# Estimating Travel Time

The objective of this document is proposing a prediction model for estimating the travel time of two specified locations at a given departure time. The main idea here is predicting the velocity of the trip. Given the distance between starting and ending point of the trip, it is possible to easily compute the Travel Time. According to the given data, different features including the time of the day, day of the week, month, travel distance, and distance to the center of the city (New York) are used. Different prediction models (Linear, GLM and Deep Neural Network) are compared, and the GLM is used for genrating the final results.

# Preparation

Import required libraries

In [136]:
```
import numpy as np
import pandas as pd
from geopy.distance import vincenty
from datetime import datetime
from datetime import timedelta
from datetime import time

import statsmodels.api as sm



from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.cross_validation import KFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error


import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns



from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers.normalization import BatchNormalization


%matplotlib inline
```

# Reading data

In [169]:
```
df_train = pd.read_csv('train.csv',index_col= 'row_id')
df_test  = pd.read_csv('test.csv',index_col= 'row_id')
df_train.head()
```

```
/Users/z002df6/anaconda/lib/python3.6/site-packages/numpy/lib/arrayseto
ps.py:463: FutureWarning: elementwise comparison failed; returning scal
ar instead, but in the future will perform elementwise comparison
  mask |= (ar1 == a)
```

Out[169]:

| | start_lng | start_lat | end_lng | end_lat | start_timestamp | duration |
|---|---|---|---|---|---|---|
| **row_id** | | | | | | |
| **0** | -74.009087 | 40.713818 | -74.004326 | 40.719986 | 1420950819 | 112 |
| **1** | -73.971176 | 40.762428 | -74.004181 | 40.742653 | 1420950819 | 1159 |
| **2** | -73.994957 | 40.745079 | -73.999939 | 40.734650 | 1421377541 | 281 |
| **3** | -73.991127 | 40.750080 | -73.988609 | 40.734890 | 1421377542 | 636 |
| **4** | -73.945511 | 40.773724 | -73.987434 | 40.755707 | 1422173586 | 705 |

# Feature engineering

It is clear that the travel time of trip depends on the starting and ending point. In other words, the most uncertain component in the prediction of travel time is the velocity of the trip. Given the velocity and the distance, it is easy to compute the duration of the travel.

Also, I observed all travels in both train and test dataset are happening around New York City. Therefore, the main component in determining the velocity of is the city traffic. We know that traffic is a time-dependent phenomenon which depends on the time of the day, the day of the week, and month of the year. In addition, the traffic is usually heavier in Manhattan (downtown of the city) in comparing to the other point of the city. Therefore, if the starting or ending point of the travel is close to the Manhattan we expect higher traffic comparing to the other neighborhoods. In visualization section, I provide enough evidence from the data set to support the aforementioned claims.

According to this observation the following features are computted by using the raw data and added to the dataframe.

- Distance between starting and ending computted by vincenty formula
- The time of the day of travel (in sec far from the midnight)
- The day of the week (Monday, Tuesday, etc). For this categorical data, six dummy variables are added to datafram
- The month of the travel to capture seasnolity effect.
- The sequare of distance
- The velocity is used as the predication variable.

In [156]:
```
def distance(row):
    source = (row['start_lat'], row['start_lng'])
    dest = ( row['end_lat'], row['end_lng'])
    return vincenty(source,dest).miles


Manhattan = (40.7831, -73.9712)
def pickup_to_MH(row):
    '''find the distance between pick up point and Manhattan center'''
    source = (row['start_lat'], row['start_lng'])
    return vincenty(source,Manhattan).miles

def dropoff_to_MH(row):
    '''find the distance between dropoff point and Manhattan center'''
    dest = ( row['end_lat'], row['end_lng'])
    return vincenty(dest,Manhattan).miles

def day_of_week(ep):
    return datetime.fromtimestamp(ep).strftime("%A")


def month(ep):
    return datetime.fromtimestamp(ep).month

def time_of_day(ep):
    ref = datetime(2015, 1, 1, 0, 0, 0)
    sec = (datetime.fromtimestamp(ep)- ref).seconds
    return min(sec, 86400- sec)

def year(ep):
    return datetime.fromtimestamp(ep).year

def add_features(df_train_s):

    # Add day of the week and the dummy variable
    DD  = df_train_s['start_timestamp'].map(day_of_week)
    df_train_s['day'] = DD

    DD  = pd.get_dummies( DD,prefix='day', drop_first=True)
    df_train_s = pd.concat([df_train_s, DD],axis =1 )

    # Month, time of the dat, df_train_s
    df_train_s['month']  = df_train_s['start_timestamp'].map(month)
    df_train_s['time_of_day']  = df_train_s['start_timestamp'].map(time_
of_day)

    # distance between start and end of the trip
    df_train_s['distance']   = df_train_s.apply(lambda x :distance(x), a
xis=1 )
    df_train_s['distance2'] = df_train_s['distance']**2

    # distance between start, end, and center of Manhatan
    df_train_s['pickup_MH']  = df_train_s.apply(pickup_to_MH, axis=1 )
    df_train_s['dropoff_MH'] = df_train_s.apply(dropoff_to_MH, axis=1 )
    return df_train_s
```

Now, we can easily add all of the above features to both traing and test data set. Due to time limtation and calculation power I only used 10% of the traing data.

