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

In [3]:
```
pd.set_option('display.max_columns', 100)
```

In [4]:
```
df_taxi = pd.read_csv('../input/green_tripdata_2015-09.csv')
```

In [5]:
```
type(df_taxi['Dropoff_latitude'][0])
```

Out[5]:
```
numpy.float64
```

In [6]:
```
# df_taxi.columns
```

Creating Trip_time column. Here time is in seconds.

In [7]:
```
t1 = pd.to_datetime(df_taxi['lpep_pickup_datetime'])
t2 = pd.to_datetime(df_taxi['Lpep_dropoff_datetime'])
df_taxi['Trip_time']=(t2-t1).astype('timedelta64[s]')
df_taxi['Trip_time'].head()
```

Out[7]:
```
0      4.0
1      4.0
2    154.0
3    246.0
4    246.0
Name: Trip_time, dtype: float64
```

Question 1

•Programmatically download and load into your favorite analytical tool the trip data for September 2015.

•Report how many rows and columns of data you have loaded.

Answer: Rows: 1494926 and Columns: 21

In [8]:
```
df_taxi.shape
```

Out[8]:
```
(1494926, 22)
```

In [9]:
```
print("Number of Rows are : ",df_taxi.shape[0])
print("Number of Columns are : ",df_taxi.shape[1])
```

```
Number of Rows are :  1494926
Number of Columns are :  22
```

Question 2

• Plot a histogram of the number of the trip distance (“Trip Distance”).

• Report any structure you find and any hypotheses you have about that structure.

Number of 0 distance Trips : 20,592

In [10]:
```
# df_taxi['Trip_distance'].value_counts()
```

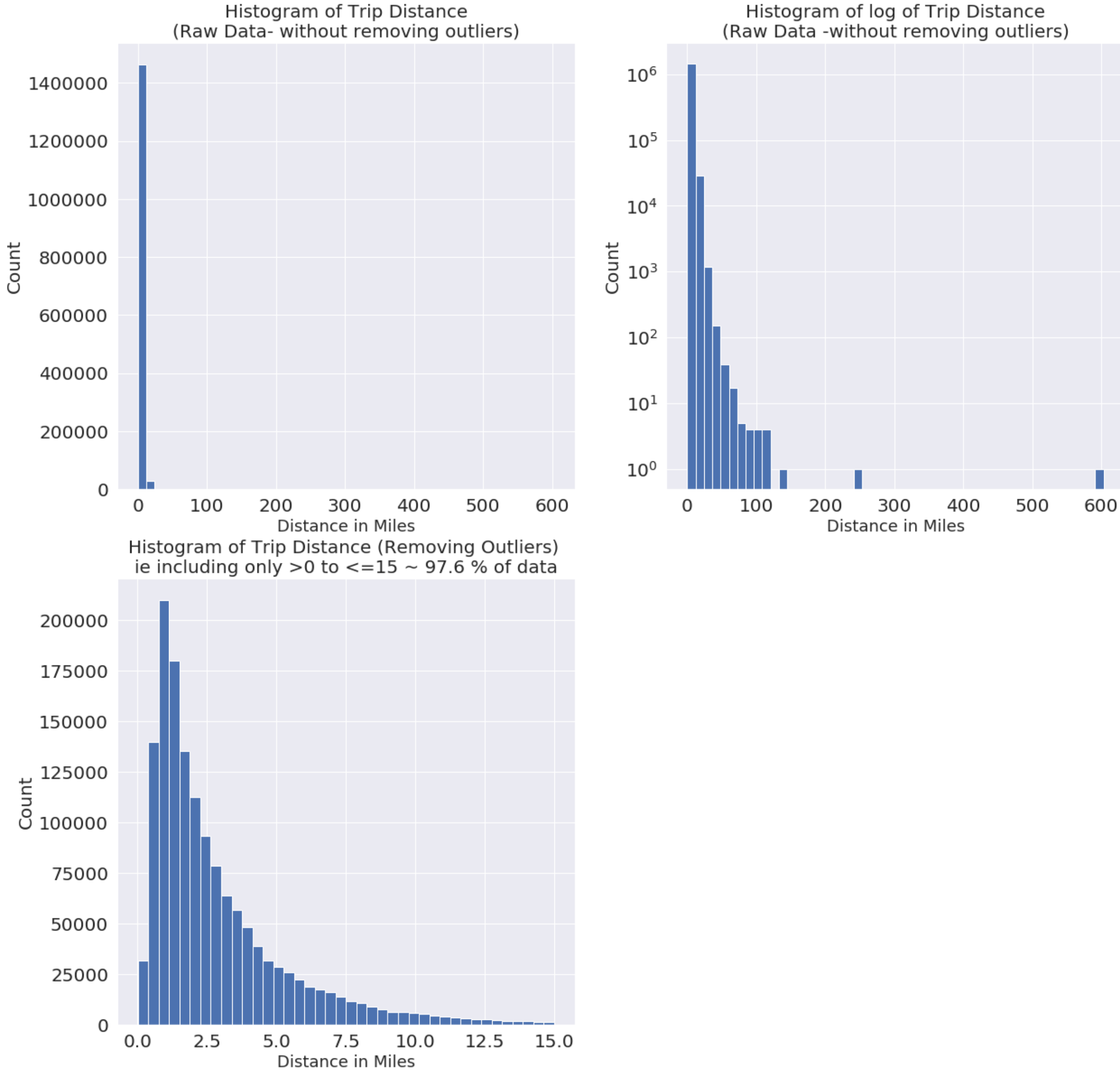
So there were nearly 20,592 trips which travelled no distance at all. For simplicity, lets keep those as an insight and remove those outliers. Also nearly 97.68% of data is between 0 and 15. So we will plot histogram of this much data.

In [11]:
```
(df_taxi[ (df_taxi['Trip_distance']>25) & (df_taxi['Trip_distance']<2500) ].shape[
```

Out[11]:
```
0.0008194385541491686
```

Q2.A • Plot a histogram of the number of the trip distance (“Trip Distance”).

In [12]:
```
import seaborn as sns
import matplotlib.mlab as mlab
import scipy.stats as stat
import matplotlib

sns.set()
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
plt.figure(1,figsize=(20,20))

plt.subplot(221)
plt.hist(df_taxi['Trip_distance'],bins=50)
plt.title('Histogram of Trip Distance\n(Raw Data- without removing outliers)',size
plt.xlabel("Distance in Miles",size=18)
plt.grid(True)
plt.ylabel("Count",size=20)


plt.subplot(222)
plt.hist(df_taxi['Trip_distance'],log=True,bins=50)
plt.title('Histogram of log of Trip Distance\n(Raw Data -without removing outliers
plt.xlabel("Distance in Miles",size=18)
plt.ylabel("Count",size=20)
plt.grid(True)

df_taxi_subset1 = df_taxi[(df_taxi['Trip_distance'] >0) & (df_taxi['Trip_distance'

plt.subplot(223)
plt.hist(df_taxi_subset1['Trip_distance'],bins=40)
plt.title('Histogram of Trip Distance (Removing Outliers) \nie including only >0 t
plt.xlabel("Distance in Miles",size=18)
plt.ylabel("Count",size=20)
plt.grid(True)

plt.show()
```



1. Raw data of Trip_distance is plotted.(Without removing outliers)

2. Logarithm of Trip_distance is plotted.(Without removing outliers)

3. Removing Outliers(trip_distance=0 and trip_distance>15) and then plotting histogram.

Q2.B • Report any structure you find and any hypotheses you have about that structure.

• Insights and Hypothesis:

• Statistical Inference:

- **Trip Distance is clearly Right Skewed ie it has mean greater than the median and it is Assymetric in nature.**

- **The distribution has a "structure of LogNormal Distribution".**

- **There are nearly 20,500 zero distance trips.**

• Qualitative Inferences:

- **Most of the people taking taxi travel less distance ie.**
  - **83% people travel distance less than 5 miles in taxi**
  - **12.6% people travel between 5 and 10 miles**
  - **3.5% people travel between 10 and 25 miles**
  - **0.08% people travel more than 25 miles which is too much rare.**

- **There are 10 trips with > 100 miles distance, out of which extreme two are 603 miles and 246 miles respectively.**

Question 3

• Report mean and median trip distance grouped by hour of day.

• We’d like to get a rough sense of identifying trips that originate or terminate at one of the NYC area airports. Can you provide a count of how many transactions fit this criteria, the average fare, and any other interesting characteristics of these trips.

In [13]:
```
### Creating time (in Hour) column
```

In [14]:
```
import datetime
df_taxi['hour']=[ t.hour for t in pd.to_datetime(df_taxi['lpep_pickup_datetime'])]
df_taxi['hour'].tail()
```

Out[14]:
```
1494921    23
1494922    23
1494923    23
1494924    23
1494925    23
Name: hour, dtype: int64
```

In [15]:
```
df_taxi.columns
```

Out[15]:
```
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_surcharge',
       'Total_amount', 'Payment_type', 'Trip_type ', 'Trip_time', 'hour'],
      dtype='object')
```

Q3.A • Report mean and median trip distance grouped by hour of day.

