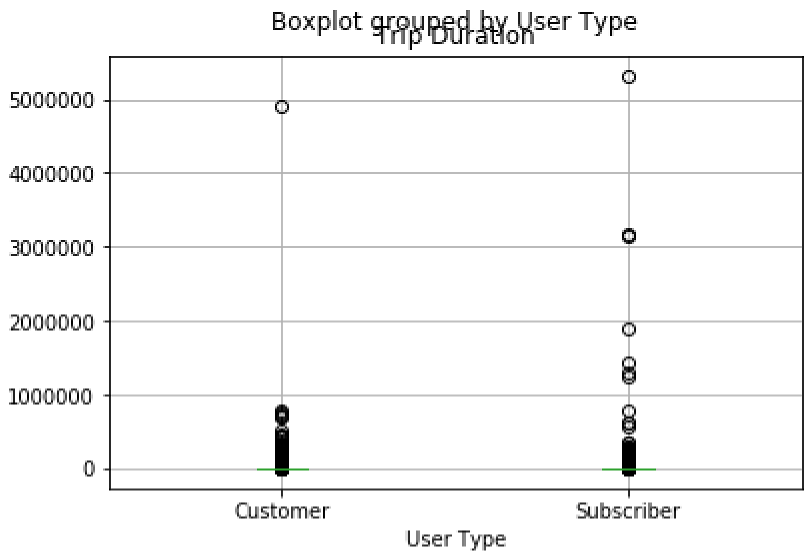
Average trip Duration (secands)per User Type with Anomalies/Inconsistency

In [83]:
```
#Average trip Duration per User Type with Anomalies/Inconsistency
ax2 = sns.barplot('User Type', 'Avg. Trip Duration', data = TD_user)
ax2.set_title('Average Trip Duration by User Type (with Inconsistency)')

ax2.set_xticklabels(ax2.get_xticklabels(),rotation=40, ha = 'right')
ax2.set_ylabel('Avg. Trip Duration (Seconds)')
for index, row in TD_user.iterrows():
    ax2.text(index,row['Avg. Trip Duration']-70,(str(row['Avg. Trip Duration'])+"  Seconds"),
             color='white', ha="center", fontsize = 10)
plt.show()
```



Boxplots are more informative to visualize breakdown of data

In [84]:
```
#Boxplots are more informative to visualize breakdown of data
del(TD_user)
df.boxplot('Trip Duration', by = 'User Type')
plt.show()
```

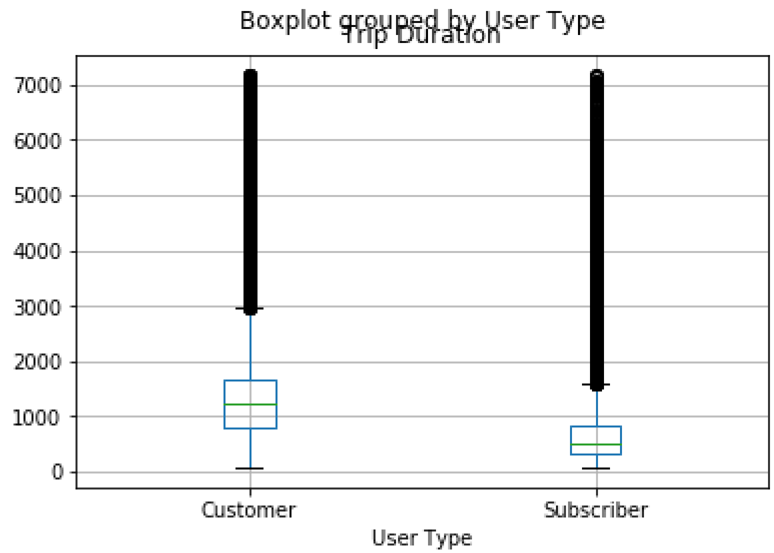


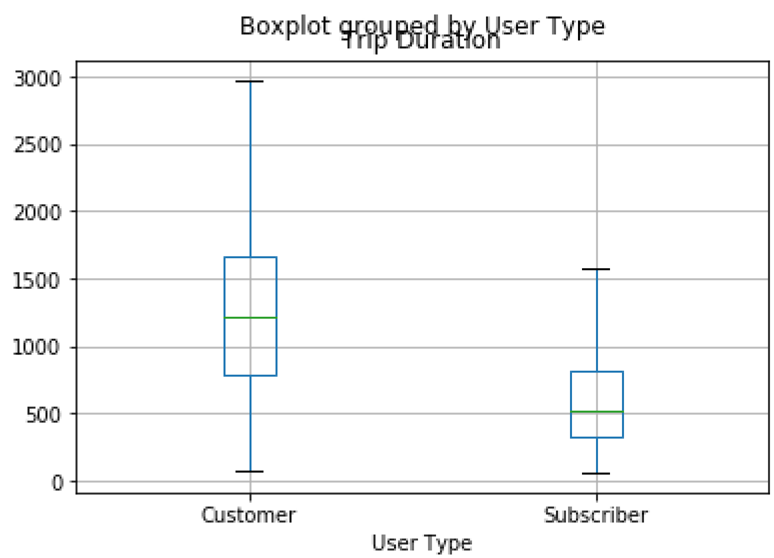
Any trip which lasts longer than 2 hours (7,200 seconds).Remove Inconsistency /Anomalies

In [85]:
```
df = df.drop(df.index[(df['Trip Duration'] > 7200)])
```

Boxplots are more informative to visualize breakdown of data

In [86]:
```
df.boxplot('Trip Duration', by = 'User Type')
plt.show()
#Boxplot without outliers
df.boxplot('Trip Duration', by = 'User Type',showfliers=False)
plt.show()
```



