In [1]:
```
# Import libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from haversine import haversine
import warnings
import matplotlib.cm as cm
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error,r2_score
from sklearn import linear_model
```

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.

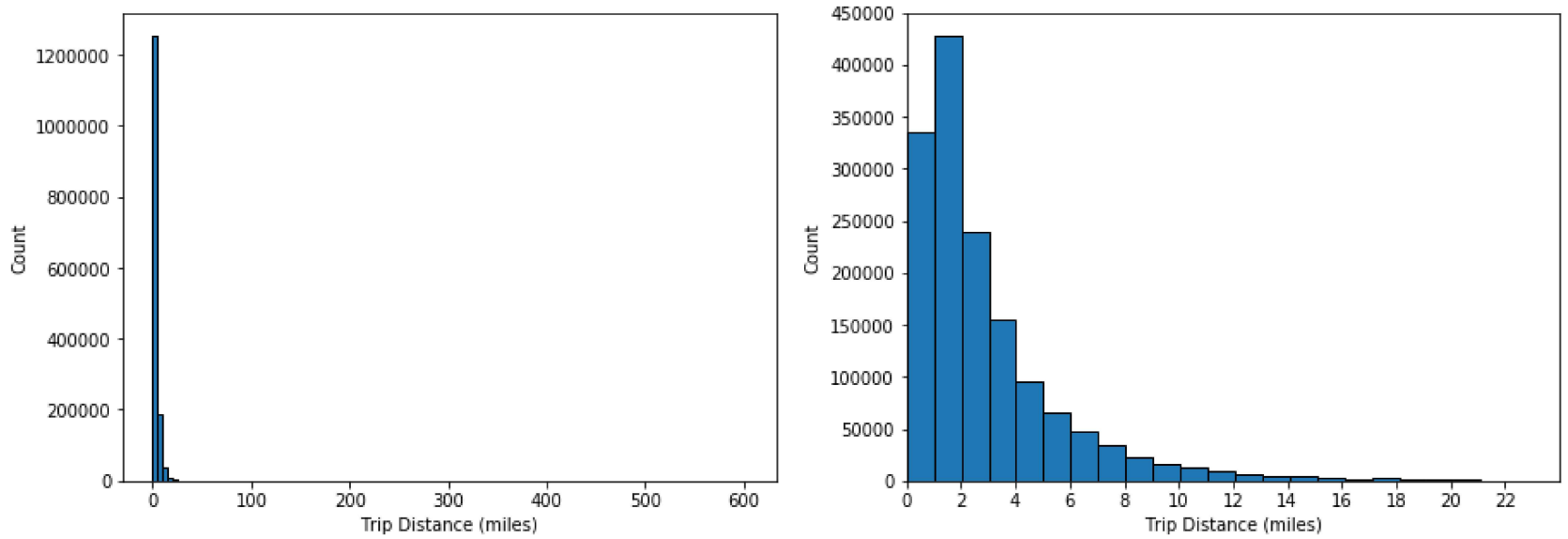
In [2]:
```
# Download data and read into a dataframe
url = "https://s3.amazonaws.com/nyc-tlc/trip+data/green_tripdata_2015-09.csv"
data = pd.read_csv(url)
```

In [3]:
```
# Number of rows and columns in the dataframe
print ("Number of rows in the file: {0}".format(data.shape[0]))
print ("Number of columns in the file: {0}".format(data.shape[1]))
```

```
Number of rows in the file: 1494926
Number of columns in the file: 21
```

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.

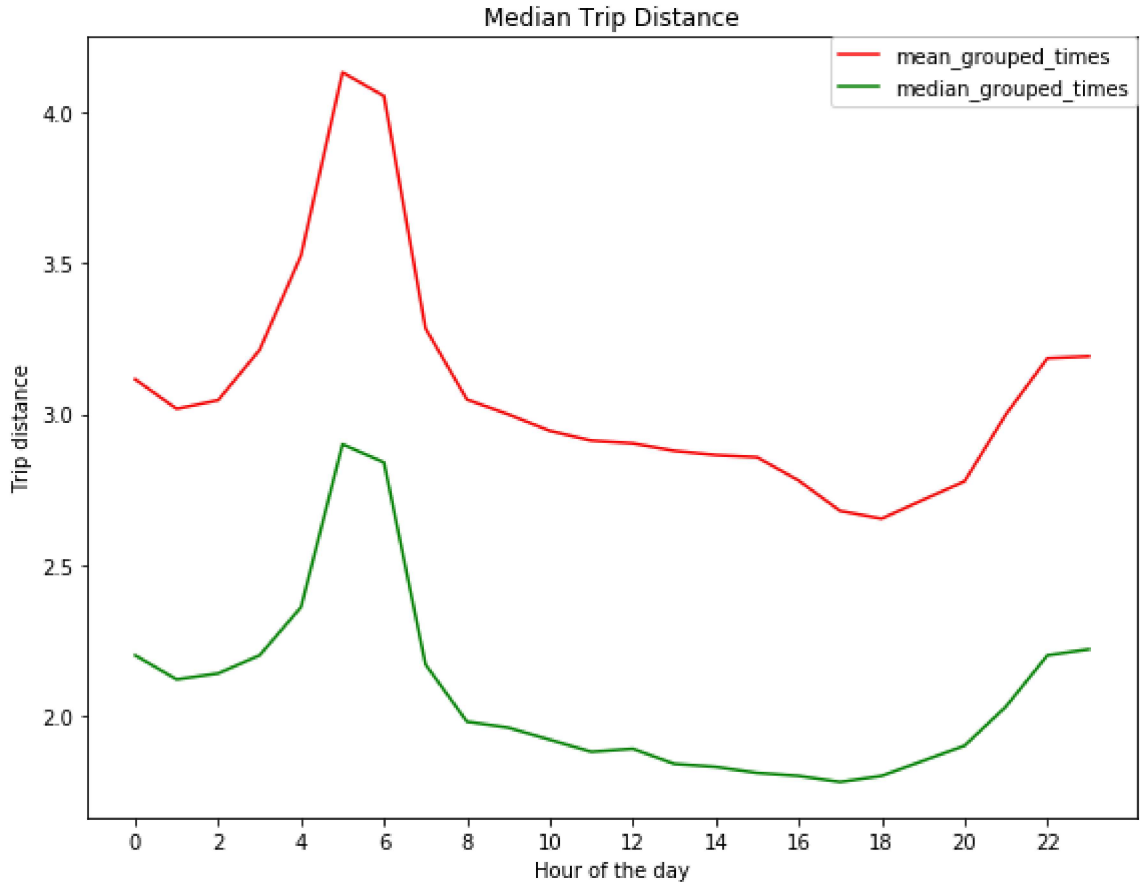
In [4]:
```
# histogram of the number of the trip distance ("Trip Distance")
f, (plt1, plt2) = plt.subplots(1, 2, sharey=True,figsize=(15,5))

plt.subplot(1,2,1)
plt.hist(data['Trip_distance'], bins=120,edgecolor='black')
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Count')


plt.subplot(1,2,2)
plt.hist(data['Trip_distance'], bins=600,edgecolor='black')
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Count')
plt.xlim(0, 24)
plt.xticks(range(0,24,2))
plt.show()
```



- This shows that most (almost 1,250,000 trips) of the trips are less than 5 miles long.
- The Histogram here is skewed to the right, implying that the dataset has very few trips with distances larger than 15 miles.
- The mean trip distance will be definely larger than the median trip distance value.
- In the right plot, the outliers have been removed, the distances beyond 24 miles, to capture the larger chunk of data.

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 [5]:
```
# mean and median trip distance grouped by hour of day
times = pd.to_datetime(data.lpep_pickup_datetime)

mean_grouped_times = data.groupby([times.dt.hour]).Trip_distance.mean()
median_grouped_times = data.groupby([times.dt.hour]).Trip_distance.median()

fig, axes = plt.subplots(nrows=1, ncols=1,figsize=(8,6))
fig.tight_layout()

#plt.figure(1)
plt.plot(np.unique(times.dt.hour),mean_grouped_times,'r', label='mean_grouped_times')
plt.xlabel("Hour of the day")
plt.ylabel("Trip distance")
plt.title("Mean Trip Distance")
plt.xticks(range(0,24,2))

plt.plot(np.unique(times.dt.hour),median_grouped_times,'g', label='median_grouped_times')
plt.xlabel("Hour of the day")
plt.ylabel("Trip distance")
plt.title("Median Trip Distance")
plt.xticks(range(0,24,2))

plt.legend(bbox_to_anchor=(1, 1), loc=1, borderaxespad=0.)

plt.show()
```



Analysis:

Looking at the graph above we can observe that mornings peaks are larger than evening peaks. This could be due to people preferring cabs in the morning to not get late to the work. Some of these people may wish to come back in cabs, hence the evening peak. Some of them may return using public transport. This could also be due to morning or late night taxi rides to places far off like airports because usually flights are during nights and early hours.