Answer 3.A : Mean and Median Trip Distance Grouped by Hour of the day are in below Data Frame

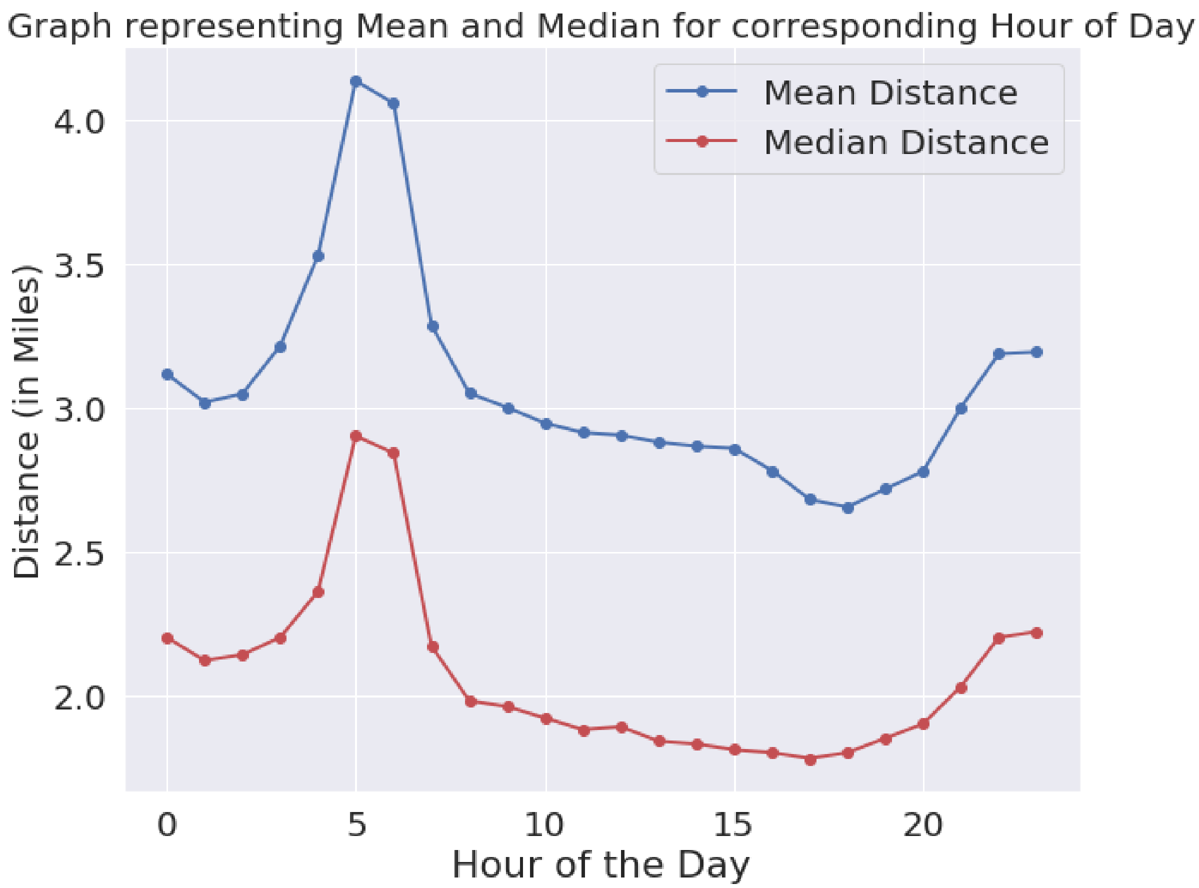
In [16]:
```
df_hour=pd.DataFrame()
df_hour['Hour']=range(0,24)
df_hour['Mean_distance']=df_taxi.groupby('hour').mean()['Trip_distance']
df_hour['Median_distance']=df_taxi.groupby('hour').median()['Trip_distance']
df_hour.head()
```

Out[16]:

	Hour	Mean_distance	Median_distance
0	0	3.115276	2.20
1	1	3.017347	2.12
2	2	3.046176	2.14
3	3	3.212945	2.20
4	4	3.526555	2.36

3.A (Continued) Graphs Representing Mean, Median and Hour of the day.

In [17]:
```
plt.figure(figsize=(10,8))
plt.plot(df_hour['Hour'],df_hour['Mean_distance'],'bo-',linewidth=2,label='Mean Di
plt.plot(df_hour['Hour'],df_hour['Median_distance'],'ro-',linewidth=2,label='Media
plt.legend(loc='upper right',fontsize=20)
plt.title('Graph representing Mean and Median for corresponding Hour of Day',size=
plt.xlabel("Hour of the Day",size=22)
plt.ylabel("Distance (in Miles)",size=20)
plt.grid(True)
```



Inference from above graph:

- **There are more long distance trips in morning 5 AM to 6 AM as compared to whole day.**

- **Also we can see that morning mean and median trips are much longer than as compared to evening trips. This might be due to reason that people might be taking taxi in morning if they do not want to be late to work and in evening they take more public transportation. It might be any other case too.**

Q3.B • We’d like to get a rough sense of identifying trips that originate or terminate at one of the NYC area airports. Can you provide a count of how many transactions fit this criteria, the average fare, and any other interesting characteristics of these trips.

I have taken three airports into consideration and Logitudes and Latitudes are found from google maps. The references are also given below.

Latitude	Longitude	Airport
40.6551573	-73.7800813	John F Kennedy (JFK)
40.6884901	-74.1743015	Newark (EWR) across the river from Manhattan in New Jersey
40.77262	-73.8759	La Guardia (LGA) in the Queens borough of NYC

References: Latitudes and Longitudes from Google Maps Coordinates
https://www.google.com/maps/place/John+F.+Kennedy+International+Airport/@40.6551573,-73.7800
73.7781394?hl=en-US
(https://www.google.com/maps/place/John+F.+Kennedy+International+Airport/@40.6551573,-73.780(
73.7781394?hl=en-US)
https://www.google.com/maps/place/Newark+Liberty+International+Airport/@40.6895232,-74.175240
74.1744624!3m4!1s0x89c252e1c5ec0cef:0xb3f3b437c5d7f286!8m2!3d40.6895314!4d-
74.1744624?hl=en-US
(https://www.google.com/maps/place/Newark+Liberty+International+Airport/@40.6895232,-74.17524(
74.1744624!3m4!1s0x89c252e1c5ec0cef:0xb3f3b437c5d7f286!8m2!3d40.6895314!4d-
74.1744624?hl=en-US) https://www.google.com/maps/search/La+Guardia+
(LGA)+in+the+Queens/@40.77262,-73.8759,1362m/data=!3m1!1e3?hl=en-US
(https://www.google.com/maps/search/La+Guardia+
(LGA)+in+the+Queens/@40.77262,-73.8759,1362m/data=!3m1!1e3?hl=en-US)

Approach: We will take small area around the latitide and longitudes to get an average of all the taxi that dropped off to and picked up from near airport and not only exact coordinates of airport because all taxis pickup and drop around airport (near parking, etc..).

Note that I could also have used RateCodeId = 2 and 3 which involves JFK and Newark, but it was "NEVER MENTIONED" in documentation that those rateCodeID=2 "involves trips from JFK". In short, we had some incomplete informations regarding RateCodeID. So I decided not to use it.

I have used compact Numpy Vectorization to perform operation on whole DataFrame to calculate Which trips involve pickup or drop off to Airport.

In [1]:
```
airportlist2=[]

a=pd.DataFrame()
a['Pickup_latitude']=df_taxi['Pickup_latitude'].copy()
a['Dropoff_latitude']=df_taxi['Dropoff_latitude'].copy()
a['Pickup_longitude']=df_taxi['Pickup_longitude'].copy()
a['Dropoff_longitude']=df_taxi['Dropoff_longitude'].copy()
a.head()
a=np.array(a)
x=((a[:,0] > 40.63899) & (a[:,0] < 40.666573) & (a[:,2] > -73.810) & (a[:,2]< -73.
airportlist2=[1]
airportlist3=['None' for x in range(0,len(a))]
airportlist2=np.multiply(airportlist2,x)
for p in range(0,len(airportlist2)):
    if(airportlist2[p]==1):
        airportlist3[p]='JFK'
x=((a[:,0] > 40.66994) & (a[:,0] < 40.70669) & (a[:,2] > -74.1959) & (a[:,2]<-74.1

airportlist2=[1]
airportlist2=np.multiply(airportlist2,x)
for p in range(0,len(airportlist2)):
    if(airportlist2[p]==1):
        airportlist3[p]='EWR'
x=((a[:,0] > 40.7659) &  (a[:,0]< 40.7830) &  (a[:,2] >-73.889) & (a[:,2]<-73.8500

airportlist2=[1]
airportlist2=np.multiply(airportlist2,x)
for p in range(0,len(airportlist2)):
    if(airportlist2[p]==1):
        airportlist3[p]='LGA'
```

In [19]:
```
df_taxi['Airport']=airportlist3
df_taxi['Airport'].value_counts()
```

Out[19]:
```
None    1454711
LGA       25715
JFK       13775
EWR         725
Name: Airport, dtype: int64
```

Can you provide a count of how many transactions fit this criteria?

Please find below the approximate number of count that indicates number of drop off or pick up at any of the three given airports.

Airport	Number of Trips
LGA	25,715

Airport	Number of Trips
JFK	13,775
EWR	725
None	1,454,711

In [20]:
```
df_taxi['Airport'].value_counts()
```

Out[20]:
```
None    1454711
LGA       25715
JFK       13775
EWR         725
Name: Airport, dtype: int64
```

The average fare and any other interesting characteristics of these trips.

There were many insights from Data that were noticed. They are:

Comparison between Airport Trips and Rest all Trips.

Feature	Airport Trips	Non Airport Trips
Fare Amount	28.00	12.54
Total Amount	33.91	15.03
Tip Amount	3.74	1.23
Trip Distance	8.73	2.96
Trip Time	1700.67	1215.75
Payment type-Credit card	65%	52%

> ## We can clearly see that Average Fare Amount, Total Amount, Tip Amount, Trip Distance and Trip Time are more for airports and less for Non Airport trips.

Airport	Tolls_amount	Trip_distance	Fare_amount	Extra	Tip_%	Trip_time
EWR	9.545710	20.35	75.72	0.193	11.02	3244.25
JFK	0.975228	13.41	40.62	0.202	9.38	2376.71
LGA	1.019293	5.90	19.90	0.263	9.75	1295.01
None	0.094498	2.80	12.11	0.354	6.55	1202.35

It can be inferred from above that EWR is far away from NYC core city which leads to higher Fare, higher Toll, higher Trip time and higher tips. BUT WAIT!!! There is a interesting characteristic here! EWR has lowest

Extra as compared to other trips. So Why is that?

