New York City Tax Fare Prediction

Raymond Chen

January 25, 2019

Background

Tourists travel in an unfamiliar city may have a taxi ride, but sometimes they don't know reasonable taxi fare in this city. Few wicked taxi drivers may charge unreasonable fare by sneakily taking long route or adding initial charge. If tourists have a tool to predict reasonable taxi fare based on some simple features like time, pickup location or dropoff location, they can notice unusual charge, take some actions and prevent from fraud. tourists make their budget on travel expense conveniently.

For personal reason, when I have a business trip and have to make a budget in advance, I would use this tool to plan my means of transport. If I have sufficient budget, I can take a taxi for a more comfortable trip. Otherwise, maybe I need to take a train or a bus.

This type of problem is so-called regression problem that demands to predict one continuous target value (e.g. taxi fare) using a set of features. There are many academic research addresses on it : For example, long term travel time is predicted from time, wind speed, temperature ,... etc. features using several state of the art regression methods [1] (https://www.researchgate.net/publication/230819938_Comparing_state-of-the-art_regression_methods_for_long_term_travel_time_prediction/download), Internet slangs for sentiment score is predicted [2] (https://www.researchgate.net/publication/283318703_Detection_and_Scoring_of_Internet_Slangs_for_Sentir and sentiment score is predicted using Tweets' messages [3] (https://www.researchgate.net/deref/http%3A%2F%2Faclweb.org%2Fanthology%2FS%2FS13%2FS13 2053.pdf).

Problem Statement

Our target is to predict taxi fare in New York city, and we have several features like pickup GPS location, dropoff GPS location, or number of passengers, etc. to help us build a model to predict. This is a regression problem and we can express it as:

$y = f(x0, x1, x2, ...)$

where y is taxi fare for a ride, $x0$, $x1$, ... are features like time, GPS location, etc. of this ride, and f is a function or model we want to derive.

Given a dataset with many samples having ground truth taxi fare and features, we can apply different machine learning algorithms or even deep neural network to train a model based on them, i.e. finding some set of parameters that can describe the model mathematically. After model is developed, we can predict taxi fare for a given features.

After model is developed, we can evaluate the model performance using certain metric that can describe the value difference between predicted taxi fare y and ground truth taxi fare y. coefficient of determination (R^2) is used as our evaluation metric in this project.

Datasets and Inputs

In this project, [New York City Taxi Fare Prediction dataset (https://www.kaggle.com/c/new-york-city-taxi-fare-prediction#description)](https://www.kaggle.com/c/new-york-city-taxi-fare-prediction#description) provided in Kaggle is used.

File description

- train.csv - Input features and target fare_amount values for the training set (about 55M rows).
- test.csv - Input features for the test set (about 10K rows). Our goal is to predict fare_amount for each row.
- sample_submission.csv - a sample submission file in the correct format (columns key and fare_amount). This file 'predicts' fare_amount to be $11.35 for all rows, which is the mean fare_amount from the training set.

Data fields

Input (features, X)

- key - Unique string identifying each row in both the training and test sets.
- pickup_datetime - timestamp value indicating when the taxi ride started.
- pickup_longitude - float for longitude coordinate of where the taxi ride started.
- pickup_latitude - float for latitude coordinate of where the taxi ride started.
- dropoff_longitude - float for longitude coordinate of where the taxi ride ended.
- dropoff_latitude - float for latitude coordinate of where the taxi ride ended.
- passenger_count - integer indicating the number of passengers in the taxi ride.

Output (target, y)

- fare_amout - float dollar amount of the cost of the taxi ride. This value is only in the training set

PART A. Explore Data Analysis (EDA)

0. Import necessary modules

In [1]:

```
import pandas as pd
from IPython.display import display
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import scipy
import seaborn as sns
%matplotlib inline
```

1. Load Data

Use 200K data to explore data

In [2]:

```
n_samples = 200000
df=pd.read_csv('all/train.csv', nrows = n_samples)
display(df.head())
df.info()
```

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropo
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 8 columns):
key                  200000 non-null object
fare_amount          200000 non-null float64
pickup_datetime      200000 non-null object
pickup_longitude     200000 non-null float64
pickup_latitude      200000 non-null float64
dropoff_longitude    199999 non-null float64
dropoff_latitude     199999 non-null float64
passenger_count      200000 non-null int64
dtypes: float64(5), int64(1), object(2)
memory usage: 12.2+ MB
```

In [3]:

```
df.describe()
```

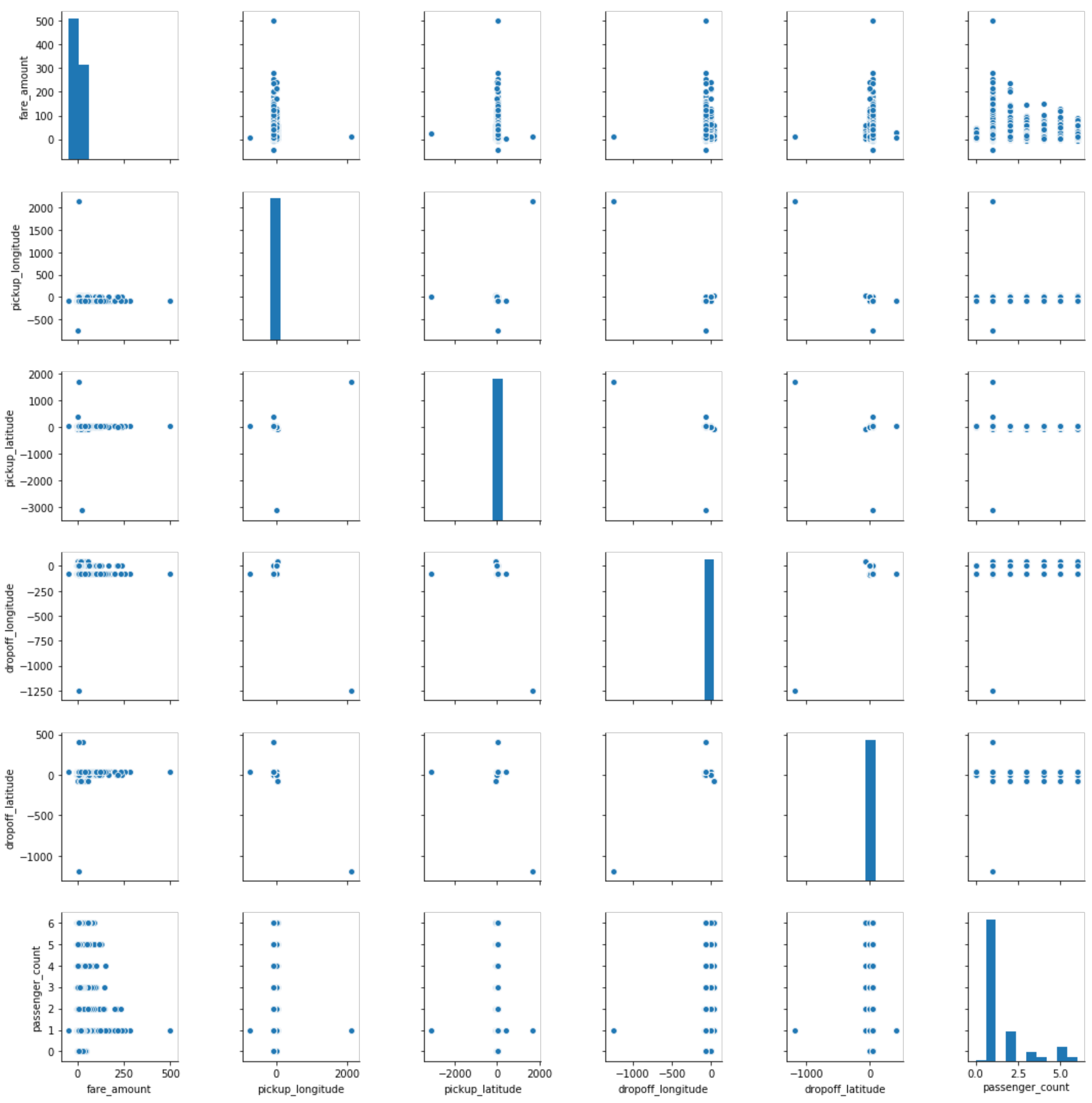
Out[3]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pa
count	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	
mean	11.342877	-72.506121	39.922326	-72.518673	39.925579	
std	9.837855	11.608097	10.048947	10.724226	6.751120	
min	-44.900000	-736.550000	-3116.285383	-1251.195890	-1189.615440	
25%	6.000000	-73.992050	40.735007	-73.991295	40.734092	
50%	8.500000	-73.981743	40.752761	-73.980072	40.753225	
75%	12.500000	-73.967068	40.767127	-73.963508	40.768070	
max	500.000000	2140.601160	1703.092772	40.851027	404.616667	

In [4]:

```python
def show_pairplot(df):
    sns.pairplot(df[df.columns], height=2.5)
    plt.tight_layout()
    plt.show()

show_pairplot(df)
```

```
/Users/RAYMOND/miniconda3/envs/ml-capstone/lib/python3.5/site-packages/nump
y/lib/histograms.py:746: RuntimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
/Users/RAYMOND/miniconda3/envs/ml-capstone/lib/python3.5/site-packages/nump
y/lib/histograms.py:747: RuntimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```



2. Remove N.A. and outliers

2-1. Remove N.A.

In [5]:

```python
def check_total_samples(df):
    print('Total number of samples = ', len(df))

def check_na(df):
    print(df.isnull().sum())

def remove_na(df):
    df = df.dropna(how = 'any', axis = 'rows')
    return df
```

In [6]:

```python
check_total_samples(df)
check_na(df)
df = remove_na(df)
check_total_samples(df)
```

```
Total number of samples =  200000
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
Total number of samples =  199999
```

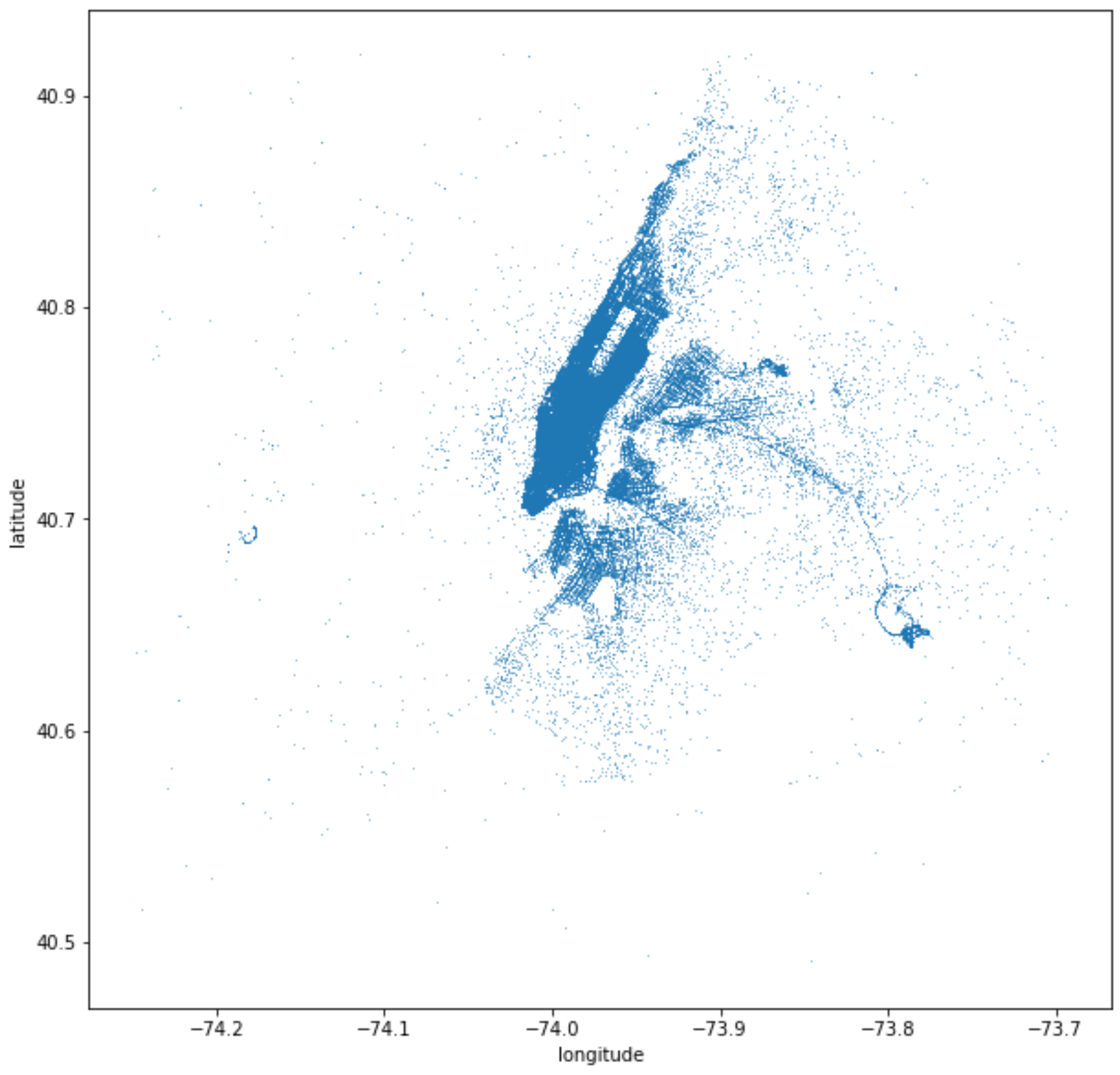
2-2. Remove GPS outliers

Note that GPS location (latitude/longitude) should be within New York city. I use google map (https://goo.gl/maps/AMDNdfcBVvx) to manually find out the boundary of latitude and longitude and implement GPS outliers removal in the following remove_GPS_outliers() function.