In [6]:
```
# Trips that originate or terminate at one of the NYC area airports

# Three major airports in NYC - John F Kennedy International Airport (JFK),
#                               La Guardia Airport (LGA),
#                               Newark Liberty International Airport (EWR)

# If the pick up or drop off location is within 1 mile radius of any airports' lat-lon range - tag it to t
hat particular airport

# Function to tag rows
def tag_airport(row):

    jfk_latlon = (40.639722, -73.778889)
    lga_latlon = (40.77725, -73.872611)
    ewr_latlon = (40.6925, -74.168611)

    if (haversine(jfk_latlon,(row[6],row[5]),miles=True)<=1):
        return "JFK_pick"
    elif (haversine(jfk_latlon,(row[8],row[7]),miles=True)<=1):
        return "JFK_drop"
    elif (haversine(lga_latlon,(row[6],row[5]),miles=True)<=1):
        return "LGA_pick"
    elif (haversine(lga_latlon,(row[8],row[7]),miles=True)<=1):
        return "LGA_drop"
    elif (haversine(ewr_latlon,(row[6],row[5]),miles=True)<=1):
        return "EWR_pick"
    elif (haversine(ewr_latlon,(row[8],row[7]),miles=True)<=1):
        return "EWR_drop"
    else:
        return "N/A"

# Tag a trip if it is to/from airport
data["Trip_airport"] = data.apply(lambda row: tag_airport(row), axis=1)
```

In [7]:
```
# Average fare amount, median passenger count, number of trips to each airport, and average trip distance
airport_data = data[data['Trip_airport'] != 'N/A'].groupby('Trip_airport').agg({'Fare_amount':'mean','Pass
enger_count':'median','Trip_airport':'size','Trip_distance':'mean'})
non_airport_data = data[data['Trip_airport'] == 'N/A']


print ("Total number of trips to/from airport: {0}".format(airport_data.Trip_airport.sum()))
print ("Total average fare to/from airport: {0}".format(airport_data.Fare_amount.mean()))
print ("Total average distance to/from airport: {0}".format(airport_data.Trip_distance.mean()))
print ("************************************************************")
print ("Total number of trips to/from non-airport: {0} ".format(non_airport_data.shape[0]))
print ("Total average fare to/from non-airport: {0} ".format(non_airport_data.Fare_amount.mean()))
print ("Total average distance to/from non-airport: {0} ".format(non_airport_data.Trip_distance.mean()))
```

```
Total number of trips to/from airport: 40881
Total average fare to/from airport: 42.15816318335252
Total average distance to/from airport: 8.693610024041119
************************************************************
Total number of trips to/from non-airport: 1454045
Total average fare to/from non-airport: 12.12709272409044
Total average distance to/from non-airport: 2.8124662991859517
```

- The average trip distance to LGA airport is very less indicating that it is within the city and is accessible to most of the New Yorkers. This led to many people using LGA airport to get away.
- Airport drop-offs are significantly many compared to airport pick-ups. This is probably because many people prefer getting to the airport by themselves but while returning they make some arrangements for pick-up before hand like family or friends.
- Average fare to airports is higher than non-aiport trips as airports have higher pricing and flat rate options.
- Many people might not be preferring airport pick-ups for the same reason.

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). We will validate a sample.

In [8]:
```
# Derived attribute for tip percentage
data = data[(data.Total_amount>=2.5)]
data['tip_percentage'] = 100*data['Tip_amount']/data['Total_amount']
```

In [9]:
```
# Extracting the time and date characteristics
data['pickup_day'] = pd.to_datetime(data['lpep_pickup_datetime'], format = '%Y-%m-%d %H:%M:%S').map(lambda
 x: x.weekday_name)
data['pickup_hour'] = pd.to_datetime(data['lpep_pickup_datetime'], format = '%Y-%m-%d %H:%M:%S').map(lambd
a x: x.hour)

data['drop_day'] = pd.to_datetime(data['Lpep_dropoff_datetime'], format = '%Y-%m-%d %H:%M:%S').map(lambda
x: x.weekday_name)
data['drop_hour'] = pd.to_datetime(data['Lpep_dropoff_datetime'], format = '%Y-%m-%d %H:%M:%S').map(lambda
 x: x.hour)
```

In [10]:
```
#Calculating trip duration and trip_speed values

import datetime as dt
data['Pickup_dt'] = data.lpep_pickup_datetime.apply(lambda x:dt.datetime.strptime(x,"%Y-%m-%d %H:%M:%S"))
data['Dropoff_dt'] = data.Lpep_dropoff_datetime.apply(lambda x:dt.datetime.strptime(x,"%Y-%m-%d %H:%M:%S"))

data['Trip_duration'] = ((data.Dropoff_dt-data.Pickup_dt).apply(lambda x:x.total_seconds()/60.))
data['Speed_mph'] = data.Trip_distance/(data.Trip_duration/60)
```

In [11]:
```
# Tip percentage comparison for airport and non airport trips
airport_data = data[data['Trip_airport'] != 'N/A']
non_airport_data = data[data['Trip_airport'] == 'N/A']

a_mean_grouped_tip_percent = airport_data.groupby('Trip_airport').tip_percentage.mean()
na_mean_grouped_tip_percent = non_airport_data.groupby('Trip_airport').tip_percentage.mean()

print ("Mean tip percent for airport trips\n{0}".format(a_mean_grouped_tip_percent))
print ("**********************************************************")
print ("Mean tip percent for non-airport trips\n{0}".format(na_mean_grouped_tip_percent))
```

```
Mean tip percent for airport trips
Trip_airport
EWR_drop    10.931781
EWR_pick    15.034394
JFK_drop     9.648137
JFK_pick     8.770687
LGA_drop     9.455125
LGA_pick     6.771311
Name: tip_percentage, dtype: float64
**********************************************************
Mean tip percent for non-airport trips
Trip_airport
N/A    6.587324
Name: tip_percentage, dtype: float64
```

- The airport drop offs usually have higher tip percentages inspite of higher fare amounts. This could be because drivers are helpful in dropping the passengers at the exact airport terminals and also helping passengers with luggage unloading.
- The interesting find here is that EWR airport trips have higher average tip percentages for pick ups than drop offs.
- The average tip percentage for non-airport trips is lower than airport trips as non-airport trips are short and people might be taking taxi everyday and not generous enough to tip everyday.

In [12]:
```
# Non airport non zero
nonzero_nonairport_data = non_airport_data[non_airport_data['tip_percentage'] != 0.]
mean_grouped_tip_percent = nonzero_nonairport_data.tip_percentage.mean()
median_grouped_tip_percent = nonzero_nonairport_data.tip_percentage.median()

print ("Mean tip percent {0}".format(mean_grouped_tip_percent))
print ("Median tip percent {0}".format(median_grouped_tip_percent))
```

```
Mean tip percent 16.45210135894283
Median tip percent 16.666666666666668
```

Most of the taxi trips within city do not offer tip. While calculating mean, the mean is pushed lower due to many 0% tip percentages. If we filter out such values. The average tip chosen by many passengers is approximately 16%

In [13]:
```
drop = airport_data[airport_data['Trip_airport'].str.contains("drop")]
pick = airport_data[airport_data['Trip_airport'].str.contains("pick")]
```

In [14]:
```
drop_mean_grouped_tip_percent = drop.groupby(['Trip_airport', 'drop_day']).agg({'tip_percentage':'mean','drop
_day':'size'})
drop_median_grouped_tip_percent = drop.groupby(['Trip_airport', 'drop_day']).agg({'tip_percentage':'median',
'drop_day':'size'})

print ("Mean tip percent by drop off day from the airport \n{0}".format(drop_mean_grouped_tip_percent))
print ("**************************************************************")


pick_mean_grouped_tip_percent = pick.groupby(['Trip_airport', 'pickup_day']).agg({'tip_percentage':'mean','pi
ckup_day':'size'})
pick_median_grouped_tip_percent = pick.groupby(['Trip_airport', 'pickup_day']).agg({'tip_percentage':'median'
,'pickup_day':'size'})

print ("Mean tip percent by pick up day from the airport \n{0}".format(pick_mean_grouped_tip_percent))
```

Mean tip percent by drop off day from the airport

Trip_airport	drop_day	tip_percentage	drop_day
EWR_drop	Friday	10.502195	93
	Monday	12.207175	112
	Saturday	8.703358	94
	Sunday	12.348927	103
	Thursday	10.355468	70
	Tuesday	9.789486	107
	Wednesday	12.398218	84
JFK_drop	Friday	10.488054	2012
	Monday	9.336400	1664
	Saturday	9.153618	1648
	Sunday	9.349281	1720
	Thursday	10.107090	1830
	Tuesday	9.252773	1782
	Wednesday	9.659674	1948
LGA_drop	Friday	10.482137	5441
	Monday	9.348570	3381
	Saturday	7.789381	2668
	Sunday	8.085666	3013
	Thursday	10.431867	4102
	Tuesday	9.229667	3470
	Wednesday	9.485983	3838

****

Mean tip percent by pick up day from the airport

Trip_airport	pickup_day	tip_percentage	pickup_day
EWR_pick	Friday	7.333333	5
	Monday	11.719298	5
	Saturday	2.346316	4
	Sunday	7.377984	7
	Thursday	29.685752	7
	Tuesday	42.662707	3
	Wednesday	11.782332	8
JFK_pick	Friday	11.505393	34
	Monday	3.458791	41
	Saturday	9.217310	35
	Sunday	14.599999	47
	Thursday	7.398441	37
	Tuesday	8.657561	46
	Wednesday	6.179620	42
LGA_pick	Friday	5.822292	222

Monday	7.760447	179
Saturday	5.523684	172
Sunday	7.423183	186
Thursday	7.780126	174
Tuesday	7.085761	165
Wednesday	6.183898	171

Predict Modelling of tip percentage -

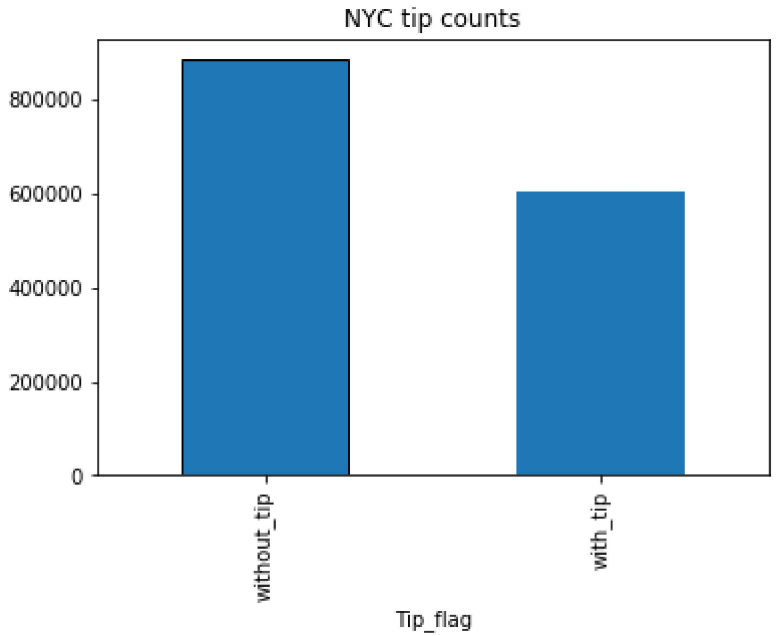
A typical predictive modelling process involves following steps

1. Exploratory Data Analysis
2. Data Cleaning
3. Feature Engineering/Extraction
4. Model selection and training
5. Evaluation of the model

First we observe nature of taxi trips tipping nature in correlation with trip duration, fare_amount, week, day of the week and payment type in the data analysis process.