> **It is lowest for EWR and highest for Non Airport trips because Extras include Rush Hour and Over night charges which are not possible mostly on Airport highways because the Rush Hour would effect city roads and not Airport Highways and Over night charges are also less for Airport People because people generally have tendency to go airport and take flight during day time instead of night time. Though sometimes there might be negligible weight of overnight charges for airports but not more than inside city.**

In [21]:
```
# Average Fare of All trips vs Average Fare of Airport Trips
print("Average fare Amount for ALL TRIPS ",df_taxi['Fare_amount'].mean())

print("Average fare Amount for AIRPORT TRIPS ",df_taxi[df_taxi['Airport']!='None']
# Total Fare of All trips vs Total Fare of Airport Trips
print("Average Total Amount for ALL TRIPS ",df_taxi['Total_amount'].mean())

print("Average Total Amount for AIRPORT TRIPS ",df_taxi[df_taxi['Airport']!='None'
# Tip Amount of All trips vs Tip Amount of Airport Trips
print("Tip Amount for ALL TRIPS ",df_taxi['Tip_amount'].mean())

print("Tip Amount for AIRPORT TRIPS ",df_taxi[df_taxi['Airport']!='None']['Tip_amo
# Trip Distance of All trips vs Trip Distance of Airport Trips
print("Trip Distance for ALL TRIPS ",df_taxi['Trip_distance'].mean())

print("Trip Distance for AIRPORT TRIPS ",df_taxi[df_taxi['Airport']!='None']['Trip
# Trip Time of All trips vs Trip Distance of Airport Trips
print("Trip Time for ALL TRIPS ",df_taxi['Trip_time'].mean())

print("Trip Time for AIRPORT TRIPS ",df_taxi[df_taxi['Airport']!='None']['Trip_tim
# Trip Time of All trips vs Trip Distance of Airport Trips
print("Payment Type for ALL TRIPS ",df_taxi['Payment_type'].value_counts())

print("Payment Type for AIRPORT TRIPS ",df_taxi[df_taxi['Airport']!='None']['Payme
# df_taxi.groupby('Airport').mean()['Tolls_amount']
# df_taxi.groupby('Airport').mean()['Trip_distance']
# df_taxi.groupby('Airport').mean()['Fare_amount']
# df_taxi.groupby('Airport').mean()['Extra']
# df_taxi.groupby('Airport').mean()['Tip_%']
# df_taxi.groupby('Airport').mean()['Trip_time']
```

```
Average fare Amount for ALL TRIPS  12.54319751613129
Average fare Amount for AIRPORT TRIPS  28.006853910232493
Average Total Amount for ALL TRIPS  15.032145751981083
Average Total Amount for AIRPORT TRIPS  33.91363073481325
Tip Amount for ALL TRIPS  1.2357267048672937
Tip Amount for AIRPORT TRIPS  3.7494947159020917
Trip Distance for ALL TRIPS  2.9681408511189864
Trip Distance for AIRPORT TRIPS  8.736023623026162
Trip Time for ALL TRIPS  1215.7577853351938
Trip Time for AIRPORT TRIPS  1700.6724107919931
Payment Type for ALL TRIPS  2    783699
1    701287
3      5498
4      4368
5        74
Name: Payment_type, dtype: int64
Payment Type for AIRPORT TRIPS  1    25921
2    14198
4       57
3       38
5        1
Name: Payment_type, dtype: int64
```

Question 4

• Build a derived variable for tip as a percentage of the total fare.

• Build a predictive model for tip as a percentage of the total fare. Use as much of the data as you like (or all of it). Provide an estimate of performance using an appropriate sample, and show your work.

Answer 4.A

I have created new column named 'Tip_%' which will be representing: Tip as Percentage of the 'Total_amount' (which is total fare).

Note that there was another column named Fare_amount which could also have been taken, but for simplicity I have assumed that we need to consider Total_amount column.

In [22]:
```
df_taxi['Tip_%']=100*df_taxi['Tip_amount']/df_taxi['Total_amount']
print("Initially there are {} NA Values ".format(df_taxi['Tip_%'].isna().sum()))
df_taxi['Tip_%'].fillna(0,inplace=True)
df_taxi.drop('Tip_amount',axis=1,inplace=True)
print("We replace all NA values by 0.")
print("Just for confirming now # of NA values are : ",df_taxi['Tip_%'].isna().sum(
```

```
Initially there are 4172 NA Values
We replace all NA values by 0.
Just for confirming now # of NA values are :  0
```

There were 4172 NA Values in Tip_% column created, so I replaced them with 0% which means there was 0% tip.

In [23]:
```
(100*df_taxi['Tip_%'].round().value_counts()/len(df_taxi)).round().iloc[:5]
```

Out[23]:
```
0.0     60.0
17.0    21.0
20.0     5.0
23.0     3.0
13.0     1.0
Name: Tip_%, dtype: float64
```

Insight 4.A :

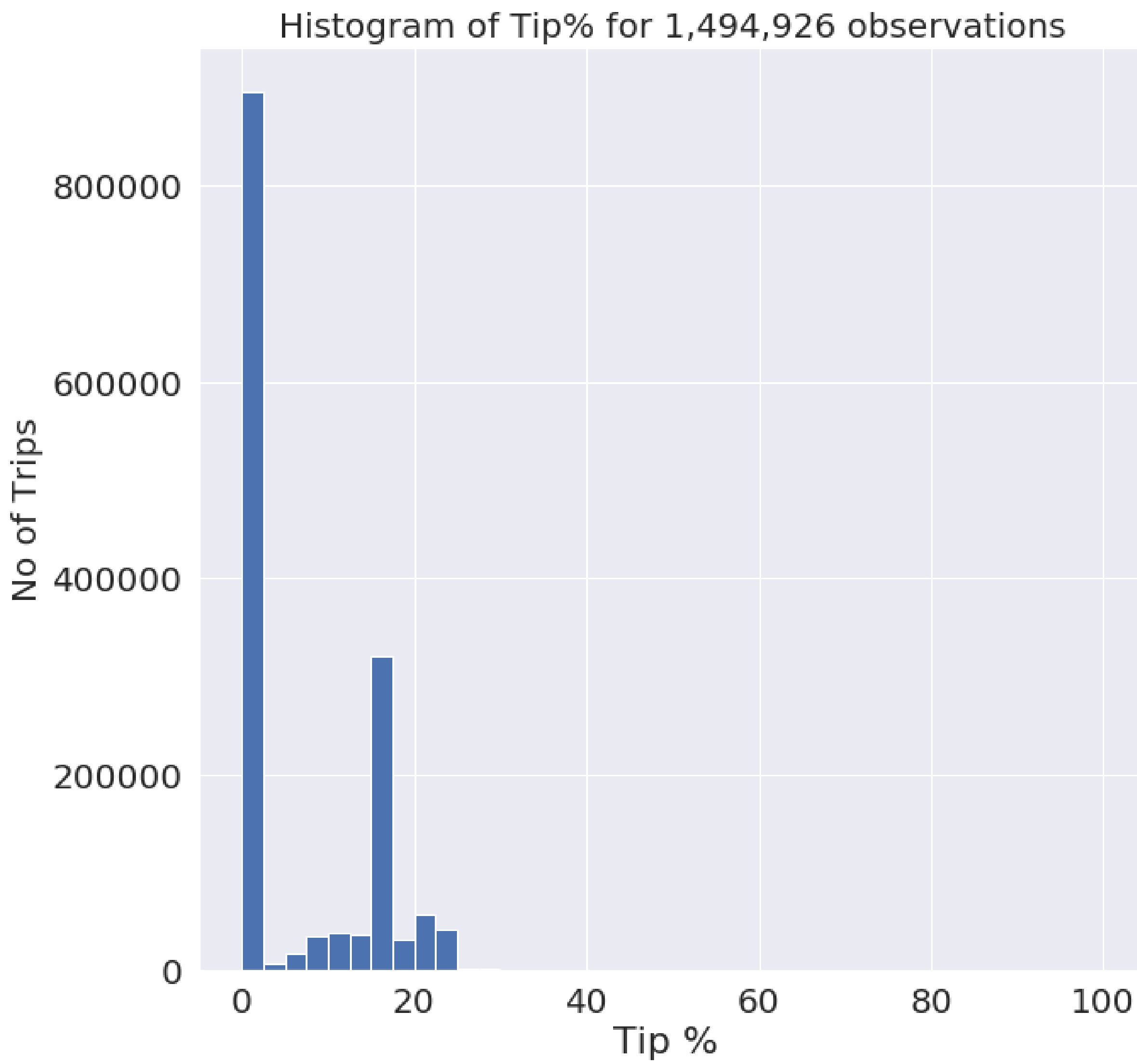
Top 5 Notable Values of Tip %(Rounded)

Almost 60% people give no tip.

Most of the people tend to give 17% or 20% of the tip or No tip.

% of tip	% of People
0.0	60.0
17.0	**21.0**
20.0	**5.0**
23.0	3.0
13.0	1.0
Other	10

In [24]:
```
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
sns.set()
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
plt.figure(figsize=(10,10))
plt.hist(df_taxi['Tip_%'],bins=40)
plt.title('Histogram of Tip% for 1,494,926 observations',size=20)
plt.xlabel("Tip % ",size=22)
plt.ylabel("No of Trips",size=20)
plt.grid(True)
```



• Build a predictive model for tip as a percentage of the total fare. Use as much of the data as you like (or all of it). Provide an estimate of performance using an appropriate sample, and show your work.

1. Data Preprocessing:

Removing Total Amount <=0. Note that we could have replaced them with something other(Mean, Median or Mode), but in our whole data of 1494926, we were having only handfull(6500) which were outliers and we dont want our model to concentrate more on these kind of outliers, so better we remove them.

Dropping 'Ehail_fee' column which was useless (100% NAN Values)

Removing rows with RateCodeID values 99 which is invalid. There were 4 rows in total and i removed it because it was having many other columns invalid value(eg: Trip_distance =0, Passenger Count=0, DropOff Longitude and Latitue both 0). There were only 4 such rows, so I removed them.

Removing rows with 0 passenger count. We removed it because there were only 267 rows in 1494926 observations.

Creating a new column with Trip_distance_Zero and Keeping its value 1 if the distance was 0.