In [24]:
```
np.random.seed(42)
df_train_s = df_train.sample(frac=0.01, replace=False)
df_train_s = add_features(df_train_s)
df_train_s['velocity'] = np.array(df_train_s['distance']/(df_train_s['du
ration']/3600))
```

In [25]:
```
df_train_s.head()
```

Out[25]:

| | start_lng | start_lat | end_lng | end_lat | start_timestamp | duration | day |
|---|---|---|---|---|---|---|---|
| **row_id** | | | | | | | |
| **9780992** | -73.989853 | 40.755650 | -74.183144 | 40.687981 | 1443634595 | 1415 | Wedne |
| **2996891** | -73.958855 | 40.774952 | -73.968918 | 40.763908 | 1425321185 | 646 | Monda |
| **937249** | -73.987465 | 40.749176 | -74.005402 | 40.727180 | 1420942464 | 762 | Saturd |
| **4483011** | -73.965347 | 40.774792 | -73.964058 | 40.770973 | 1428517239 | 164 | Wedne |
| **1285264** | -74.006622 | 40.744011 | -74.008812 | 40.704350 | 1423735890 | 715 | Thursd |

In [170]:
```
# adding the feature to test set.
df_test = add_features(df_test)
```

## Removing Outlires

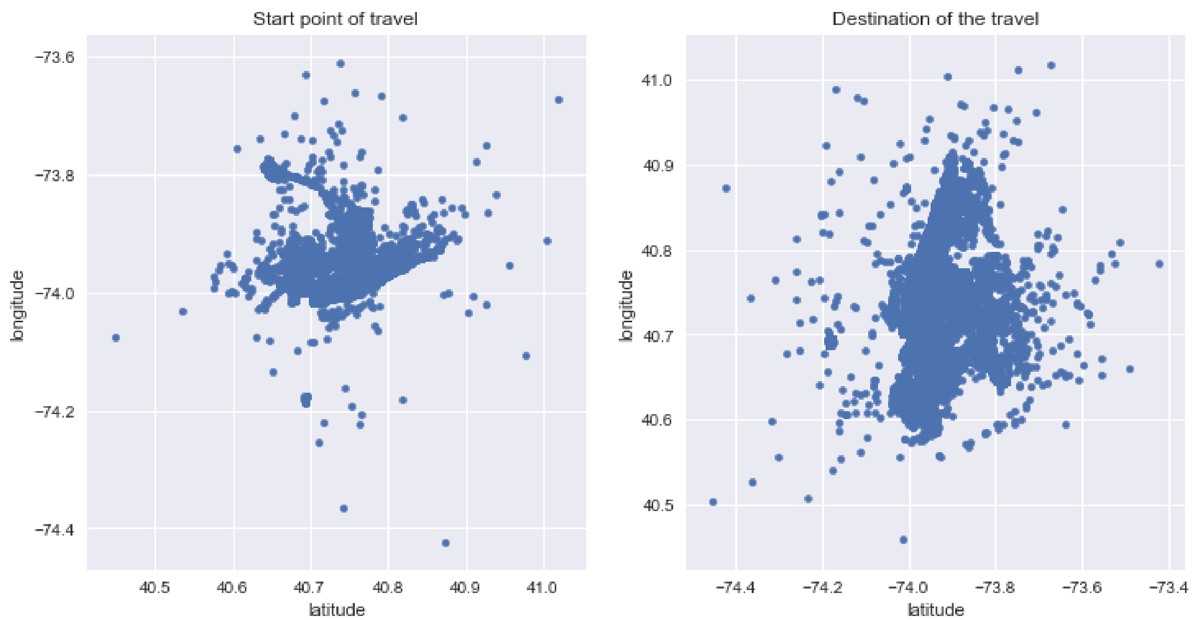
The following functions are used to compute these features. Considering the speed limit and the fact the usual trafic in New York, it is reseanable to assume that always the speed show not exceed 90 mph. Therefore, I remove the points with more than this number as the outlires. Also, I removed the data with less than .5 mph. Specificlly, there exists many samples with zero distance between starting and ending point which might happen becouse GPS problem.

In [41]:
```
df_train_s = df_train_s[df_train_s['velocity']<90]
df_train_s = df_train_s[df_train_s['velocity']>.5]
```

## Data Visulazation

First we look at the starting and ending point of the trips which happens in New York.