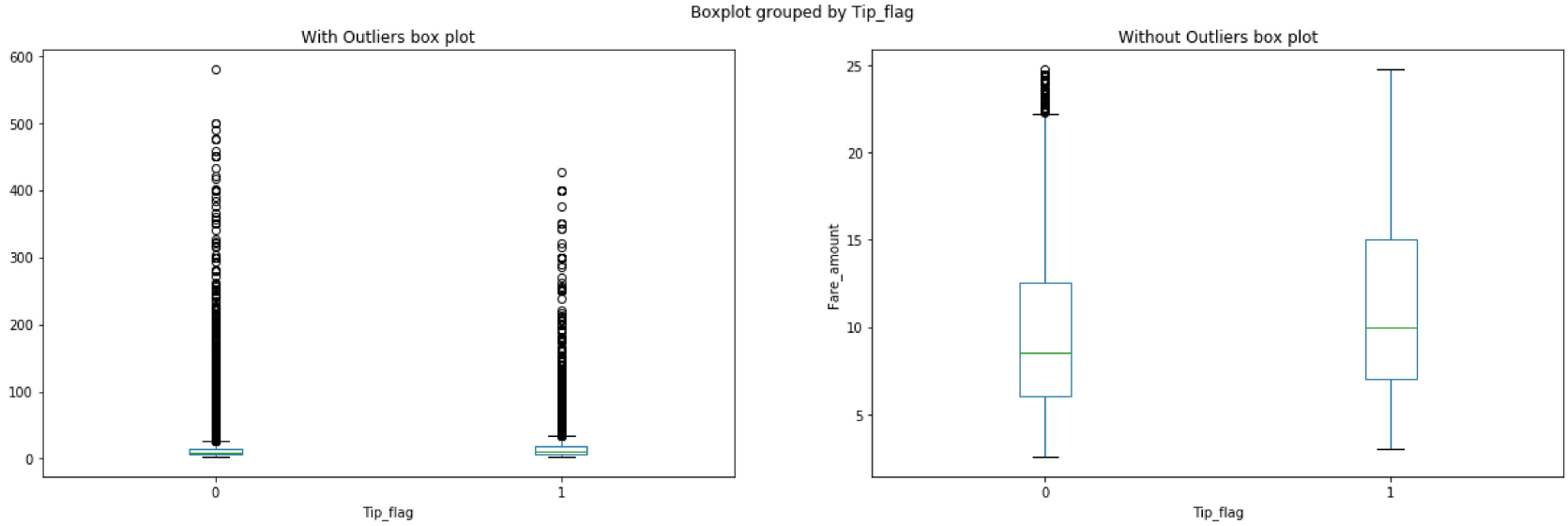
I built a classification model to see what is characteristics that effect tipping drivers. Is there a definitive nature?

In [15]:
```
# Predictive model for tip percentages
df = pd.DataFrame(data)

# Step 1 - Discriptive Analysis of tips
df['Tip_flag'] = df['Tip_amount'].apply(lambda x: 1 if x > 0 else 0)
df['lpep_pickup_datetime'] = pd.to_datetime(df['lpep_pickup_datetime'])
df['Lpep_dropoff_datetime'] = pd.to_datetime(df['Lpep_dropoff_datetime'])
df['Tip_flag'] = df['Tip_flag'].astype('int')


plt.figure();

df.groupby('Tip_flag').size().plot(kind='bar',edgecolor='black')
plt.xticks(range(2),('without_tip','with_tip'))
plt.title("NYC tip counts")
plt.show()
```



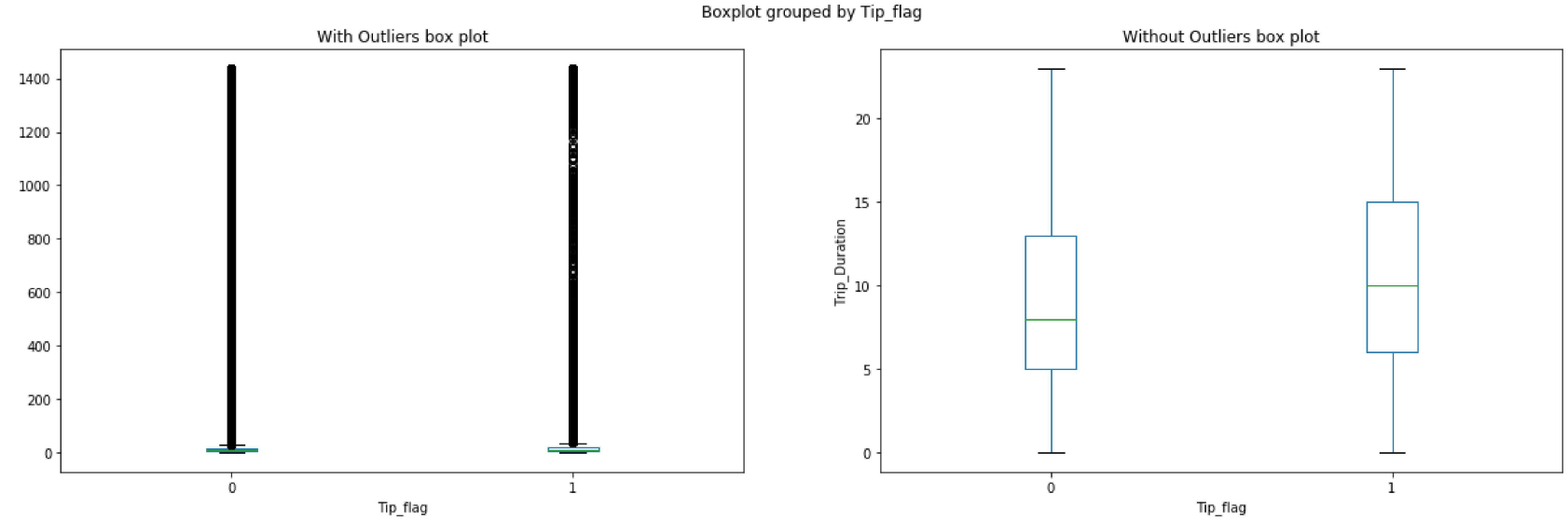
Here, we can see that most of the trips ~60% of them do not tip their drivers. The possible reasons could be very short trips or maybe surcharges or less average speed during traffic.

In [16]:
```
import warnings
import numpy as np
warnings.filterwarnings('ignore')

df = df[df.Fare_amount>2.5]

df['Fare_amount'] = df['Fare_amount'].abs()

fig, axs = plt.subplots(1,2,figsize=(20,6),edgecolor='black')

b1 = df.boxplot(column='Fare_amount',by='Tip_flag',ax=axs[0],grid=False)
axs[0].set_title('With Outliers box plot')
b2 = df[df.Fare_amount<25].boxplot(column='Fare_amount',by='Tip_flag',ax=axs[1],grid=False)
axs[1].set_title('Without Outliers box plot')

plt.ylabel("Fare_amount")

plt.show()
```



Taxi trips with average fare amount higher are more likely to tip drivers

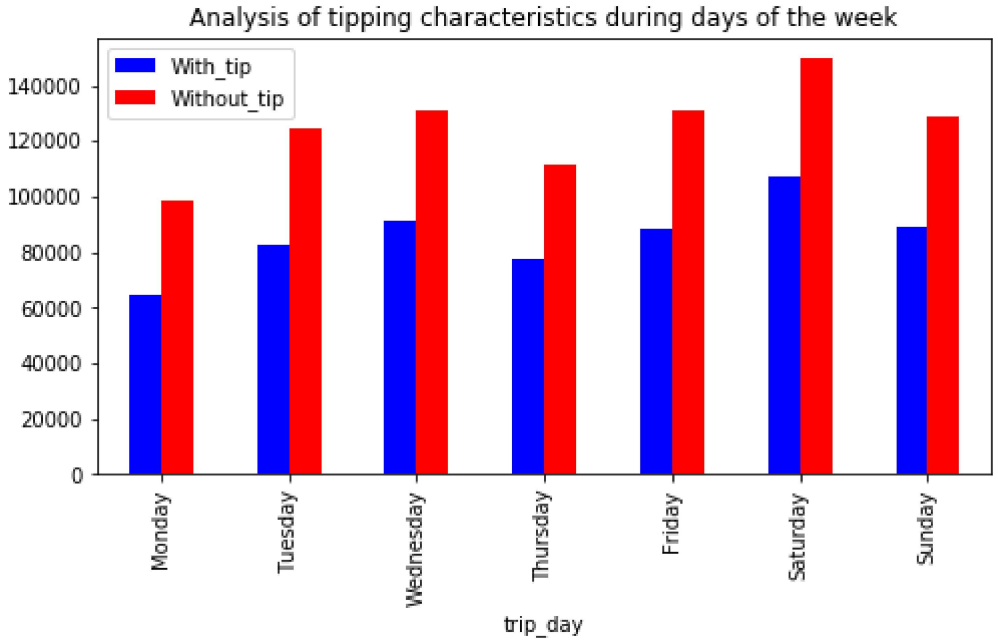
In [17]:
```
df['Lpep_dropoff_datetime'] = pd.to_datetime(df['Lpep_dropoff_datetime'])
df['lpep_pickup_datetime']=pd.to_datetime(df['lpep_pickup_datetime'])


df['Trip_duration'] = (df['Lpep_dropoff_datetime'] - df['lpep_pickup_datetime']).astype('timedelta64[m]')

fig, axs = plt.subplots(1,2,figsize=(20,6),edgecolor='black')

plt.suptitle("Analysis of Tip_flag vs. Trip_Duration")

b1 = df.boxplot(column='Trip_duration',by='Tip_flag',ax=axs[0],grid=False)
axs[0].set_title('With Outliers box plot')


b2 = df[df.Trip_duration < 24].boxplot(column='Trip_duration',by='Tip_flag',ax=axs[1],grid=False)
axs[1].set_title('Without Outliers box plot')

plt.ylabel("Trip_Duration")
plt.show()

print (df['Trip_duration'].head())
```




```
0    0.0
1    0.0
2    2.0
3    4.0
4    4.0
Name: Trip_duration, dtype: float64
```

Taxi trips with more average trip duration are likely to be tipped. This could be because drivers engaging in small talk and gaining appreciation from passengers or maybe because the drivers might have chosen to help passengers through fastest routes during traffic.