Replacing Trip_Distance having value 0 with mean distance 3.0117792040867934 (which we got by doing mean of Trip_distance values - excluding values with distance 0)

removing Fare_amount ==0 (There were 288 such columns)

In [25]:
```
print("Removing {} rows with Total Amount <=0 ".format(df_taxi[(df_taxi['Total_amo
df_taxi_tips=df_taxi[(df_taxi['Total_amount']>0)]

df_taxi_tips.drop(columns='Ehail_fee',inplace=True)
print("Ehail_fee column with 100% missing data dropped.")

i=df_taxi_tips[(df_taxi_tips['RateCodeID']==99)].index
df_taxi_tips.drop(index=i,axis=0,inplace=True)
print("Rows with RateCodeID = 99 dropped. There were 4 such rows ")


i=df_taxi_tips[(df_taxi_tips['Passenger_count']==0)].index
df_taxi_tips.drop(index=i,axis=0,inplace=True)
print("Rows with Passenger_count = 0 dropped")

print("Mean is :",df_taxi_tips[df_taxi_tips['Trip_distance']>0]['Trip_distance'].m

df_taxi_tips['Trip_distance_Zero']= df_taxi_tips.apply(
    lambda row: 1 if (row['Trip_distance']==0) else 0,
    axis=1
)
print("New column Trip_distance_Zero created.")
df_taxi_tips['Trip_distance'].replace(0,3.0117792040867934,inplace=True)
print("Trip_Distance having values 0, replaced with mean = ",3.0117792040867934)

print("Removing {} rows with Fare Amount <=0 ".format(df_taxi_tips[(df_taxi_tips['
df_taxi_tips=df_taxi_tips[(df_taxi['Fare_amount']>0)]
```

```
Removing 6589 rows with Total Amount <=0

/opt/conda/lib/python3.6/site-packages/pandas/core/frame.py:3697: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/sta
ble/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy)
  errors=errors)

Ehail_fee column with 100% missing data dropped.
Rows with RateCodeID = 99 dropped. There were 4 such rows
Rows with Passenger_count = 0 dropped
Mean is : 3.0117792040867934

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:20: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/sta
ble/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy)
/opt/conda/lib/python3.6/site-packages/pandas/core/generic.py:5890: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/sta
ble/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy)
  self._update_inplace(new_data)
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:27: UserWarning:
 Boolean Series key will be reindexed to match DataFrame index.

New column Trip_distance_Zero created.
Trip_Distance having values 0, replaced with mean =  3.0117792040867934
Removing 288 rows with Fare Amount <=0
```

In [26]:
```
df_taxi_tips=df_taxi_tips.reset_index(drop=True)
```

2. Feature Engineering

> **Creating Day of Month Column (1,2,3,...30). The day of the month might act useful if there is some big occasion for US on that day or there is some good news for NYC people on that day or traffic is more on that day due ot some reasons and people would pay tip according to that situation.**

> **Creating Week Column(1,2,3,4,5- Maximum 5th week can start). The Week currently running might also affect the tips because there might be human intuition to give more tip the week they receive salary or Week of Public Holiday when their mood is very good.**

> **Creating Day of Week Column(1,2,3,4,5,6,7) which might be usefull if we want to predict tip because people might be willing to pay high tip on Weekends as they would be in joyfull mood.**

> **Creating Speed from distance in mile and time in seconds. Replacing Nan values with 0.**

> **Most Important: Creating 36 sub-regions in NYC from where a taxi is picking up or where taxi is dropping off. This is very usefull because there might be regions in NYC where people are rich as compared to other regions. Say from this 36 regions, there are 7 very Porshe Regions and people who are picked up from that locations would give more tips as compared to other people from other locations. Thus, Tips can be very highly dependent on this feature set.**

Treating Pickup Sub-Regions as a Category. One Hot Encoding of this 36 Sub-Regions. We cannot treat SubRegions as a continuous variable because it might not be beneficial while we are using ML algorithms. Intuition behind this: ML Algorithm treats Sub Region as a Continuous quantity ie. If subregion is 1,2,...36, ML algorithm with treat subregion 1 as smaller than subregion 36 because it treats it numerically. Thus we want to treat them as categorical variable so it extracts good features from it.

Treating Week Column and Day of Week column as a Category because they would be more usefull if we treat them as a category because we DO NOT want ML algorithm to infer that 1 as Monday is less than 2 as Tuesday and Sunday-7 is greatest. We want Monday, Tuesday,...Sunday to be treated as categories.

Creating another 36 subregions for DropOff Locations. Initially I tried keeping Pickup and Dropoff as in same group, but model was performing better when I tried it doing different groups. Thus, we now have 36 regions of Pickup and 36 regions of Drop Off.

Treating: 1. VendorID 2. RateCodeID 3. Store_and_fwd_flag 4. Payment_type 5. Hour of Day and 6. Trip_type as Categorical Variables. So I did One Hot Encoding of all these columns. Now, the reason of treating them as categories is same as previous one.

Removing Object Columns (Pickup Date, Date and Time, Date and DropOff Date) which cannot be fitted into ML Model while training.

In [27]:

```
import datetime
timedf = pd.DataFrame()
timedf['Day']=[ t.day for t in pd.to_datetime(df_taxi_tips['lpep_pickup_datetime']]
print("Day column created")
timedf['DateandTime']=df_taxi_tips['lpep_pickup_datetime']
timedf['Date']=[ t.date() for t in pd.to_datetime(df_taxi_tips['lpep_pickup_dateti
print("Date Column Created")
baseweek=datetime.datetime(2015,9,1).isocalendar()[1]

timedf['Week']=timedf['Date'].apply(lambda x : x.isocalendar()[1])-baseweek+1
print("Week of Month Column Created")
timedf['DayofWeek']=timedf['Date'].apply(lambda x : x.isocalendar()[2])
print("Day of Week Column Created")

timedf['Speed']=df_taxi_tips.Trip_distance/(df_taxi_tips.Trip_time/3600)
print('Initially there are {} null values in Speed column'.format(timedf.Speed.isn
timedf['Speed'].fillna(0,inplace=True)
print('Now there are {} null values in Speed column'.format(timedf.Speed.isna().su
d={}
d[40.56]={-73.98:1}
d[40.62]={-73.98:2}
d[40.65]={-74.01:3,-73.98:4,-73.95:5,-73.92:6}
d[40.68]={-74.01:7,-73.98:8,-73.95:9,-73.92:10 ,-73.83:11}
d[40.71]={-73.98:12,-73.95:13,-73.89:14,-73.86:15,-73.83:16}
d[40.74]={-73.98:17,-73.95:18,-73.92 :19,-73.89:20,-73.86:21,-73.83:22 }
d[40.77]={-73.98:23,-73.95:24,-73.92:25}
d[40.80]={-73.98:26,-73.95:27,-73.92:28}
d[40.83]={-73.95:29,-73.92:30,-73.89:31,-73.86:32}
d[40.86]={-73.95:33,-73.92:34,-73.89:35}


timedf["Pickup_latitude"] = df_taxi_tips.Pickup_latitude
timedf["Pickup_longitude"] = df_taxi_tips.Pickup_longitude
step = 0.03
to_bin = lambda x: np.floor( x/ step) * step
timedf["latbin"] = df_taxi_tips.Pickup_latitude.map(to_bin)
timedf["lonbin"] = df_taxi_tips.Pickup_longitude.map(to_bin)

timedf.head()

arr=np.array(timedf)
group=[]
for x in range(0,len(timedf)):
    if(arr[:,8][x] in d.keys()):
        if(arr[:,9][x] in d[arr[:,8][x]].keys()):
            g=d[arr[:,8][x]][arr[:,9][x]]
            group.append(g)
        else:
            group.append(0)
    else:
        group.append(0)
timedf['Group1']=group
timedf.Group1.value_counts()

timedf["Dropoff_latitude"] = df_taxi_tips.Dropoff_latitude
timedf["Dropoff_longitude"] = df_taxi_tips.Dropoff_longitude
```

```
step = 0.03
to_bin = lambda x: np.floor( x/ step) * step
timedf["latbin2"] = df_taxi_tips.Dropoff_latitude.map(to_bin)
timedf["lonbin2"] = df_taxi_tips.Dropoff_longitude.map(to_bin)
groups = timedf.groupby(("latbin2", "lonbin2"))
timedf.head()

arr=np.array(timedf)
group=[]
for x in range(0,len(timedf)):
    if(arr[:,13][x] in d.keys()):
        if(arr[:,14][x] in d[arr[:,13][x]].keys()):
            g=d[arr[:,13][x]][arr[:,14][x]]
            group.append(g)
        else:
            group.append(0)
    else:
        group.append(0)
timedf['Group2']=group
timedf.Group2.value_counts()
timedf=timedf.reset_index(drop=True)
df_taxi_tips=pd.concat([df_taxi_tips,timedf],axis=1)
```

```
Day column created
Date Column Created
Week of Month Column Created
Day of Week Column Created
Initially there are 0 null values in Speed column
Now there are 0 null values in Speed column

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:61: FutureWarnin
g: Interpreting tuple 'by' as a list of keys, rather than a single key. Use 'b
y=[...]' instead of 'by=(...)'. In the future, a tuple will always mean a sing
le key.
```

In [28]:
```
# df_taxi_tips.head()
```

In [29]:
```
df_taxi_tips['Group1'].isna().sum()
```

Out[29]:
```
0
```

In [30]:
```
_tips.shape
ckup=df_taxi_tips[['Group1','Group2','Week','DayofWeek','Airport','VendorID','RateC
```

In [31]:
```
pd.get_dummies(df_taxi_tips['Group2'])
df_taxi_tips = pd.concat([df_taxi_tips,pd.get_dummies(df_taxi_tips['Group2'], pref
print("One Hot Encoding of Group2 done and shape is :",df_taxi_tips.shape[1])
df_taxi_tips = pd.concat([df_taxi_tips,pd.get_dummies(df_taxi_tips['Group1'], pref
print("One Hot Encoding of Group1 done and shape is :",df_taxi_tips.shape[1])
df_taxi_tips = pd.concat([df_taxi_tips,pd.get_dummies(df_taxi_tips['DayofWeek'], p
df_taxi_tips = pd.concat([df_taxi_tips,pd.get_dummies(df_taxi_tips['Airport'], pre
df_taxi_tips = pd.concat([df_taxi_tips,pd.get_dummies(df_taxi_tips['hour'], prefix
df_taxi_tips = pd.concat([df_taxi_tips,pd.get_dummies(df_taxi_tips['Week'], prefix
df_taxi_tips = pd.concat([df_taxi_tips,pd.get_dummies(df_taxi_tips['VendorID'], pr
df_taxi_tips = pd.concat([df_taxi_tips,pd.get_dummies(df_taxi_tips['RateCodeID'],
df_taxi_tips = pd.concat([df_taxi_tips,pd.get_dummies(df_taxi_tips['Store_and_fwd_
df_taxi_tips = pd.concat([df_taxi_tips,pd.get_dummies(df_taxi_tips['Trip_type '],
df_taxi_tips = pd.concat([df_taxi_tips,pd.get_dummies(df_taxi_tips['Payment_type']
df_taxi_tips.dtypes[df_taxi_tips.dtypes=='object']
df_taxi_tips.drop(['lpep_pickup_datetime', 'Lpep_dropoff_datetime','Date', 'Datean
df_taxi_tips.Speed[(df_taxi_tips.Speed<240)&(df_taxi_tips.Speed>0)].mean()
indices = df_taxi_tips[(df_taxi_tips.Speed>240)].index
df_taxi_tips.loc[indices,'Speed']=13.32423148992669
```

```
One Hot Encoding of Group2 done and shape is : 70
One Hot Encoding of Group1 done and shape is : 100
```