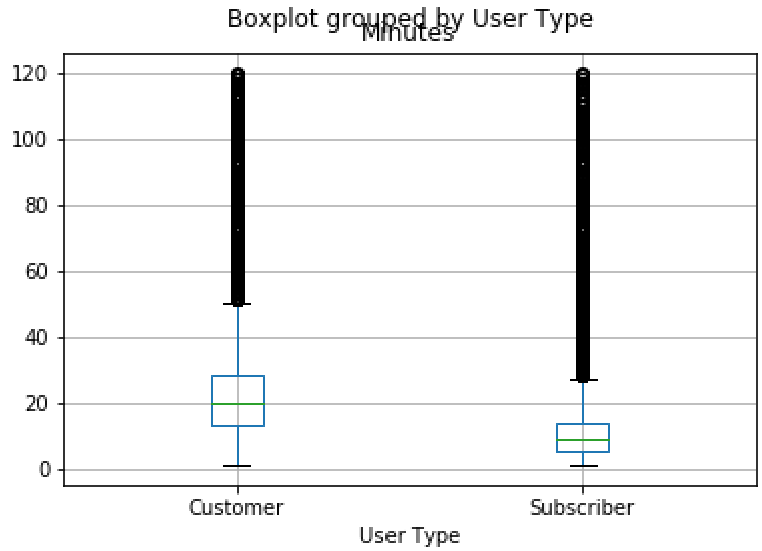


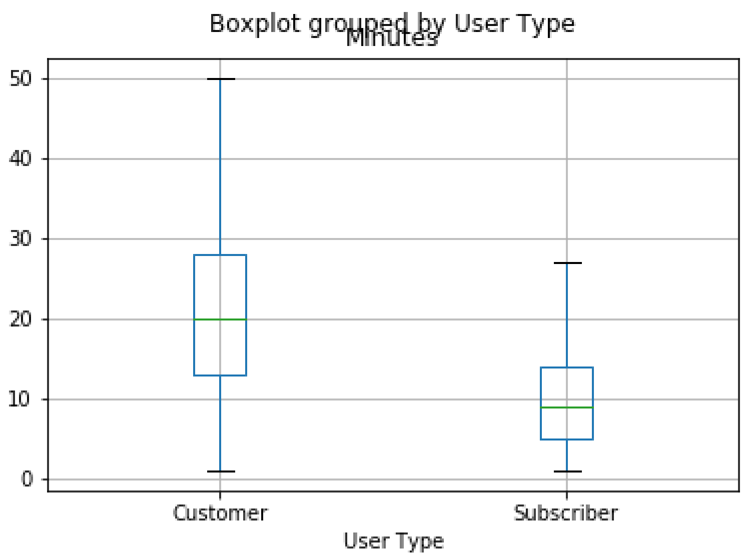
Add Minutes column for Trip Duration

In [87]:
```
df['Minutes'] = df['Trip Duration']/60
#For Visual purposes, rounded
df['Minutes'] = round(df['Minutes'])
df['Minutes'] = df['Minutes'].astype(int)
```

Final Boxplot with some outliers. Could turn of outliers with showfliers = False

In [88]:
```
df.boxplot('Minutes', by = 'User Type')
plt.show()
df.boxplot('Minutes', by = 'User Type', showfliers = False)
plt.show()
```





In [89]:
```
TD_user2 = pd.DataFrame()
TD_user2['Avg. Trip Duration'] = round(df.groupby('User Type')['Minutes'].mean(),1)
TD_user2 = TD_user2.reset_index()
TD_user2['User Type'] = TD_user2['User Type'].astype('object')
```

In [90]:
```
TD_user2['Avg. Trip Duration']
```

Out[90]:
```
0    23.2
1    10.9
Name: Avg. Trip Duration, dtype: float64
```

In [91]:
```
TD_user2['User Type']
```

Out[91]:
```
0      Customer
1    Subscriber
Name: User Type, dtype: object
```

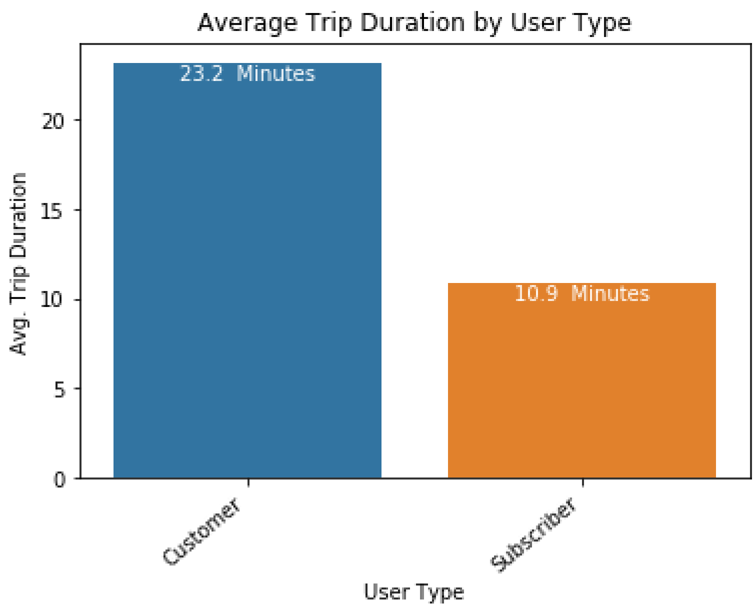
In [92]:
```
TD_user2
```

Out[92]:

	User Type	Avg. Trip Duration
0	Customer	23.2
1	Subscriber	10.9

Average Trip Duration Based on Minutes

In [93]:
```
#Average Trip Duration Based on Minutes
ax3 = sns.barplot('User Type', 'Avg. Trip Duration', data = TD_user2)
ax3.set_title('Average Trip Duration by User Type')
#rcParams['figure.figsize'] = 12,10
ax3.set_xticklabels(ax3.get_xticklabels(),rotation=40, ha = 'right')
for index, row in TD_user2.iterrows():
    ax3.text(row.name,row['Avg. Trip Duration']-1,(str(row['Avg. Trip Duration'])+"  Minutes"),
             color='white', ha="center", fontsize = 10)
plt.show()
```



In [94]:
```
del(TD_user2)
```

undo rounding for modeliing purposes

In [95]:
```
df['Minutes'] = df['Trip Duration']/60
```

QUESTION 3: Most Popular Trip

To get most popular trips, the most convenient way to do this is by using the groupby function in pandas. It's analogous to a Pivot table.

The groupby function makes it extremely easy and convenient to identify the most popular trips. Visuals and transformations can be found below

In [96]:
```
#Identify the 10 most popular trips
trips_df = pd.DataFrame()
trips_df = df.groupby(['Start Station Name','End Station Name']).size().reset_index(name = 'Number of Trips')
trips_df = trips_df.sort_values('Number of Trips', ascending = False)
```

In [97]:
```
trips_df.head()
```

Out[97]:

	Start Station Name	End Station Name	Number of Trips
46225	E 7 St & Avenue A	Cooper Square & E 7 St	440
83306	W 21 St & 6 Ave	9 Ave & W 22 St	367
55741	Greenwich Ave & Charles St	Greenwich Ave & Charles St	353
38414	E 33 St & 2 Ave	W 33 St & 7 Ave	317
70416	Pershing Square North	E 24 St & Park Ave S	305

In [98]:
```
trips_df["Start Station Name"] = trips_df["Start Station Name"].astype(str)
trips_df["End Station Name"] = trips_df["End Station Name"].astype(str)
```

In [99]:
```
trips_df["Trip"] = trips_df["Start Station Name"] + " to " + trips_df["End Station Name"]
```

In [100]:
```
trips_df = trips_df[:10]
print(trips_df)
```

```
                   Start Station Name              End Station Name  \
46225              E 7 St & Avenue A        Cooper Square & E 7 St   
83306                W 21 St & 6 Ave               9 Ave & W 22 St   
55741       Greenwich Ave & Charles St    Greenwich Ave & Charles St   
38414                E 33 St & 2 Ave               W 33 St & 7 Ave   
70416          Pershing Square North          E 24 St & Park Ave S   
68002             N 6 St & Bedford Ave  Wythe Ave & Metropolitan Ave   
70348          Pershing Square North            Broadway & W 32 St   
70606          Pershing Square North               W 33 St & 7 Ave   
98238   Wythe Ave & Metropolitan Ave          N 6 St & Bedford Ave   
72758       Richardson St & N Henry St     Graham Ave & Conselyea St   

       Number of Trips                                               Trip  
46225              440          E 7 St & Avenue A to Cooper Square & E 7 St  
83306              367                  W 21 St & 6 Ave to 9 Ave & W 22 St  
55741              353  Greenwich Ave & Charles St to Greenwich Ave & ...  
38414              317                  E 33 St & 2 Ave to W 33 St & 7 Ave  
70416              305     Pershing Square North to E 24 St & Park Ave S  
68002              295  N 6 St & Bedford Ave to Wythe Ave & Metropolit...  
70348              293       Pershing Square North to Broadway & W 32 St  
70606              276          Pershing Square North to W 33 St & 7 Ave  
98238              255  Wythe Ave & Metropolitan Ave to N 6 St & Bedfo...  
72758              253  Richardson St & N Henry St to Graham Ave & Con...  
```

