kaggle

Mohit Sharma

NYC Taxi Fare Kernel

5
voters

last run 18 days ago · IPython Notebook HTML · 142 views
using data from New York City Taxi Fare Prediction · Public

Tags beginner data visualization starter code tutorial

Notebook

Import the libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import os
os.listdir('../input')
```

Out[1]:
```
['GCP-Coupons-Instructions.rtf',
 'train.csv',
 'sample_submission.csv',
 'test.csv']
```

Import Dataset

In [2]:
```
# I try to run train.csv file but Kaggle kernel didn't have that much power to excute 55 Millions row,
# so I am skipping good portion of data
train_df = pd.read_csv('../input/train.csv', nrows = 10_000_000)
```

In [3]:
```
train_df.head()
```

Out[3]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitu
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655

In [4]:
```
train_df.columns
```

Out[4]:
```
Index(['key', 'fare_amount', 'pickup_datetime', 'pickup_longitude',
       'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude',
       'passenger_count'],
      dtype='object')
```

In [5]:
```
train_df.dtypes
```

Out[5]:
```
key                 object
fare_amount        float64
pickup_datetime     object
pickup_longitude   float64
pickup_latitude    float64
dropoff_longitude  float64
dropoff_latitude   float64
passenger_count      int64
dtype: object
```

In [6]:
```
train_df.describe()
```

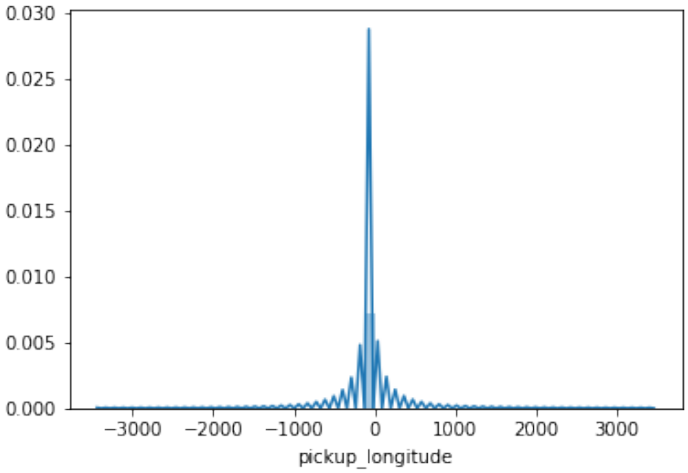
Out[6]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_c

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_c
count	1.000000e+07	1.000000e+07	1.000000e+07	9.999931e+06	9.999931e+06	1.000000e+
mean	1.133854e+01	-7.250775e+01	3.991934e+01	-7.250897e+01	3.991913e+01	1.684793e+
std	9.799930e+00	1.299421e+01	9.322539e+00	1.287532e+01	9.237280e+00	1.323423e+
min	-1.077500e+02	-3.439245e+03	-3.492264e+03	-3.426601e+03	-3.488080e+03	0.000000e+
25%	6.000000e+00	-7.399207e+01	4.073491e+01	-7.399139e+01	4.073403e+01	1.000000e+
50%	8.500000e+00	-7.398181e+01	4.075263e+01	-7.398016e+01	4.075316e+01	1.000000e+
75%	1.250000e+01	-7.396710e+01	4.076712e+01	-7.396367e+01	4.076810e+01	2.000000e+
max	1.273310e+03	3.457626e+03	3.344459e+03	3.457622e+03	3.351403e+03	2.080000e+

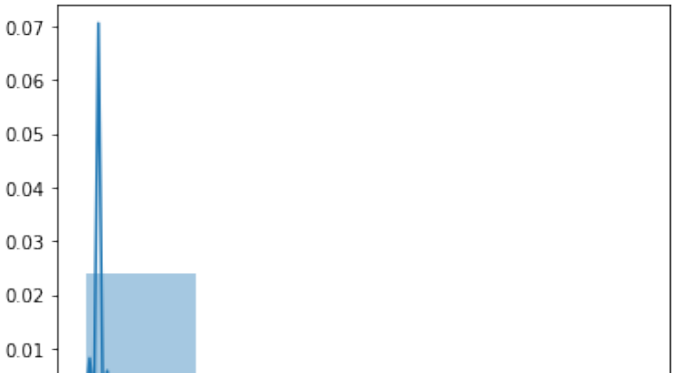
In [7]:
```
sns.distplot(train_df['pickup_longitude'])
```

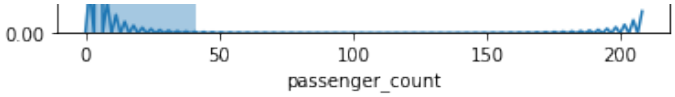
Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5f7d948d0>
```



