```
In [1]: %matplotlib inline # Shows the data visualizations after running the cod
        e below its code block
```

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        import urllib

        from sklearn import svm
        from sklearn.ensemble import RandomForestClassifier
```

Question 1

```
In [4]: dataCSV = urllib.request.urlopen("https://s3.amazonaws.com/nyc-tlc/trip+
        data/green_tripdata_2015-09.csv").read()
        with open("green_tripdata_2015-09_1.csv", 'wb') as greenTaxiData: # reme
        mber to change the file name for saving later on
            greenTaxiData.write(dataCSV)
```

programmatically download and load data into pandas DataFrame object

```
In [9]: fileURL = "https://s3.amazonaws.com/nyc-tlc/trip+data/green_tripdata_201
        5-09.csv"
        taxiData = pd.read_csv(fileURL)
```

Rows & Columns of data loaded

Rows : 1494926

Columns : 21

```
In [10]: # Show the dimension of the loaded data
         taxiData.shape

Out[10]: (1494926, 21)
```

```
In [11]: print("Loaded Data contains {} rows and {} columns".format(*taxiData.sha
         pe))

         Loaded Data contains 1494926 rows and 21 columns
```

In [12]:
```
# A quick lookthrough of the first and last 5 rows of data and its value
s
taxiData.head(5)
```

Out[12]:

	VendorID	lpep_pickup_datetime	Lpep_dropoff_datetime	Store_and_fwd_flag	RateCo
0	2	2015-09-01 00:02:34	2015-09-01 00:02:38	N	5
1	2	2015-09-01 00:04:20	2015-09-01 00:04:24	N	5
2	2	2015-09-01 00:01:50	2015-09-01 00:04:24	N	1
3	2	2015-09-01 00:02:36	2015-09-01 00:06:42	N	1
4	2	2015-09-01 00:00:14	2015-09-01 00:04:20	N	1

5 rows × 21 columns

In [13]:
```
taxiData.tail(5)
```

Out[13]:

	VendorID	lpep_pickup_datetime	Lpep_dropoff_datetime	Store_and_fwd_flag	F
1494921	1	2015-09-30 23:00:01	2015-09-30 23:17:21	N	1
1494922	1	2015-09-30 23:00:05	2015-09-30 23:08:13	N	1
1494923	1	2015-09-30 23:00:30	2015-09-30 23:08:39	N	1
1494924	1	2015-09-30 23:00:10	2015-09-30 23:03:49	N	1
1494925	1	2015-09-30 23:00:11	2015-09-30 23:05:36	N	1

5 rows × 21 columns

In [14]:
```
# Total number of data recorded
taxiData.Trip_distance.size
```

Out[14]:
```
1494926
```

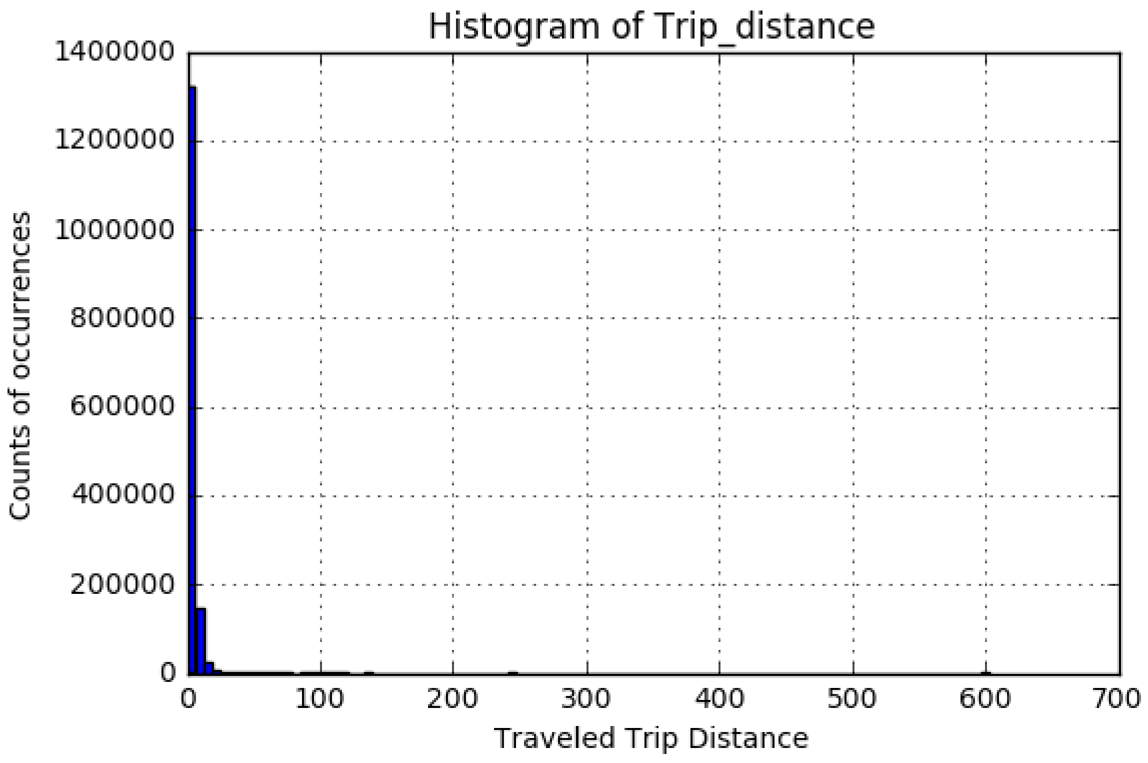
Question 2

In [119]:
```
# Calculate the MAX, MIN, MEAN and MEDIAN of Trip_distance
taxiData.Trip_distance.max() # 603.10000000000002 --> 605
taxiData.Trip_distance.min() # 0.0 -- > 0
taxiData.Trip_distance.size # 1494926 --> 1500000
taxiData.Trip_distance.mean() # 2.9681408511189864
taxiData.Trip_distance.median() # 1.98
print("""
Information Regarding Trip_distance:
    MAX : {},
    MIN : {},
    MEAN : {},
    MEDIAN : {}
""".format(taxiData.Trip_distance.max(),taxiData.Trip_distance.min(), ta
xiData.Trip_distance.mean(),
           taxiData.Trip_distance.median()))
```

```
Information Regarding Trip_distance:
    MAX : 603.1,
    MIN : 0.0,
    MEAN : 2.9681408511189864,
    MEDIAN : 1.98
```

In [120]:
```
# The histogram of all the data recorded in Trip_distance
plt.hist(taxiData.Trip_distance, bins = 100, range = [taxiData.Trip_dist
ance.min(),taxiData.Trip_distance.max()])