In [32]:
```
taxi_backup=pd.concat([df_taxi_tips,taxi_backup],axis=1)
# taxi_backup is used later
# df_taxi_tips.head()
```

In [33]:
```
# taxi_backup.head()
```

Loading Libraries to use later

In [34]:
```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import datetime
import time
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_predict, GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import (roc_curve, auc, accuracy_score)
```

Model Building

In [34]:
```
df_taxi_tips.corr()['Tip_%']
```

Out[34]:
```
Pickup_longitude          -0.002563
Pickup_latitude           -0.004883
Dropoff_longitude         -0.018082
Dropoff_latitude           0.009585
Passenger_count            0.001373
Trip_distance              0.095788
Fare_amount                0.085370
Extra                      0.012975
MTA_tax                    0.066664
Tolls_amount               0.043230
improvement_surcharge      0.065931
Total_amount               0.230755
Trip_time                 -0.011282
hour                       0.011478
Tip_%                      1.000000
Trip_distance_Zero        -0.006107
Day                        0.023592
Speed                      0.040409
Pickup_latitude           -0.004883
Pickup_longitude          -0.002563
latbin                    -0.004760
lonbin                    -0.002704
Dropoff_latitude           0.009585
Dropoff_longitude         -0.018082
latbin2                    0.009664
lonbin2                   -0.018158
Group2_0.0                 0.110138
Group2_2.0                 0.006359
Group2_4.0                 0.042346
Group2_5.0                -0.013654
                             ...
hour_23.0                  0.016474
hour_nan                        NaN
Week_1.0                  -0.022723
Week_2.0                  -0.001339
Week_3.0                   0.009844
Week_4.0                   0.009175
Week_5.0                   0.005688
Week_nan                        NaN
VendorID_1.0               0.006277
VendorID_2.0              -0.006277
VendorID_nan                    NaN
RateCodeID_1.0             0.063264
RateCodeID_2.0            -0.004626
RateCodeID_3.0            -0.003294
RateCodeID_4.0            -0.000616
RateCodeID_5.0            -0.066901
RateCodeID_6.0            -0.003497
RateCodeID_nan                  NaN
Store_and_fwd_flag_N       0.010237
Store_and_fwd_flag_Y      -0.010237
Store_and_fwd_flag_nan          NaN
Trip_type_1.0              0.066439
Trip_type_2.0             -0.066439
```

```
Trip_type_nan                  NaN
Payment_type_1.0          0.799281
Payment_type_2.0         -0.791298
Payment_type_3.0         -0.038804
Payment_type_4.0         -0.036314
Payment_type_5.0         -0.005246
Payment_type_nan               NaN
Name: Tip_%, Length: 154, dtype: float64
```

In [35]:

```
#Creating Train Test split from here. X_test and y_test is TEST SET and they will
X_train, X_test, y_train, y_test = train_test_split(np.array(df_taxi_tips.drop('Ti

# X_train and y_train will be further splitted into X_train_new and X_valid and sa
X_train_new, X_validation, y_train_new, y_validation = train_test_split(X_train,y_

print('Training Features Shape:', X_train.shape)
print('Training Label Shape:', y_train.shape)
print('Testing Features Shape:', X_test.shape)
print('Testing Label Shape:', y_test.shape)
```

```
Training Features Shape: (1413392, 153)
Training Label Shape: (1413392,)
Testing Features Shape: (74390, 153)
Testing Label Shape: (74390,)
```

In [36]:

```
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor(max_depth=10, n_estimators=100,random_state=42, ver
forest.fit( X_train_new, y_train_new)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed: 33.7min finished
```

Out[36]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
           oob_score=False, random_state=42, verbose=True,
           warm_start=False)
```

In [37]:

```
from sklearn.metrics import mean_squared_error
ypred2=forest.predict(X_test)
error2 = mean_squared_error(ypred2, y_test)
print("Our MSE(Mean Squared Error ) is : ",error2)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.

Our MSE(Mean Squared Error ) is :  15.045183080201236

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.3s finished
```

Type *Markdown* and LaTeX: α^2

In [38]:

```
#Creating Train Test split from here. X_test and y_test is TEST SET and they will
X_train, X_test, y_train, y_test = train_test_split(np.array(df_taxi_tips.drop('Ti

# X_train and y_train will be further splitted into X_train_new and X_valid and sa
X_train_new, X_validation, y_train_new, y_validation = train_test_split(X_train,y_

# Creating Evaluation Set using X_validation and y_validation which we created in
validation_set = [ ( X_validation, y_validation ) ]

#This parameters are found by Hyperparameter tuning and still further better could
param = { "silent":False, "max_depth":10, "n_estimators":500}

# Setting up the XGBRegressor model and parameters as mentioned above are passed.
xgbmodel = xgb.XGBRegressor( **param )

# We will set evaluation set/ validation set to eval_set parameter
# early stopping =50 is highly prefered in order to prevent overfitting.
# Also we have used rmse as our metric which is Root mean squared error.
xgbmodel.fit( X_train_new, y_train_new, eval_metric="rmse",early_stopping_rounds=5
```

```
s, 862 extra nodes, 0 pruned nodes, max_depth=10
[348]	validation_0-rmse:0.238203
[02:38:35] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 688 extra nodes, 0 pruned nodes, max_depth=10
[349]	validation_0-rmse:0.238169
[02:38:48] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 1066 extra nodes, 0 pruned nodes, max_depth=10

[350]	validation_0-rmse:0.23802
[02:39:00] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 1414 extra nodes, 0 pruned nodes, max_depth=10
[351]	validation_0-rmse:0.23783
[02:39:13] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 1122 extra nodes, 0 pruned nodes, max_depth=10
[352]	validation_0-rmse:0.237682
[02:39:27] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 1260 extra nodes, 0 pruned nodes, max_depth=10
[353]	validation_0-rmse:0.237495
[02:39:39] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 1292 extra nodes, 0 pruned nodes, max_depth=10
[354]	validation_0-rmse:0.237367
```

In [39]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
ypred=xgbmodel.predict(X_test)
error = mean_absolute_error(ypred, y_test)
print("Our XGBOOST MAE(Mean Absolute Error ) is : ",error)
```

```
Our XGBOOST MAE(Mean Absolute Error ) is :  0.050911784915922034
```

In [40]:
```
from sklearn.metrics import mean_squared_error
ypred=xgbmodel.predict(X_test)
error = mean_squared_error(ypred, y_test)
print("Our XGBOOST MSE(Mean Squared Error ) is : ",error)
```

```
Our XGBOOST MSE(Mean Squared Error ) is :  0.053042014532034186
```

In [41]:
```
resultdf=pd.DataFrame()
resultdf['y_test']=y_test
resultdf['ypred']=ypred
resultdf['y_test-ypred']=y_test-ypred
resultdf
```

Out[41]:

	y_test	ypred	y_test-ypred
0	0.000000	-0.001933	0.001933
1	0.000000	-0.000533	0.000533
2	0.000000	0.000164	-0.000164
3	0.000000	-0.001161	0.001161
4	16.666667	16.660606	0.006060
5	19.745223	19.757586	-0.012363
6	14.705882	14.674166	0.031717
7	0.000000	-0.001365	0.001365
8	0.000000	0.000686	-0.000686
9	32.258065	32.116093	0.141972
10	16.666667	16.746353	-0.079686
11	0.000000	-0.004377	0.004377
12	0.000000	-0.076782	0.076782
13	20.000000	19.956116	0.043884
14	20.000000	19.879057	0.120943
15	0.000000	0.001350	-0.001350
16	0.000000	0.000311	-0.000311
17	0.000000	0.001003	-0.001003
18	16.571429	16.556349	0.015080
19	0.000000	0.000967	-0.000967
20	16.618076	16.646595	-0.028519
21	16.666667	16.663891	0.002776
22	16.666667	16.662252	0.004414
23	0.000000	0.000131	-0.000131
24	0.000000	0.001444	-0.001444
25	0.000000	0.000523	-0.000523
26	0.000000	0.003728	-0.003728
27	7.000000	7.229422	-0.229422
28	16.619718	16.611153	0.008566
29	0.000000	0.001412	-0.001412
...	...	...	...
74360	0.000000	-0.000432	0.000432

	y_test	ypred	y_test-ypred
74361	0.000000	0.001169	-0.001169
74362	0.000000	0.003976	-0.003976
74363	0.000000	-0.000620	0.000620
74364	0.000000	0.000244	-0.000244
74365	0.000000	0.000568	-0.000568
74366	20.000000	19.977346	0.022654
74367	13.076923	13.227209	-0.150286
74368	0.000000	-0.001216	0.001216
74369	16.619718	16.668232	-0.048514
74370	16.666667	16.661812	0.004855
74371	16.666667	16.818777	-0.152110
74372	19.965577	19.947577	0.018000
74373	16.666667	16.651121	0.015546
74374	16.666667	16.636814	0.029853
74375	13.409962	13.355430	0.054532
74376	16.666667	16.637623	0.029044
74377	0.000000	0.000932	-0.000932
74378	16.666667	16.653013	0.013653
74379	0.000000	-0.000899	0.000899
74380	16.666667	16.662125	0.004542
74381	0.000000	-0.000068	0.000068
74382	13.274336	13.679052	-0.404716
74383	0.000000	0.001348	-0.001348
74384	0.000000	-0.013423	0.013423
74385	23.076923	23.161402	-0.084479
74386	16.666667	16.656908	0.009759
74387	19.971671	19.910744	0.060928
74388	0.000000	-0.000646	0.000646
74389	0.000000	0.155815	-0.155815