In [8]:
```
sns.distplot(train_df['passenger_count'], bins=5, kde=True)
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5f7d91320>
```



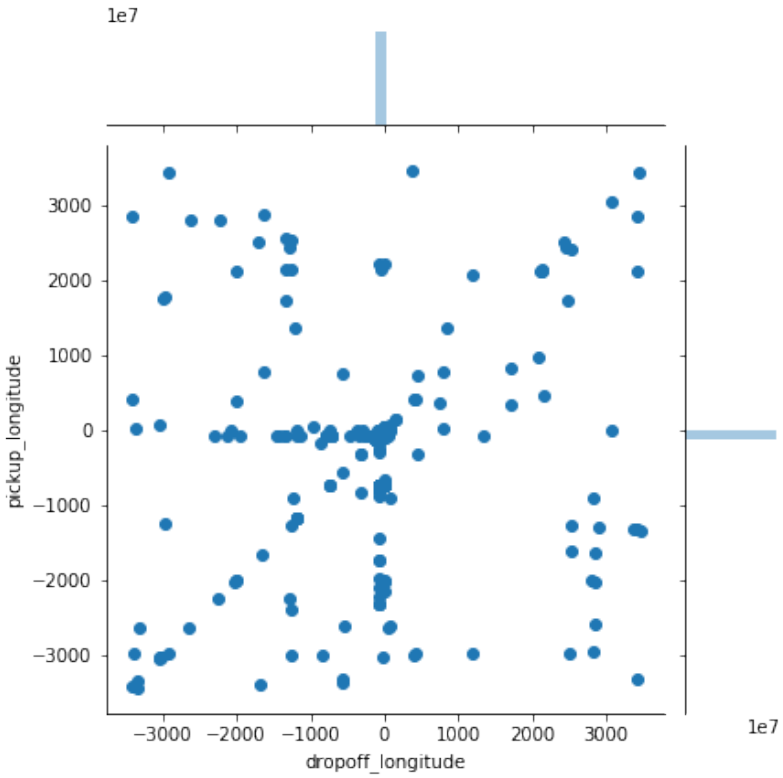


In [9]:

```
sns.jointplot(x = 'dropoff_longitude', y = 'pickup_longitude', data = train_df)
```

Out[9]:

```
<seaborn.axisgrid.JointGrid at 0x7fb5f7462940>
```



In [10]:

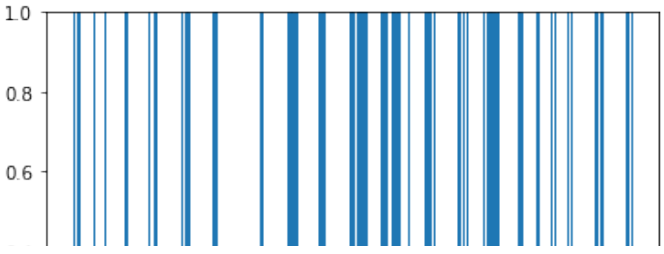
```
diff_longitude = (train_df['pickup_longitude'] - train_df['dropoff_longitude']).abs()
```

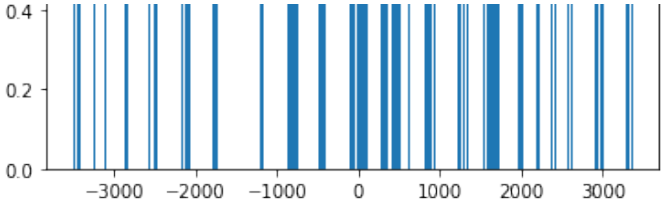
In [11]:

```
sns.rugplot(train_df['pickup_latitude'], height=1.0)
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5f71b9470>
```





In [12]:
```
diff_longitude.head()
```

Out[12]:
```
0    0.002701
1    0.036780
2    0.008504
3    0.004437
4    0.011440
dtype: float64
```

In [13]:
```
diff_latitude = (train_df['pickup_latitude']- train_df['dropoff_latitude'] )
.abs()
```

In [14]:
```
diff_latitude.head()
```

Out[14]:
```
0    0.009041
1    0.070701
2    0.010708
3    0.024949
4    0.015754
dtype: float64
```

In [15]:
```
train_df['diff_longitude'] = diff_longitude
```

In [16]:
```
train_df['diff_latitude'] = diff_latitude
```

In [17]:
```
train_df.columns  #Columns added
```

Out[17]:
```
Index(['key', 'fare_amount', 'pickup_datetime', 'pickup_longitude',
       'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude',
       'passenger_count', 'diff_longitude', 'diff_latitude'],
      dtype='object')
```

In [18]:
```
train_df.describe()
```

Out[18]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_c
count	1.000000e+07	1.000000e+07	1.000000e+07	9.999931e+06	9.999931e+06	1.000000e+
mean	1.133854e+01	-7.250775e+01	3.991934e+01	-7.250897e+01	3.991913e+01	1.684793e+
std	9.799930e+00	1.299421e+01	9.322539e+00	1.287532e+01	9.237280e+00	1.323423e+
min	-1.077500e+02	-3.439245e+03	-3.492264e+03	-3.426601e+03	-3.488080e+03	0.000000e+
25%	6.000000e+00	-7.399207e+01	4.073491e+01	-7.399139e+01	4.073403e+01	1.000000e+
50%	8.500000e+00	-7.398181e+01	4.075263e+01	-7.398016e+01	4.075316e+01	1.000000e+
75%	1.250000e+01	-7.396710e+01	4.076712e+01	-7.396367e+01	4.076810e+01	2.000000e+
max	1.273310e+03	3.457626e+03	3.344459e+03	3.457622e+03	3.351403e+03	2.080000e+

Checking out the Missing Values and Outliers

In [19]:
```
train_df.isnull().sum()
```

Out[19]:
```
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    69
dropoff_latitude     69
passenger_count      0
diff_longitude       69
diff_latitude        69
dtype: int64
```

We can see the missing data is very less so we can ignore it by dropping it.

In [20]:
```
print(train_df['dropoff_latitude'].isnull().sum())
```

```
69
```

In [21]:
```
# These are location where we have missing Values
train_df.dropoff_latitude[train_df.dropoff_latitude != train_df.dropoff_lati
tude].index.values
```

Out[21]:
```
array([ 120227,  245696,  340533,  428108,  471472,  524834,  574023,
        580338,  794694,  895400, 1220978, 1476796, 1521628, 1882440,
       2087156, 2267436, 2277566, 2455721, 2455848, 2637865, 2664981,
       2747686, 2794177, 3162290, 3244924, 3310378, 3700567, 3941824,
       3952804, 4114839, 4165644, 4236846, 4617652, 4789267, 4835072,
       4854887, 5591752, 5616035, 5784187, 6189379, 6269652, 6358428,
       6442547, 6501722, 6571093, 6660408, 6678592, 7191178, 7844202,
       8131337, 8160692, 8190328, 8552586, 8631332, 8862512, 8891498,
       8913939, 9028651, 9060096, 9088217, 9093119, 9145845, 9354560,
       9496338, 9536062, 9609188, 9699243, 9715861, 9754957])
```

In [22]:
```
len(train_df)
```

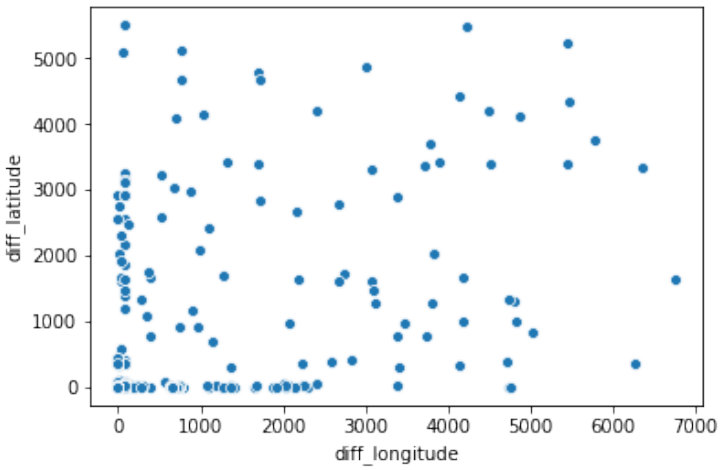
Out[22]:
```
10000000
```

In [23]:
```
train_df = train_df.dropna(how = 'any', axis = 'rows')
len(train_df)
```

Out[23]:
```
9999931
```

In [24]:
```
sns.scatterplot(x='diff_longitude', y='diff_latitude', data=train_df)
```