In [30]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

ax = df_train_s.plot.scatter( 'start_lat','start_lng',
                             ax = axes[0],
                             title='Start point of travel')
ax.set(xlabel="latitude", ylabel='longitude')
ax = df_train_s.plot.scatter('end_lng','end_lat',
                        ax = axes[1],
                        title='Destination of the travel')
ax.set(xlabel="latitude", ylabel='longitude')
plt.show()
```




Here are some statitcs about the volacity, distance of each trip and its duration. Also, we looked at the density function of the volacity. A log-normal or Gamma distribution are approprate candiatdes for this distribution.

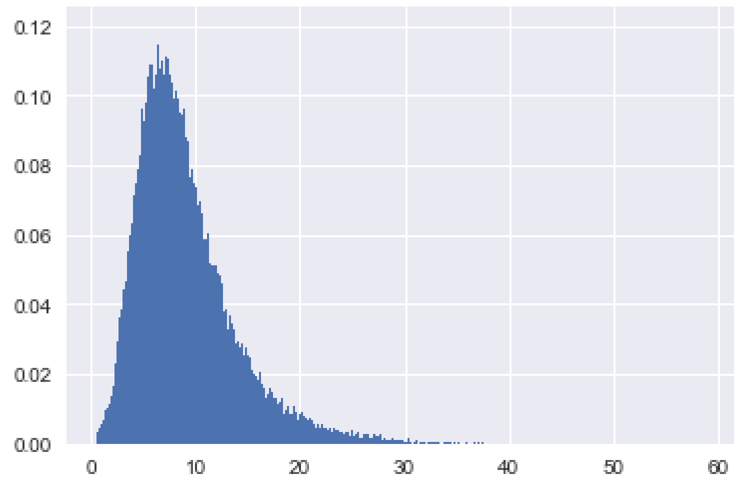
In [42]:
```
df_train_s[['distance', 'duration','velocity']].describe()
```

Out[42]:

| | distance | duration | velocity |
|---|---|---|---|
| **count** | 128036.000000 | 128036.000000 | 128036.000000 |
| **mean** | 2.169607 | 844.237621 | 9.091566 |
| **std** | 2.427614 | 668.033514 | 4.828527 |
| **min** | 0.000420 | 1.000000 | 0.500455 |
| **25%** | 0.791981 | 404.000000 | 5.766028 |
| **50%** | 1.342186 | 669.000000 | 8.094952 |
| **75%** | 2.455935 | 1081.000000 | 11.294384 |
| **max** | 29.962159 | 25693.000000 | 58.945106 |

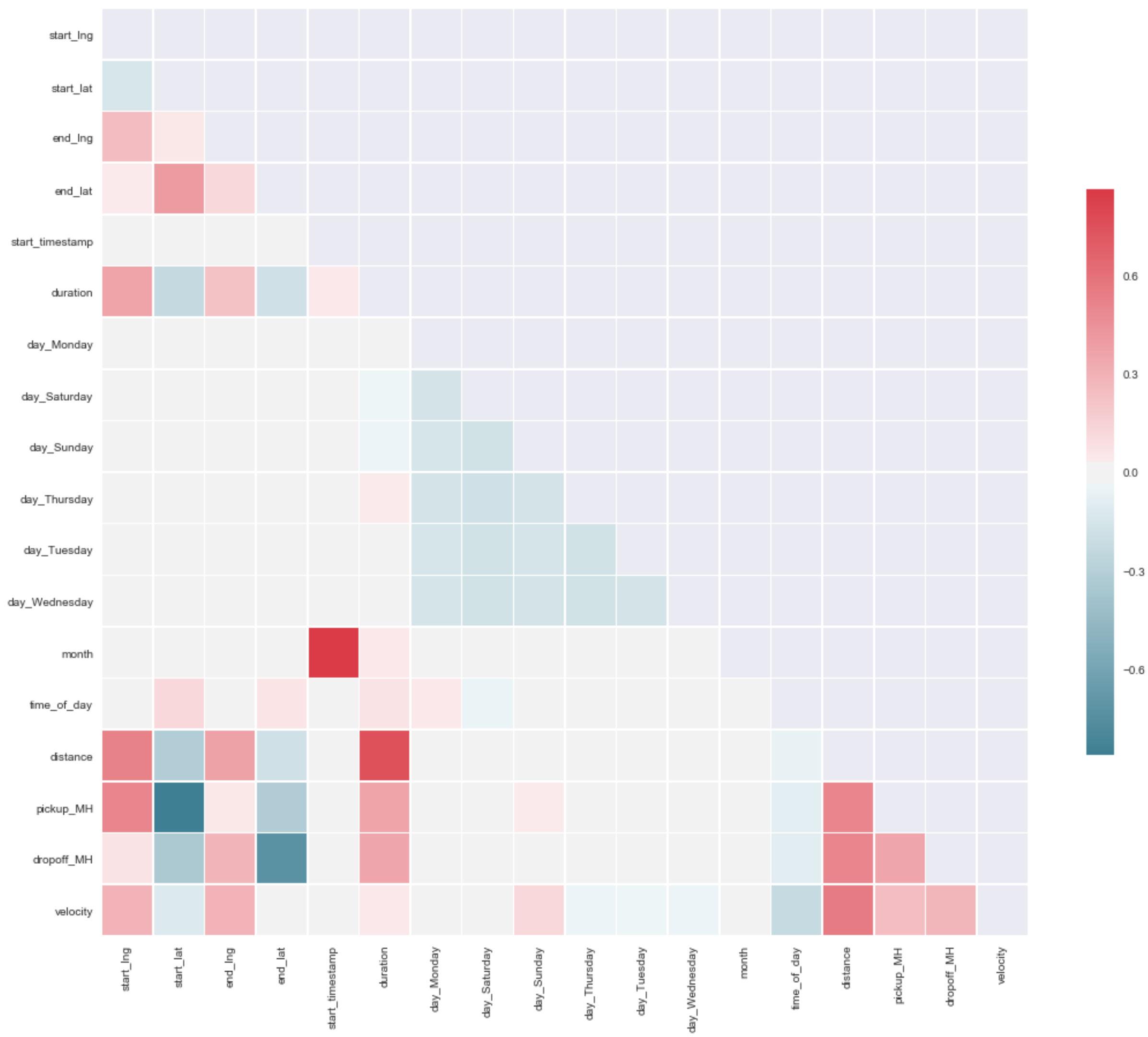
In [43]:
```
df_train_s['velocity'].hist(bins=1000,normed=True)
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x12a992978>



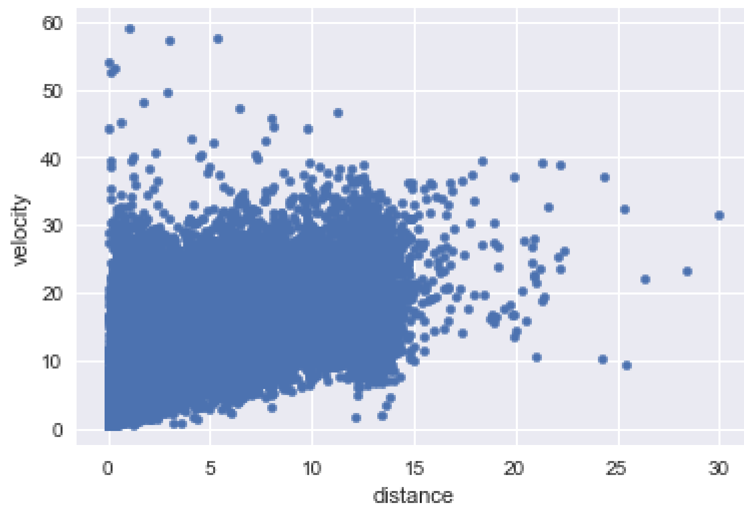

## Corrolation matrix

In [44]:
```
corr = df_train_s.corr()

# generate a mask for the lower triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# set up the matplotlib figure
f, ax = plt.subplots(figsize=(18, 18))

# generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3,
            square=True,
            linewidths=.5, cbar_kws={"shrink": .5}, ax=ax)

plt.show()
```