plt.xlabel('Traveled Trip Distance')
plt.ylabel('Counts of occurrences')
plt.title('Histogram of Trip_distance')
plt.grid(True)
```



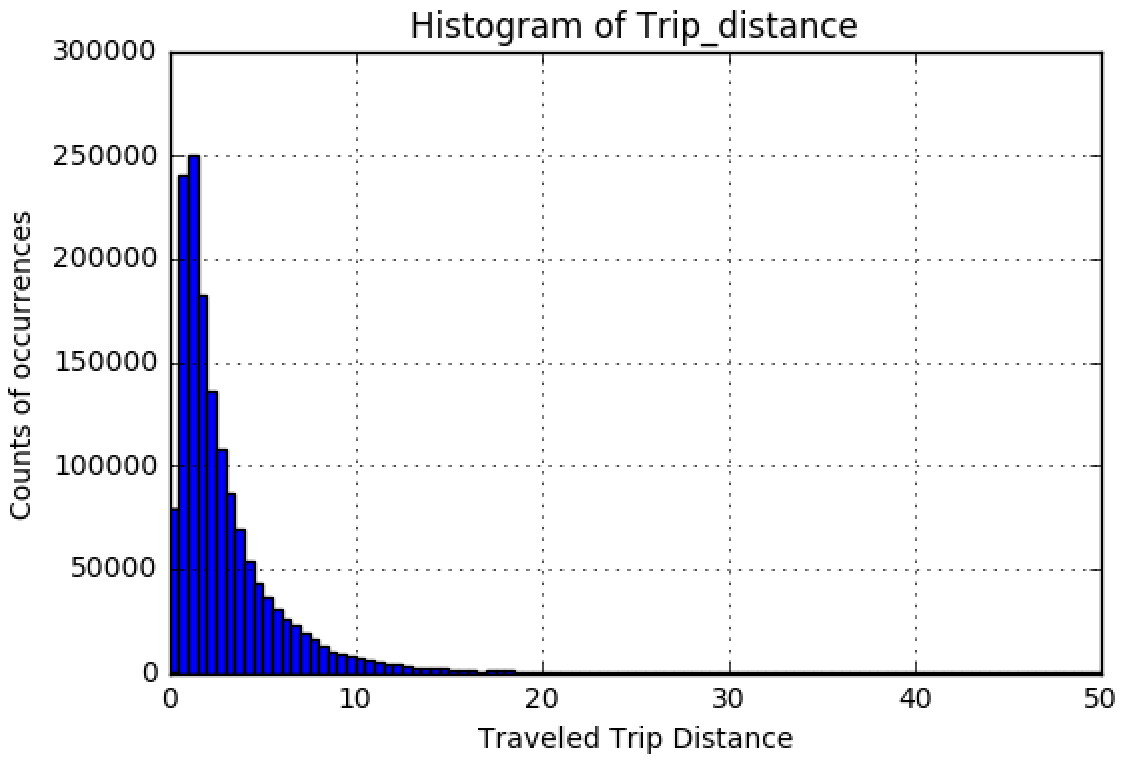
As we can see from above, most trip distances are small and seem to mostly happen within 20 miles

Thus I divided the histogram to show data within a smaller range for a more accurate look at the values

and distributions of the data.

We first plot a histogram with the distance of 0 ~ 50

In [121]:
```
# Histogram with the range of 0 ~ 50 miles
plt.hist(taxiData.Trip_distance, bins = 100, range = [taxiData.Trip_dist
ance.min(),50])
plt.xlabel('Traveled Trip Distance')
plt.ylabel('Counts of occurrences')
plt.title('Histogram of Trip_distance')
plt.grid(True)
```



From the histogram of distance between 0 and 50, we can see the distribution is highly right skewed and that most people travel within 10 miles. The number of travels for distance larger than 10 miles is comparably small.

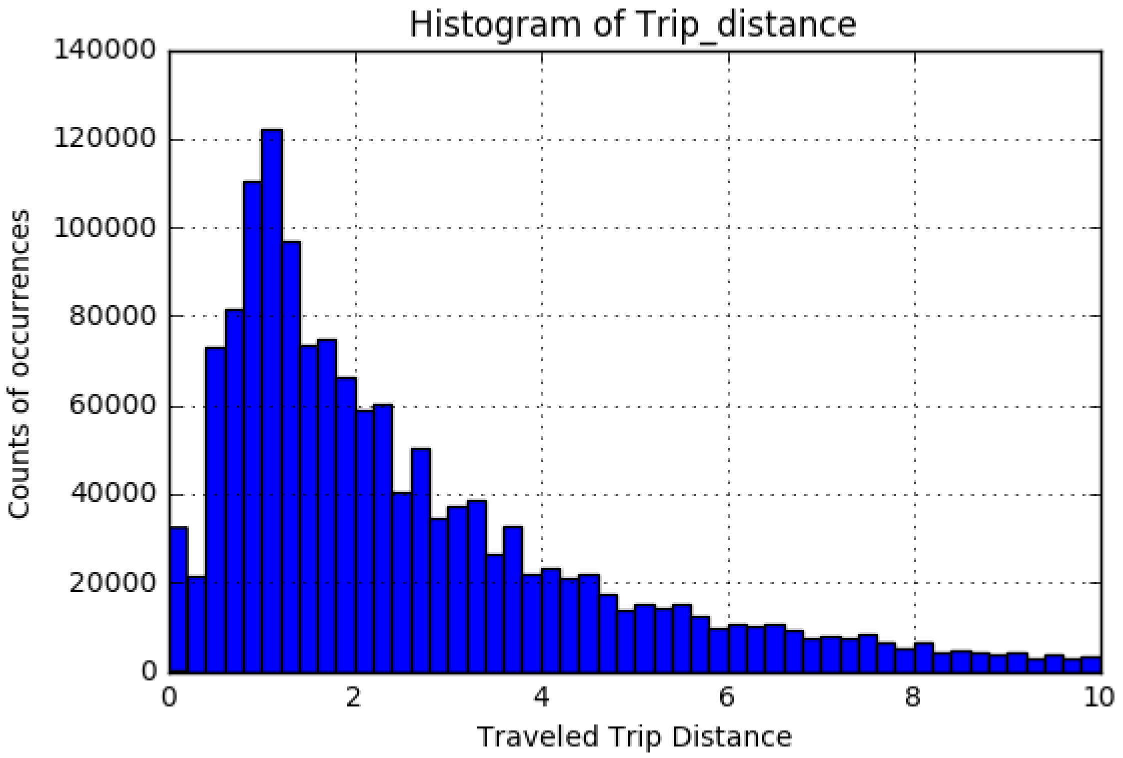
As there seems to be very little occurences of travel distance between 10 to 20 miles, and close to none with distances larger than 20 miles.

Yet as the unit of counts for travel distance is very large, with a unit of 50000. What may seem close to zero counts

may actually still have a significant value such as 500 or more.

Thus, I further plot two more histograms with the range of 0 ~ 10 miles and 10 ~ 20 miles each for better and more accurate visualization of the data value and its distribution.

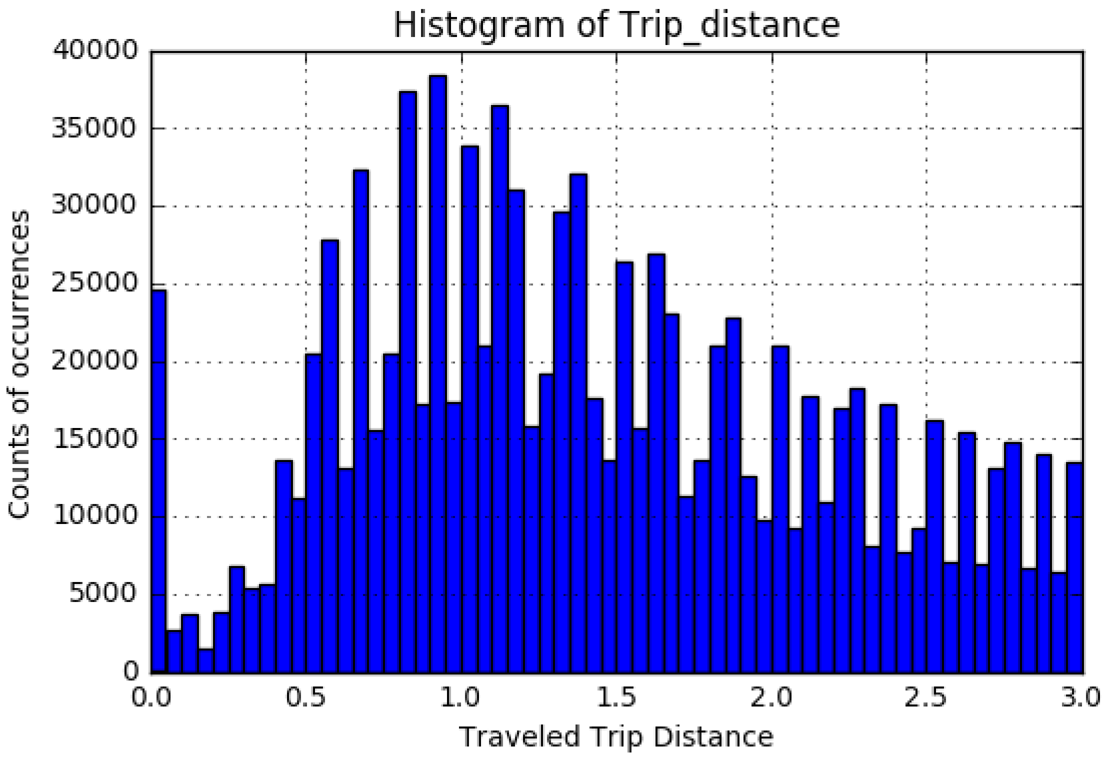
```
In [122]: # Histogram with the range of 0 ~ 10 miles
          plt.hist(taxiData.Trip_distance, bins = 50, range = [0,10])
          plt.xlabel('Traveled Trip Distance')
          plt.ylabel('Counts of occurrences')
          plt.title('Histogram of Trip_distance')
          plt.grid(True)
```



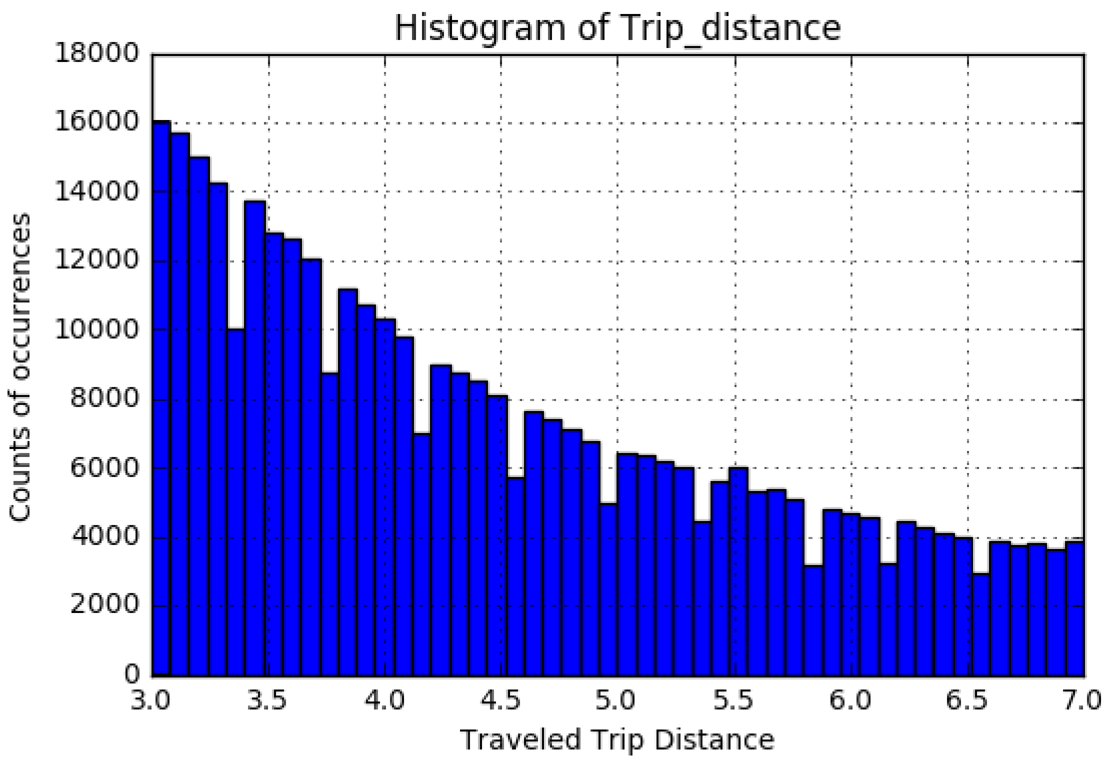
The above histogram gives us better insight for the values. However, the unit still seems a bit too big that we can't really observe the numbers of those less than 2000.

Thus I created three more histograms to display trip distance with the ranges of 0~3, 3~7 and 7~10 miles for a clearer view of the values.