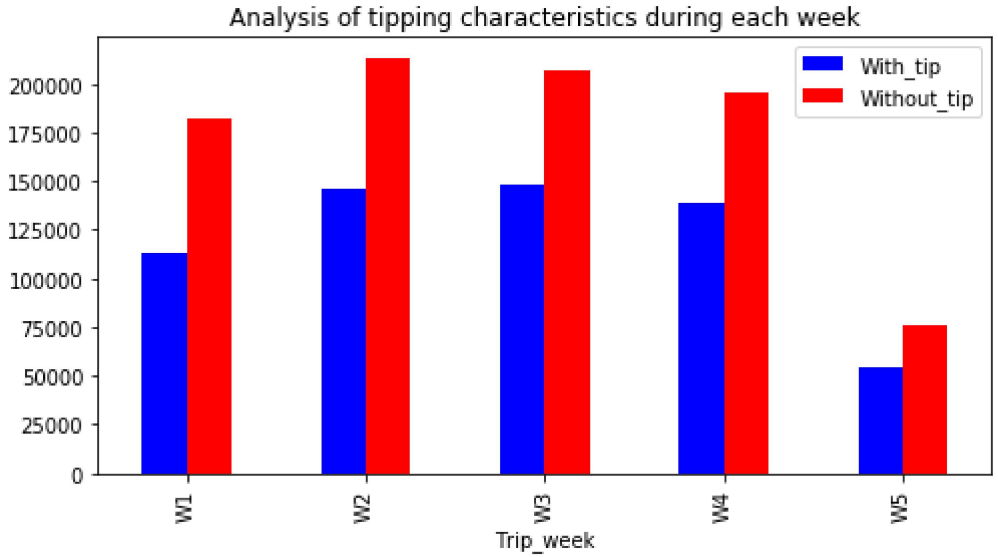
In [18]:
```
df['trip_day'] = df['lpep_pickup_datetime'].dt.dayofweek

subset = df.groupby(['trip_day','Tip_flag']).size().unstack()
subset = subset.reindex_axis(sorted(subset.columns,reverse=True), axis=1)

plt.rcParams["figure.figsize"] = [8,4]
subset.plot(kind = 'bar', colormap = cm.bwr, width = 0.5)
plt.xticks(range(7),('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'))
plt.title("Analysis of tipping characteristics during days of the week")
plt.legend(labels=['With_tip','Without_tip'])
plt.show()
```



More people are likely to tip drivers on weekends as the number of trips is higher and people might be hanging out as discussed before.

In [19]:
```
df['Trip_week'] =  df['lpep_pickup_datetime'].dt.week
df['Trip_week'] = df['Trip_week'] - 36

print (df['Trip_week'].describe())
subset2 = df.groupby(['Trip_week','Tip_flag']).size().unstack()
subset2 = subset2.reindex_axis(sorted(subset.columns,reverse=True), axis=1)

plt.rcParams["figure.figsize"] = [8,4]
subset2.plot(kind = 'bar', colormap = cm.bwr, width = 0.5)
plt.xticks(range(5),('W1','W2','W3','W4','W5'))
plt.title("Analysis of tipping characteristics during each week")
plt.legend(labels=['With_tip','Without_tip'])
plt.show()
```

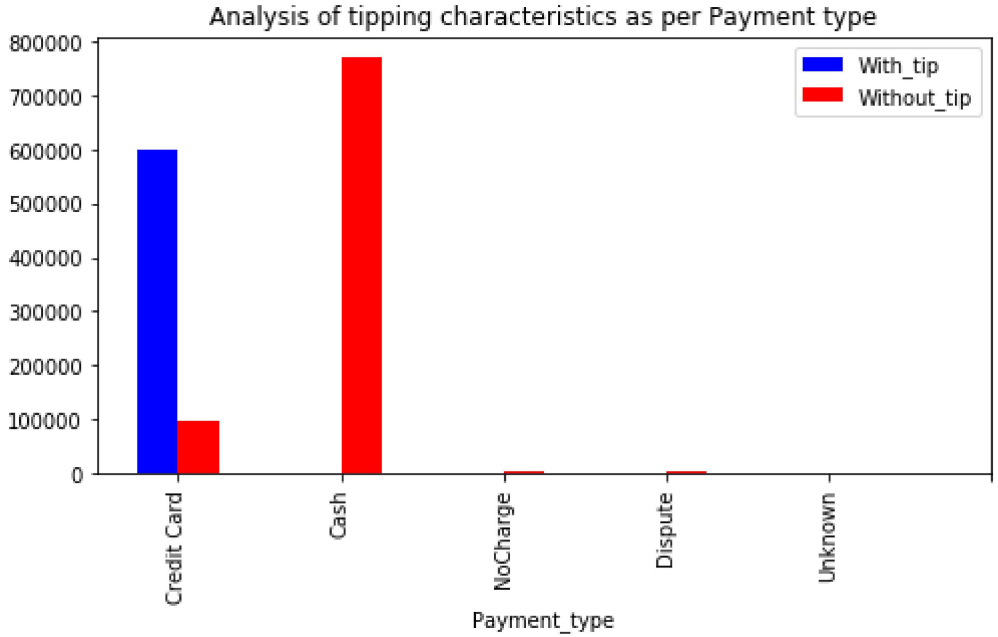
```
count    1.475973e+06
mean     1.759721e+00
std      1.251719e+00
min      0.000000e+00
25%      1.000000e+00
50%      2.000000e+00
75%      3.000000e+00
max      4.000000e+00
Name: Trip_week, dtype: float64
```



More tips given during mid-week due to more trips and also because people are not broke mid-month.

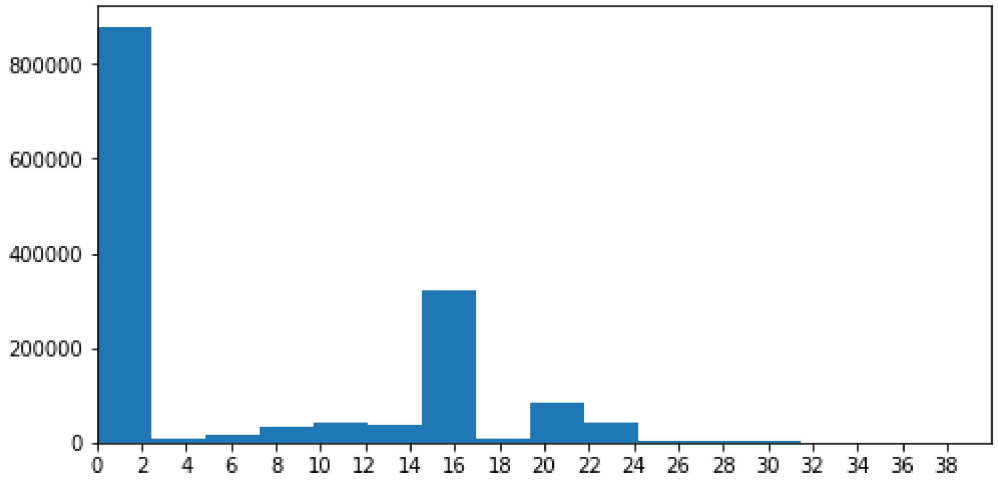
In [20]:
```
subset3 = df.groupby(['Payment_type','Tip_flag']).size().unstack()
subset3 = subset3.reindex_axis(sorted(subset.columns,reverse=True), axis=1)

plt.rcParams["figure.figsize"] = [8,4]
subset3.plot(kind = 'bar', colormap = cm.bwr, width = 0.5)
print (subset3)

plt.title("Analysis of tipping characteristics as per Payment type")
plt.xticks(range(6),('Credit Card','Cash','NoCharge','Dispute','Unknown'))
plt.legend(labels=['With_tip','Without_tip'])
plt.show()

print ((subset3[1][1]/df[df.Tip_flag==1].shape[0])*100,"% tip givers pay through credit card")
```

```
Tip_flag             1         0
Payment_type
1             600960.0   97646.0
2                  1.0  771331.0
3                  NaN    2860.0
4                  1.0    3122.0
5                  NaN      52.0
```




```
99.9996672003 % tip givers pay through credit card
```

In [21]:
```
df['tip_percentage'] = 100*df['Tip_amount']/df['Total_amount']

plt.hist(df['tip_percentage'],bins=40)
plt.xlim(0,40)
plt.xticks(range(0,40,2))
plt.show()

print (df[(df['tip_percentage']==0)].shape)

print ((df[(df['tip_percentage']==0)].shape[0])/df.shape[0]*100,"% of customers do not tip Taxi drivers")
print ((df[(df['tip_percentage']>13) & (df['tip_percentage']<18) ].shape[0])/df.shape[0]*100,"% of customers
 tip Taxi drivers between 14% and 17%")
```




```
(875011, 34)
59.28367253330515 % of customers do not tip Taxi drivers
23.77895801616967 % of customers tip Taxi drivers between 14% and 17%
```