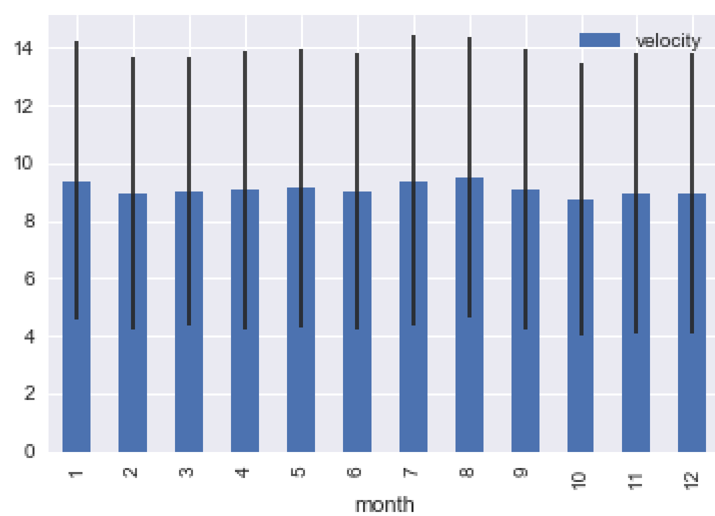
In [53]:
```
df_train_s.plot.scatter( 'distance','velocity')
```

Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x1338054a8>




In [48]:
```
### Seanility and time Effect on Velocity
gr= df_train_s[['velocity','month']].groupby(by='month')
gr.mean().plot.bar(yerr=gr.std())
```

Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x1340c4eb8>




## Data preprocessing

Let's split our data to train and test set in fraction of $\frac{4}{1}$ to facilate comparing the results. This test set is differenet from the given test set.

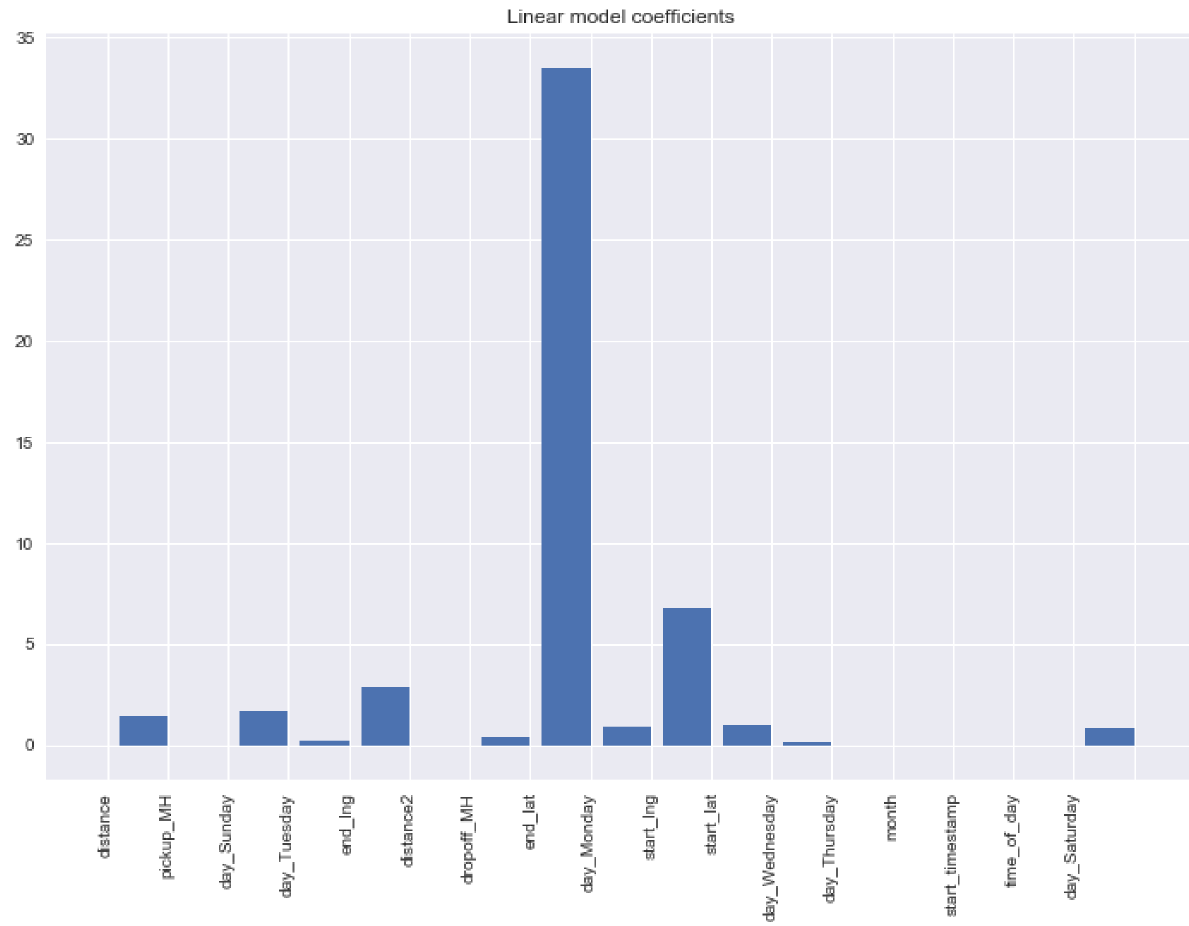
In [105]:
```
cl = list(set(df_train_s.keys())-{'velocity','duration','day'})
X  = np.array(df_train_s[cl])
X1 = np.insert(X, 0, 1, axis=1)
y = np.array(df_train_s['velocity'])


X_train, X_test, y_train, y_test = train_test_split(X1, y, test_size=0.2
, random_state=42)

dist_train = X_train[:,1]
dist_test  = X_test[:,1]
```

In [106]:
```
list(enumerate(cl))
dist_train.mean()
```

Out[106]: 2.1668461824987508

# Linear Model

In [204]:
```
model_sk = LinearRegression()
model_sk.fit(X_train, y_train)

plt.figure(figsize=(12, 8))
plt.bar(np.arange(model_sk.coef_.shape[0]) - 0.4, model_sk.coef_)
plt.xticks(np.arange(model_sk.coef_.shape[0]), cl, rotation='vertical')
plt.xlim([-1, model_sk.coef_.shape[0]])
plt.title("Linear model coefficients")
plt.show()
```




The folling chart also provide better understading. Excepet X12 (dummy for sunday) all the other variables are significant; the p-value is zero and null-hypothesis is rejected.