74390 rows × 3 columns

In [42]:

```
import math

print("RMSE :",math.sqrt((resultdf['y_test-ypred']**2).mean()))
```

```
RMSE : 0.230308520320114
```

Printing feature_importances

In [43]:
```
print(len(xgbmodel.feature_importances_))
```

```
153
```

In [44]:
```
xgbmodel.feature_importances_
```

Out[44]:
```
array([1.2228319e-01, 7.6405019e-02, 5.3895041e-02, 7.4943647e-02,
       5.4284511e-03, 3.6148317e-02, 1.6809243e-01, 3.7547648e-02,
       5.7731150e-03, 2.2361772e-02, 4.9217958e-03, 2.0096299e-01,
       4.1604340e-02, 1.7891485e-02, 5.0320884e-04, 1.7057400e-02,
       1.9949127e-02, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 1.2959350e-03, 1.8267169e-04, 3.9980974e-04,
       3.6879003e-04, 2.0679814e-04, 5.9971464e-04, 5.8937469e-04,
       2.5505104e-04, 1.8267169e-04, 4.1014966e-04, 2.2403132e-04,
       2.3781786e-04, 3.3777030e-04, 1.9301160e-04, 6.1005453e-04,
       3.7568330e-04, 5.3078192e-04, 3.2398375e-04, 3.6534338e-04,
       2.7573085e-05, 1.9094362e-03, 2.3712853e-03, 1.4751601e-03,
       2.3574987e-03, 7.4102666e-04, 4.0670301e-04, 2.8951740e-04,
       3.8257657e-04, 4.4461602e-04, 2.3437123e-04, 0.0000000e+00,
       1.0374374e-03, 1.5854524e-04, 4.6184918e-04, 3.1709048e-04,
       1.9301160e-04, 5.5835501e-04, 3.3432367e-04, 1.6543851e-04,
       1.7233178e-04, 6.2384107e-04, 2.7228423e-04, 3.0675059e-04,
       2.7228423e-04, 1.0339907e-04, 4.9286889e-04, 7.4102666e-04,
       4.9976219e-04, 3.9980974e-04, 3.0675059e-04, 1.1029234e-04,
       9.8918448e-04, 1.7750174e-03, 6.5141416e-04, 9.2714501e-04,
       4.3082947e-04, 4.1014966e-04, 2.6194431e-04, 2.9985732e-04,
       1.9645823e-04, 1.3786543e-04, 0.0000000e+00, 1.3579745e-03,
       1.6371519e-03, 1.7715708e-03, 1.6061323e-03, 1.8473967e-03,
       1.7715708e-03, 1.4544802e-03, 0.0000000e+00, 2.4815777e-04,
       1.5165197e-04, 4.1359628e-04, 3.2053713e-04, 0.0000000e+00,
       0.0000000e+00, 4.2738282e-04, 4.0670301e-04, 3.9980974e-04,
       4.8597564e-04, 3.7912992e-04, 5.7558814e-04, 6.6175405e-04,
       7.0656033e-04, 8.4097910e-04, 8.1685267e-04, 7.8583293e-04,
       6.9277378e-04, 7.5825985e-04, 9.5471810e-04, 9.0646523e-04,
       9.9263107e-04, 1.1270499e-03, 1.1477297e-03, 9.9263107e-04,
       9.2369836e-04, 8.8578538e-04, 6.6520070e-04, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 1.5199664e-03, 1.8025904e-03,
       1.2338955e-03, 0.0000000e+00, 0.0000000e+00, 5.7352018e-03,
       0.0000000e+00, 0.0000000e+00, 1.7715708e-03, 1.7922506e-04,
       7.9272620e-05, 2.8607078e-04, 1.1890894e-03, 0.0000000e+00,
       0.0000000e+00, 3.5500349e-04, 0.0000000e+00, 0.0000000e+00,
       5.6869490e-04, 0.0000000e+00, 0.0000000e+00, 1.7881146e-02,
       2.0335150e-04, 5.5146171e-05, 3.2053713e-04, 0.0000000e+00,
       0.0000000e+00], dtype=float32)
```

In [46]:
```
d={}
for i in range(0,153):
    d[df_taxi_tips.columns[i]]=xgbmodel.feature_importances_[i]
    print(d[df_taxi_tips.columns[i]])
```

In [47]:
```
d=sorted(d.items(),key= lambda value: value[1],reverse=True)
d
```

```
('lonbin', 0.0),
('latbin2', 0.0),
('Group2_35.0', 0.0),
('Group1_35.0', 0.0),
('DayofWeek_7.0', 0.0),
('Airport_None', 0.0),
('Airport_nan', 0.0),
('hour_22.0', 0.0),
('hour_23.0', 0.0),
('hour_nan', 0.0),
('Week_4.0', 0.0),
('Week_5.0', 0.0),
('VendorID_1.0', 0.0),
('VendorID_2.0', 0.0),
('RateCodeID_5.0', 0.0),
('RateCodeID_6.0', 0.0),
('Store_and_fwd_flag_N', 0.0),
('Store_and_fwd_flag_Y', 0.0),
('Trip_type_1.0', 0.0),
('Trip_type_2.0', 0.0),
```

Code for Parameter Tuning which could be used to tune Hyper Parameters.

In [48]:
```
# import xgboost as xgb
# xgb_model =  xgb.XGBRegressor()
# parameters = {'objective':['reg:linear'],
#               'learning_rate': [0.01,0.05,0.1,0.2,0.3,0.5],
#               'max_depth': [6,8,10,12],
#               'min_child_weight': [8,10,12],
#               'silent': [1],
#               'subsample': [0.8],
#               'colsample_bytree': [0.7],
#               'n_estimators': [5,50,100,500,1000],
#               'seed': [42]}


# clf = GridSearchCV(xgb_model, parameters,cv=5,scoring='neg_mean_squared_error',v

# clf.fit(X_train, y_train)

# clf.grid_scores_
```

Question 5 : Choose only one of these options to answer for Question 5. There is no preference as to which one you choose. Please select the question that you feel your particular skills and/or

expertise are best suited to. If you answer more than one, only the first will be scored.

• Option A: Distributions

• Build a derived variable representing the average speed over the course of a trip.

• Can you perform a test to determine if the average trip speeds are materially the same in all weeks of September? If you decide they are not the same, can you form a hypothesis regarding why they differ?

• Can you build up a hypothesis of average trip speed as a function of time of day?

5.A.1 • Build a derived variable representing the average speed over the course of a trip.

Answer 5.A.1

• I have already build a column 'Speed' that was derived in Feature Engineering Section. It was derived from distance and time varaible. We had distance in Miles and time in Seconds. So Speed was derived by

$$Speed = \frac{Distance(inMiles)}{Duration(inseconds) * (3600)}$$

Thus, our derived varaible

$$Speed = \frac{Distance(inMiles)}{Duration(inHour)}$$

Now, I took mean of all Speed Values which were >0 and <240 and that mean turned out to be 13.324 and I replaced 1. Nan, 2. float('inf') 3. NULL and 4. Speed >240 values with our mean value 13.324 which would be our best estimate as we have very much large amount of data and Mean imputation turns out to be best in such cases.

In [35]:
```
df_taxi_tips['Speed'].head()
```

Out[35]:
```
0    13.324231
1    13.324231
2    13.792208
3    10.829268
4     8.926829
Name: Speed, dtype: float64
```

Question 5.A.2

• Can you perform a test to determine if the average trip speeds are materially the same in all weeks of September? If you decide they are not the same, can you form a hypothesis regarding why they differ?

Lets have a look at average speed of taxi in all weeks.

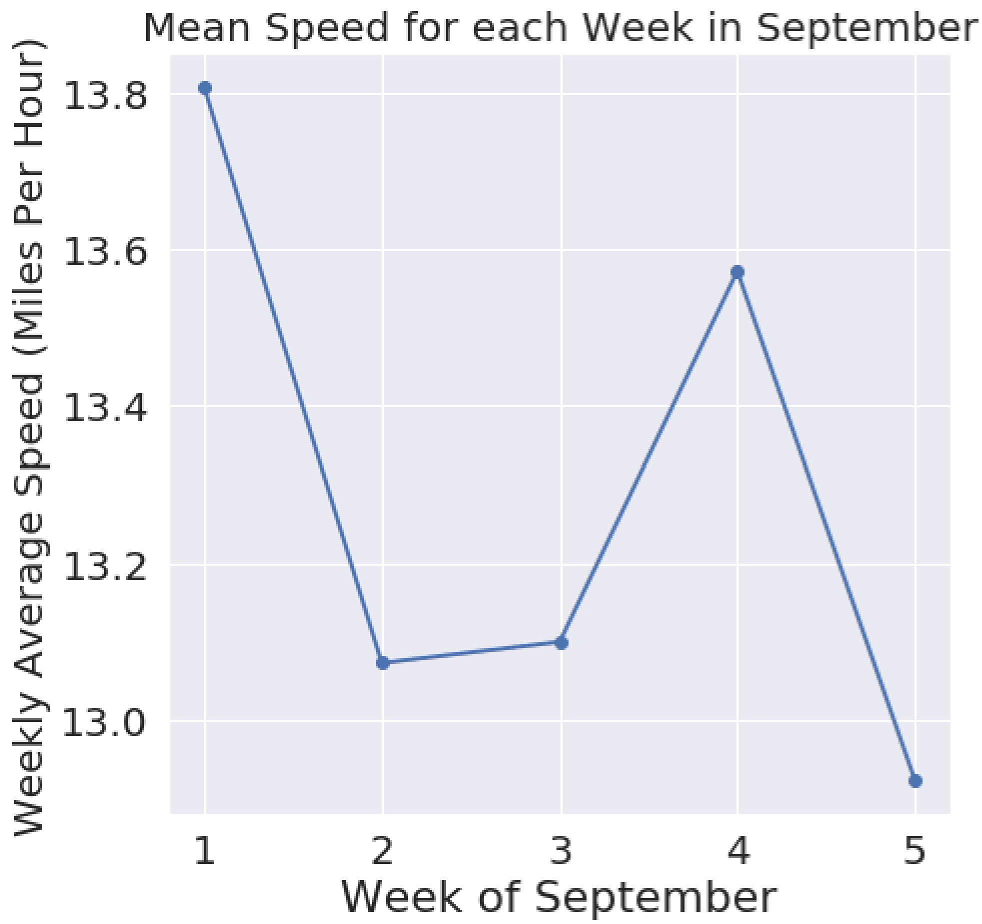
In [36]:
```
taxi_backup.groupby('Week').mean()['Speed']
```