```
In [123]: # Histogram with the range of 0 ~ 3 miles
          plt.hist(taxiData.Trip_distance, bins = 60, range = [0, 3])
          plt.xlabel('Traveled Trip Distance')
          plt.ylabel('Counts of occurrences')
          plt.title('Histogram of Trip_distance')
          plt.grid(True)
```

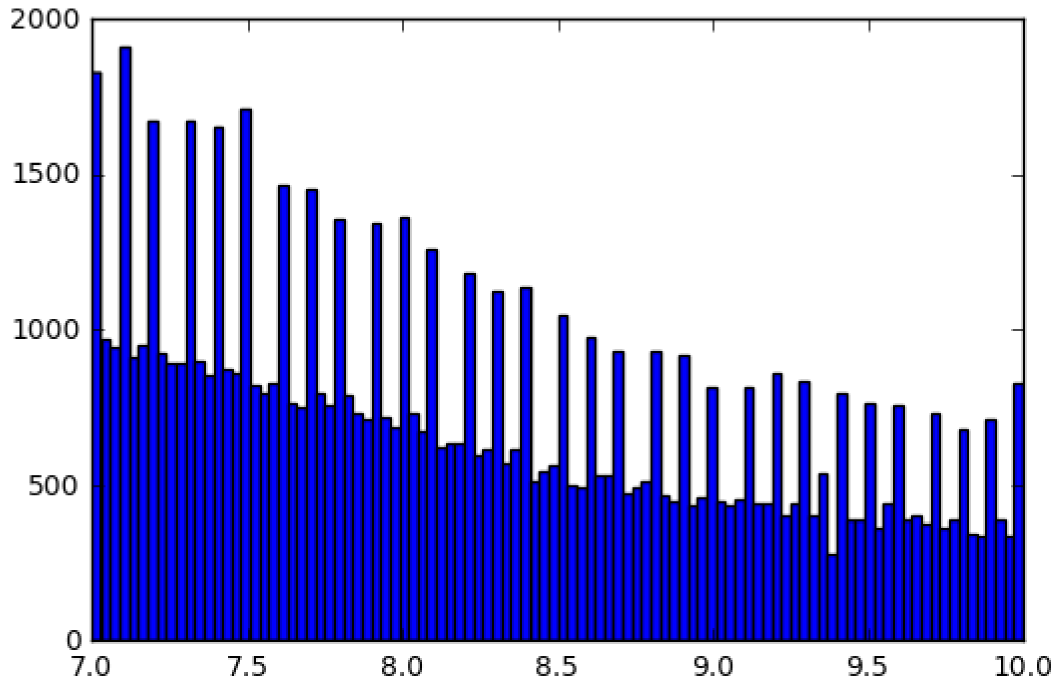