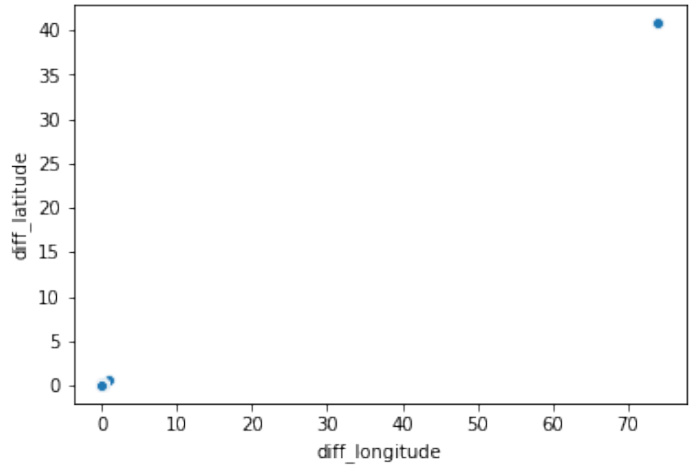
Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb51c030ef0>
```



We expect most of these values to be very small (likely between 0 and 1) since it should all be differences between GPS coordinates within one city. For reference, one degree of latitude is about 69 miles. However, we can see the dataset has extreme values which do not make sense. Let's remove those values from our training set. Based on the scatterplot, it looks like we can safely exclude values above 5 (though remember the scatterplot is only showing the first 2000 rows...)

In [25]:
```
sns.scatterplot(x='diff_longitude', y='diff_latitude', data=train_df.iloc[ :
2000])
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb408aed1d0>
```



In [26]:
```
print('Old size: %d' % len(train_df))
train_df = train_df[(train_df.diff_longitude < 5.0) & (train_df.diff_latitud
e < 5.0)]
print('New size: %d' % len(train_df))
```

```
Old size: 9999931
New size: 9979187
```

Train our model

Our model will take the form $X \cdot w = y$ where X is a matrix of input features, and y is a column of the target variable, fare_amount, for each row. The weight column w is what we will "learn".

First let's setup our input matrix X and target column y from our training set. The matrix X should consist of the two GPS coordinate differences, plus a third term of 1 to allow the model to learn a constant bias term. The column y should consist of the target fare_amount values

In [27]:
```
train_X = np.column_stack( (train_df.diff_latitude, train_df.diff_longitude,
np.ones(len(train_df))))
```

In [28]:
```
train_y = np.array(train_df['fare_amount'])
```

In [29]:
```
print(train_X.shape)
print(train_y.shape)
```

```
(9979187, 3)
(9979187,)
```

Now let's use numpy's lstsq library function to find the optimal weight column

In [30]:
```
# The lstsq function returns several things, and we only care about the actual
weight vector w.
(w,_,_,_) = np.linalg.lstsq(train_X, train_y)
print(w)
```

```
[ 76.95503724 147.16176525   6.39545245]
```

These weights pass a quick sanity check, since we'd expect the first two values -- the weights for the absolute longitude and latitude differences -- to be positive, as more distance should imply a higher fare, and we'd expect the bias term to loosely represent the cost of a very short ride.

Sidenote: we can actually calculate the weight column w directly using the Ordinary Least Squares method: $w=(X^T \cdot X)^{-1} \cdot X^T \cdot y$

In [31]:
```
w_OLS = np.matmul(np.matmul(np.linalg.inv(np.matmul(train_X.T, train_X)), tr
ain_X.T), train_y)
print(w_OLS)
```

```
[ 76.95503724 147.16176525   6.39545245]
```

Make predictions on the test set

Now let's load up our test inputs and predict the fare_amounts for them using our learned weights!

In [32]:
```
test_df = pd.read_csv('../input/test.csv')
test_df.dtypes
```

Out[32]:
```
key                 object
pickup_datetime     object
pickup_longitude   float64
pickup_latitude    float64
dropoff_longitude  float64
dropoff_latitude   float64
passenger_count      int64
dtype: object
```

In [33]:
```
def add_travel_vector_features(df):
    df['abs_diff_longitude'] = (df.dropoff_longitude - df.pickup_longitude).
abs()
    df['abs_diff_latitude'] = (df.dropoff_latitude - df.pickup_latitude).abs
()

add_travel_vector_features(test_df)
```

In [34]:
```
def get_input_matrix(df):
    return np.column_stack((df.abs_diff_longitude, df.abs_diff_latitude, np.
ones(len(df))))
```

In [35]:
```
test_X = get_input_matrix(test_df)
```

In [36]:
```
# Predict fare_amount on the test set using our model (w) trained on the train
ing set.
test_y_predictions = np.matmul(test_X, w).round(decimals = 2)

# Write the predictions to a CSV file which we can submit to the competition.
submission = pd.DataFrame(
    {'key': test_df.key, 'fare_amount': test_y_predictions},
    columns = ['key', 'fare_amount'])
submission.to_csv('submission.csv', index = False)

print(os.listdir('.'))
```

```
['script.ipynb', 'submission.csv', '__output__.json']
```

Thank you!! Check out my blog on THEMENYOUWANTTOBE (http://themenyouwanttobe.wordpress.com)