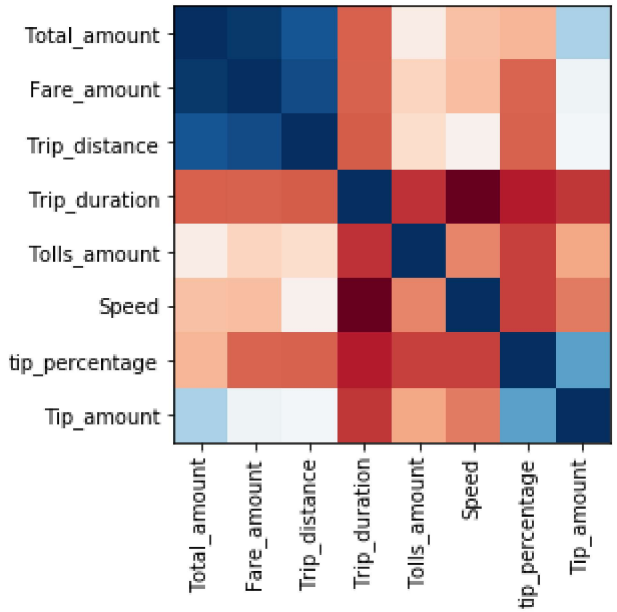
In [22]:
```
# After observing these characteristics, we will clean data off the outliers
#Data Cleaning
predict_data = df.copy()

predict_data = predict_data[predict_data['Tip_amount'] >= 0]
predict_data = predict_data[predict_data['Trip_distance'] > 0]
predict_data = predict_data[predict_data['Trip_duration'] > 0]
predict_data = predict_data[predict_data['Total_amount'] >= 2.5]

predict_data = predict_data[(predict_data['RateCodeID'] <=6) & (predict_data['RateCodeID'] > 0)]
predict_data['Fare_amount'] = predict_data['Fare_amount'].abs()
predict_data['Extra'] = predict_data['Extra'].abs()
predict_data['MTA_tax'] =predict_data['MTA_tax'] .abs()
predict_data['Tip_amount'] =predict_data['Tip_amount'] .abs()

predict_data['improvement_surcharge'] = predict_data['improvement_surcharge'].abs()
predict_data['Tolls_amount'] = predict_data['Tolls_amount'].abs()
predict_data['Speed'] = predict_data['Trip_distance']/(predict_data['Trip_duration']/60)
predict_data = predict_data.drop(['Store_and_fwd_flag'],axis=1)
predict_data = predict_data.drop(['lpep_pickup_datetime'],axis=1)
predict_data = predict_data.drop(['Lpep_dropoff_datetime'],axis=1)
predict_data = predict_data.drop(['Ehail_fee'],axis=1)

predict_data.head()
```

Out[22]:

	VendorID	RateCodeID	Pickup_longitude	Pickup_latitude	Dropoff_longitude	Dropoff_latitude	Passenger_count	Trip_dis
2	2	1	-73.921410	40.766708	-73.914413	40.764687	1	0.59
3	2	1	-73.921387	40.766678	-73.931427	40.771584	1	0.74
4	2	1	-73.955482	40.714046	-73.944412	40.714729	1	0.61
5	2	1	-73.945297	40.808186	-73.937668	40.821198	1	1.07
6	2	1	-73.890877	40.746426	-73.876923	40.756306	1	1.43

5 rows × 31 columns

In [23]:
```
#Feature Engineering and Feature Ranking
cols = predict_data.columns.tolist()

cols.insert(22, cols.pop(cols.index('tip_percentage')))
cols.insert(24, cols.pop(cols.index('Tip_flag')))
predict_data = predict_data.reindex(columns= cols)

continuous_cols=['Total_amount','Fare_amount','Trip_distance','Trip_duration','Tolls_amount','Speed','tip_
percentage','Tip_amount']

cor_mat = predict_data[continuous_cols].corr()
plt.imshow(cor_mat,cmap=cm.RdBu)
plt.xticks(range(len(continuous_cols)),continuous_cols,rotation='vertical')
plt.yticks(range(len(continuous_cols)),continuous_cols)

plt.show()
```



The correlation matrix shows that

- tip amount is positively correlated to trip_distance, fare_amount, total_amount
- tip amount is negatively correlated to trip_duration, tolls_amount, speed

To score the features based on their effect on tip_flag that states tip given or not given, I used Random Forest Regressor algorithm. This builds a model to see which features describe the training data well.

In [24]:
```
# Random forest regressor

predict_data1 = predict_data.drop(['tip_percentage','drop_hour','drop_day','pickup_hour','Dropoff_dt','pickup
_day','Pickup_dt','Speed_mph','Trip_airport', 'Tip_amount','Pickup_latitude','Pickup_longitude','Dropoff_long
itude','Dropoff_latitude','VendorID'],axis=1)
cols = predict_data1.columns.tolist()
cols.insert(15, cols.pop(cols.index('Tip_flag')))
predict_data1 = predict_data1.reindex(columns= cols)

predict_data_arr = predict_data1.values

X = predict_data_arr[:,0:15]
Y = predict_data_arr[:,15]
```

In [25]:
```
rf = RandomForestRegressor()
rf.fit(X, Y)
print ("Features sorted by their score:")
print (pd.DataFrame(sorted(zip(rf.feature_importances_, predict_data1.columns), reverse=True),columns=['Impor
tance','Feature']))
```

```
Features sorted by their score:
    Importance               Feature
0     0.772753          Payment_type
1     0.116550          Total_amount
2     0.081322           Fare_amount
3     0.027123                 Extra
4     0.001148          Tolls_amount
5     0.000301         Trip_distance
6     0.000269                 Speed
7     0.000151         Trip_duration
8     0.000105             trip_day
9     0.000082            Trip_week
10    0.000055       Passenger_count
11    0.000049  improvement_surcharge
12    0.000035               MTA_tax
13    0.000032            RateCodeID
14    0.000025            Trip_type
```

In [26]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
rfc = RandomForestClassifier()

# Cross Validation
scores = cross_val_score(rfc, X_train, y_train, cv=5)
print (scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

# Fit model
model = rfc.fit(X_train, y_train)
predictions = rfc.predict(X_test)
```

```
[ 0.99391684  0.99440984  0.99419121  0.99306371  0.99350098]
Accuracy: 0.99 (+/- 0.00)
```

In [27]:
```
accuracy = accuracy_score(y_test, predictions)
print ("Accuracy of RandomForest Classifier : ",accuracy*100,"%")
```

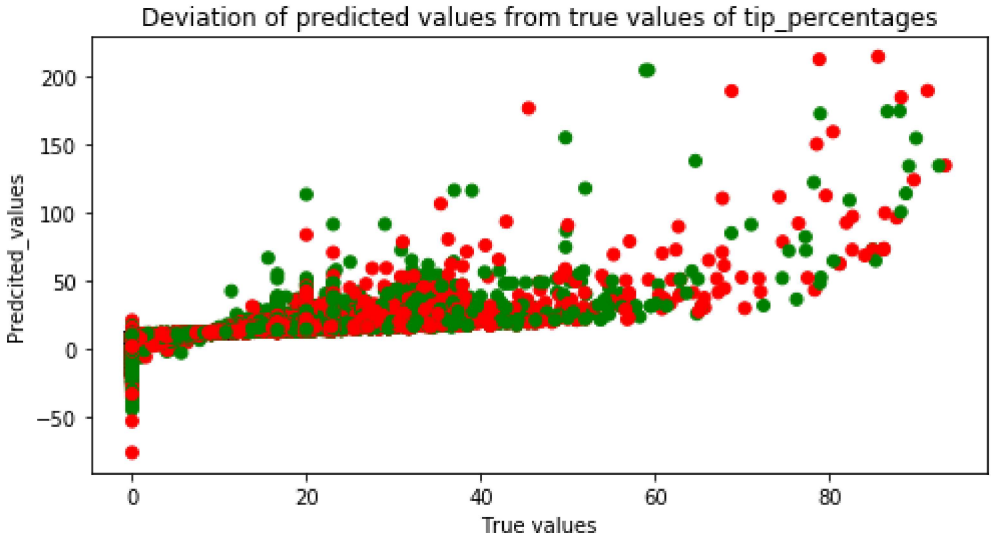
```
Accuracy of RandomForest Classifier :  99.3950285338 %
```

Here we can observe that the Random Forest Classifier classifies whether a given trip instance will be tipped or not tipped with an accuracy of 99% approximately.

Next, we use Linear regression method the predict tip percentages.A Linear regression model typically formas a mathematical relation between dependent and independent variables. The variable tip_percentage we are trying to estimate sis the dependent variable while others in the feature list are independent variables.

In [28]:
```
predict_data2 = predict_data.drop(['Tip_flag','drop_hour','drop_day','pickup_hour','Dropoff_dt','pickup_day',
'Pickup_dt','Speed_mph','Trip_airport', 'Tip_amount','Pickup_latitude','Pickup_longitude','Dropoff_longitude'
,'Dropoff_latitude','VendorID'],axis=1)
cols = predict_data2.columns.tolist()
cols.insert(15, cols.pop(cols.index('tip_percentage')))
predict_data2 = predict_data2.reindex(columns= cols)

predict_data_arr = predict_data2.values

X1 = predict_data_arr[:,0:15]
Y1 = predict_data_arr[:,15]

X_train, X_test, y_train, y_test = train_test_split(X1, Y1, test_size=0.2, random_state=42)
lm = linear_model.LinearRegression()
lm.fit(X_train,y_train)

percent_predictions = lm.predict(X_test)

print("Mean Square Error of Linear Regression Model:",mean_squared_error(y_test, percent_predictions))
print("R-square Error of Linear Regression Model:",r2_score(y_test, percent_predictions))

colors = ['green','red']
plt.scatter(y_test,percent_predictions,c=colors,label=("True","Predicted"))
plt.xlabel('True values')
plt.ylabel('Predcited_values')
plt.title("Deviation of predicted values from true values of tip_percentages")
plt.show()
```