```
In [124]: # Histogram with the range of 3 ~ 7 miles
          plt.hist(taxiData.Trip_distance, bins = 50, range = [3, 7])
          plt.xlabel('Traveled Trip Distance')
          plt.ylabel('Counts of occurrences')
          plt.title('Histogram of Trip_distance')
          plt.grid(True)
```

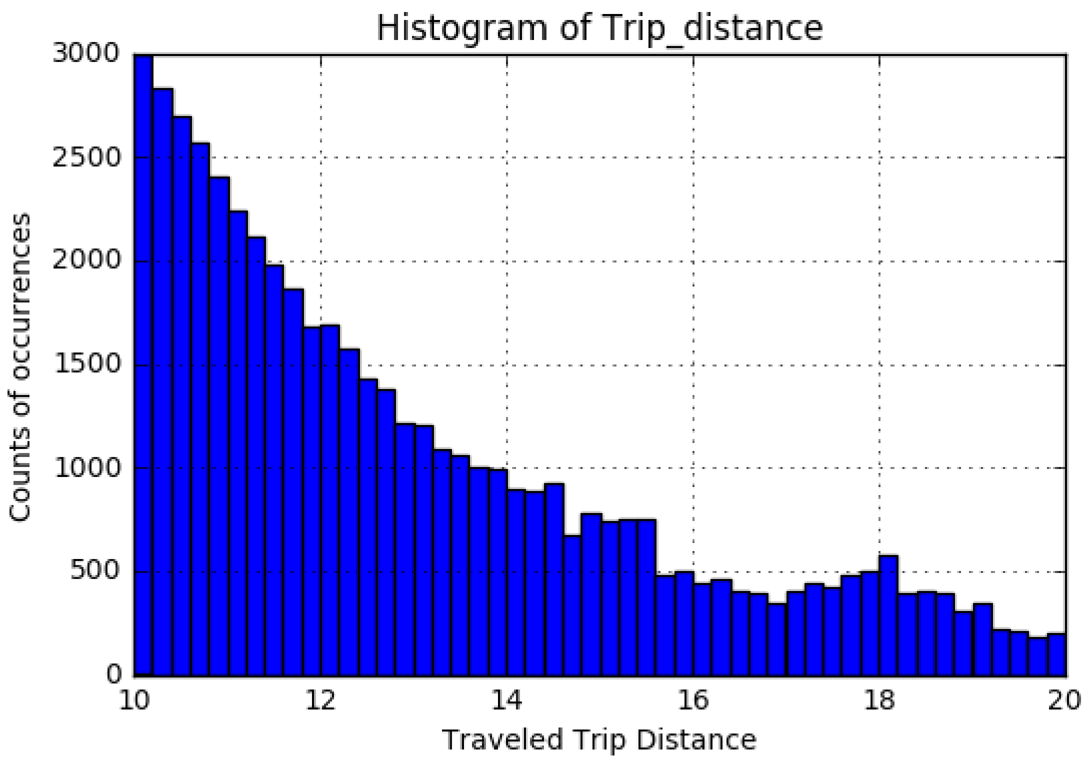


```
In [125]: # Histogram with the range of 7 ~ 10 miles
          plt.hist(taxiData.Trip_distance, bins = 100, range = [7, 10])
```

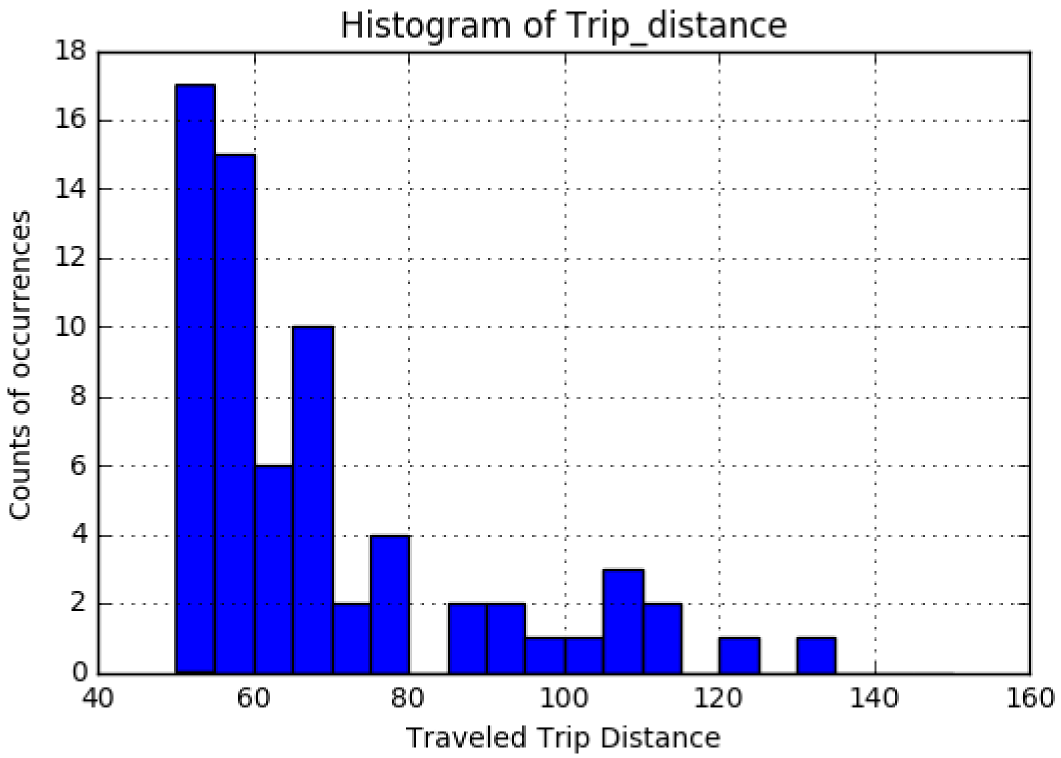
```
Out[125]: (array([ 1828.,   968.,   943.,  1912.,   914.,   949.,  1672.,   927.,
                    893.,   892.,  1675.,   897.,   854.,  1652.,   877.,   862.,
                   1713.,   819.,   798.,   829.,  1470.,   762.,   752.,  1457.,
                    797.,   755.,  1358.,   793.,   729.,   713.,  1344.,   718.,
                    686.,  1365.,   730.,   677.,  1259.,   624.,   634.,   635.,
                   1181.,   596.,   617.,  1124.,   573.,   618.,  1138.,   514.,
                    547.,   566.,  1047.,   497.,   492.,   977.,   533.,   529.,
                    930.,   471.,   493.,   514.,   929.,   465.,   451.,   916.,
                    438.,   463.,   816.,   448.,   437.,   453.,   814.,   444.,
                    442.,   859.,   405.,   444.,   834.,   405.,   541.,   282.,
                    796.,   388.,   388.,   767.,   367.,   439.,   758.,   389.,
                    401.,   377.,   731.,   367.,   387.,   680.,   345.,   339.,
                    711.,   388.,   339.,   830.]),
            array([  7.  ,   7.03,   7.06,   7.09,   7.12,   7.15,   7.18,   7.21,
                    7.24,   7.27,   7.3 ,   7.33,   7.36,   7.39,   7.42,   7.45,
                    7.48,   7.51,   7.54,   7.57,   7.6 ,   7.63,   7.66,   7.69,
                    7.72,   7.75,   7.78,   7.81,   7.84,   7.87,   7.9 ,   7.93,
                    7.96,   7.99,   8.02,   8.05,   8.08,   8.11,   8.14,   8.17,
                    8.2 ,   8.23,   8.26,   8.29,   8.32,   8.35,   8.38,   8.41,
                    8.44,   8.47,   8.5 ,   8.53,   8.56,   8.59,   8.62,   8.65,
                    8.68,   8.71,   8.74,   8.77,   8.8 ,   8.83,   8.86,   8.89,
                    8.92,   8.95,   8.98,   9.01,   9.04,   9.07,   9.1 ,   9.13,
                    9.16,   9.19,   9.22,   9.25,   9.28,   9.31,   9.34,   9.37,
                    9.4 ,   9.43,   9.46,   9.49,   9.52,   9.55,   9.58,   9.61,
                    9.64,   9.67,   9.7 ,   9.73,   9.76,   9.79,   9.82,   9.85,
                    9.88,   9.91,   9.94,   9.97,  10.  ]),
            <a list of 100 Patch objects>)
```