In [101]:
```
trips_df = trips_df.drop(['Start Station Name', "End Station Name"], axis = 1)
```

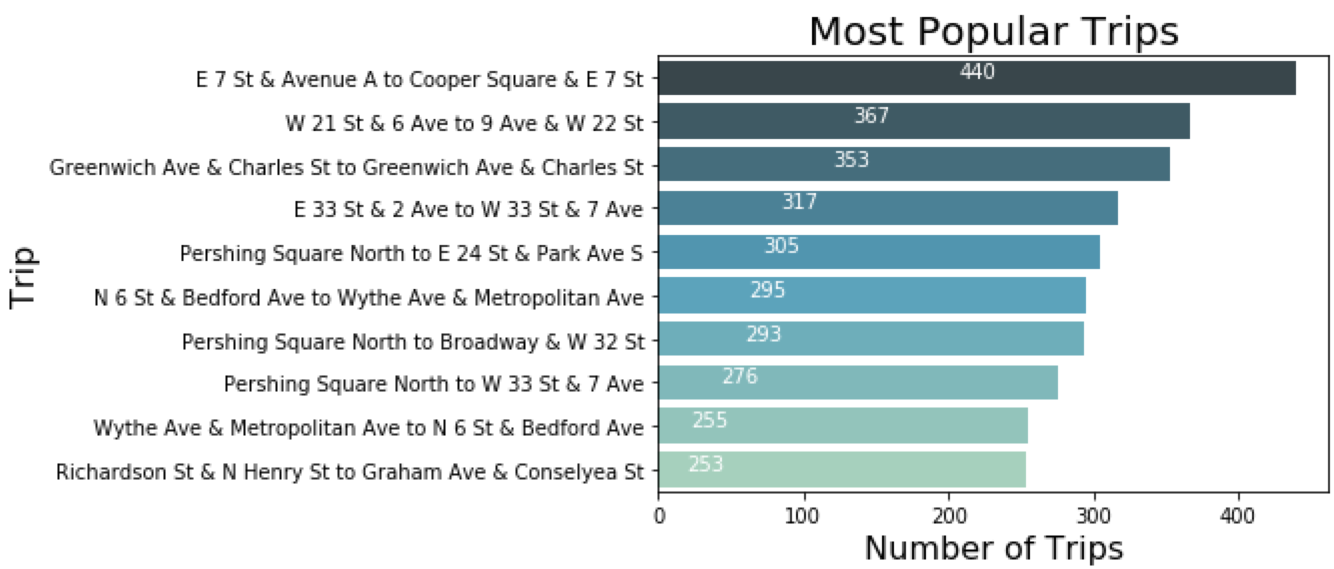
In [102]:
```
trips_df = trips_df.reset_index()
```

In [103]:
```
trips_df.head()
```

Out[103]:

	index	Number of Trips	Trip
0	46225	440	E 7 St & Avenue A to Cooper Square & E 7 St
1	83306	367	W 21 St & 6 Ave to 9 Ave & W 22 St
2	55741	353	Greenwich Ave & Charles St to Greenwich Ave & ...
3	38414	317	E 33 St & 2 Ave to W 33 St & 7 Ave
4	70416	305	Pershing Square North to E 24 St & Park Ave S

In [104]:
```
ax4 = sns.barplot('Number of Trips','Trip', data = trips_df,palette="GnBu_d")
ax4.set_title('Most Popular Trips', fontsize = 20)
ax4.set_ylabel("Trip",fontsize=16)
ax4.set_xlabel("Number of Trips",fontsize=16)
for index, row in trips_df.iterrows():
    ax4.text(row['Number of Trips']-220,index,row['Number of Trips'],
             color='white', ha="center",fontsize = 10)
plt.show()
```



Deleting Trip_df dataframe

In [105]:
```
del(trips_df)
```

QUESTION 4: Rider Performance by Gender and Age

ASK;- Rider performance by Gender and Age based on avg trip distance (station to station), median speed (trip duration/distance traveled) Let's make sure the data we're working with here is clean.Ask: Rider performance by Gender and Age based on avg trip distance (station to station), median speed (trip duration/distance traveled) Let's make sure the data we're working with here is clean. Missing Gender and Birth Year values - Check missing_table above No for Gender. Yes for Birth Year ~10% Missing Birth year. Not a big chunk of data. Can either impute missing values or drop it. Since it's less than 10% of the data, it's safe to assume the rest of the 90% is a representative sample of data and we can replace the birth year with the median, based on gender and Start Station ID. I chose this method because most people the same age live in similar neighborhoods (i.e: young people in east village, older people in Upper West Side, etc.). This will be done after anomalies are removed and speed is calculated. Are there anomalies? For Birth Year, there are some people born prior to 1957. I can believe some 60 year olds can ride a bike and that's a stretch, however, anyone "born" prior to that riding a citibike is an anomaly and false data. There could be a few senior citizens riding a bike, but probably not likely. My approach is to identify the age 2 standard deviations lower than the mean. After calculating this number, mean-2stdev, I removed the tail end of the data, birth year prior to 1957. Calculate an Age column to make visuals easier to interpret. Calculate trip distance (Miles) Caulculate Speed (min/mile) and (mile/hr) (min/mile): Can be used like sprint time (how fast does this person run) (mile/hr): Conventional approach. Miles/hour is an easy to understand unit of measure and one most people are used to seeing. So the visual will be created based on this understanding. Dealing with "circular" trips Circular trips are trips which start and end at the same station. The distance for these trips will come out to 0, however, that is not the case. These points will skew the data and visuals. Will be removing them to account for this issue. For the model, this data is also irrelevant. Because if someone is going on a circular trip, the only person who knows how long the trip is going to take is themself, assuming they know that. So it's safe to drop this data for the model.

In [106]:
```
#Drop the tail end of birth years 2 standard deviations below the mean
#df['Birth Year'].mean()-(2*df['Birth Year'].std())
df = df.drop(df.index[(df['Birth Year'] < 1957)])
```

Reset Index to avoid issues in future calculations

In [107]:
```
#Reset Index to avoid issues in future calculations
df = df.reset_index()
df = df.drop('index',axis =1)
```

In [108]:
```
#Combine coordinates to calculate distance based on Vincenty
df['Start Coordinates'] = list(zip(df['Start Station Latitude'], df['Start Station Longitude']))
df['End Coordinates'] = list(zip(df['End Station Latitude'], df['End Station Longitude']))
```