```
Mean Square Error of Linear Regression Model: 14.8474147029
R-square Error of Linear Regression Model: 0.799483370841
```

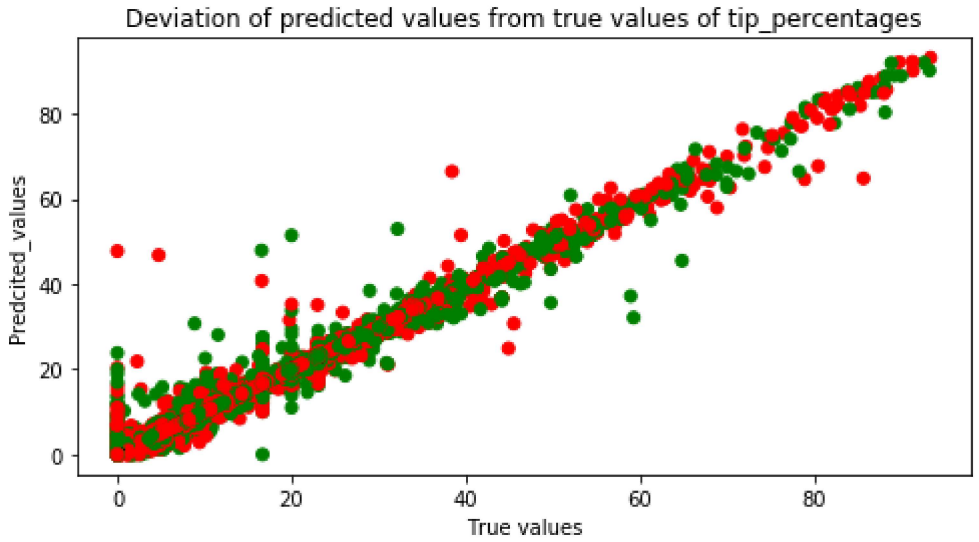


This model is trained and tested and produces a r-square error of 0.799. Though this score is just enough for a good model this willnot always work the same way in real world scenarios where we encounter large number of missing values and outliers. The linear model is prone to error due to bias as it assumes there exists a linear relationship between variables.

To further improve the performance of regression model, I used Random Forest Regression model to predict tip percentages. Random Forest Regression is a piecewise function. This method uses model emsembling technique where it combines outcome of multiple models. All base models are constructed independent of all other models. Eventually combining all these models to improve the performance. I mainly used Random Forest Regression as it is able to capture non-linear interaction between the features and the target.

In [29]:
```
# Random forest regressor
predict_data4 = predict_data2.copy()

predict_data4_arr = predict_data4.values
X3 = predict_data4_arr[:,0:15]
Y3 = predict_data4_arr[:,15]

X_train, X_test, y_train, y_test = train_test_split(X3, Y3, test_size=0.3, random_state=42)

rfr = RandomForestRegressor()
rfr.fit(X_train, y_train)
y_pred = rfr.predict(X_test)

print("Mean Square Error with RandomForest Regression: ",mean_squared_error(y_test, y_pred))
print("R-square Error with RandomForest Regression: ",r2_score(y_test, y_pred))

plt.scatter(y_test,y_pred,c=colors)
plt.xlabel('True values')
plt.ylabel('Predcited_values')
plt.title("Deviation of predicted values from true values of tip_percentages")
plt.show()
```

```
Mean Square Error with RandomForest Regression:  0.0828809359911
R-square Error with RandomForest Regression:  0.998881359178
```



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 best suits your particular skills and/or expertise. If you answer more than one, only the first will be scored.

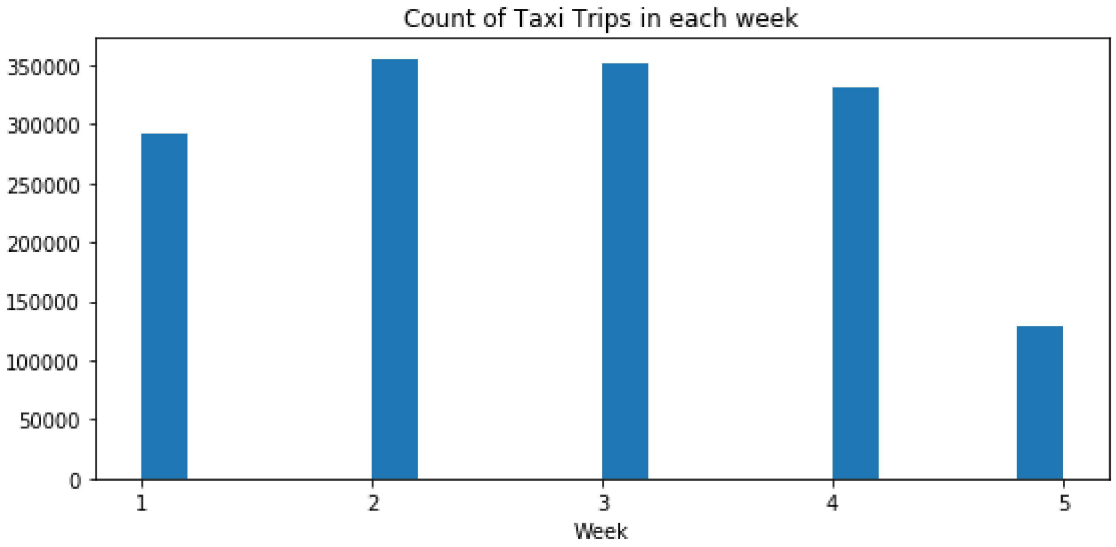
Option A: Distributions

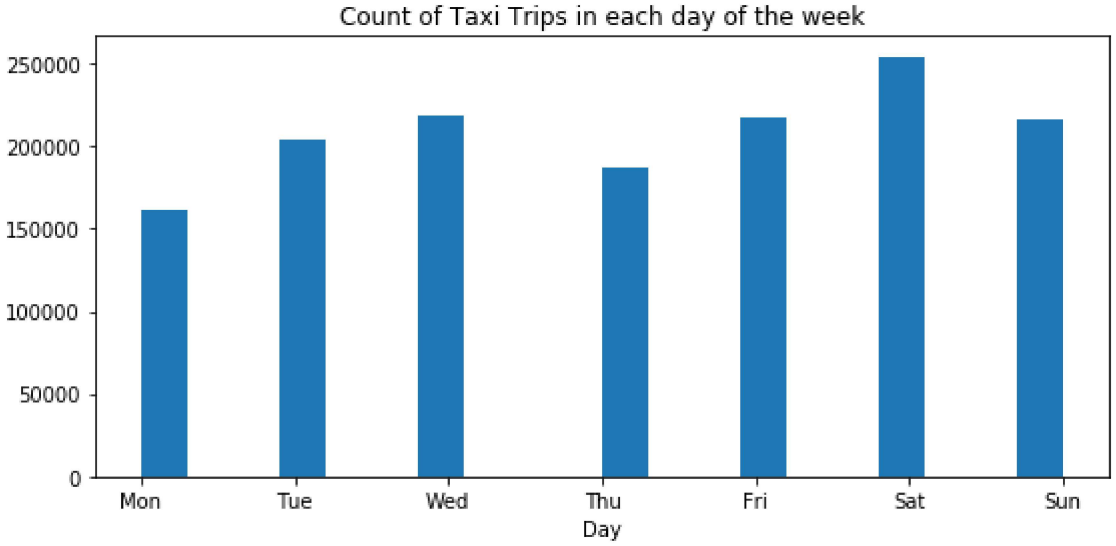
Build a derived variable representing the average speed over the course of a trip.

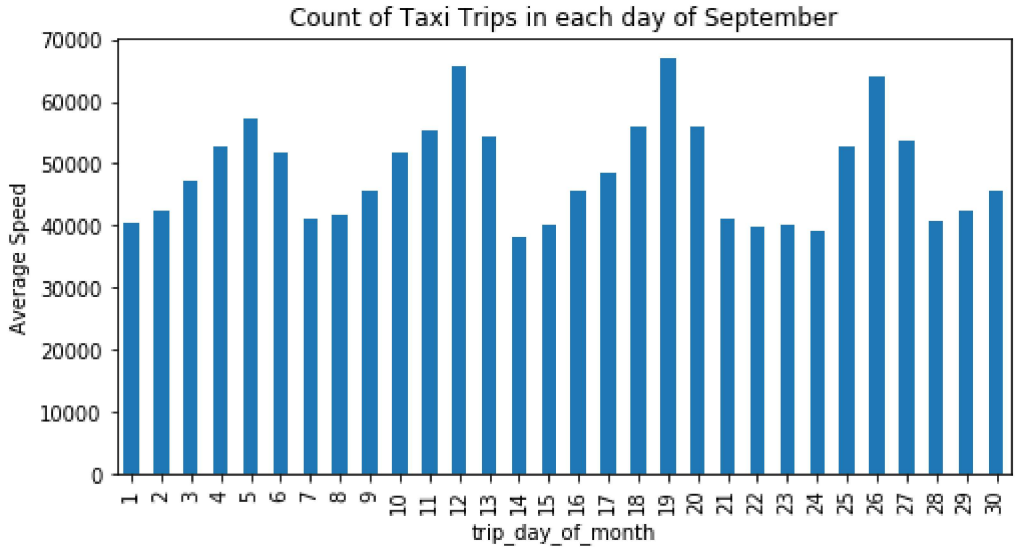
Can you perform a test to determine if the average trip speeds are materially the same in all weeks of September? If you decide they are not the same, can you form a hypothesis regarding why they differ?

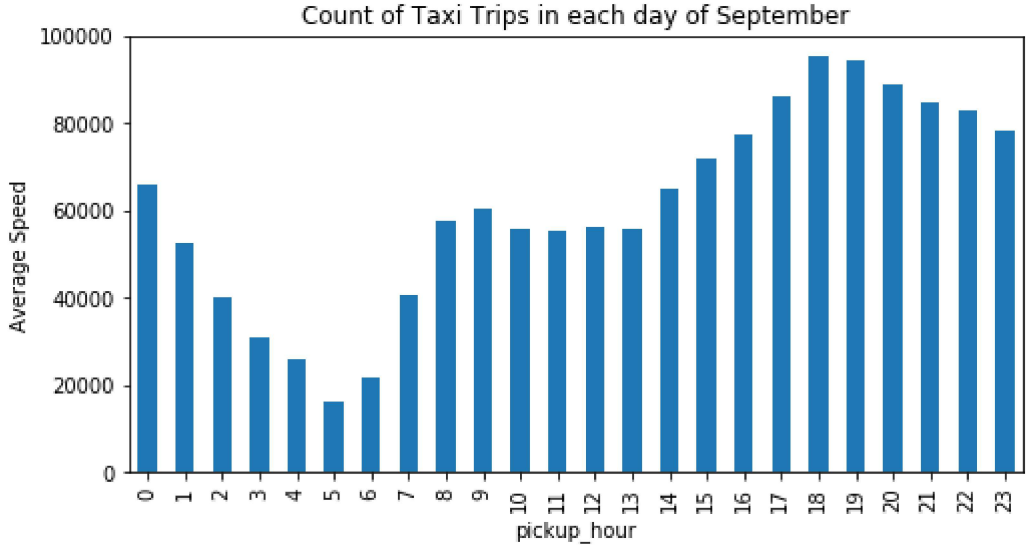
Can you build up a hypothesis of average trip speed as a function of time of day?

In [30]:
```
# Average Speed Analysis
# Question 5

df_copy = predict_data

df_copy["trip_day_of_month"] = df_copy["Pickup_dt"].dt.day

df_copy['Trip_week'].hist(bins = 20,grid=False)
plt.xlabel("Week")
plt.title("Count of Taxi Trips in each week")
plt.xticks(range(5),range(1,6))
plt.tight_layout()
plt.show()

df_copy['trip_day'].hist(bins = 20,grid=False)
plt.xlabel("Day")
plt.title("Count of Taxi Trips in each day of the week")
plt.xticks(range(7),("Mon","Tue","Wed","Thu","Fri","Sat","Sun"))
plt.tight_layout()
plt.show()

df_copy.groupby('trip_day_of_month').Speed.count().plot.bar()
plt.ylabel("Average Speed")
plt.title("Count of Taxi Trips in each day of September")
plt.xticks(range(30))
plt.show()

df_copy.groupby('pickup_hour').Speed.count().plot.bar()
plt.ylabel("Average Speed")
plt.title("Count of Taxi Trips in each day of September")
plt.xticks(range(24))
plt.show()
```


Count of Taxi Trips in each week
350000
300000
250000
200000
150000
100000
50000
0
1
2
3
4
5
Week


Count of Taxi Trips in each day of the week
250000
200000
150000
100000
50000
0
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Day


Count of Taxi Trips in each day of September
Average Speed
70000
60000
50000
40000
30000
20000
10000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
trip_day_of_month

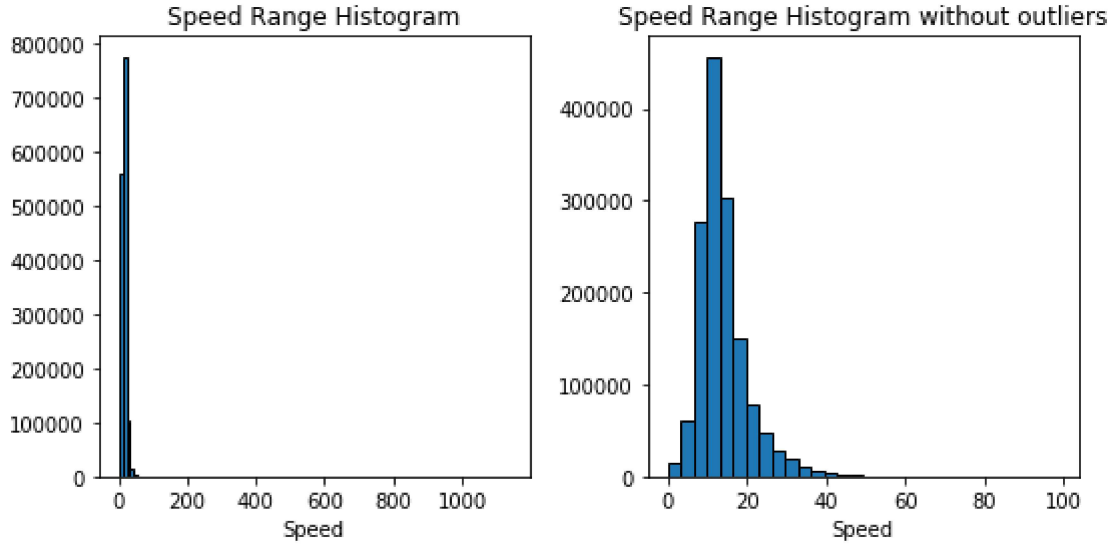

Count of Taxi Trips in each day of September
Average Speed
100000
80000
60000
40000
20000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
pickup_hour

Analysis of count of taxi trips

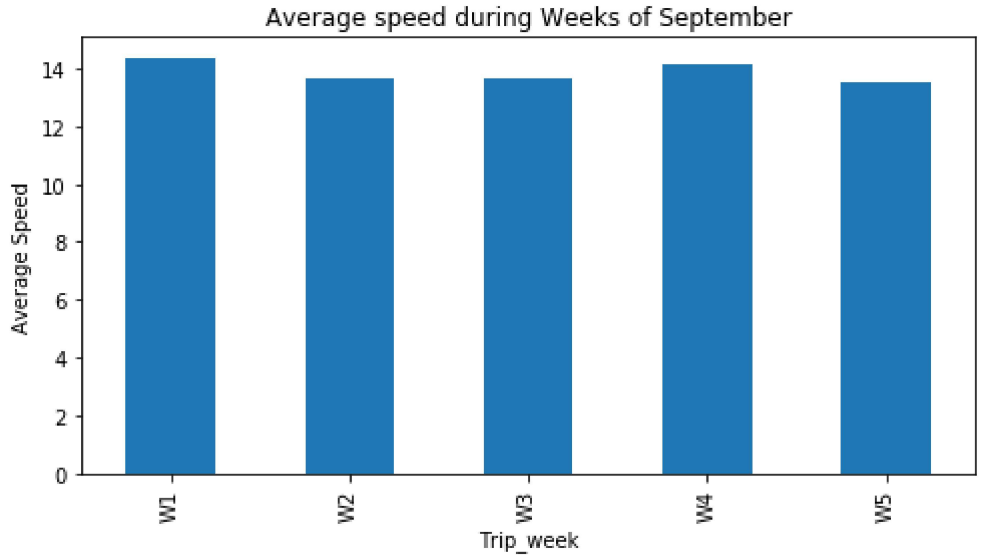
- The histogram of count of taxi trips per week shows mid of the month people prefer travelling in taxis while on month start and end there are lesser trips. This could be because people are broke towards the end of the month.
- Average speeds should be larger during start and end of the month as number of taxi trips is less , implies lesser traffic.
- The histograms of count of taxi trips shows that the number of taxi trips during weekends is significantly larger than other days of the week.
- The number of taxi trips during start of the week is significantly less than other days of the week.
- This shows that people prefer to hangout on the weekends while during the start of the week, people are serious about work and try to make arrangements like public transport for commute.
- According these findings, the average speed during weekends should be significantly lower than other days.

In [31]:
```
fig,axs = plt.subplots(1,2)

df_copy['Speed'].hist(bins=100,grid=False,ax=axs[0],edgecolor='black')
axs[0].set_xlabel("Speed")
axs[0].set_title("Speed Range Histogram")

print (df_copy[df_copy.Speed > 100].shape[0], "rows have speed greater than 100mph which is practically impos
sible in NYC")
# Remove data with speeds higher than 100
df_copy = df_copy[df_copy.Speed <= 100]

df_copy['Speed'].hist(bins=30,grid=False,ax=axs[1],edgecolor='black')
axs[1].set_xlabel("Speed")
axs[1].set_title("Speed Range Histogram without outliers")
plt.tight_layout()
plt.show()
```

```
246 rows have speed greater than 100mph which is practically impossible in NYC
```

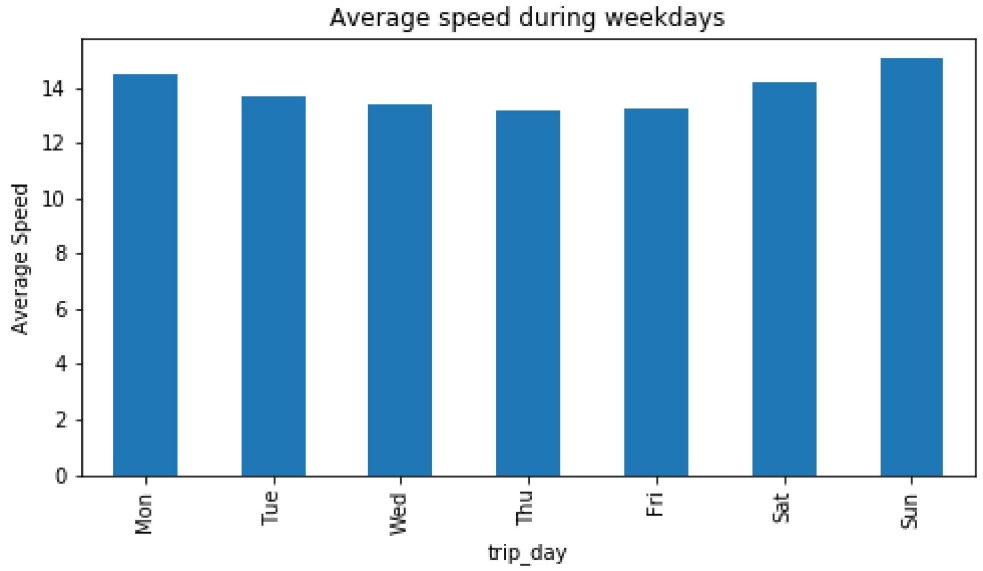


In [32]:
```
df_copy.groupby('Trip_week').Speed.mean().plot.bar()

plt.ylabel("Average Speed")
plt.title("Average speed during Weeks of September")
plt.xticks(range(5),("W1","W2","W3","W4","W5"))
plt.show()
```



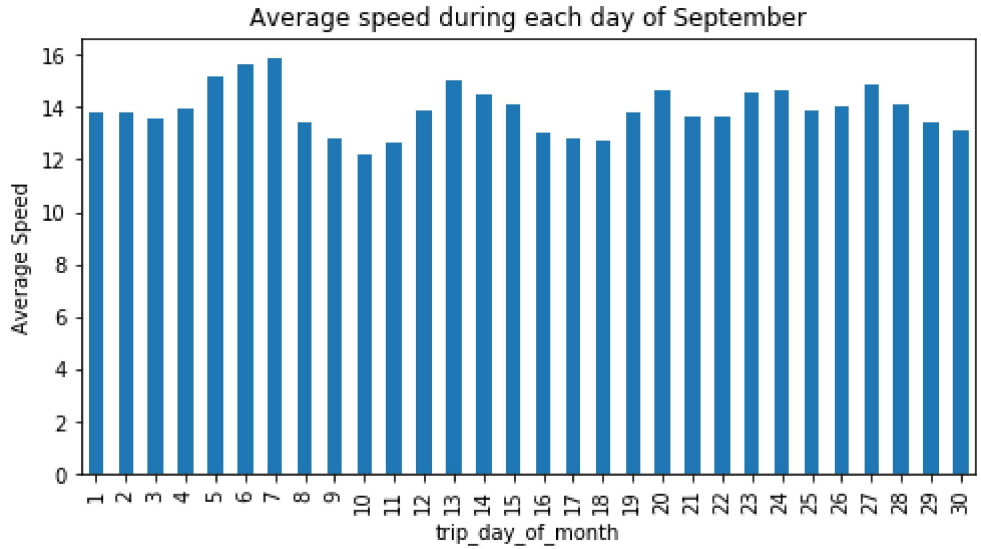
There is slight difference in average speeds per each week. We prove this statistically in later steps.

In [33]:
```
df_copy.groupby('trip_day').Speed.mean().plot.bar()
plt.ylabel("Average Speed")
plt.title("Average speed during weekdays")
plt.xticks(range(7),("Mon","Tue","Wed","Thu","Fri","Sat","Sun"))
plt.show()
```

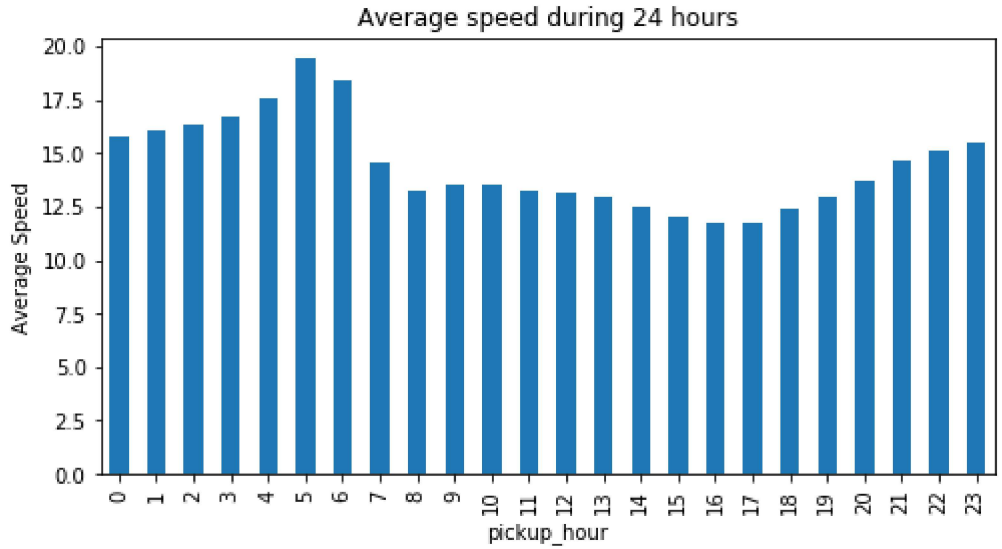


Average speed during weekends Fri-Sat is lesser due to more taxi trips.

In [34]:
```
df_copy.groupby('trip_day_of_month').Speed.mean().plot.bar()
plt.ylabel("Average Speed")
plt.title("Average speed during each day of September")
plt.xticks(range(0,30))
plt.show()
```



In [35]:
```
df_copy.groupby('pickup_hour').Speed.mean().plot.bar()

plt.ylabel("Average Speed")
plt.title("Average speed during 24 hours")
plt.xticks(range(0,24))
plt.show()
```



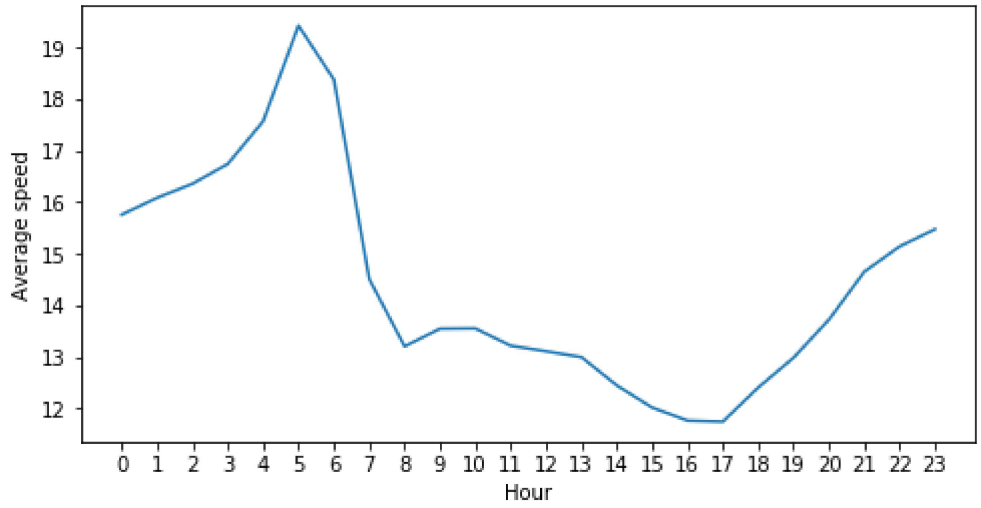
- The average speed during early hours of the day is high as there ususally wouldn't be any traffic.
- The average speed during evenings 3PM-7PM is low as people will generally leave from work and this will cause more traffic.

In [36]:
```
# Hypothesis - average trip speeds are materially the same in all weeks of September
# For testing average speeds across 5 weeks of September we will use ANOVA Testing
# Anova Testing - Analysis of Variance for comparing means of speeds through different weeks.
import scipy.stats as stats

W1 = df_copy[df_copy['Trip_week']==0].Speed.values
W2 = df_copy[df_copy['Trip_week']==1].Speed.values
W3 = df_copy[df_copy['Trip_week']==2].Speed.values
W4 = df_copy[df_copy['Trip_week']==3].Speed.values
W5 = df_copy[df_copy['Trip_week']==4].Speed.values

print (stats.f_oneway(W1,W2, W3,W4, W5))
```

```
F_onewayResult(statistic=1060.890118092, pvalue=0.0)
```

- The One way ANOVA test shows that p val is very low and test statistic value is very large.
- The F-statistic is simply a ratio of two variances. Variances are a measure of dispersion, or how far the data are scattered from the mean. Larger values represent greater dispersion of data between the groups.
- The probability that we want to calculate is the probability of observing an F-statistic that is at least as high as the value that our study obtained as test statistic. If the probability is low enough, we can conclude that our data is inconsistent with the null hypothesis.
- The p-value we calculated from F-test is significantly low implying that the null hypothesis is false.
- This shows that the difference of average speeds among different weeks is statistically true.
- This also implies week of the month effects the average speed of a trip.

In [37]:
```
from sklearn.linear_model import LinearRegression

speed_df = pd.DataFrame(df_copy.groupby('pickup_hour').Speed.mean().reset_index())

plt.plot(speed_df['pickup_hour'],speed_df['Speed'])
plt.xlabel("Hour")
plt.ylabel("Average speed")
plt.xticks(range(24))
plt.show()


X = speed_df['pickup_hour']
Y = speed_df['Speed']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)


lm = LinearRegression()
lm.fit(X_train.reshape(-1,1), y_train)
y_pred = lm.predict(X_test.reshape(-1,1))

print("Mean Square Error with Linear Regression: ",mean_squared_error(y_test, y_pred))
```




```
Mean Square Error with Linear Regression:  3.07708418151
```

In [38]:
```
print ("Intercept value: ",lm.intercept_," | Hour of the day coefficient: ",lm.coef_[0])
```

```
Intercept value:  16.7267743108  | Hour of the day coefficient:  -0.167371122331
```

From the above method the mathematical relation between avaerage speed and time of the day can be modelled as -

Average_speed = (Hour of the day coefficient) * Hour_of_the_day + Intercept value

which will be,
Avg_speed = (lm.coef_[0] * time_of_the_day) + lm.intercept_

Avg_speed = (-0.168499510409 * time_of_the_day) + 16.7833981947

** time_of_the_day = hour_of_the_day