```
In [126]: # Histogram with the range of 10 ~ 20 miles
          plt.hist(taxiData.Trip_distance, bins = 50, range = [10,20])
          plt.xlabel('Traveled Trip Distance')
          plt.ylabel('Counts of occurrences')
          plt.title('Histogram of Trip_distance')
          plt.grid(True)
```



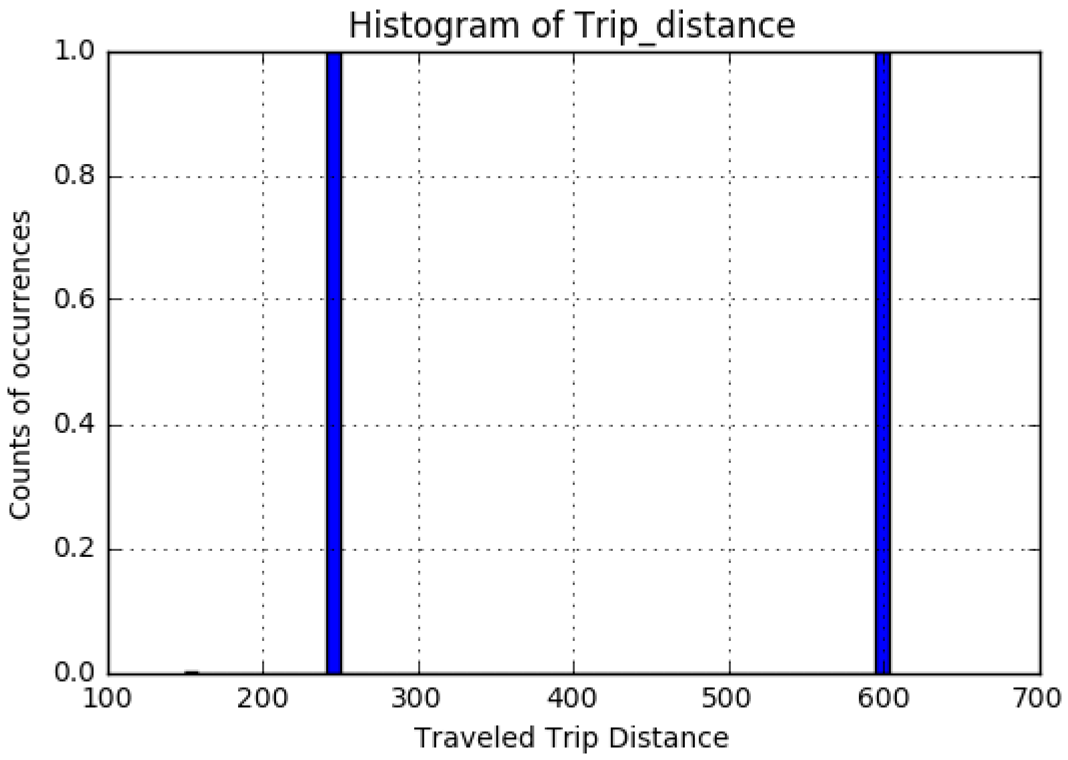
In [127]:
```
# The histogram plots with trip distance values ranging from 50 to 150 m
iles
plt.hist(taxiData.Trip_distance, bins = 20, range = [50,150])
plt.xlabel('Traveled Trip Distance')
plt.ylabel('Counts of occurrences')
plt.title('Histogram of Trip_distance')
plt.grid(True)
```



The histogram plot with the rest of the trip distance from 150 miles to the max value of the Trip_distance counted with the code "taxiData.Trip_distance.max()", which is 603.1 (about 605).

And we could see it is very very very very rare for trip distances to happen with a value larger than 150 miles. with a probability of (2 / 1494926 = 1.33785886391701e-06)

In [128]:
```
# The histogram plot with the rest of the trip distance from 150 miles t
o the max value of the Trip_distance (about 605)
plt.hist(taxiData.Trip_distance, bins = 50, range = [150,taxiData.Trip_d
istance.max()])
plt.xlabel('Traveled Trip Distance')
plt.ylabel('Counts of occurrences')
plt.title('Histogram of Trip_distance')
plt.grid(True)
```



Below is the statistical summary of the Trip_distance variable.

In [23]:
```
# Generate the statistical summary of the Trip_distance variable.
taxiData.Trip_distance.describe()
```

Out[23]:
```
count    1.494926e+06
mean     2.968141e+00
std      3.076621e+00
min      0.000000e+00
25%      1.100000e+00
50%      1.980000e+00
75%      3.740000e+00
max      6.031000e+02
Name: Trip_distance, dtype: float64
```

We can see that the mean is around 3 miles. Yet still many extreme values exists as there are still over thousands of occurrences with a value larger than 13 miles which is the value larger than 3 standard deviations from the mean.

With the data and histogram, I interpret that the population that are not within the densely populated areas of Manhattan mostly reside around a 3 miles radius from the densely populated areas where green taxis are not allowed, and that there may not be a lot of parking spaces a lot of people are willing to take taxis and pay the fees instead of driving to and from the densly populated areas of Manhattan.

(Of course, this is based on my guess that most of these passengers are heading to the populated areas of Manhattan)

Another hypothesis could be that communities outside the densly populated area are mostly spread around a 3 miles radius from places such as downtown, airports, plazas, bars or school.

Question 3

```
In [25]: # Generate all the column names of the data so we would not need to cons
         tantly scroll upwards
         taxiData.columns
```

```
Out[25]: Index(['VendorID', 'lpep_pickup_datetime', 'Lpep_dropoff_datetime',
                'Store_and_fwd_flag', 'RateCodeID', 'Pickup_longitude',
                'Pickup_latitude', 'Dropoff_longitude', 'Dropoff_latitude',
                'Passenger_count', 'Trip_distance', 'Fare_amount', 'Extra', 'MTA
         _tax',
                'Tip_amount', 'Tolls_amount', 'Ehail_fee', 'improvement_surcharg
         e',
                'Total_amount', 'Payment_type', 'Trip_type '],
               dtype='object')
```

The method I came up in order to report the mean and median value of trip distance grouped by hour of day, is to first create a new pandas dataframe, "newdf", object containing only the trip distance values, then create a function to apply to all the values of the pickup time (the 'lpep_pickup_datetime' variable) to extract the 'Hours' value within each row of data, assign those values to a new column in the new pandas dataframe, "newdf", with a new variable name, "Hour_of_day". Lastly, we group the "newdf" dataframe by the variable, "Hour_of_day", to find the mean and median value of trip distance grouped by hour of day.

```
In [32]: # Create a dataframe than contains only the trip distance values
         newdf = pd.DataFrame(taxiData.Trip_distance)
```

Create a function, "getHour", that will be applied to all values of the 'lpep_pickup_datetime' variable to extract the "Hours" values of each row of data

```
In [34]: # Create the getHour function
         def getHour(x):
             """
             A function that would be applied to all values within a pandas Serie
         s object to extract the "Hours" values.
             The values in each data would be in the format of a string "YEAR-MON
         TH-DAY HOUR:MINUTE:SECOND"
             Exp. '2015-09-01 00:02:34'

             :params x: pandas Series object
             :return:
             """
             return x.split()[1].split(':')[0]
```

```
In [36]: # Apply the "getHour" function to all values of the 'lpep_pickup_datetim
         e' variable,
         # then assign those values to a new column in the new pandas dataframe,
          "newdf",
         # with a new variable name, "Hour_of_day".
         newdf["Hour_of_day"] = taxiData.lpep_pickup_datetime.apply(getHour)
```

Finally calculate the mean and median trip distance grouped by hour of day.

The output will show the two columns and twenty four rows of data.

The First column shows rows from the first to the last with values of the "00" Hour of day to the "23" Hour of day.

The Second column shows the mean trip distance traveled within the corresponding Hour of day

In [37]:
```
# The MEAN trip distance grouped by hour of day
newdf.groupby([newdf.Hour_of_day]).Trip_distance.mean()
```

Out[37]:
```
Hour_of_day
00    3.115276
01    3.017347
02    3.046176
03    3.212945
04    3.526555
05    4.133474
06    4.055149
07    3.284394
08    3.048450
09    2.999105
10    2.944482
11    2.912015
12    2.903065
13    2.878294
14    2.864304
15    2.857040
16    2.779852
17    2.679114
18    2.653222
19    2.715597
20    2.777052
21    2.999189
22    3.185394
23    3.191538
Name: Trip_distance, dtype: float64
```

The output will show the two columns and twenty four rows of data.

The First column shows rows from the first to the last with values of the "00" Hour of day to the "23" Hour of day.

The Second column shows the median trip distance traveled within the corresponding Hour of day

```
In [38]: # The MEDIAN trip distance grouped by hour of day
         newdf.groupby([newdf.Hour_of_day]).Trip_distance.median()
```

```
Out[38]: Hour_of_day
         00    2.20
         01    2.12
         02    2.14
         03    2.20
         04    2.36
         05    2.90
         06    2.84
         07    2.17
         08    1.98
         09    1.96
         10    1.92
         11    1.88
         12    1.89
         13    1.84
         14    1.83
         15    1.81
         16    1.80
         17    1.78
         18    1.80
         19    1.85
         20    1.90
         21    2.03
         22    2.20
         23    2.22
         Name: Trip_distance, dtype: float64
```

Output the column names and frist 10 rows of the data to prevent constantly have to scroll up to look at values

In [39]:
```
taxiData.head(10)
```

Out[39]:

	VendorID	lpep_pickup_datetime	Lpep_dropoff_datetime	Store_and_fwd_flag	RateCo
0	2	2015-09-01 00:02:34	2015-09-01 00:02:38	N	5
1	2	2015-09-01 00:04:20	2015-09-01 00:04:24	N	5
2	2	2015-09-01 00:01:50	2015-09-01 00:04:24	N	1
3	2	2015-09-01 00:02:36	2015-09-01 00:06:42	N	1
4	2	2015-09-01 00:00:14	2015-09-01 00:04:20	N	1
5	2	2015-09-01 00:00:39	2015-09-01 00:05:20	N	1
6	2	2015-09-01 00:00:52	2015-09-01 00:05:50	N	1
7	2	2015-09-01 00:02:15	2015-09-01 00:05:34	N	1
8	2	2015-09-01 00:02:36	2015-09-01 00:07:20	N	1
9	2	2015-09-01 00:02:13	2015-09-01 00:07:23	N	1

10 rows × 21 columns

In [40]:
```
taxiData.columns
```

Out[40]:
```
Index(['VendorID', 'lpep_pickup_datetime', 'Lpep_dropoff_datetime',
       'Store_and_fwd_flag', 'RateCodeID', 'Pickup_longitude',
       'Pickup_latitude', 'Dropoff_longitude', 'Dropoff_latitude',
       'Passenger_count', 'Trip_distance', 'Fare_amount', 'Extra', 'MTA
_tax',
       'Tip_amount', 'Tolls_amount', 'Ehail_fee', 'improvement_surcharg
e',
       'Total_amount', 'Payment_type', 'Trip_type '],
      dtype='object')
```

In [41]:
```
# Create a new pandas dataframe object named "bufferdf" to record the on
ly two column values,
# "RateCodeID"&"Fare_amount" for usage of grouping and mean calculation
 of the fare amount for different RateCodeID
bufferdf = taxiData[['RateCodeID','Fare_amount']]
```

In [42]:
```
# Find the transacations that fit the criteria of trips that terminated
 at one of the NYC airports, identified
# by the RateCodeID values equal to 2 or 3 or 4, each representing the a
irports JFK, Newark, Nassau or Westchester
bufferdf.Fare_amount[(bufferdf.Fare_amount==2) |
(bufferdf.Fare_amount==3) | (bufferdf.Fare_amount==4)].apply(int).size
```

Out[42]:
```
46048
```

The Number of the transacations that fit the criteria of trips that terminated at one of the NYC airports:

46048

```
In [43]: # Total numbers of recorded data = 1494926
         taxiData.Trip_distance.size
```

```
Out[43]: 1494926
```

```
In [44]: # Group the values together based on the RateCodeID values and assign th
         e dataframe back to the 'bufferdf' variable.
         bufferdf = bufferdf.groupby([bufferdf.RateCodeID])
```

The MEAN of the fare amount the RateCodeID values equal to 2 or 3 or 4, each representing the airports JFK, Newark, Nassau or Westchester

```
In [45]: # Get the mean values of the airports in NYC with the index number of 2,
          3 and 4.
         bufferdf.Fare_amount.mean().ix[[2,3,4]]
```

```
Out[45]: RateCodeID
         2    49.021871
         3    48.798568
         4    60.164886
         Name: Fare_amount, dtype: float64
```

The MEAN of Fare Amount that the transacations that fit the criteria of trips that terminated at one of the NYC airports.

JFK(2) : 49.021871

Newark(3) : 48.798568

Nassau or Westchester(4) : 60.164886

```
In [46]: # The mean of all the RateCodeID values
         bufferdf.Fare_amount.mean()
```

```
Out[46]: RateCodeID
         1     12.244825
         2     49.021871
         3     48.798568
         4     60.164886
         5     18.082432
         6      2.138889
         99    12.183333
         Name: Fare_amount, dtype: float64
```

By comparing the mean of ALL the fare amount of the RateCodeID to the fare amounts of trips that terminated at one of the NYC airports, identified by the RateCodeID values equal to 2 or 3 or 4. We can see that the average fare amount for trips that terminate at the NYC airports are alot more higher than other termination points.

Furthermore, from the previous findings, we see that a total number of 46048 transacations fit the criteria of trips that terminated at one of the NYC airports, however that is a small amount compared to the whole data which has 1494926, with a portion of 46048/1494926 = 0.03080286248282524 (about only 3 percent).

As we know from the previous historgram plots of the trip distance, we know that the portions of trips with distances over 20 miles is very very small, and as we know that taxi charges more with the amount of distance it covers for each service. Thus, we can speculate that the extreme values of trip distances may result from passengers traveling long distances to one of the NYC airports.

Question 4

In case of messing up the orignal dataframe values. I first create another new dataframe 'taxiData2' to serve as a copy of the original 'taxiData' to perform data manipulations on.

In [67]:
```
# Create another new dataframe 'taxiData2' to serve as a copy of the ori
ginal 'taxiData'
# to perform data manipulations on
taxiData2 = taxiData
```

In order to calculate tip as a percentage of the total fare, we need to divide each row's "Tip_amount" with the "Fare_amount" data value. Thus, we need to check the "Fare_amount" values to make sure if any of it equals zero (0).

In [68]:
```
# Checking if there are any values of Fare_amount that equals to 0
(taxiData2.Fare_amount == 0).any() # This Returns True, meaning there ar
e values that equal to 0
```

Out[68]:
```
True
```

```
In [69]: # Also, we need to check if there were any misrecorded values that may m
         ake a fare amount less than 0.
         (taxiData2.Fare_amount < 0).any() # This Returns True, meaning there are
          values that are negative

Out[69]: True
```

```
In [70]: # Same goes for the Tip_amount, we need to be sure there are no negative
          values recorded. If so, set them to 0.
         (taxiData2.Tip_amount < 0).any() # This Returns True, meaning there are
          values that are negative

Out[70]: False
```

```
In [71]: # Set negative values in Tip_amount to 0
         taxiData2.loc[taxiData2.Tip_amount < 0, "Tip_amount"] = 0
```

```
In [72]: # Double check if there are no more negative values in dataframe's "Tip_
         amount".
         (taxiData2.Tip_amount < 0).any() # This Returns False, proving we have s
         uccessfully changed the values with no negative

Out[72]: False
```

We can see that there are values of "Fare_amount" that equals 0, and as the divisor can not be 0, we must set the original values of "Fare_amount" that equals to 0 to 1 to avoid divisor having the 0 value, we also would need to perform the same procedures for those records that have negative values.

```
In [73]: # In order to set the values that are originally 0 or with negative valu
         es to 1 in "Fare_amount", we need to first
         # find their dataframe indexes within the "Fare_amount" column. Then sel
         ect the "Fare_amount" column, which each
         # value would only be 0 or negative then assign those values as 1.
         taxiData2.loc[taxiData2.Fare_amount <=0, "Fare_amount"] = 1
```

```
In [74]: # Double check if there are no more values of 0 or negative values in da
         taframe's "Fare_amount".
         (taxiData2.Fare_amount <= 0).any() # This Returns False, proving we have
          successfully changed the values with no negative

Out[74]: False
```

The returned result is "False", confirming that we have successfully changed the original values of 0 in the "Fare_amount" to 1, avoiding any potential errors that could arise when we calculate the tip as a percentage of the total fare when we need to divide each row's "Tip_amount" with the Fare_amount" data value.

Now that all the values that may mislead us and that may cause an error are dealt with, we can go along to build a derived variable, named "Tip_percentage_of_total_fare", for tip as a percentage of the total fare by dividing each row's "Tip_amount" with "Fare_amount" data value.

In [75]:
```
# Create variable 'Tip_percentage_of_total_fare' to store the calculatio
ns of tip as a percentage of the total fare
# by dividing each row's "Tip_amount" with "Fare_amount" data value
Tip_percentage_of_total_fare = taxiData.Tip_amount / taxiData2.Fare_amou
nt
```

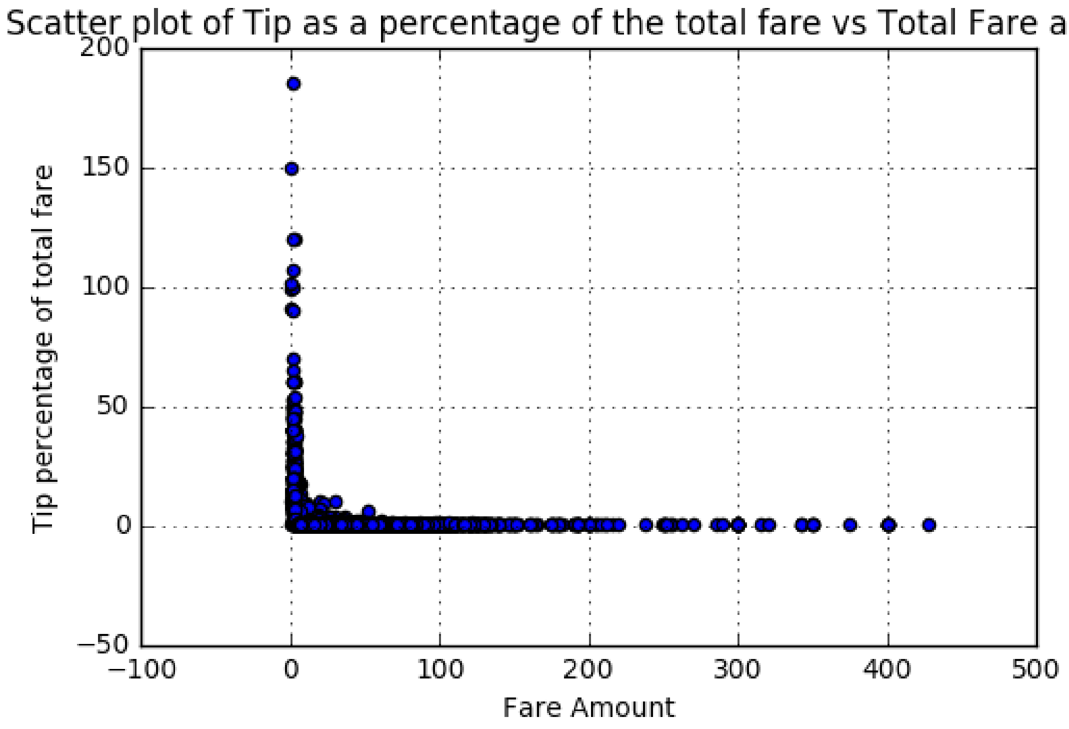
In [76]:
```
# Change the variables that had their values changed back to the origina
l 'taxiData' dataframe.
# Then add the new variable "Tip_percentage_of_total_fare" as a new colu
mn of data with its variable name
# as the column name.
taxiData = taxiData2
taxiData["Tip_percentage_of_total_fare"] = Tip_percentage_of_total_fare
```

Because the tips that are 0 means they are not automatically generated due to the passengers not using credit card as a payment. Thus, in order to get a more accurate mean of the tips, we exclude the Tip_amounts that are 0 (Which also means that the Tip_percentage_of_total_fare would be 0 as well) and calculate the mean of those that are not 0.

In [117]:
```
# Exclude the Tip_amounts that are 0 and calculate the mean of those tha
t are not 0.
taxiData[Tip_percentage_of_total_fare !=
0].Tip_percentage_of_total_fare.mean() # tip percentage mean is around
 0.25
print("MEAN of Tip as a percentage of the total fare = {}"
      .format(taxiData[Tip_percentage_of_total_fare != 0].Tip_percentage
_of_total_fare.mean()))
```

```
MEAN of Tip as a percentage of the total fare = 0.24961780650979515
```

```
In [118]: # create a scatter plot to look at the distribution to decide which pred
          ictive model to use
          plt.scatter(taxiData[Tip_percentage_of_total_fare!=0].Fare_amount,
                      taxiData[Tip_percentage_of_total_fare!=0].Tip_percentage_of_
          total_fare)
          plt.xlabel('Fare Amount')
          plt.ylabel('Tip percentage of total fare')
          plt.title('Scatter plot of Tip as a percentage of the total fare vs Tota
          l Fare amount')
          plt.grid(True)
```



As the scatter plot clearly shows that it is non-linear (more like exponential), I decided to use RandomForest as a prediction model

First, we Create training and testing data, with Tip_percentage_of_total_fare as the dependent variable and Fare_amount as the independent factor.

Use the front 70% of data as training data for the model and the las 30% of data as the testing model.

Currently we are looking at Tip_percentage_of_total_fare that does not have a value of 0 for a more accurate approach.