Install geopy to find distance using given coordinates

In [109]:
```
!pip install geopy
from geopy.distance import geodesic

dist = []
for i in range(len(df)):
    dist.append(geodesic(df['Start Coordinates'][i],df['End Coordinates'][i]).miles)
    if (i%1000000==0):
        print(i)
```

```
Requirement already satisfied: geopy in c:\users\mahwish\anaconda3\lib\site-packages
Requirement already satisfied: geographiclib<2,>=1.49 in c:\users\mahwish\anaconda3\lib\site-packages (from g
eopy)
0

You are using pip version 9.0.3, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

In [110]:
```
 #print(dist[i])
```

In [111]:
```
#Reset Index to avoid issues in future calculations
df = df.reset_index()
df = df.drop('index',axis =1)
```

In [112]:
```
df['Distance'] = dist
```

In [113]:
```
del(dist)
#Replace missing birth year by median based on speed and gender
df['Birth Year'] = df.groupby(['Gender','Start Station ID'])['Birth Year'].transform(lambda x: x.fillna(x.med
ian()))
```

In [114]:
```
#Still have a few nulls, but few so Comfortable dropping these.
df = df.dropna(subset=['Birth Year'])
```

In [115]:
```
#Calculate age and drop circular/roundtrips
df['Age'] = 2018 - df['Birth Year']
df['Age'] = df['Age'].astype(int)
```

In [116]:
```
df = df.drop(df.index[(df['Distance'] == 0)])
```

. Followed the same reasoning as behind Birth Year. People in similar locations tend to also work in a similar industry or location

In [117]:
```
df['Distance'] = df.groupby(['Gender','Start Station ID'])['Distance'].transform(lambda x: x.fillna(x.median
()))
```

Caulculate Speed (min/mile) and (mile/hr)

(min/mile): Can be used like sprint time (how fast does this person run) (mile/hr): Conventional approach. Miles/hour is an easy to understand unit of measure and one most people are used to seeing. So the visual will be created based on this understanding.

In [118]:
```
df['min_mile'] = round(df['Minutes']/df['Distance'], 2)
df['mile_hour'] = round(df['Distance']/(df['Minutes']/60),2)
```

Let's check for data integrity to make sure all the numbers look as expected. Only numerical data included

In [119]:
```
#Let's check for data integrity to make sure all the numbers look as expected. Only numerical data
#included
print(round(df.describe(),2))
```

```
       Trip Duration  Start Station ID  Start Station Latitude  \
count      670501.00         670501.00               670501.00   
mean          664.19           1217.57                   40.74   
std           505.29           1274.98                    0.03   
min            61.00             72.00                   40.65   
25%           328.00            358.00                   40.72   
50%           518.00            482.00                   40.74   
75%           840.00           3090.00                   40.76   
max          7193.00           3443.00                   40.80   

       Start Station Longitude  End Station ID  End Station Latitude  \
count                670501.00       670501.00             670501.00   
mean                    -73.98         1192.23                 40.74   
std                       0.02         1263.49                  0.03   
min                     -74.02           72.00                 40.65   
25%                     -74.00          357.00                 40.72   
50%                     -73.99          479.00                 40.74   
75%                     -73.98         3076.00                 40.75   
max                     -73.93         3447.00                 40.80   

       End Station Longitude    Bike ID  Birth Year     Gender    Minutes  \
count              670501.00  670501.00   670501.00  670501.00  670501.00   
mean                  -73.99   21738.08     1978.65       1.17      11.07   
std                     0.02    4199.95       10.28       0.47       8.42   
min                   -74.03   14529.00     1957.00       0.00       1.02   
25%                   -74.00   17878.00     1970.00       1.00       5.47   
50%                   -73.99   21329.00     1980.00       1.00       8.63   
75%                   -73.98   25812.00     1987.00       1.00      14.00   
max                   -73.93   27325.00     2000.00       2.00     119.88   

        Distance        Age   min_mile  mile_hour  
count  670501.00  670501.00  670501.00  670501.00  
mean        1.05      39.35      11.82       6.02  
std         0.78      10.28      14.40       1.71  
min         0.02      18.00       3.44       0.01  
25%         0.53      31.00       8.45       5.00  
50%         0.83      38.00       9.96       6.02  
75%         1.31      48.00      12.01       7.10  
max         8.98      61.00    4116.27      17.46  
```

In [120]:

```
df = df.drop(df.index[(df['Distance'] == 0)])
```

Observations We still have trips less than 90 seconds, however they seem to be legitimate trips. Checked using the code in cell above. We have some Start Coordinates as (0.0,0.0). These are trips which were taken away for repair or for other purposes. These should be dropped. If kept, the distance for these trips is 5,389 miles. For this reason I've dropped any points where the distance is greater than 30 miles. Additionally, we have some missing values. Since it's a tiny portion, let's replace missing values based on Gender and start location. These One some trips, the speed of the biker is more than 200 mph. This could be due to the formula used for distance calculation or some other error. The fastest cyclist in the world on a flat surface ever recorded biked at 82mph. It's safe to assume none of the citibike riders can approach this speed. Due to this and the fact that an average cyclist speed is 10mph, I've decided to remove all data where the speed in mph is greater than 20 mph and less than 0.1 mph. ~1.5k data points

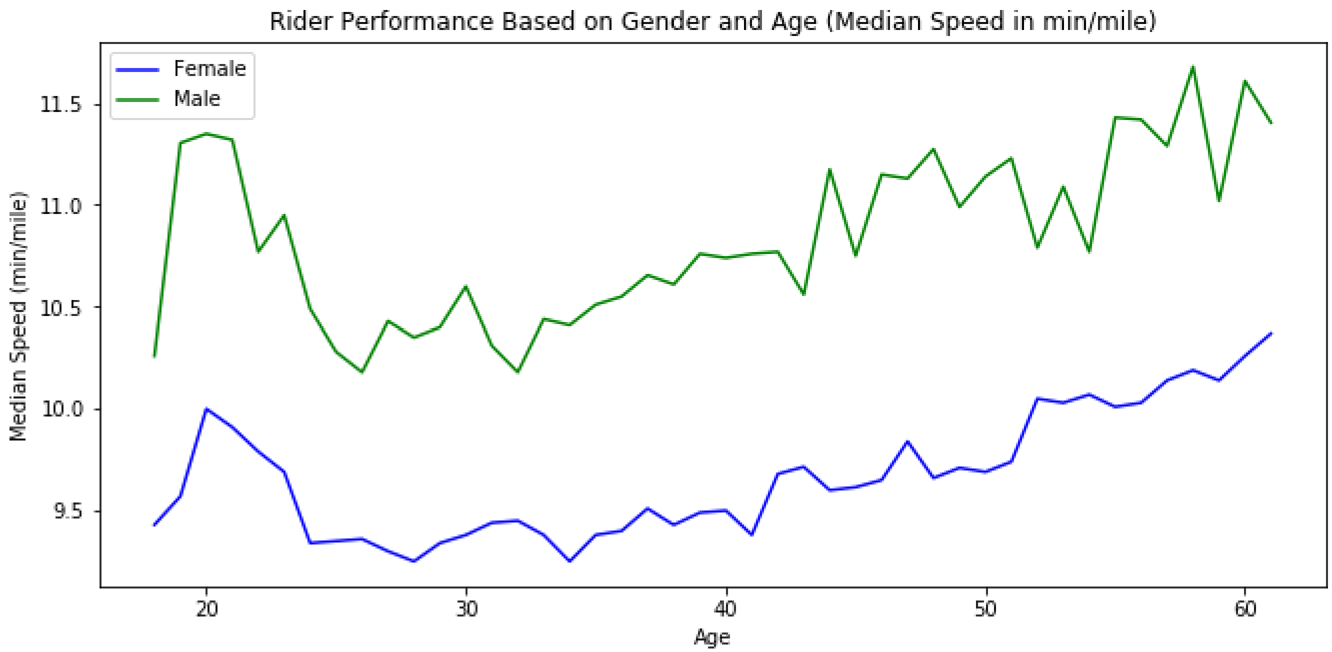
In [121]:
```
df = df[df['Distance'] < 30]
```

In [122]:
```
#1-Done in two steps to ensure data integrity, could've used an or statement as well.
df = df[df['mile_hour']<20]
#2
df = df[df['mile_hour']> (df['mile_hour'].mean()-(2*df['mile_hour'].std()))]
```

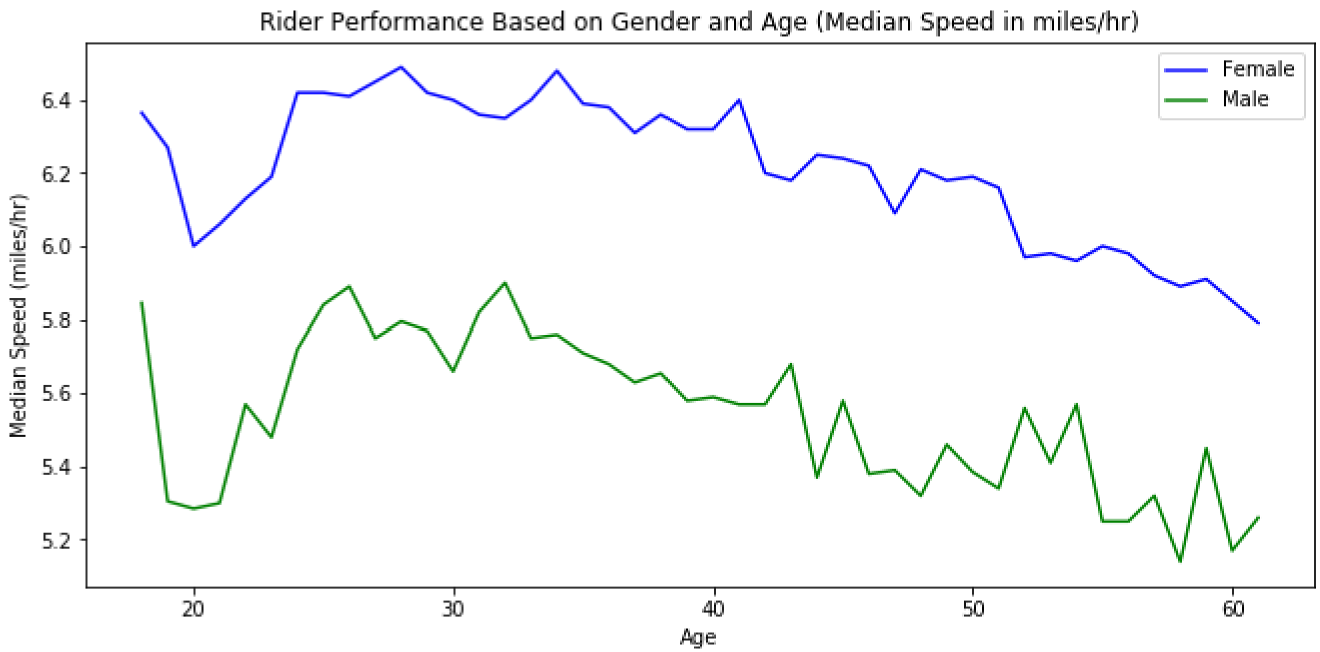
In [123]:
```
#Dropping unknown to make the visual more informative.
#Unknown gender may be important for the model, which is why I created a copy of the original dataframe.
df1 = df.drop(df.index[(df['Gender'] == 0)])
```

In [124]:
```
#Rider performance by age and Gender in Min/Mile
```

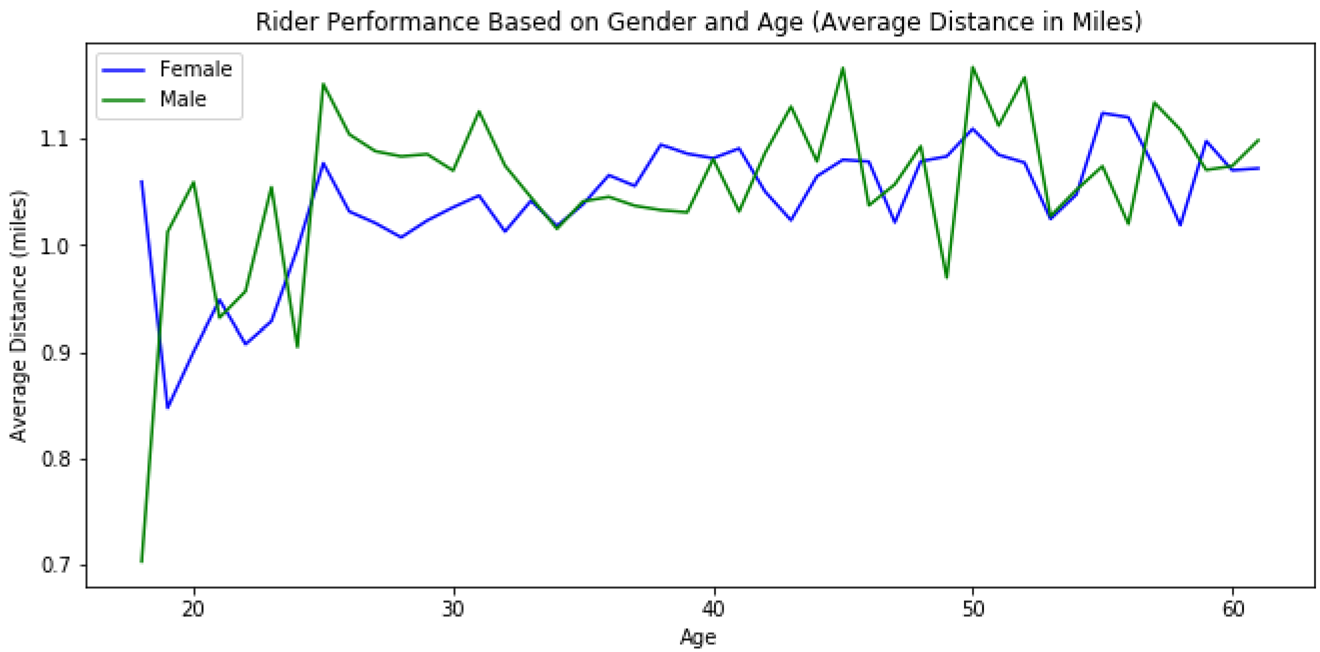
In [125]:
```
#Rider performance by age and Gender in Min/Mile
fig, ax5 = plt.subplots(figsize=(11,5))
df1.groupby(['Age','Gender']).median()['min_mile'].unstack().plot(ax=ax5, color ="bg")
ax5.legend(['Female','Male'])
plt.ylabel('Median Speed (min/mile)')
plt.title('Rider Performance Based on Gender and Age (Median Speed in min/mile)')
plt.show()
```



In [126]:
```
#Rider performance by age and Gender in Miles/hr
del([fig,ax5])
fig1, ax6 = plt.subplots(figsize=(11,5))
df1.groupby(['Age','Gender']).median()['mile_hour'].unstack().plot(ax=ax6,color ="bg")
ax6.legend(['Female', 'Male'])
plt.ylabel('Median Speed (miles/hr)')
plt.title('Rider Performance Based on Gender and Age (Median Speed in miles/hr)')
plt.show()
```



In [127]:
```
#Rider performance by age and Gender in Averge Distance
del([fig1,ax6])
fig2, ax7 = plt.subplots(figsize=(11,5))
df1.groupby(['Age','Gender']).mean()['Distance'].unstack().plot(ax=ax7,color ="bg")
ax7.legend(['Female', 'Male'])
plt.ylabel('Average Distance (miles)')
plt.title('Rider Performance Based on Gender and Age (Average Distance in Miles)')
plt.show()
```



QUESTION 5

Busiest Bike by Times and Minutes Used

What is the busiest bike in NYC in 2017?

Bike 26022

How many times was it used? times 250

How many minutes was it in use? Minutes 2984

Busiest bike and count can be identified by a groupby function

Function above will also identify the number of times the bike was used A similar groupby function which calls for the sum on minutes can identify the number of minutes the bike was used.

Bike usage based on number of times used

In [128]:
```
#Bike usage based on number of times used
del(df1)
bike_use_df = pd.DataFrame()
bike_use_df = df.groupby(['Bike ID']).size().reset_index(name = 'Number of Times Used')
bike_use_df = bike_use_df.sort_values('Number of Times Used', ascending = False)
#bike_use_df.to_csv('Q5.csv')
bike_use_df = bike_use_df[:10]
bike_use_df['Bike ID'] = bike_use_df['Bike ID'].astype(str)
bike_use_df['Bike ID'] = ('Bike ' + bike_use_df['Bike ID'])
bike_use_df = bike_use_df.reset_index()
```

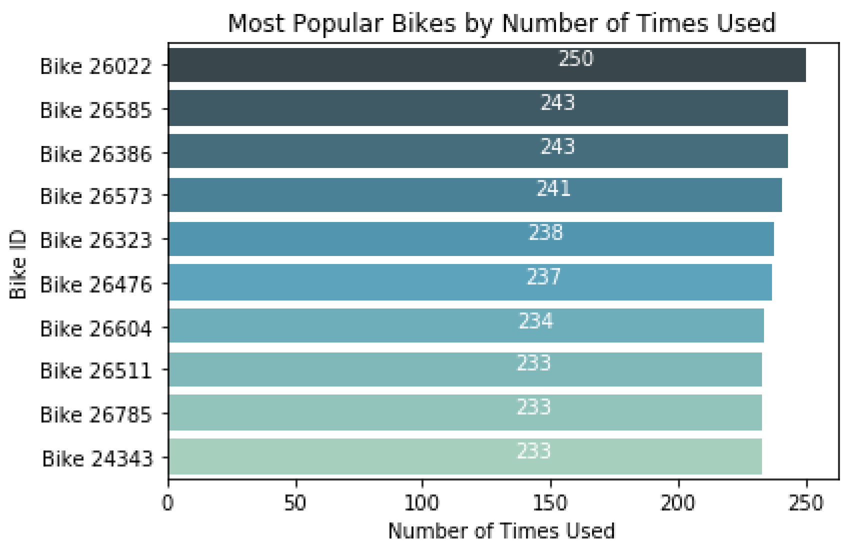
In [129]:
```
bike_use_df.head()
```

Out[129]:

	index	Bike ID	Number of Times Used
0	7088	Bike 26022	250
1	7470	Bike 26585	243
2	7282	Bike 26386	243
3	7460	Bike 26573	241
4	7221	Bike 26323	238

Visual of most used bike based on Number of Trips

In [130]:
```
ax8 = sns.barplot('Number of Times Used', 'Bike ID',data = bike_use_df, palette="GnBu_d")
ax8.set_title('Most Popular Bikes by Number of Times Used')
for index, row in bike_use_df.iterrows():
    ax8.text(row['Number of Times Used']-90,index,row['Number of Times Used'], color='white', ha="center", fo
ntsize =10)
plt.show()
```



Bike usage based on minutes used

In [131]:
```
del(ax8)
```

In [132]:
```
bike_min_df = pd.DataFrame()
bike_min_df['Minutes Used'] = df.groupby('Bike ID')['Minutes'].sum()
bike_min_df = bike_min_df.reset_index()
bike_min_df = bike_min_df.sort_values('Minutes Used', ascending = False)
bike_min_df['Bike ID'] = bike_min_df['Bike ID'].astype(str)
bike_min_df['Bike ID'] = ('Bike ' + bike_min_df['Bike ID'])
bike_min_df = bike_min_df[:10]
bike_min_df = bike_min_df.reset_index()
```

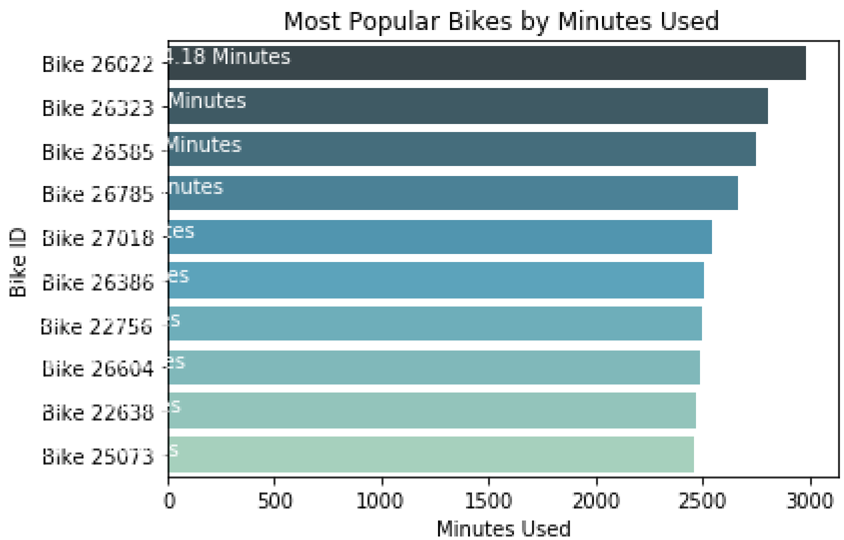
In [133]:
```
bike_min_df.head()
```

Out[133]:

	index	Bike ID	Minutes Used
0	7088	Bike 26022	2984.183333
1	7221	Bike 26323	2810.500000
2	7470	Bike 26585	2751.966667
3	7648	Bike 26785	2671.283333
4	7878	Bike 27018	2540.733333

#Visual of most used bike based on number of minutes used

In [134]:
```
ax9 = sns.barplot('Minutes Used', 'Bike ID',data = bike_min_df, palette="GnBu_d")
ax9.set_title('Most Popular Bikes by Minutes Used')
rcParams['figure.figsize'] = 11,6
for index, row in bike_min_df.iterrows():
    ax9.text(row['Minutes Used']-2800,index,str(round(row['Minutes Used'],2))+' Minutes',
             color='white', ha="center")
plt.show()
```



1) Predictive Model - Baseline Model

Ask: Build a model that can predict how long a trip will take given a starting point and destination.

Let's now begin to train out regression model! We will need to first split up our data into an X array that contains the features to train on, and a y array with the target variable ,in this case the Minutes columnAssumptions on how the Kiosk will work: After speaking to Daniel Yawitz (if you're looking at this, thanks for the clarification), I was told that we should assume that when a user inputs the start and end station, they swipe their key fob (if they're a subscriber) and enter their info on the kiosk (if they're a "Customer") prior to entering the start and end station. This means that we would know their gender and age. Thus these variables can be used in building the model.

Step 1.

This dataset is massive,Let's work on a random subsample while we build and evaluate models. If I tried to build and evaluate a model on the entire dataset, each run would take me ~10+ minutes depending on the model.I also made the same visuals on the sample to visualize the data.

Step 2.

Let's get a baseline. If I were to just run a simple multi-variate linear regression, what would my model look like and how accurate would it be? Need to prepare the data for a multivariate regression 1)Drop irrelevant columns Trip Duration: We have the minutes column, which is the target variable Stop Time: In the real world, we won't have this information when predicting the trip duration. Start Station ID: Start Station Name captures this information Start Station Latitude: Start Station Name captures this information Start Station Longitude: Start Station Name captures this information Start Coordinates: Start Station Name captures this information End Station ID: End Station Name captures this information End Station Latitude: End Station Name captures this information End Station Longitude: End Station Name captures this information End Coordinates: End Station Name captures this information Bike Id: We won't know what bike the user is going to end up using Min_Mile: Effectively the same information as end time when combined with distance. We won't have this information in the real world. mile_hour: Effectively the same information as end time when combined with distance. We won't have this information in the real world. (Speed * Distance = Trip Duration): Which is why speed is dropped Birth Year: Age captures this information Start Station Name and End Station Name: The distance variable captures the same information. For the model, if a user is inputting start and end station, we can build a simple function to calculate the distance which would capture the same information. 2) Basic cleaning of data FOR NOW.This is only being done for the baseline model Start Time: Requires reformatting. Will do this after baseline model Dumify categorical variables Scale Age Don't scale distance, since it does not just represent distance, but is also indicative of the trip the rider is making (start and station) 3) Anomalies in Trip Duration I'm going to come back to an observation from earlier. Any trip which lasts longer than 45 minutes(2,700 seconds) probably indicates a stolen bike, incorrect docking of the bike, or an anomaly. No rider would plan to go over the maximum 45 minutes allowed. Age was removed after an initial run indicated it had no effect on the model. This was also clearly indicated in some of the visuals above.

In [136]:
```
print(round(df.describe(),2))
```

```
       Trip Duration  Start Station ID  Start Station Latitude  \
count      649110.00         649110.00               649110.00
mean          635.93           1214.20                   40.74
std           451.03           1273.17                    0.03
min            61.00             72.00                   40.65
25%           323.00            359.00                   40.72
50%           507.00            482.00                   40.74
75%           810.00           3090.00                   40.76
max          7062.00           3443.00                   40.80

       Start Station Longitude  End Station ID  End Station Latitude  \
count                649110.00       649110.00             649110.00
mean                    -73.98         1188.18                 40.74
std                       0.02         1261.28                  0.03
min                     -74.02           72.00                 40.65
25%                     -74.00          357.00                 40.72
50%                     -73.99          479.00                 40.74
75%                     -73.98         3071.00                 40.75
max                     -73.93         3447.00                 40.80

       End Station Longitude    Bike ID  Birth Year     Gender    Minutes  \
count              649110.00  649110.00   649110.00  649110.00  649110.00
mean                  -73.99   21746.11     1978.69       1.18      10.60
std                     0.02    4200.11       10.26       0.46       7.52
min                   -74.03   14529.00     1957.00       0.00       1.02
25%                   -74.00   17886.00     1971.00       1.00       5.38
50%                   -73.99   21339.00     1980.00       1.00       8.45
75%                   -73.98   25816.00     1987.00       1.00      13.50
max                   -73.93   27325.00     2000.00       2.00     117.70

        Distance        Age   min_mile  mile_hour
count  649110.00  649110.00  649110.00  649110.00
mean        1.06      39.31      10.40       6.17
std         0.78      10.26       2.88       1.53
min         0.06      18.00       3.44       2.60
25%         0.54      31.00       8.40       5.11
50%         0.84      38.00       9.86       6.09
75%         1.33      47.00      11.74       7.14
max         8.98      61.00      23.12      17.46
```

Observations

We have some unreasonable speeds in min_mile column, but that's fine. Some people may have walked with their bike or stopped at multiple destinations before docking. The sample data below seems to be representative of the entire dataset above.#Training a Linear Regression Model #We have the minutes column, which is the target variable #1min =60 sec #45min =45*60= 2700 sec=Trip duration more than this number will be dropped from the data set.

Cleaning up anomalies

In [137]:
```
#Cleaning up anomalies
df = df.drop(df.index[(df['Trip Duration'] > 2700)])
```

Training a Linear Regression Model We have the minutes column, which is the target variable 1min =60 sec 45min =45*60= 2700 sec=Trip duration more than this number will be dropped from the data set.

In [142]:
```
#Let's work with a random sample of the data and inspect it thoroughly to ensure it's representative
df_sample = df.sample(frac = 0.1, random_state = 0)
print(df_sample.describe())
```

```
       Trip Duration  Start Station ID  Start Station Latitude  \
count   64764.000000       64764.00000            64764.000000   
mean      629.120453        1211.78539               40.737337   
std       434.848317        1273.24560                0.026285   
min        61.000000          72.00000               40.646768   
25%       322.000000         358.00000               40.720874   
50%       505.500000         481.50000               40.739355   
75%       805.000000        3088.00000               40.755003   
max      2700.000000        3443.00000               40.804213   

       Start Station Longitude  End Station ID  End Station Latitude  \
count             64764.000000    64764.000000          64764.000000   
mean                -73.985055     1189.697455             40.737094   
std                   0.015974     1261.254191              0.025941   
min                 -74.017134       72.000000             40.646768   
25%                 -73.995299      357.000000             40.720828   
50%                 -73.987216      480.000000             40.739355   
75%                 -73.976806     3071.000000             40.754666   
max                 -73.929891     3443.000000             40.804213   

       End Station Longitude       Bike ID    Birth Year        Gender  \
count           64764.000000  64764.000000  64764.000000  64764.000000   
mean              -73.985361  21741.058860   1978.714764      1.179220   
std                 0.016033   4200.926265     10.247366      0.458099   
min               -74.017134  14529.000000   1957.000000      0.000000   
25%               -73.995960  17896.000000   1971.000000      1.000000   
50%               -73.987586  21347.000000   1981.000000      1.000000   
75%               -73.977061  25805.000000   1987.000000      1.000000   
max               -73.929891  27325.000000   2000.000000      2.000000   

            Minutes      Distance           Age      min_mile     mile_hour  
count  64764.000000  64764.000000  64764.000000  64764.000000  64764.000000  
mean      10.485341      1.051823     39.284031     10.399077      6.166821  
std        7.247472      0.766826     10.247372      2.882319      1.533457  
min        1.016667      0.067453     18.000000      4.240000      2.600000  
25%        5.366667      0.533514     31.000000      8.420000      5.110000  
50%        8.425000      0.835016     37.000000      9.860000      6.080000  
75%       13.416667      1.320865     47.000000     11.740000      7.130000  
max       45.000000      7.021076     61.000000     23.120000     14.140000  
```

Build a Baseline Model

In [143]:
```
#Drop Irrelevant data
def drop_data(df):
    df = df.drop(['Trip Duration','Stop Time','Start Station ID','Start Station Latitude','Start Station Long
itude',
                  'Start Coordinates','End Station ID', 'End Station Latitude', 'End Station Longitude',
                  'End Coordinates','Bike ID', 'Start Station Name','Birth Year','End Station Name','min_mil
e',
                  'mile_hour','Age'], axis = 1)
    return df

df_basemodel = drop_data(df_sample)
```

In [144]:
```
df_basemodel = df_basemodel.drop('Start Time', axis =1)
```

In [145]:
```
#Dummify categorical data and avoid dummy variable trap
df_basemodel = pd.get_dummies(df_basemodel, drop_first = True)
df_basemodel.dtypes
```

Out[145]:
```
Gender                       int64
Minutes                    float64
Distance                   float64
User Type_Subscriber         uint8
dtype: object
```

In [146]:
```
df_basemodel.head()
```

Out[146]:

	Gender	Minutes	Distance	User Type_Subscriber
259203	1	8.550000	0.756068	1
360096	1	6.633333	0.697781	1
434602	1	5.900000	1.023749	1
478347	1	4.000000	0.370099	1
347879	2	15.583333	1.488621	1

In [147]:
```
##As We have the minutes column, which is the target variable
df_basemodel.corr().loc[:,'Minutes']
```

Out[147]:
```
Gender                  0.005635
Minutes                 1.000000
Distance                0.899097
User Type_Subscriber   -0.152969
Name: Minutes, dtype: float64
```

Training a Linear Regression Base Model¶

Let’s now begin to train out regression model! We will need to first split up our data into an X array that contains the features to train on, and a y array with the target variable ,in this case the Minutes column

In [151]:
```
#Train Test Split
#Predictor variable
X = df_basemodel[['Gender','Distance','User Type_Subscriber']]
#Target variable
y = df_basemodel['Minutes']
```

#Train Test Split #Our goal is to create a model that generalises well to new data. Our test set serves as a proxy #for new data.Trained data is the data on which we apply the linear regression algorithm. #And finally we test that algorithm on the test data.The code for splitting is as follows: #from sklearn.cross_validation import train_test_split

In [153]:
```
from sklearn.cross_validation import train_test_split
```

```
C:\Users\Mahwish\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module
was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes a
nd functions are moved. Also note that the interface of the new CV iterators are different from that of this
module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [154]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

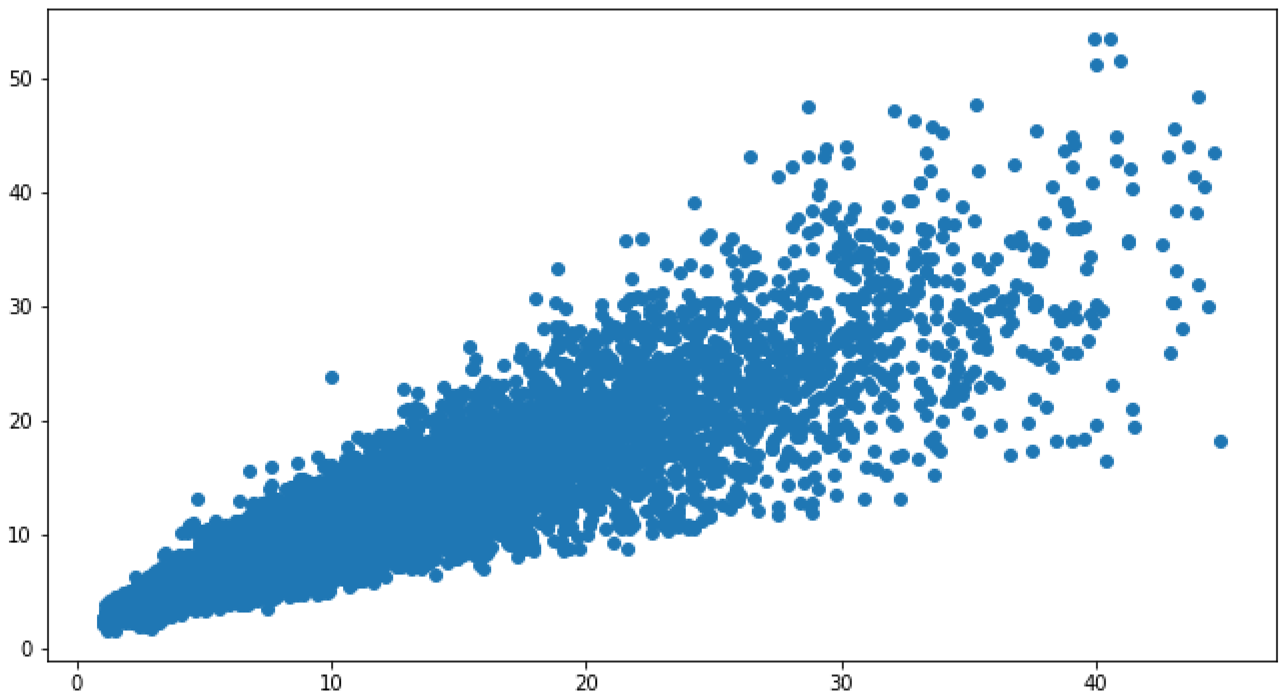
In [155]:
```
#Creating and Training the Model

from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
#The above code fits the linear regression model on the training data.
```

Out[155]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [156]:
```
#Predictions from our Model
#Let’s grab predictions off the test set and see how well it did!
predictions = lm.predict(X_test)
```

In [157]:
```
#Let’s visualise the prediction
plt.scatter(y_test,predictions)
```

Out[157]:
```
<matplotlib.collections.PathCollection at 0x1eb09215160>
```



In [158]:
```
#Model Accuracy
lm.score(X_test,y_test)
```

Out[158]: 0.8226371686192211

In [159]:
```
#Using Statsmodel because it has the summary function.
import statsmodels.api as sm
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)
lm_OLS = sm.OLS(y_train, X_train).fit()
lm_OLS.summary()
```

Out[159]:

OLS Regression Results

Dep. Variable:	Minutes	**R-squared:**	0.821
Model:	OLS	**Adj. R-squared:**	0.821
Method:	Least Squares	**F-statistic:**	7.946e+04
Date:	Tue, 20 Nov 2018	**Prob (F-statistic):**	0.00
Time:	23:34:00	**Log-Likelihood:**	-1.3167e+05
No. Observations:	51811	**AIC:**	2.634e+05
Df Residuals:	51807	**BIC:**	2.634e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	6.6541	0.098	68.047	0.000	6.462	6.846
Gender	1.0105	0.032	31.987	0.000	0.949	1.072
Distance	8.4293	0.018	479.888	0.000	8.395	8.464
User Type_Subscriber	-6.3531	0.103	-61.659	0.000	-6.555	-6.151

Omnibus:	16487.274	**Durbin-Watson:**	2.010
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	133659.054
Skew:	1.306	**Prob(JB):**	0.00
Kurtosis:	10.423	**Cond. No.**	22.3

Hence the model is successfully built