In [7]:

```python
def remove_GPS_outliers (df):
    long_min = -74.26
    long_max = -73.69
    lat_min = 40.49
    lat_max = 40.92
    df = df [ (df['pickup_latitude'] > lat_min) & (df['pickup_latitude'] < lat_max) ]
    df = df [ (df['dropoff_latitude'] > lat_min) & (df['dropoff_latitude'] < lat_max) ]
    df = df [ (df['pickup_longitude'] > long_min) & (df['pickup_longitude'] < long_max) ]
    df = df [ (df['dropoff_longitude'] > long_min) & (df['dropoff_longitude'] < long_max) ]
    return df

def show_GPS (df) :
    long = list(df['pickup_longitude']) + list(df['dropoff_longitude'])
    lat = list(df['pickup_latitude'])+ list(df['dropoff_latitude'])
    plt.figure(figsize = (10,10))
    plt.plot(long,lat,',', alpha = 0.5, markersize = 0.5)
    plt.xlabel('longitude')
    plt.ylabel('latitude')
    plt.show()

check_total_samples(df)
df = remove_GPS_outliers(df)
check_total_samples(df)
show_GPS(df)
```

```
Total number of samples =  199999
Total number of samples =  195568
```



2-3. Remove fare outliers

Note that minimum value of fare_amount shouldn't be less than $2.5 according to taxi information for yellow cab (http://www.nyc.gov/html/tlc/downloads/pdf/taxi_information.pdf) from New York city government. On the other hand, we limit the maximum value of fare_amount to 150 (too expansive fare is abnormal)

In [8]:

```
def remove_fare_outliers (df):
    df = df [ (df['fare_amount'] > 2.5) & (df['fare_amount'] < 300) ]
    return df

df = remove_fare_outliers(df)
df.describe()
```

Out[8]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pa
count	194874.000000	194874.000000	194874.000000	194874.000000	194874.000000	
mean	11.315164	-73.975587	40.750919	-73.974474	40.751276	
std	9.462376	0.034129	0.026725	0.034037	0.030726	
min	2.510000	-74.248263	40.492546	-74.244448	40.490235	
25%	6.000000	-73.992270	40.736662	-73.991477	40.735672	
50%	8.500000	-73.982062	40.753523	-73.980528	40.753946	
75%	12.500000	-73.968390	40.767512	-73.965297	40.768362	
max	255.000000	-73.702795	40.918290	-73.694137	40.918868	

In [9]:

```
check_total_samples(df)
```

```
Total number of samples =  194874
```

3. Transform data

In [10]:

```
def transform_pickup_datetime(df):
    df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
    df['pickup_year'] = df['pickup_datetime'].dt.year
    df['pickup_month'] = df['pickup_datetime'].dt.month
    df['pickup_weekday'] = df['pickup_datetime'].dt.weekday
    df['pickup_hour'] = df['pickup_datetime'].dt.hour
    return df

df = transform_pickup_datetime(df)
df.head()
```

Out[10]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropo
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21	-73.844311	40.721319	
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16	-74.016048	40.711303	
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00	-73.982738	40.761270	
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42	-73.987130	40.733143	
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00	-73.968095	40.768008	

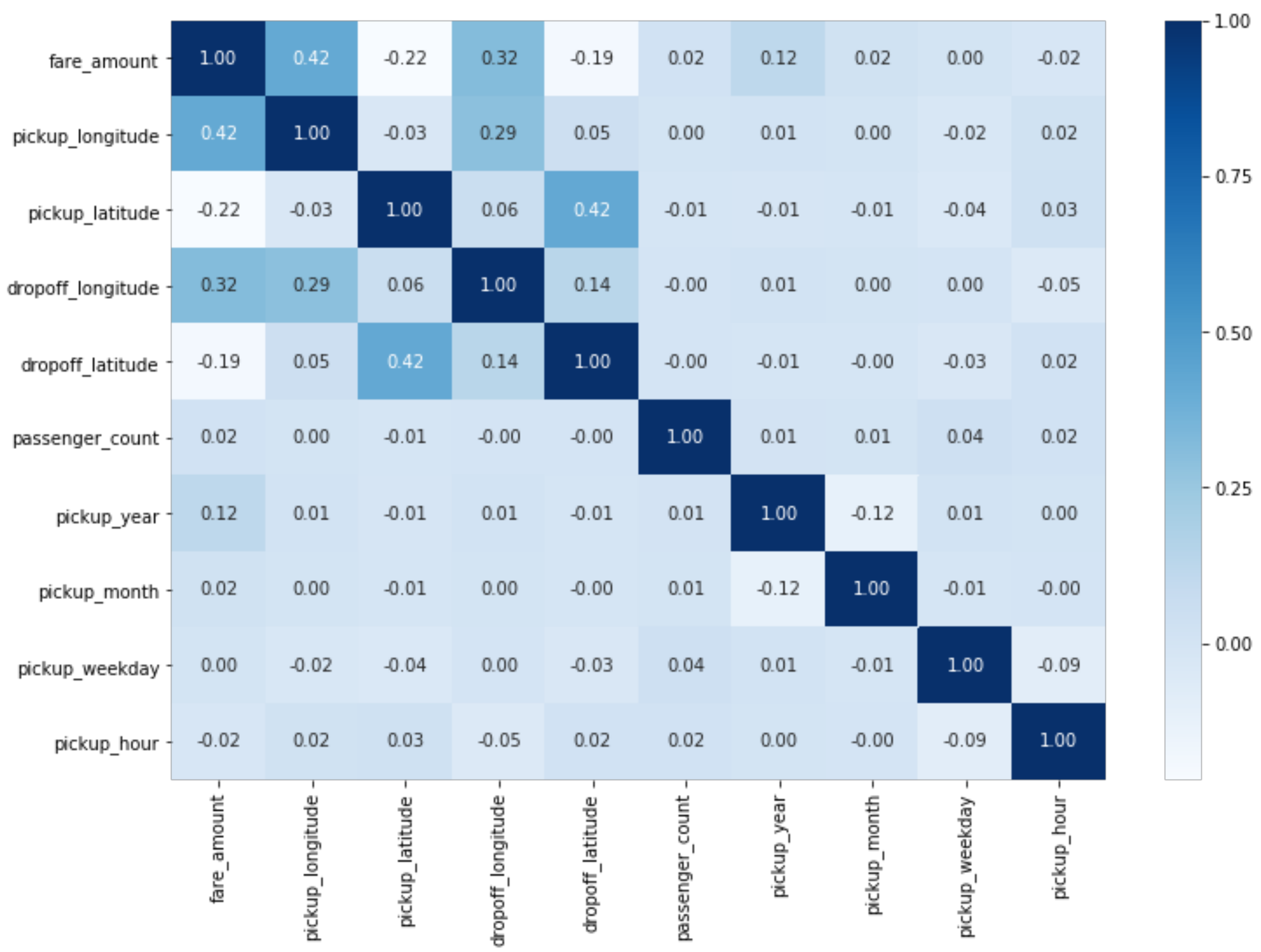
4. Explore Data and Feature Selection

In [11]:

```
def show_corr(df):
    plt.figure(figsize=(12,8))
    cols = df.columns.values
    corr = np.corrcoef(df[cols].values.T)
    sns.heatmap(corr, annot=True,cmap="Blues", fmt='.2f', annot_kws={'size':10}, yticklabels=cols, x
    plt.show()
```

In [12]:

```
df.pop('key')
df.pop('pickup_datetime')
show_corr(df)
```



	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_year	pickup_month	pickup_weekday	pickup_hour
fare_amount	1.00	0.42	-0.22	0.32	-0.19	0.02	0.12	0.02	0.00	-0.02
pickup_longitude	0.42	1.00	-0.03	0.29	0.05	0.00	0.01	0.00	-0.02	0.02
pickup_latitude	-0.22	-0.03	1.00	0.06	0.42	-0.01	-0.01	-0.01	-0.04	0.03
dropoff_longitude	0.32	0.29	0.06	1.00	0.14	-0.00	0.01	0.00	0.00	-0.05
dropoff_latitude	-0.19	0.05	0.42	0.14	1.00	-0.00	-0.01	-0.00	-0.03	0.02
passenger_count	0.02	0.00	-0.01	-0.00	-0.00	1.00	0.01	0.01	0.04	0.02
pickup_year	0.12	0.01	-0.01	0.01	-0.01	0.01	1.00	-0.12	0.01	0.00
pickup_month	0.02	0.00	-0.01	0.00	-0.00	0.01	-0.12	1.00	-0.01	-0.00
pickup_weekday	0.00	-0.02	-0.04	0.00	-0.03	0.04	0.01	-0.01	1.00	-0.09
pickup_hour	-0.02	0.02	0.03	-0.05	0.02	0.02	0.00	-0.00	-0.09	1.00

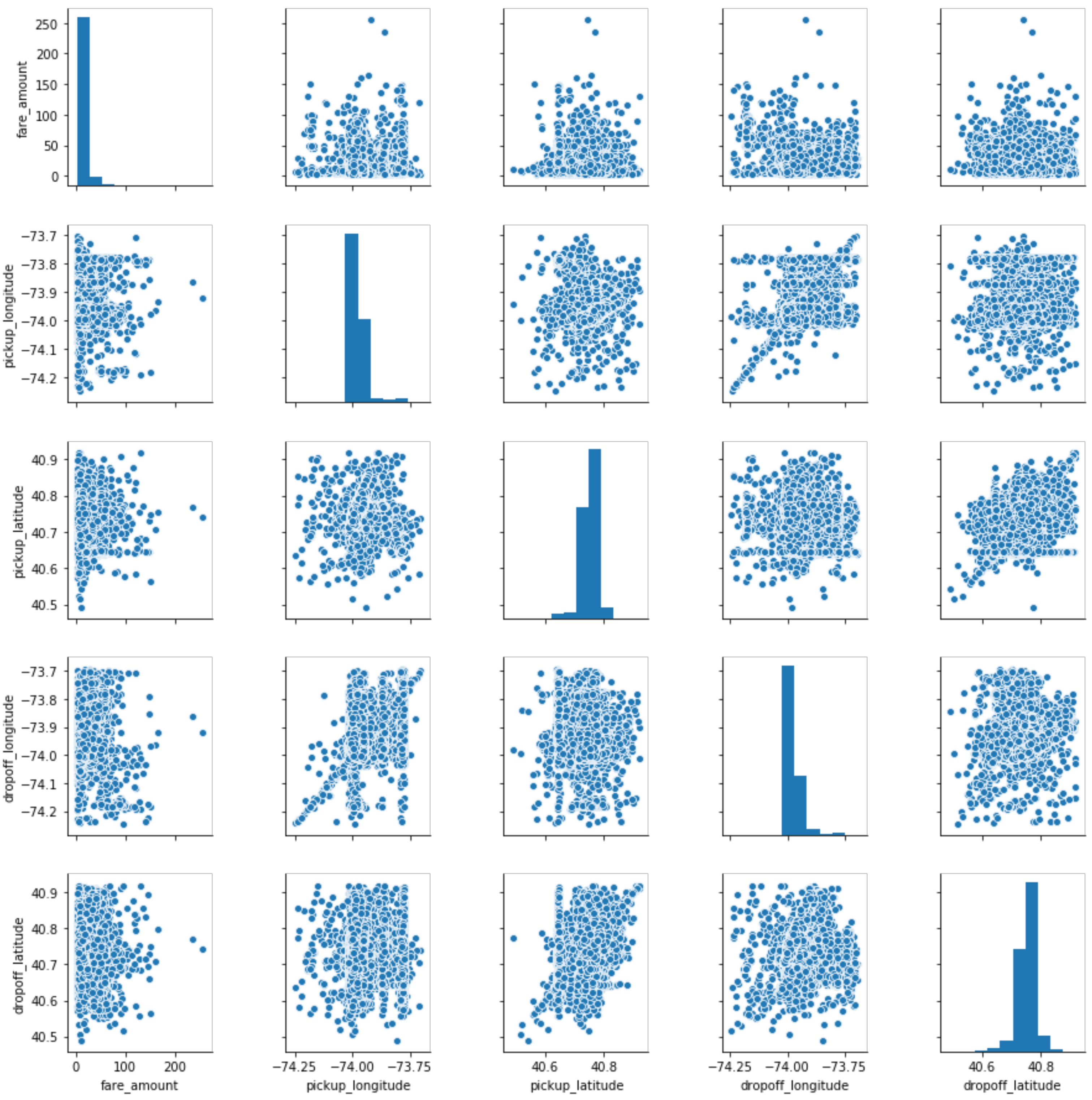
Here are selected features having significant correlation (absolute correlation value > 0.15) with target 'fare_amount' :

- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude

Other insignificant features are removed.

In [13]:

```python
df.pop('passenger_count')
df.pop('pickup_year')
df.pop('pickup_month')
df.pop('pickup_weekday')
df.pop('pickup_hour')
show_pairplot(df)
```



5. Feature extraction

Intuitively, taxi fare should have some relationships with travel distance between pickup location (longitude/latitude) and dropoff location (longitude/latitude). We add two typical distance metrics between pickup location and dropoff location : Manhattan distance (https://en.wikipedia.org/wiki/Taxicab_geometry) and Euclidean distance (https://en.wikipedia.org/wiki/Euclidean_distance).

In [14]:

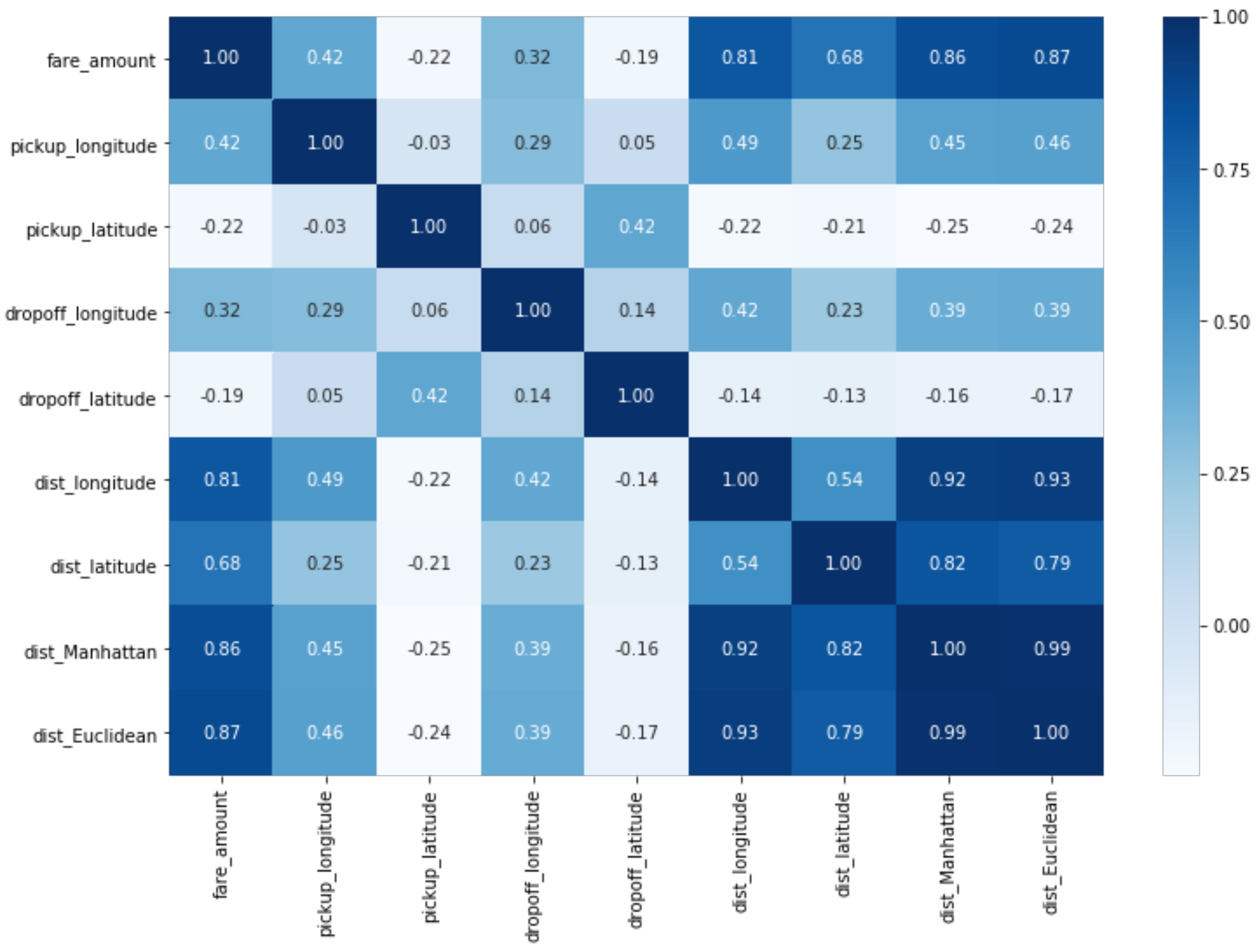
```python
def add_GPS_dist(df):
    df['dist_longitude'] = (df['dropoff_longitude'] - df['pickup_longitude']).abs()
    df['dist_latitude'] = (df['dropoff_latitude'] - df['pickup_latitude']).abs()
    df['dist_Manhattan']  = df['dist_longitude'] + df['dist_latitude']
    df['dist_Euclidean'] = np.sqrt(df['dist_longitude']**2 + df['dist_latitude']**2)

add_GPS_dist(df)
df.describe()
```

Out[14]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	d
count	194874.000000	194874.000000	194874.000000	194874.000000	194874.000000	19
mean	11.315164	-73.975587	40.750919	-73.974474	40.751276	
std	9.462376	0.034129	0.026725	0.034037	0.030726	
min	2.510000	-74.248263	40.492546	-74.244448	40.490235	
25%	6.000000	-73.992270	40.736662	-73.991477	40.735672	
50%	8.500000	-73.982062	40.753523	-73.980528	40.753946	
75%	12.500000	-73.968390	40.767512	-73.965297	40.768362	
max	255.000000	-73.702795	40.918290	-73.694137	40.918868	

In [15]:

```
show_corr(df)
```



	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	dist_longitude	dist_latitude	dist_Manhattan	dist_Euclidean
fare_amount	1.00	0.42	-0.22	0.32	-0.19	0.81	0.68	0.86	0.87
pickup_longitude	0.42	1.00	-0.03	0.29	0.05	0.49	0.25	0.45	0.46
pickup_latitude	-0.22	-0.03	1.00	0.06	0.42	-0.22	-0.21	-0.25	-0.24
dropoff_longitude	0.32	0.29	0.06	1.00	0.14	0.42	0.23	0.39	0.39
dropoff_latitude	-0.19	0.05	0.42	0.14	1.00	-0.14	-0.13	-0.16	-0.17
dist_longitude	0.81	0.49	-0.22	0.42	-0.14	1.00	0.54	0.92	0.93
dist_latitude	0.68	0.25	-0.21	0.23	-0.13	0.54	1.00	0.82	0.79
dist_Manhattan	0.86	0.45	-0.25	0.39	-0.16	0.92	0.82	1.00	0.99
dist_Euclidean	0.87	0.46	-0.24	0.39	-0.17	0.93	0.79	0.99	1.00

After extracting distance of longitude, distance of longitude latitude, distance of Manhattan, distance of Euclidean, we have four additional features that highly correled (absolute value > 0.6) with target 'fare_amount' :

- dist_longitude
- dist_latitude
- dist_Manhattan
- dist_Euclidean

6. Data Standardize

Several machine algorithms performs better when data standardized. I apply StandardScaler in sklearn to standardize features (not including the target, 'fare_amound').

In [16]:

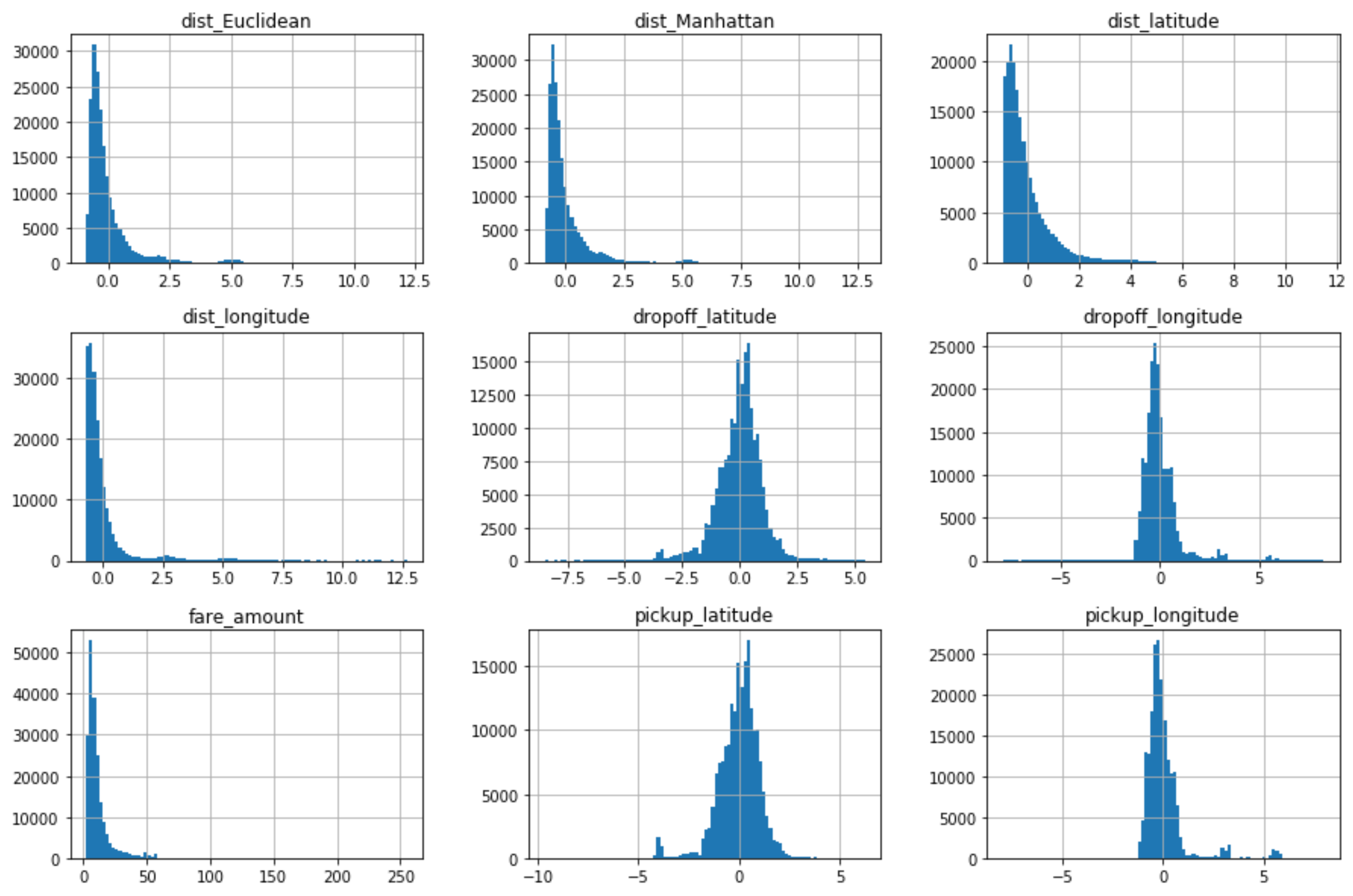
```python
#from sklearn.preprocessing import MinMaxScaler
#scaler = MinMaxScaler() # default=(0, 1)
from sklearn.preprocessing import StandardScaler
def data_standardize(df):
    scaler = StandardScaler()
    cols = df.columns[1:]
    df_standardized=pd.DataFrame(data = df)
    df_standardized[cols] = scaler.fit_transform(df[cols])
    return df_standardized

df = data_standardize(df)
df.describe()
```

Out[16]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	di
count	194874.000000	1.948740e+05	1.948740e+05	1.948740e+05	1.948740e+05	1
mean	11.315164	2.458006e-13	7.950499e-14	1.023217e-13	1.287564e-13	[illegible]
std	9.462376	1.000003e+00	1.000003e+00	1.000003e+00	1.000003e+00	1
min	2.510000	-7.989544e+00	-9.667958e+00	-7.931708e+00	-8.495833e+00	-6
25%	6.000000	-4.888206e-01	-5.334885e-01	-4.995354e-01	-5.078387e-01	-4
50%	8.500000	-1.897211e-01	9.742612e-02	-1.778588e-01	8.689076e-02	-2
75%	12.500000	2.108754e-01	6.208746e-01	2.696137e-01	5.560903e-01	3
max	255.000000	7.992942e+00	6.262773e+00	8.236178e+00	5.454460e+00	1

In [17]:

```
df.hist(bins=100, figsize=(15,10));
```



PART B. Data Preprocess

0. Import necessary modules

In [18]:

```
import pandas as pd
from IPython.display import display
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import scipy
import seaborn as sns
%matplotlib inline
```

1. Load Data

Instead of using whole dataset, I use 50K samples (sample row from 0 to 50K) in "train.csv" file for training and validation (90% data for training, 10% data for validation). Also, I use 10K samples (sample row from 50K to 60K) in "train.csv" for testing.

In [19]:

```python
n_train_valid_samples = 50000 # 50K
n_test_samples = 10000  #10K

df_train_valid=pd.read_csv('all/train.csv', nrows = n_train_valid_samples)
df_train_valid.describe()
```

Out[19]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pa
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	
mean	11.364171	-72.509756	39.933759	-72.504616	39.926251	
std	9.685557	10.393860	6.224857	10.407570	6.014737	
min	-5.000000	-75.423848	-74.006893	-84.654241	-74.006377	
25%	6.000000	-73.992062	40.734880	-73.991152	40.734371	
50%	8.500000	-73.981840	40.752678	-73.980082	40.753372	
75%	12.500000	-73.967148	40.767360	-73.963584	40.768167	
max	200.000000	40.783472	401.083332	40.851027	43.415190	

2. Preprocess and Split Data

In [20]:

```python
def preprocess(df):
    df = remove_na(df)
    df = transform_pickup_datetime(df)
    df.pop('key')
    df.pop('pickup_datetime')
    df = remove_GPS_outliers(df)
    df = remove_fare_outliers(df)
    add_GPS_dist(df)
    df.pop('passenger_count')
    df.pop('pickup_year')
    df.pop('pickup_month')
    df.pop('pickup_weekday')
    df.pop('pickup_hour')
    df = data_standardize(df)
    return df
```

In [21]:

```python
df_train_valid = preprocess(df_train_valid)
df_train_valid.describe()
```

Out[21]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	dis
count	48694.000000	4.869400e+04	4.869400e+04	4.869400e+04	4.869400e+04	4.
mean	11.340768	-1.506822e-13	-1.931195e-13	-1.221188e-13	6.217125e-14	5.
std	9.381403	1.000010e+00	1.000010e+00	1.000010e+00	1.000010e+00	1.
min	2.900000	-7.986515e+00	-8.503057e+00	-7.862797e+00	-7.241180e+00	-6.
25%	6.000000	-4.890409e-01	-5.339814e-01	-4.972472e-01	-5.037757e-01	-4.
50%	8.500000	-1.908392e-01	9.384223e-02	-1.790887e-01	8.817515e-02	-2.
75%	12.500000	2.088434e-01	6.261311e-01	2.626755e-01	5.563444e-01	4.
max	165.000000	7.991825e+00	6.225639e+00	8.030137e+00	5.236496e+00	1.

In [22]:

```python
from sklearn.model_selection import train_test_split
X, y = df_train_valid.iloc[:,1:].values, df_train_valid.iloc[:,0].values
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size = 0.1, random_state = 0)

print("Preprocessed training set has {} samples.".format(X_train.shape[0]))
print("Preprocessed validation set has {} samples.".format(X_valid.shape[0]))
```

```
Preprocessed training set has 43824 samples.
Preprocessed validation set has 4870 samples.
```

PART C. Model Training and Selection

0. Define evaluation metric

In [23]:

```python
from sklearn.metrics import r2_score

# Unit Test
y_true = [10.2, 20.3, 38.4, 10.8, 4.2]
y_predict = [9.9, 12.0, 45.1, 3.8, 5.3]
print("Unit Test of Evaluation Metric : r2_score")
print("y_true = ", y_true)
print("y_predict = ", y_predict)

print("R^2 : {}".format(r2_score(y_true, y_predict)))
```

```
Unit Test of Evaluation Metric : r2_score
y_true =  [10.2, 20.3, 38.4, 10.8, 4.2]
y_predict =  [9.9, 12.0, 45.1, 3.8, 5.3]
R^2 : 0.7711984471391438
```

1. Candidate Model Training

Multiple linear model is used as benchmark model and the other three models with hyperparameter grid-searching are experimented. The experiment models are :

- Polynomial Regression
- Random Forest Regression
- Multiple Layer Perceptron (MLP) Regression

Model 1 : Multiple Linear Regression (benchmark model)

In [24]:

```python
from sklearn.linear_model import LinearRegression
def Linear_reg(X_train, X_valid, y_train, y_valid):
    est_linear = LinearRegression()
    est_linear.fit(X_train, y_train)

    y_train_predict = est_linear.predict(X_train)
    r2_train = r2_score(y_train, y_train_predict)

    y_valid_predict = est_linear.predict(X_valid)
    r2_valid = r2_score(y_valid, y_valid_predict)

    return r2_train, r2_valid, est_linear

r2_train, r2_valid, est_linear = Linear_reg(X_train, X_valid, y_train, y_valid)
print("R^2 (train) : ", r2_train)
print("R^2 (valid) : ", r2_valid)
```

```
R^2 (train) :  0.7729741316254147
R^2 (valid) :  0.809754753525481
```

Model 2 : Polynomial Regression

In [25]:

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

def Poly_reg(X_train, X_valid, y_train, y_valid, Poly_para={'degree' : 2}):
    poly = PolynomialFeatures(degree=Poly_para['degree'])
    X_train_ = poly.fit_transform(X_train)
    est_poly = LinearRegression()
    est_poly.fit(X_train_,y_train)

    y_train_predict = est_poly.predict(X_train_)
    r2_train = r2_score(y_train, y_train_predict)

    X_valid_ = poly.fit_transform(X_valid)

    y_valid_predict = est_poly.predict(X_valid_)
    r2_valid = r2_score(y_valid, y_valid_predict)

    return r2_train, r2_valid

for deg in range(1,5):
    Poly_para={'degree' : deg}
    r2_train, r2_valid = Poly_reg(X_train, X_valid, y_train, y_valid, Poly_para)
    print("Polynomial degree", deg)
    print("R^2 (train) : ", r2_train)
    print("R^2 (valid) : ", r2_valid)

```

```
Polynomial degree 1
R^2 (train) :  0.7729843036205323
R^2 (valid) :  0.8098699723098052
Polynomial degree 2
R^2 (train) :  0.7922608093755379
R^2 (valid) :  0.8272034943361776
Polynomial degree 3
R^2 (train) :  0.8021432880961903
R^2 (valid) :  0.8338251696982749
Polynomial degree 4
R^2 (train) :  0.8112781920025682
R^2 (valid) :  0.8025435308497839
```

We can found polynomial degree = 3 is the best model among polynomial regression models because it has high training R^2 value and validation R^2 is also the peak value before reverting.

Model 3 : Random Forest Regression

In [28]:

```python
from sklearn.ensemble import RandomForestRegressor
def RF_reg(X_train, X_valid, y_train, y_valid, RF_para={'n_estimators': 50}):
    est_rf = RandomForestRegressor(n_estimators=RF_para['n_estimators'], random_state=1)
    est_rf.fit(X_train,y_train)

    y_train_predict = est_rf.predict(X_train)
    r2_train = r2_score(y_train, y_train_predict)

    y_valid_predict = est_rf.predict(X_valid)
    r2_valid = r2_score(y_valid, y_valid_predict)

    return r2_train, r2_valid, est_rf


for n in range(50,250,50):
    RF_para = {'n_estimators': n}
    r2_train, r2_valid, est_rf = RF_reg(X_train, X_valid, y_train, y_valid, RF_para)
    print("Random Forest number of estimators : ", n)
    print("R^2 (train) : ", r2_train)
    print("R^2 (valid) : ", r2_valid)
```

```
Random Forest number of estimators :  50
R^2 (train) :  0.9733243999496467
R^2 (valid) :  0.8533891018408051
Random Forest number of estimators :  100
R^2 (train) :  0.9744579157443554
R^2 (valid) :  0.8541811948795832
Random Forest number of estimators :  150
R^2 (train) :  0.9747864419162937
R^2 (valid) :  0.8541748899779061
Random Forest number of estimators :  200
R^2 (train) :  0.9750437379662068
R^2 (valid) :  0.8540403418177958
```

We choose number of estimators = 100 as the best model among random forest regression models because it has high training R^2 value and validation R^2 saturates.

Model 4 : Multiple Layer Perceptron Regression

In [29]:

```python
from sklearn.neural_network import MLPRegressor
def MLP_reg(X_train, X_valid, y_train, y_valid, MLP_para={'hidden_layer_sizes' : (10, 10, 10) , 'randor
    est_mlp= MLPRegressor(solver=MLP_para['solver'], alpha=MLP_para['alpha'], hidden_layer_sizes
    est_mlp.fit(X_train, y_train)
#    est_mlp.score(X_train, y_train)

    y_train_predict = est_mlp.predict(X_train)
    r2_train = r2_score(y_train, y_train_predict)

    y_valid_predict = est_mlp.predict(X_valid)
    r2_valid = r2_score(y_valid, y_valid_predict)
    return r2_train, r2_valid, est_mlp

for num_node in range(10, 50, 10):
    MLP_para = {'hidden_layer_sizes' : (num_node, num_node, num_node) , 'random_state' : 1, 'alpha
    r2_train, r2_valid, est_mlp = MLP_reg(X_train, X_valid, y_train, y_valid, MLP_para)
    print("MLP hidden layer =3, number of hidden nodes = ", num_node)
    print("R^2 (train) : ", r2_train)
    print("R^2 (valid) : ", r2_valid)
```

```
MLP hidden layer =3, number of hidden nodes =  10
R^2 (train) :  0.8158648517376604
R^2 (valid) :  0.8479220681658175
MLP hidden layer =3, number of hidden nodes =  20
R^2 (train) :  0.8246558228373615
R^2 (valid) :  0.850849950748826
MLP hidden layer =3, number of hidden nodes =  30
R^2 (train) :  0.8257800335298277
R^2 (valid) :  0.8500624174084247
MLP hidden layer =3, number of hidden nodes =  40
R^2 (train) :  0.8237470179256542
R^2 (valid) :  0.8448035054943113
```

We can found number of hidden nodes = 20 is the best model because it has high training R^2 value and validation R^2 is also the peak value before reverting.

Among all models, random forest regression model with number of estimators = 30 has highest training R^2 = 0.974 and validation R^2 = 0.854. Thus, it seems to be our best model candidate. On the other hand, the benchmark Multiple Linear Regression model has R^2(train) = 0.773 and R^(valid) = 0.810.

2. Sensitivity Analysis

To justify the sensitivity of best model and benchmark model, I used data of sample rows range from 100K to 130K in 'train.csv' file to test model sensitivity. Three sensitivity batch data with 10K samples are used.

In [30]:

```python
n_sensitivity_samples = 10000 #10K

df_sensitivity1 =pd.read_csv('all/train.csv', nrows = n_sensitivity_samples, skiprows=list(range(1, 100
df_sensitivity2 =pd.read_csv('all/train.csv', nrows = n_sensitivity_samples, skiprows=list(range(1, 10(
df_sensitivity3 =pd.read_csv('all/train.csv', nrows = n_sensitivity_samples, skiprows=list(range(1, 10(

df_sensitivity1 = preprocess(df_sensitivity1)
df_sensitivity2 = preprocess(df_sensitivity2)
df_sensitivity3 = preprocess(df_sensitivity3)
X1, y1 = df_sensitivity1.iloc[:,1:].values, df_sensitivity1.iloc[:,0].values
X2, y2 = df_sensitivity2.iloc[:,1:].values, df_sensitivity2.iloc[:,0].values
X3, y3 = df_sensitivity3.iloc[:,1:].values, df_sensitivity3.iloc[:,0].values
X_sensitivity = [X1, X2, X3]
y_sensitivity = [y1, y2, y3]
```

```
/Users/RAYMOND/miniconda3/envs/ml-capstone/lib/python3.5/site-packages/ipyker
nel/__main__.py:2: SettingWithCopyWarning:
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/stable/i
ndexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)
  from ipykernel import kernelapp as app
/Users/RAYMOND/miniconda3/envs/ml-capstone/lib/python3.5/site-packages/ipyker
nel/__main__.py:3: SettingWithCopyWarning:
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/stable/i
ndexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)
  app.launch_new_instance()
/Users/RAYMOND/miniconda3/envs/ml-capstone/lib/python3.5/site-packages/ipyker
nel/__main__.py:4: SettingWithCopyWarning:
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/stable/i
ndexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)
/Users/RAYMOND/miniconda3/envs/ml-capstone/lib/python3.5/site-packages/ipyker
nel/__main__.py:5: SettingWithCopyWarning:
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/stable/i
ndexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)
/Users/RAYMOND/miniconda3/envs/ml-capstone/lib/python3.5/site-packages/ipyker
nel/__main__.py:6: SettingWithCopyWarning:
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/stable/i
ndexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy)
```

In [31]:

```python
r2_train_benchmark, r2_valid_benchmark, est_benchmark = Linear_reg(X_train, X_valid, y_train, y_va
print('Sensitivity of benchmark model (linear regression model) : ')
r2_benchmark=[]
for X, y in zip(X_sensitivity, y_sensitivity):
    y_predict = est_linear.predict (X)
    r2 = r2_score(y, y_predict)
    r2_benchmark.append(r2)
    print('R^2 :', r2)
print('Average R^2 :', sum(r2_benchmark) / float(len(r2_benchmark)))


r2_train_best, r2_valid_best, est_best = RF_reg(X_train, X_valid, y_train, y_valid, RF_para = {'n_estin
print('Sensitivity of best model (random forest regression model) : ')

r2_best=[]
for X, y in zip(X_sensitivity, y_sensitivity):
    y_predict = est_best.predict (X)
    r2 = r2_score(y, y_predict)
    r2_best.append(r2)
    print('R^2 :', r2)
print('Average R^2 :', sum(r2_best) / float(len(r2_best)) )
```

```
Sensitivity of benchmark model (linear regression model) :
R^2 : -3.082568191050259e+20
R^2 : -1.2677135170730803e+20
R^2 : -5.474876256878982e+20
Average R^2 : -3.2750526550007736e+20
Sensitivity of best model (random forest regression model) :
R^2 : 0.83434451537281
R^2 : 0.7276965321528759
R^2 : 0.8062664829977666
Average R^2 : 0.7894358435078175
```

We can find the benchmark model (multiple linear regression model) is very sensitive to data, compared with R^2 for valid dataset 0.810, the average R^2 for sensitivity batches is -3.2750526550007736e+20. It has very large variation. On the other hand, our best model (random forest regression model) is not that sensitive to data, the R^2 for valid data is 0.854, and average R^2 for sensitivity batches is 0.789. It's comparable and has similar performance among different dataset.

PART D. Model Evaluation

1. Load Data

In [32]:

```
df_test=pd.read_csv('all/train.csv', nrows = n_test_samples, skiprows=list(range(1, n_train_valid_sar
df_test.describe()
```

Out[32]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pa
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	
mean	11.160682	-72.571975	39.952387	-72.607937	39.977419	
std	9.501261	10.208912	6.011664	10.078934	5.934254	
min	-44.900000	-74.718822	-73.983715	-74.718822	-73.989357	
25%	6.000000	-73.992041	40.735733	-73.991301	40.734766	
50%	8.500000	-73.981863	40.752918	-73.980158	40.753690	
75%	12.500000	-73.967923	40.767247	-73.963460	40.768415	
max	149.000000	40.766100	41.366138	40.757492	41.366138	

2. Preprocess Data

In [33]:

```
df_test = preprocess(df_test)
X_test, y_test = df_test.iloc[:,1:].values, df_test.iloc[:,0].values
print("Preprocessed testing set has {} samples.".format(X_test.shape[0]))
```

```
Preprocessed testing set has 9754 samples.
```

Model evaluation

In [34]:

```
X_test, y_test = df.iloc[:,1:].values, df.iloc[:,0].values

y_test_predict_benchmark= est_benchmark.predict(X_test)
y_test_predict_best = est_best.predict(X_test)

r2_test_benchmark = r2_score(y_test, y_test_predict_benchmark)
r2_test_best = r2_score(y_test, y_test_predict_best)

print("R^2 (test) of Linear Regression : ", r2_test_benchmark)
print("R^2 (test) of Random Forest Regression : ", r2_valid)
```

```
R^2 (test) of Linear Regression :  -3.5642511557435007e+19
R^2 (test) of Random Forest Regression :  0.8448035054943113
```

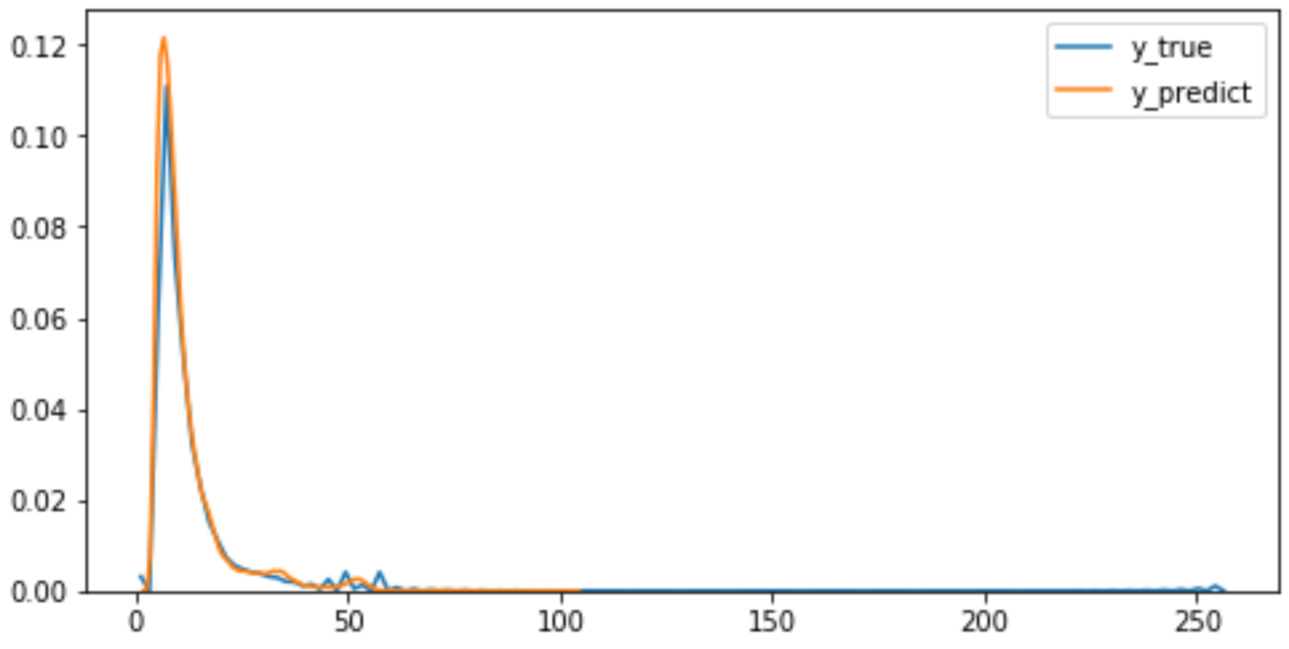
In [35]:

```
def show_series(y_true, y_predict):
    plt.figure(figsize=(8,4))
    sns.kdeplot(y_true,label='y_true')
    sns.kdeplot(y_predict,label='y_predict')
```

In [36]:

```
show_series(y_test, y_test_predict_best)
```

```
/Users/RAYMOND/miniconda3/envs/ml-capstone/lib/python3.5/site-packages/scipy/
stats/stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional in
dexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be
interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or
a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



We can see R^2 (test) of Random Forest Regression = 0.8448035054943113, which is comparable to R^2 in train and valid data. It performs pretty good as expected. On the other hand, the benchmark model (linear model) is terrible because R^2 = -3.5642511557435007e+19 < 0. Predicting average value of taxi fare would even have better R^2 than the benchmark model.

In [ ]:

```

```

Present Slides Themes Help