```
In [79]: # Creating training and testing data for both variables
         xtrain = taxiData[Tip_percentage_of_total_fare!=0].Fare_amount[:int(taxi
         Data.size*0.7)]
         ytrain = taxiData[Tip_percentage_of_total_fare!=0].Tip_percentage_of_tot
         al_fare[:int(taxiData.size*0.7)]
         xtest = taxiData[Tip_percentage_of_total_fare!=0].Fare_amount[int(taxiDa
         ta.size*0.7):]
         ytest = taxiData[Tip_percentage_of_total_fare!=0].Tip_percentage_of_tota
         l_fare[int(taxiData.size*0.7):]
```

```
In [ ]: from sklearn import svm

        x = xtrain
        # reshape the x variable so it could fit into the model
        x = x.reshape(-1,1)
        y = ytrain

        # Using RandomForestClassifier as the prediction model
        svmRegressionModel = svm.SVR(kernel='poly')
        svmRegressionModel.fit(x,y)
```

```
In [ ]: from sklearn.ensemble import RandomForestClassifier

        #Using RandomForestRegressor as the prediction model
        randomForestModel = RandomForestClassifier(n_estimators=1000)
        # Train the model using the training sets
        randomForestModel.fit(xtrain, ytrain)
        # Predict Output
        predicted= randomForestModel.predict(x_test)
```

```
In [ ]:
```

Question 5.

Option E.

I would like to know the usage of credit cards for taxis, such as how often they are used, in what conditions, travel distances, amounts paid, areas traveled to and how many total passengers were there for a person to pay with credit cards.

I hope this may give us some breef insights to the behaviors of people who use their credit cards related to transportation.

In order to do analysis on credit cards, we first must categorize the credit card usage. Thus categorize transactions using credit cards as the number 1, and those with cash as the number 0.

Then we group the data based on this new information, and then look through each variable based on the usage of credit cards for payments to see if there are any interesting findings.

First, create another dataframe names 'Used_credit_card' to store only the Tip_amount in the original data in order to categorize the values to represent '1' as credit card used for payment, and '0' for NOT using a credit card.

In [100]:
```
# create another dataframe names 'Used_credit_card' to store only the Ti
p_amount in the original data
Used_credit_card = pd.DataFrame(taxiData.Tip_amount)

# Filter through the values and categorize the values so that we represe
nt '1' as credit card used for payment,
# and '0' for NOT using a credit card
Used_credit_card.loc[Used_credit_card.Tip_amount > 0, "Tip_amount"] = 1
#Catergorize 1 as credit cards used for payment

# Add this new data back to the original dataframe with a new column nam
e "Used_credit_card"
taxiData["Used_credit_card"] = Used_credit_card

# Add our previous processed data "Hour_of_day" in the 'newdf' dataframe
 to the original dataframe
taxiData["Hour_of_day"] = newdf.Hour_of_day

# Group the data based on "Used_credit_card"
gpCreditCard = taxiData.groupby([taxiData.Used_credit_card])
```

After the completion of data process, we then look at the statistical information of different data based on credit card usage

```
In [102]: # Information of Trip_distance based on credit card usage
          gpCreditCard.Trip_distance.describe()

Out[102]: Used_credit_card
          0.0               count    892194.000000
                            mean          2.605003
                            std           2.885074
                            min           0.000000
                            25%           0.980000
                            50%           1.700000
                            75%           3.200000
                            max         603.100000
          1.0               count    602732.000000
                            mean          3.505676
                            std           3.266777
                            min           0.000000
                            25%           1.320000
                            50%           2.500000
                            75%           4.540000
                            max         246.280000
          Name: Trip_distance, dtype: float64
```

We can see that the distance of a trip of using a credit card has a higher mean and standard deviation than those that pay in cash. It also has a maximum travel distance (extreme value) that is not as large as the max travel distance of the payments using cash. And as we know, the more grounds you cover taking a taxi, the higher the fee gets. Thus, we can make an inference that even though payments of using a credit card has only a slightly higher mean, those who use credit cards for transportation payment are more likely to occur when the fare is higher and are more likely to travel a longer distance, but they do not tend to travel very far off somewhere.

In [103]:
```
# Information of Tip_amount based on credit card usage
gpCreditCard.Tip_amount.describe()
```

Out[103]:
```
Used_credit_card
0.0               count    892194.000000
                  mean          0.000000
                  std           0.000000
                  min           0.000000
                  25%           0.000000
                  50%           0.000000
                  75%           0.000000
                  max           0.000000
1.0               count    602732.000000
                  mean          3.065131
                  std           3.008482
                  min           0.010000
                  25%           1.580000
                  50%           2.320000
                  75%           3.700000
                  max         300.000000
Name: Tip_amount, dtype: float64
```

Combined with the previous Trip_distance information, we can see that the tip amount highly correlates to Trip_distance.

In [104]:
```
# Information of Passenger_count based on credit card usage
gpCreditCard.Passenger_count.describe()
```

Out[104]:
```
Used_credit_card
0.0               count    892194.000000
                  mean          1.370014
                  std           1.037308
                  min           0.000000
                  25%           1.000000
                  50%           1.000000
                  75%           1.000000
                  max           9.000000
1.0               count    602732.000000
                  mean          1.371462
                  std           1.042553
                  min           0.000000
                  25%           1.000000
                  50%           1.000000
                  75%           1.000000
                  max           9.000000
Name: Passenger_count, dtype: float64
```

From the information, we can not tell any difference of how the number of passengers could affect the method of payment. As I initially guessed that as there are more passengers in a taxi, passengers may tend to pay with credit card then have others pay them back through echecks or mobile apps. But we can see now that is not the case. Meaning that the number of passengers would not be a factor to using credit cards for payments.

However, we may guess that perhaps passengers that travel in groups may usually be in the same family, as one member of the family pays for everyone for taking the taxi, and do not need others to pay them back, whether with cash or credit card.

```
In [105]: # Information of Hour_of_day based on credit card usage
          gpCreditCard.Hour_of_day.describe()
```

```
Out[105]: Used_credit_card
          0.0               count     892194
                            unique        24
                            top           18
                            freq       57515
          1.0               count     602732
                            unique        24
                            top           19
                            freq       40511
          Name: Hour_of_day, dtype: object
```

The information above also shows that the Hour of day doesn't affect the payment type of passengers taking a taxi, as their highest frequency payments only differs 1 hour. However, we can see that most transactions for both types of payments happen at night, which we may be able to use to extract important information combined with other data.

All in all, it seems as though the information we have can not give as much insight regarding credit card usage with taxis. However, if there were other data and information such as what kind of stores, landmarks, recreations, housing, communities around the area of Manhattan we could probably make more inference regarding credit card usage and transportation.