Out[36]:
```
Week
1    13.806461
2    13.073065
3    13.099998
4    13.571924
5    12.923634
Name: Speed, dtype: float64
```

In [37]:
```
taxi_backup.groupby('DayofWeek').mean()['Speed'].values
```

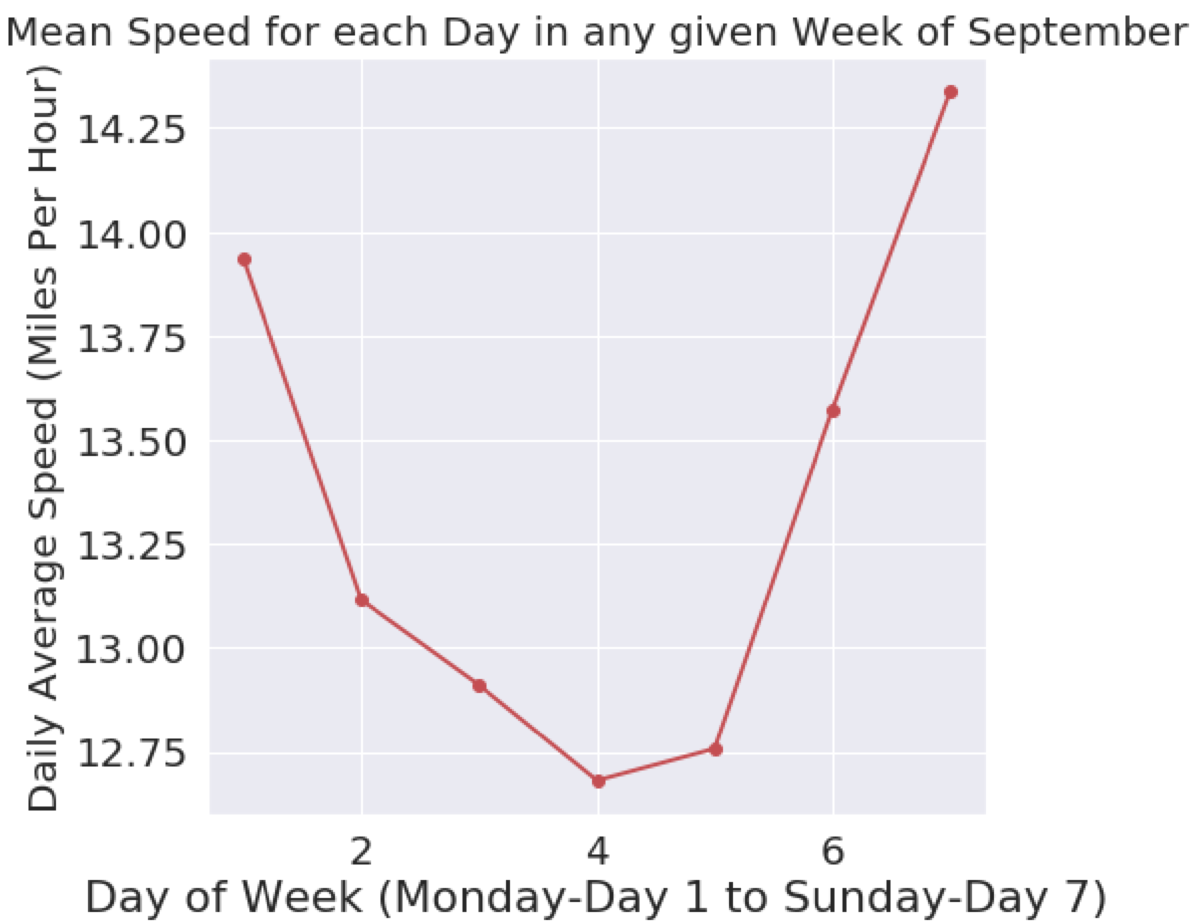
Out[37]:
```
array([13.93498571, 13.11474977, 12.91042952, 12.68070843, 12.75760886,
       13.57337905, 14.33794771])
```

In [38]:
```
plt.figure(figsize=(7,7))
plt.plot([1,2,3,4,5],taxi_backup.groupby('Week').mean()['Speed'].values,'bo-',line
plt.title('Mean Speed for each Week in September',size=20)
plt.xlabel("Week of September",size=22)
plt.ylabel("Weekly Average Speed (Miles Per Hour)",size=20)
plt.grid(True)
```



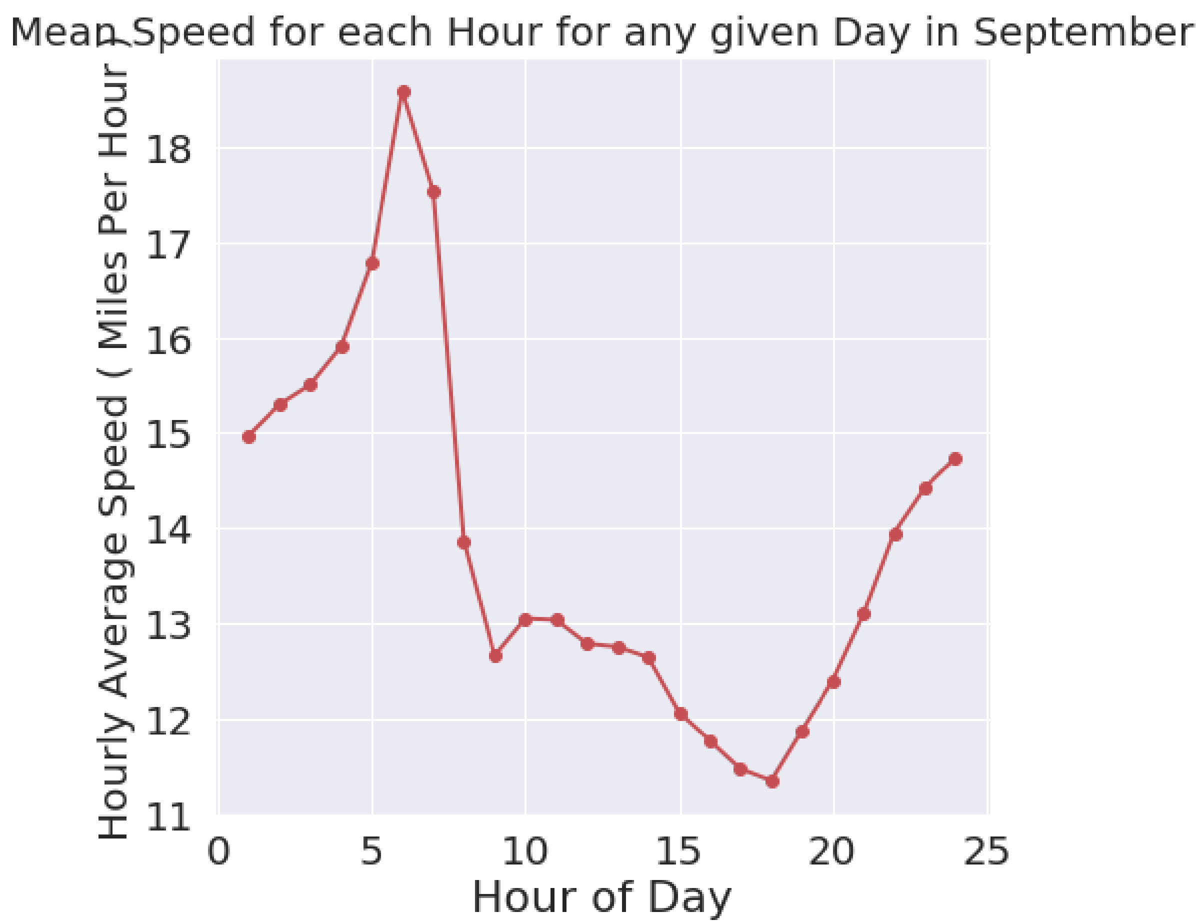
We can see from above graph that the 2nd and 3rd week has low Mean Speed as compared to 1st and 4th. Moreover, 5th week is having lowest.

In [39]:
```
plt.figure(figsize=(7,7))
plt.plot([1,2,3,4,5,6,7],taxi_backup.groupby('DayofWeek').mean()['Speed'].values,'
plt.title('Mean Speed for each Day in any given Week of September',size=20)
plt.xlabel("Day of Week (Monday-Day 1 to Sunday-Day 7)",size=22)
plt.ylabel("Daily Average Speed (Miles Per Hour)",size=20)
plt.grid(True)
```



We can see that there is very high speed on Monday , while there are low speeds on 3rd, 4th and 5th day . Then again from Saturday the speed increases and we have on Sunday a speed of 14.337.

In [40]:
```
plt.figure(figsize=(7,7))
plt.plot(range(1,25),taxi_backup.groupby('hour').mean()['Speed'].values,'ro-',line
plt.title('Mean Speed for each Hour for any given Day in September',size=20)
plt.xlabel("Hour of Day",size=22)
plt.ylabel("Hourly Average Speed ( Miles Per Hour )",size=20)
plt.grid(True)
```



We can observe from figure that we have a very steep high speed in morning 4 to 6 AM, while a noteworthy change is seen in evening 3 to 6 PM, when there is so much less average speed as compared ot whole day.

Can you perform a test to determine if the average trip speeds are materially the same in all weeks of September?

Test: One Way Anova test.

Why One Way?

> **A one-way ANOVA is used when we have 1 categorical Independent Variable with 3+ categories or groups, and 1 continuous dependent variable. This is Design of One Way Anova Test.**

In our case: We can see that there is One continuous Dependent Variable: Speed (Speed is Dependent on Week and Speed is Continuous) AND We have one categorical independent variable which is Week and it has 3+ Categories(Week 1 to Week 5). Thus I have used Anova Test here.