In [205]:
```
linear_model = sm.OLS(y_train, X_train)
linear_results = linear_model.fit()
print(linear_results.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
Dep. Variable:                      y   R-squared:
  0.398
Model:                            OLS   Adj. R-squared:
  0.398
Method:                 Least Squares   F-statistic:
  3980.
Date:                Thu, 28 Dec 2017   Prob (F-statistic):
   0.00
Time:                        08:56:07   Log-Likelihood:                   -2.8
052e+05
No. Observations:              102428   AIC:                                5.
611e+05
Df Residuals:                  102410   BIC:                                5.
612e+05
Df Model:                          17

Covariance Type:            nonrobust

==============================================================================
=======
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-------
const       -685.1248     83.582     -8.197      0.000    -848.945    -
521.305
x1             1.4070      0.014    103.644      0.000       1.380
  1.434
x2            -0.0323      0.017     -1.918      0.055      -0.065
  0.001
x3             1.6923      0.044     38.571      0.000       1.606
  1.778
x4             0.2024      0.044      4.639      0.000       0.117
  0.288
x5             2.8453      0.407      6.998      0.000       2.048
  3.642
x6            -0.0436      0.001    -39.441      0.000      -0.046
-0.041
x7             0.4158      0.011     38.292      0.000       0.395
  0.437
x8            33.5198      0.662     50.657      0.000      32.223
34.817
x9             0.8994      0.045     20.201      0.000       0.812
  0.987
x10            6.7762      0.537     12.609      0.000       5.723
  7.830
x11            1.0109      1.158      0.873      0.383      -1.258
  3.280
x12            0.0904      0.043      2.090      0.037       0.006
  0.175
x13           -0.0179      0.043     -0.420      0.674      -0.102
  0.066
x14           -0.0305      0.041     -0.747      0.455      -0.111
```

```
  0.050
x15         -2.58e-09   1.55e-08     -0.166      0.868    -3.3e-08
2.78e-08
x16        -6.837e-05   9.26e-07    -73.847      0.000   -7.02e-05   -
6.66e-05
x17            0.8166      0.043     19.188      0.000       0.733
  0.900
==============================================================================
=======
Omnibus:                    30504.903   Durbin-Watson:
  1.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           140
710.310
Skew:                           1.387   Prob(JB):
   0.00
Kurtosis:                       8.027   Cond. No.
1.03e+13
==============================================================================
=======

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.03e+13. This might indicate that t
here are
strong multicollinearity or other numerical problems.
```

## Generalized Linear Model

I tried GLM with gamma fammaly.

In [206]:
```
gamma_model = sm.GLM( y_train, X_train,family=sm.families.Gamma())
gamma_results = gamma_model.fit()
print(gamma_results.summary())
```

```
/Users/z002df6/anaconda/lib/python3.6/site-packages/statsmodels/genmod/
generalized_linear_model.py:244: DomainWarning: The inverse_power link
function does not respect the domain of the Gamma family.
  DomainWarning)
```

```
                 Generalized Linear Model Regression Results
==========================================================================
=======
Dep. Variable:                      y   No. Observations:
102428
Model:                            GLM   Df Residuals:
102414
Model Family:                   Gamma   Df Model:
     13
Link Function:          inverse_power   Scale:                      0.1944
2760013
Method:                          IRLS   Log-Likelihood:               -2.8
026e+05
Date:                Thu, 28 Dec 2017   Deviance:
21788.
Time:                        08:56:25   Pearson chi2:
1.99e+04
No. Iterations:                   100

==========================================================================
=======
                 coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------
-------
const          6.0060      0.345     17.384      0.000       5.329
  6.683
x1            -0.0146   9.13e-05   -159.591      0.000      -0.015
-0.014
x2            -0.0003   7.74e-05     -3.666      0.000      -0.000
-0.000
x3            -0.0203      0.000    -47.881      0.000      -0.021
-0.019
x4            -0.0027      0.000     -5.578      0.000      -0.004
-0.002
x5            -0.0393      0.002    -20.029      0.000      -0.043
-0.035
x6             0.0006   5.29e-06    119.167      0.000       0.001
  0.001
x7            -0.0022   4.74e-05    -45.802      0.000      -0.002
-0.002
x8            -0.1930      0.002   -109.229      0.000      -0.196
-0.190
x9            -0.0113      0.000    -24.825      0.000      -0.012
-0.010
x10           -0.0438      0.003    -14.727      0.000      -0.050
-0.038
x11           -0.0915      0.005    -18.119      0.000      -0.101
-0.082
x12           -0.0008      0.001     -1.572      0.116      -0.002
  0.000
x13            0.0006      0.001      1.054      0.292      -0.000
  0.002
x14            0.0012      0.000      7.548      0.000       0.001
  0.001
x15        -2.997e-10   5.78e-11     -5.184      0.000   -4.13e-10   -
```

```
1.86e-10
x16          8.222e-07   8.37e-09     98.194      0.000    8.06e-07
8.39e-07
x17            -0.0114      0.000    -26.469      0.000      -0.012
-0.011
======================================================================
=======
```

## Deep Neural Network (DNN)

Here, I am useing a DNN as a prediction model. I am using the Keras package to train the network. Network includes 3 layers. Also, between each two layer a dropout layer is add. RELU and softmax are used as the activation functions. Here, I define the model.

I normilized the data the input data to imporve the performance.

In [195]:
```
DNN_model = Sequential()
DNN_model.add(Dense(100,input_dim=X_train.shape[1],init='uniform',activa
tion='relu'))
DNN_model.add(Dropout(0.5))
DNN_model.add(Dense(50,init='uniform',activation='softmax'))
DNN_model.add(Dropout(0.5))
DNN_model.add(Dense(100,init='uniform',activation='relu'))
DNN_model.add(Dropout(0.5))
DNN_model.add(Dense(1,init='uniform',activation='relu'))

DNN_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 100)               1900
_________________________________________________________________
dropout_12 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_19 (Dense)             (None, 50)                5050
_________________________________________________________________
dropout_13 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_20 (Dense)             (None, 100)               5100
_________________________________________________________________
dropout_14 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_21 (Dense)             (None, 1)                 101
=================================================================
Total params: 12,151
Trainable params: 12,151
Non-trainable params: 0
_________________________________________________________________

/Users/z002df6/anaconda/lib/python3.6/site-packages/ipykernel_launcher.
py:2: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense
(100, input_dim=18, activation="relu", kernel_initializer="uniform")`

/Users/z002df6/anaconda/lib/python3.6/site-packages/ipykernel_launcher.
py:4: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense
(50, activation="softmax", kernel_initializer="uniform")`
  after removing the cwd from sys.path.
/Users/z002df6/anaconda/lib/python3.6/site-packages/ipykernel_launcher.
py:6: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense
(100, activation="relu", kernel_initializer="uniform")`

/Users/z002df6/anaconda/lib/python3.6/site-packages/ipykernel_launcher.
py:8: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense
(1, activation="relu", kernel_initializer="uniform")`
```

## Fitting the DNN

In [196]:
```
mn = X1.mean(axis=0)
#model.compile(loss='mean_absolute_error',optimizer='adam',metrics='[acc
uracy]')
DNN_model.compile(loss='mean_absolute_error',optimizer='adam')
history = DNN_model.fit(X_train/mn,y_train,
                    validation_data=(X_test/mn, y_test),
                    epochs =100,
                    batch_size=100,
                    verbose=2)
```

```
Train on 102428 samples, validate on 25608 samples
Epoch 1/100
2s - loss: 3.8939 - val_loss: 2.8545
Epoch 2/100
2s - loss: 3.0103 - val_loss: 2.8045
Epoch 3/100
2s - loss: 2.9702 - val_loss: 2.7933
Epoch 4/100
2s - loss: 2.9282 - val_loss: 2.7548
Epoch 5/100
2s - loss: 2.9222 - val_loss: 2.7306
Epoch 6/100
2s - loss: 2.9013 - val_loss: 2.7235
Epoch 7/100
2s - loss: 2.8852 - val_loss: 2.7172
Epoch 8/100
2s - loss: 2.8784 - val_loss: 2.7125
Epoch 9/100
2s - loss: 2.8572 - val_loss: 2.6998
Epoch 10/100
2s - loss: 2.8461 - val_loss: 2.6974
Epoch 11/100
2s - loss: 2.8372 - val_loss: 2.7010
Epoch 12/100
2s - loss: 2.8295 - val_loss: 2.6925
Epoch 13/100
2s - loss: 2.8248 - val_loss: 2.6924
Epoch 14/100
2s - loss: 2.8133 - val_loss: 2.6827
Epoch 15/100
2s - loss: 2.8123 - val_loss: 2.6869
Epoch 16/100
2s - loss: 2.7984 - val_loss: 2.6885
Epoch 17/100
2s - loss: 2.7926 - val_loss: 2.6851
Epoch 18/100
2s - loss: 2.7892 - val_loss: 2.6849
Epoch 19/100
2s - loss: 2.7831 - val_loss: 2.6757
Epoch 20/100
2s - loss: 2.7786 - val_loss: 2.6751
Epoch 21/100
2s - loss: 2.7692 - val_loss: 2.6810
Epoch 22/100
2s - loss: 2.7685 - val_loss: 2.6736
Epoch 23/100
2s - loss: 2.7593 - val_loss: 2.6658
Epoch 24/100
2s - loss: 2.7583 - val_loss: 2.6657
Epoch 25/100
2s - loss: 2.7538 - val_loss: 2.6670
Epoch 26/100
2s - loss: 2.7433 - val_loss: 2.6676
Epoch 27/100
2s - loss: 2.7470 - val_loss: 2.6666
Epoch 28/100
2s - loss: 2.7444 - val_loss: 2.6661
```

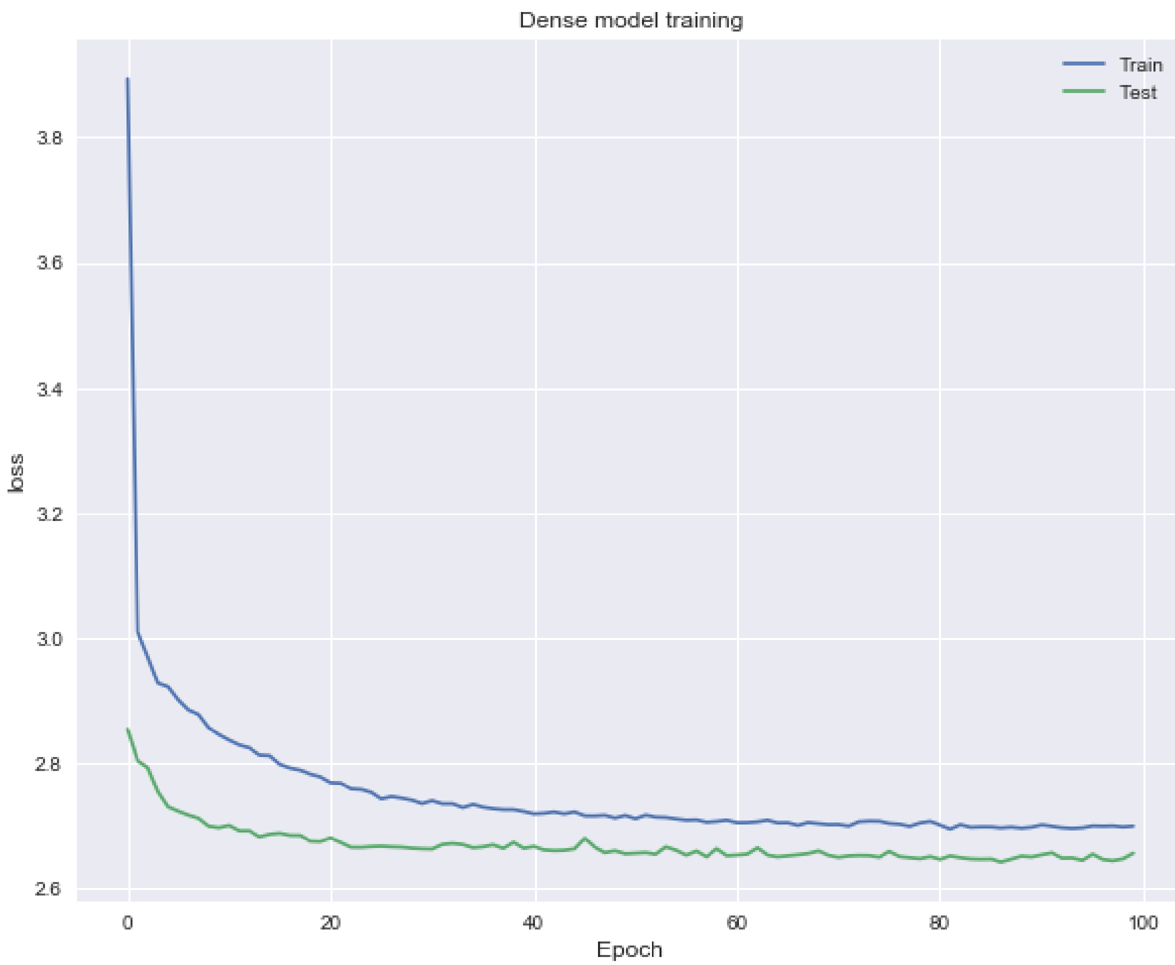
```
Epoch 29/100
2s - loss: 2.7412 - val_loss: 2.6643
Epoch 30/100
2s - loss: 2.7357 - val_loss: 2.6634
Epoch 31/100
2s - loss: 2.7403 - val_loss: 2.6630
Epoch 32/100
2s - loss: 2.7355 - val_loss: 2.6705
Epoch 33/100
2s - loss: 2.7355 - val_loss: 2.6722
Epoch 34/100
2s - loss: 2.7294 - val_loss: 2.6703
Epoch 35/100
2s - loss: 2.7345 - val_loss: 2.6651
Epoch 36/100
2s - loss: 2.7301 - val_loss: 2.6666
Epoch 37/100
2s - loss: 2.7277 - val_loss: 2.6698
Epoch 38/100
2s - loss: 2.7262 - val_loss: 2.6641
Epoch 39/100
2s - loss: 2.7261 - val_loss: 2.6739
Epoch 40/100
2s - loss: 2.7232 - val_loss: 2.6643
Epoch 41/100
2s - loss: 2.7195 - val_loss: 2.6672
Epoch 42/100
2s - loss: 2.7203 - val_loss: 2.6617
Epoch 43/100
2s - loss: 2.7225 - val_loss: 2.6604
Epoch 44/100
2s - loss: 2.7195 - val_loss: 2.6610
Epoch 45/100
2s - loss: 2.7225 - val_loss: 2.6634
Epoch 46/100
2s - loss: 2.7163 - val_loss: 2.6803
Epoch 47/100
2s - loss: 2.7161 - val_loss: 2.6661
Epoch 48/100
2s - loss: 2.7171 - val_loss: 2.6573
Epoch 49/100
2s - loss: 2.7128 - val_loss: 2.6603
Epoch 50/100
2s - loss: 2.7169 - val_loss: 2.6554
Epoch 51/100
2s - loss: 2.7117 - val_loss: 2.6563
Epoch 52/100
2s - loss: 2.7174 - val_loss: 2.6573
Epoch 53/100
2s - loss: 2.7143 - val_loss: 2.6546
Epoch 54/100
2s - loss: 2.7137 - val_loss: 2.6665
Epoch 55/100
2s - loss: 2.7114 - val_loss: 2.6611
Epoch 56/100
2s - loss: 2.7093 - val_loss: 2.6533
Epoch 57/100
```

```
2s - loss: 2.7099 - val_loss: 2.6598
Epoch 58/100
2s - loss: 2.7061 - val_loss: 2.6506
Epoch 59/100
2s - loss: 2.7074 - val_loss: 2.6636
Epoch 60/100
2s - loss: 2.7094 - val_loss: 2.6524
Epoch 61/100
2s - loss: 2.7053 - val_loss: 2.6534
Epoch 62/100
2s - loss: 2.7055 - val_loss: 2.6549
Epoch 63/100
2s - loss: 2.7068 - val_loss: 2.6650
Epoch 64/100
2s - loss: 2.7095 - val_loss: 2.6534
Epoch 65/100
2s - loss: 2.7051 - val_loss: 2.6506
Epoch 66/100
2s - loss: 2.7053 - val_loss: 2.6522
Epoch 67/100
2s - loss: 2.7011 - val_loss: 2.6539
Epoch 68/100
2s - loss: 2.7057 - val_loss: 2.6559
Epoch 69/100
2s - loss: 2.7039 - val_loss: 2.6600
Epoch 70/100
2s - loss: 2.7024 - val_loss: 2.6531
Epoch 71/100
2s - loss: 2.7025 - val_loss: 2.6498
Epoch 72/100
2s - loss: 2.7000 - val_loss: 2.6521
Epoch 73/100
2s - loss: 2.7069 - val_loss: 2.6528
Epoch 74/100
2s - loss: 2.7079 - val_loss: 2.6526
Epoch 75/100
2s - loss: 2.7077 - val_loss: 2.6500
Epoch 76/100
2s - loss: 2.7042 - val_loss: 2.6593
Epoch 77/100
2s - loss: 2.7030 - val_loss: 2.6510
Epoch 78/100
3s - loss: 2.6996 - val_loss: 2.6492
Epoch 79/100
2s - loss: 2.7050 - val_loss: 2.6476
Epoch 80/100
2s - loss: 2.7074 - val_loss: 2.6510
Epoch 81/100
2s - loss: 2.7016 - val_loss: 2.6465
Epoch 82/100
2s - loss: 2.6952 - val_loss: 2.6521
Epoch 83/100
2s - loss: 2.7023 - val_loss: 2.6492
Epoch 84/100
2s - loss: 2.6981 - val_loss: 2.6472
Epoch 85/100
2s - loss: 2.6987 - val_loss: 2.6467
```

```
Epoch 86/100
2s - loss: 2.6987 - val_loss: 2.6471
Epoch 87/100
2s - loss: 2.6971 - val_loss: 2.6423
Epoch 88/100
2s - loss: 2.6985 - val_loss: 2.6469
Epoch 89/100
2s - loss: 2.6967 - val_loss: 2.6517
Epoch 90/100
2s - loss: 2.6983 - val_loss: 2.6504
Epoch 91/100
2s - loss: 2.7020 - val_loss: 2.6539
Epoch 92/100
2s - loss: 2.6997 - val_loss: 2.6570
Epoch 93/100
2s - loss: 2.6974 - val_loss: 2.6485
Epoch 94/100
2s - loss: 2.6963 - val_loss: 2.6488
Epoch 95/100
2s - loss: 2.6973 - val_loss: 2.6450
Epoch 96/100
2s - loss: 2.7003 - val_loss: 2.6549
Epoch 97/100
2s - loss: 2.6999 - val_loss: 2.6463
Epoch 98/100
2s - loss: 2.7003 - val_loss: 2.6444
Epoch 99/100
2s - loss: 2.6987 - val_loss: 2.6474
Epoch 100/100
2s - loss: 2.6999 - val_loss: 2.6563
```

In [197]:
```
plt.figure(figsize=(10, 8))
plt.title("Dense model training", fontsize=12)
plt.plot(history.history["loss"], label="Train")
plt.plot(history.history["val_loss"], label="Test")
plt.grid("on")
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("loss", fontsize=12)
plt.legend(loc="upper right")
```

Out[197]: <matplotlib.legend.Legend at 0x1626c0f60>




# Evalution

In this part, I compare the propsed models and choose the best one. I compare the results based on mean absolute error of predicted versus actual durations, and also mean absolute percentage error which is the percantge of the error. Note that here we compare based on duration as asked in the question and not the velocity.

In [207]:
```
preds_test, preds_train = {}, {}

#Linear Model
preds_test['linear']  = linear_results.predict(X_test)
preds_train['linear'] = linear_results.predict(X_train)

#GLM (Gamma Model)

preds_test['GLM']  = gamma_results.predict(X_test)
preds_train['GLM'] = gamma_results.predict(X_train)

#Deep Learning
preds_test['DL']  = np.squeeze(DNN_model.predict(X_test/mn))
preds_train['DL'] = np.squeeze(DNN_model.predict(X_train/mn))
```

The functions are used for evalution

In [84]:
```
def mean_absolute_error(dist,y_true, y_pred ):
    """
    Args:
        dist(ndarray)   : distance between pick up and drop off
        y_true(ndarray) : true velocity
        y_pred(ndarray) : the prediction value of velocity

    """
    err = np.abs(dist/y_true - dist/y_pred)
    err = err[np.isfinite(err)]
    return np.mean(err) *3600


def mean_absolute_percentage_error(dist,y_true, y_pred ):
    """
    Args:
        dist(ndarray)   : distance between pick up and drop off
        y_true(ndarray) : true velocity
        y_pred(ndarray) : the prediction value of velocity

    """
    err = np.abs(y_true/y_pred - 1)
    err = err[np.isfinite(err)]
    return np.mean(err)*100


def evalute(dist,y_true,prediction):
    MAE, MAPE= {}, {}
    for kys, y_pred in prediction.items():
        MAE[kys]   = mean_absolute_error(dist,y_true, y_pred )
        MAPE[kys]  = mean_absolute_percentage_error(dist,y_true, y_pred
)


    return MAE, MAPE
```

In [209]:
```
MAE_train, MAPE_train = evalute(dist_train,y_train, preds_train)
MAE_test, MAPE_test = evalute(dist_test,y_test, preds_test)


pd.DataFrame([MAE_test,MAE_train, MAPE_test, MAPE_train],
             index=  ['MAE_test', 'MAE_train', 'MAPE_test', 'MAPE_train'
] ).transpose()
```

Out[209]:

| | MAE_test | MAE_train | MAPE_test | MAPE_train |
|---|---|---|---|---|
| **DL** | 7.360206 | 7.330754 | 33.989024 | 33.661978 |
| **GLM** | 6.786721 | 7.246643 | 32.516880 | 32.406202 |
| **linear** | 7.299544 | 7.264867 | 31.866012 | 31.679155 |

In [201]:
```
dist_train.mean()
```

Out[201]: 2.1668461824987508

## Generate Prediction for Test Set

By comparing the three models (linear, GLM, DNN), I choose GLM for generating the predication for the given test set.

In [212]:
```
XX  = np.array(df_test[cl])
XX = np.insert(XX, 0, 1, axis=1)

dist_x = XX[:,1]
#DNN_TD = dist_x/np.squeeze(DNN_model.predict(XX/mn))*3600
GLM_TD = dist_x/gamma_results.predict(XX)*3600
df_ans= pd.DataFrame(GLM_TD, columns =['duration'])


df_ans.index.name = 'row_id'
df_ans.to_csv('answer.csv')
df_ans= pd.DataFrame(TD, columns =['duration'])
```

## Extention and Further Idea

Here, we only use the vincenty, but by contecccting to google API and fidning the real distance between start and end point the preditor defenitlly can be improved. Also, here I only used 10% of data points becouse of the limitation on runnig the DNN. By using GPU or running over the cloud we can use all the samples.