In a given month of September, 2015:

> **Null Hypothesis: "Average Speed is almost same through all Weeks from 1 to 5."**

> **Alternate Hypothesis: "Average Speed differs among in at least two weeks with a significant amount."**

> **P value turned out to be 0.0 which means we do not have enough enough claims to support Null Hypothesis or in other terms, the probability of given event occuring is almost 0, if we assume that Null Hypothesis is true. Thus, if we assume that Average Speed is equal for all weeks, we have 0 probability for the data that we have. Thus, we will reject NULL Hypothesis.**

> **Rejecting Null Hypothesis means we CANNOT consider that we have almost same Average speeds.**

Anova Results: P value=0.0 and test statistic=485.9197592 and thus we REJECT NULL Hypothesis of means being equal.

How I achieved results?

1. Got Week 1 Speed data into variable w1, same way for w2, w3, w4 and w5.

2. After getting Speed data into seperate groups, passing them to f_oneway() method of stats imprted from scipy.

3. We will get results performing above step.

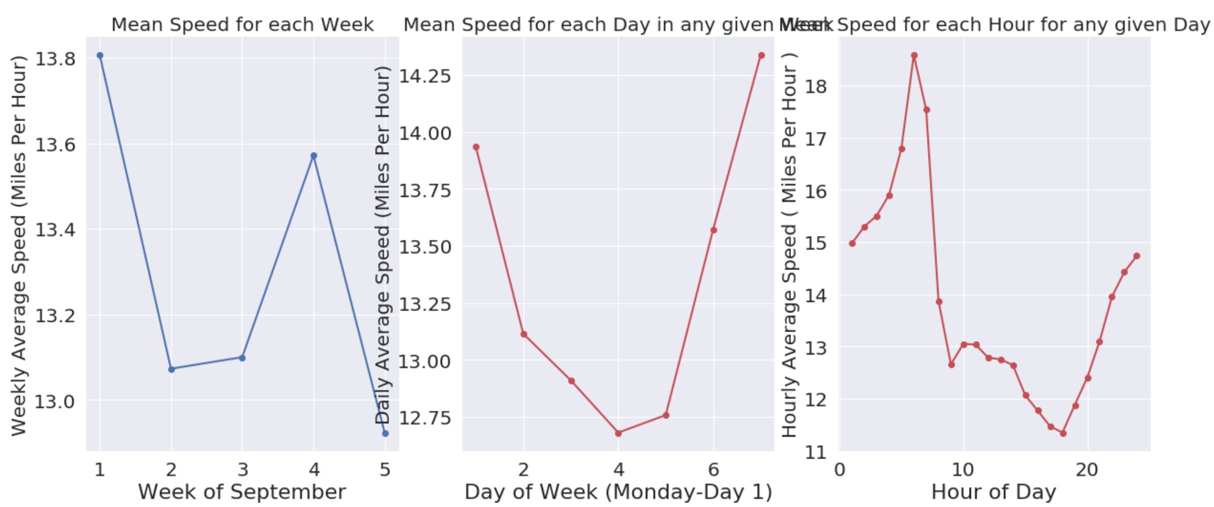
In [41]:
```
from scipy import stats
w1=taxi_backup[(taxi_backup['Week']==1)]['Speed']
w2=taxi_backup[(taxi_backup['Week']==2)]['Speed']
w3=taxi_backup[(taxi_backup['Week']==3)]['Speed']
w4=taxi_backup[(taxi_backup['Week']==4)]['Speed']
w5=taxi_backup[(taxi_backup['Week']==5)]['Speed']
# print("Total of all :",w1.shape[0]+w2.shape[0]+w3.shape[0]+w4.shape[0]+w5.shape[
# print(taxi_backup.shape[0])
stats.f_oneway(w1,w2,w3,w4,w5)
```

Out[41]:
```
F_onewayResult(statistic=485.9197592749325, pvalue=0.0)
```

If you decide they are not the same, can you form a hypothesis regarding why they differ?

In [42]:

```
sns.set()
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
plt.figure(1,figsize=(20,8))

plt.subplot(131)
plt.plot([1,2,3,4,5],taxi_backup.groupby('Week').mean()['Speed'].values,'bo-',line
plt.title('Mean Speed for each Week',size=20)
plt.xlabel("Week of September",size=22)
plt.ylabel("Weekly Average Speed (Miles Per Hour)",size=20)
plt.grid(True)

plt.subplot(132)
plt.plot([1,2,3,4,5,6,7],taxi_backup.groupby('DayofWeek').mean()['Speed'].values,'
plt.title('Mean Speed for each Day in any given Week',size=20)
plt.xlabel("Day of Week (Monday-Day 1)",size=22)
plt.ylabel("Daily Average Speed (Miles Per Hour)",size=20)
plt.grid(True)

plt.subplot(133)
plt.plot(range(1,25),taxi_backup.groupby('hour').mean()['Speed'].values,'ro-',line
plt.title('Mean Speed for each Hour for any given Day',size=20)
plt.xlabel("Hour of Day",size=22)
plt.ylabel("Hourly Average Speed ( Miles Per Hour )",size=20)
plt.grid(True)
```



Hypothesis regarding why the average speed differs:

We can see from first graph that week 5 is having too much low Average Speeds as compared to the other weeks. Now, from Second graph we can easily see that 6th and 7th day(Saturday and Sunday) are having high average speeds than most of the other days. Now a close inspection discovered that, In week 5 We have No Saturday and No Sunday. Thus the two days on which people tend to drive faster, are not there in week 5. Thus we have only low averages day in Week 5, which leads us to MAJOR difference in their means.

Week 1: Tuesday to Sunday

Week 2: All

Week 3: All

Week 4 : All

Week 5: Monday, Tuesday and Wednesday (Less Average Speed Days)

Can you build up a hypothesis of average trip speed as a function of time of day?

In [ ]:
```
sns.set()
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
plt.figure(1,figsize=(20,15))

plt.subplot(221)
plt.plot(range(1,25),taxi_backup.groupby('hour').mean()['Speed'].values,'bo-',line
plt.title('Mean Speed for each Hour for any given Day',size=20)
plt.xlabel("Hour of Day",size=22)
plt.ylabel("Hourly Average Speed ( Miles Per Hour )",size=20)
plt.grid(True)

plt.subplot(222)
plt.plot(range(1,25),taxi_backup.groupby('hour').count()['Speed'].values,'ro-',lin
plt.title('Count of Trips for each Hour for any given Day',size=20)
plt.xlabel("Hour of Day",size=22)
plt.ylabel("Count of Trips",size=20)
plt.grid(True)

plt.subplot(223)
plt.plot(df_hour['Hour'],df_hour['Mean_distance'],'bo-',linewidth=2,label='Mean Di
plt.legend(loc='upper right',fontsize=20)
plt.title('Graph representing Mean Distance corresponding Hour of Day',size=20)
plt.xlabel("Hour of the Day",size=22)
plt.ylabel("Distance (in Miles)",size=20)
plt.grid(True)

plt.subplot(224)
plt.plot(range(1,25),taxi_backup.groupby('hour').mean()['Trip_time'].values,'ro-',
plt.title('Graph representing Trip Duration for given Hour',size=20)
plt.xlabel("Hour of the Day",size=22)
plt.ylabel("Trip Duration in Seconds",size=20)
plt.grid(True)
```

From 12 Am to 5PM, the graphs of Speed and Count of Trips(First 2) are perfectly mirror images of one another. This leads to some clear explanations that as count increases ie as number of trips increases, the average speed of taxi on roads decreases.

Hypothesis for Average Speed as a function of time of day:

12 AM to 7AM

Why 12AM to 7 AM high Average Speed?

> **1A. There are high average speeds from 12 AM late night to morning 7AM. If we look closely, there are too much high average speeds from 5 to 7 which is logical. This is because, at night or early morning before 6, there might not be much traffic and taxi drivers tend to drive faster if the roads are empty.**
>
> **1B. Another reason is from 2nd graph, we can see that number of rides are less from 12 AM to 7AM exactly, and then it increases. Thus, when rides are less, the traffic would be less and thus the speeds would be more.**

7AM to 9 AM

Why from after 7 AM it drastically started to decrease untill 9AM?

> **2A . We all know that the first shift of people going to work is 8AM ot 4PM or 9AM to 5PM. Thus, We can infer that more people will be going to work after 7AM untill 9AM because their work duty starts from 8 or 9, which makes a clear logical sense.**
>
> **2B. As we can see from 2nd graph that from 7AM to 9AM, there is a lot increase in number of count of trips, thus if number of taxi on roads increases, traffic increases and this leads to decrease in average speed.**

10 AM to 3PM

> **3A . No Special changes in Graph 1.**
>
> **3B . No Special Changes in Graph 2.**

3PM to 6PM

Why was there unwanted decrease from 3PM to 6PM ?

> **4. Now, Taxi Drivers mostly changes shifts from around 4 to 5PM and that is the reason that in NYC, there is nearly 20% decrease in Taxi being available as the drivers are going back to give taxi to another drivers or they are ending their shifts during this time period. Thus, they tend to accept less rides than usually and it is mostly mentioned that NYC people generally find it very hard to get a cap from 4 PM to 5 PM.**

6PM to 12 AM.

Why Average Speed increases?

> **5. Now, NYC people nearly 80% use public transport. Thus, when all people run to their homes at 5 PM nearly from their jobs, they would directly run to public transports. Thus due to those 80% people using public transport, there is comparatively less traffic on roads and hence, more average speeds are observed. [Source : http://web.mta.info/mta/network.htm] (http://web.mta.info/mta/network.htm%5D)**
>
> **Also after having a look at graph 2, you might be wondering that How the counts are increasing and still the average speed increases. For this, please have a look at Graph 4 which indicates that Trip Duration is lower as compared to whole day, So a person might be using cab for less time and thus more trip counts.**
>
> **Also, have a look at graph 3 which indicates the distance decreases for a ride as we go from 6AM to 12 AM, which means that distance decreases for any given ride and so taxi drivers can take more trips. Thus it all justifies one another smoothly.**

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: