DOWNLOAD DATASET:

[2]:
```
!pip install numpy pandas opendatasets scikit-learn xgboost --quiet
```

[3]:
```
import opendatasets as od
```

[4]:
```
dataset_url = 'https://www.kaggle.com/c/new-york-city-taxi-fare-prediction/overview'
```

[6]:
```
%%time
od.download(dataset_url)
```

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username:
  adityadutta200021
Your Kaggle Key:
  ........
Downloading new-york-city-taxi-fare-prediction.zip to .\new-york-city-taxi-fare-prediction
100%|██████████| 1.56G/1.56G [01:13<00:00, 22.8MB/s]

Extracting archive .\new-york-city-taxi-fare-prediction/new-york-city-taxi-fare-prediction.zip to .\new-york-city-taxi-fare-prediction
CPU times: total: 21.7 s
Wall time: 1min 59s
```

[7]:
```
data_dir = './new-york-city-taxi-fare-prediction'
```

[19]:
```
import os
os.listdir(data_dir)
```

[19]:
```
['GCP-Coupons-Instructions.rtf',
 'sample_submission.csv',
 'test.csv',
 'train.csv']
```

LOAD TRAINING SET:

[26]:
```
import pandas as pd
import random
```

[27]:
```
sample_frac = 0.01
```

[28]:
```
selected_cols = 'fare_amount,pickup_datetime,pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude,passenger_count'.split(',')
selected_cols
```

[28]:
```
['fare_amount',
 'pickup_datetime',
 'pickup_longitude',
 'pickup_latitude',
 'dropoff_longitude',
 'dropoff_latitude',
 'passenger_count']
```

[29]:
```
dtypes = {
    'fare_amount': 'float32',
    'pickup_longitude': 'float32',
    'pickup_latitude': 'float32',
    'dropoff_longitude': 'float32',
    'dropoff_latitude': 'float32',
    'passenger_count': 'float32'
}
```

[30]:
```
def skip_row(row_idx):
    if row_idx == 0:
        return False
    return random.random() > sample_frac
```

[32]:
```
random.seed(42)
df = pd.read_csv(data_dir + '/train.csv', usecols = selected_cols, dtype = dtypes,parse_dates=['pickup_datetime'], skiprows = skip_row )
```

[33]:
```
df
```

[33]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	4.0	2014-12-06 20:36:22+00:00	-73.979813	40.751904	-73.979446	40.755482	1.0
1	8.0	2013-01-17 17:22:00+00:00	0.000000	0.000000	0.000000	0.000000	2.0
2	8.9	2011-06-15 18:07:00+00:00	-73.996330	40.753223	-73.978897	40.766964	3.0
3	6.9	2009-12-14 12:33:00+00:00	-73.982430	40.745747	-73.982430	40.745747	1.0
4	7.0	2013-11-06 11:26:54+00:00	-73.959061	40.781059	-73.962059	40.768604	1.0
...	...	...	...	...	...	...	...

552445	45.0	2014-02-06 23:59:45+00:00	-73.973587	40.747669	-73.999916	40.602894	1.0
552446	22.5	2015-01-05 15:29:08+00:00	-73.935928	40.799656	-73.985710	40.726952	2.0
552447	4.5	2013-02-17 22:27:00+00:00	-73.992531	40.748619	-73.998436	40.740143	1.0
552448	14.5	2013-01-27 12:41:00+00:00	-74.012115	40.706635	-73.988724	40.756218	1.0
552449	6.0	2014-10-18 07:51:00+00:00	-73.997681	40.724380	-73.994148	40.717796	1.0

552450 rows × 7 columns

LOAD TEST SET:

[34]:
```
test_df = pd.read_csv(data_dir + '/test.csv', dtype = dtypes,parse_dates=['pickup_datetime'])
test_df
```

[34]:

	key	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2015-01-27 13:08:24.0000002	2015-01-27 13:08:24+00:00	-73.973320	40.763805	-73.981430	40.743835	1.0
1	2015-01-27 13:08:24.0000003	2015-01-27 13:08:24+00:00	-73.986862	40.719383	-73.998886	40.739201	1.0
2	2011-10-08 11:53:44.0000002	2011-10-08 11:53:44+00:00	-73.982521	40.751259	-73.979652	40.746140	1.0
3	2012-12-01 21:12:12.0000002	2012-12-01 21:12:12+00:00	-73.981163	40.767807	-73.990448	40.751637	1.0
4	2012-12-01 21:12:12.0000003	2012-12-01 21:12:12+00:00	-73.966049	40.789776	-73.988564	40.744427	1.0
...	...	...	...	...	...	...	...
9909	2015-05-10 12:37:51.0000002	2015-05-10 12:37:51+00:00	-73.968124	40.796997	-73.955643	40.780388	6.0
9910	2015-01-12 17:05:51.0000001	2015-01-12 17:05:51+00:00	-73.945511	40.803600	-73.960213	40.776371	6.0
9911	2015-04-19 20:44:15.0000001	2015-04-19 20:44:15+00:00	-73.991600	40.726608	-73.789742	40.647011	6.0
9912	2015-01-31 01:05:19.0000005	2015-01-31 01:05:19+00:00	-73.985573	40.735432	-73.939178	40.801731	6.0
9913	2015-01-18 14:06:23.0000006	2015-01-18 14:06:23+00:00	-73.988022	40.754070	-74.000282	40.759220	6.0

9914 rows × 7 columns

EXPLORE DATASET:

[35]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 552450 entries, 0 to 552449
Data columns (total 7 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   fare_amount        552450 non-null  float32
 1   pickup_datetime    552450 non-null  datetime64[ns, UTC]
 2   pickup_longitude   552450 non-null  float32
 3   pickup_latitude    552450 non-null  float32
 4   dropoff_longitude  552450 non-null  float32
 5   dropoff_latitude   552450 non-null  float32
 6   passenger_count    552450 non-null  float32
dtypes: datetime64[ns, UTC](1), float32(6)
memory usage: 16.9 MB
```

[36]:
```
df.describe()
```

[36]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	552450.000000	552450.000000	552450.000000	552450.000000	552450.000000	552450.000000
mean	11.354059	-72.497063	39.910500	-72.504326	39.934265	1.684983
std	9.810809	11.622035	8.041162	12.065184	9.226158	1.341986
min	-52.000000	-1183.362793	-3084.490234	-3356.729736	-2073.150635	0.000000
25%	6.000000	-73.992020	40.734875	-73.991425	40.733990	1.000000
50%	8.500000	-73.981819	40.752621	-73.980179	40.753101	1.000000
75%	12.500000	-73.967155	40.767036	-73.963737	40.768059	2.000000
max	499.000000	2420.209473	404.983337	2467.752686	3351.403076	208.000000

[37]:
```
df['pickup_datetime'].min(), df['pickup_datetime'].max()
```

[37]:
```
(Timestamp('2009-01-01 00:11:46+0000', tz='UTC'),
 Timestamp('2015-06-30 23:59:54+0000', tz='UTC'))
```

[38]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9914 entries, 0 to 9913
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   key                9914 non-null   object
 1   pickup_datetime    9914 non-null   datetime64[ns, UTC]
 2   pickup_longitude   9914 non-null   float32
```

```
 3   pickup_latitude    9914 non-null   float32
 4   dropoff_longitude  9914 non-null   float32
 5   dropoff_latitude   9914 non-null   float32
 6   passenger_count    9914 non-null   float32
dtypes: datetime64[ns, UTC](1), float32(5), object(1)
memory usage: 348.7+ KB
```

[39]:
```
test_df.describe()
```

[39]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	9914.000000	9914.000000	9914.000000	9914.000000	9914.000000
mean	-73.974716	40.751041	-73.973656	40.751740	1.671273
std	0.042799	0.033542	0.039093	0.035436	1.278756
min	-74.252190	40.573143	-74.263245	40.568974	1.000000
25%	-73.992500	40.736125	-73.991249	40.735253	1.000000
50%	-73.982327	40.753052	-73.980015	40.754065	1.000000
75%	-73.968012	40.767113	-73.964062	40.768757	2.000000
max	-72.986534	41.709557	-72.990967	41.696682	6.000000

[40]:
```
test_df['pickup_datetime'].min(), test_df['pickup_datetime'].max()
```

[40]:
```
(Timestamp('2009-01-01 11:04:24+0000', tz='UTC'),
 Timestamp('2015-06-30 20:03:50+0000', tz='UTC'))
```

PREPARE DATASET:

SPLIT TRAINING AND VALIDATION SET:

[41]:
```
from sklearn.model_selection import train_test_split
```

[44]:
```
train_df,val_df = train_test_split(df, test_size = 0.2, random_state = 42)
```

[45]:
```
len(train_df), len(val_df)
```

[45]:
```
(441960, 110490)
```

FILL / REMOVE MISSING VALUES:

[176]:
```
train_df = train_df.dropna()
val_df = val_df.dropna()
```

EXTRACT INPUTS AND OUTPUTS:

[47]:
```
train_df.columns
```

[47]:
```
Index(['fare_amount', 'pickup_datetime', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

[48]:
```
input_cols = ['pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'passenger_count']
target_col = 'fare_amount'
```

TRAINING:

[49]:
```
train_inputs = train_df[input_cols]
train_targets = train_df[target_col]
```

VALIDATION:

[50]:
```
val_inputs = val_df[input_cols]
val_targets = val_df[target_col]
```

TEST:

[52]:
```
test_inputs = test_df[input_cols]
```

TRAINING HARDCODED AND BASELINE MODELS

HARDCODED MODEL:

[54]:
```
import numpy as np
```

[55]:
```
class MeanRegressor():
    def fit(self, inputs, targets):
        self.mean = targets.mean()

    def predict(self, inputs):
        return np.full(inputs.shape[0], self.mean)
```

[56]:
```
mean_model = MeanRegressor()
```

[57]:
```
mean_model.fit(train_inputs, train_targets)
```

[58]:
```
mean_model.mean
```

[58]:
```
11.354714
```

[60]:
```
train_preds = mean_model.predict(train_inputs)
train_preds
```

[60]:
```
array([11.354714, 11.354714, 11.354714, ..., 11.354714, 11.354714,
       11.354714], dtype=float32)
```

[61]:
```
val_preds = mean_model.predict(val_inputs)
val_preds
```

[61]:
```
array([11.354714, 11.354714, 11.354714, ..., 11.354714, 11.354714,
       11.354714], dtype=float32)
```

[62]:
```
from sklearn.metrics import mean_squared_error, root_mean_squared_error
```

[64]:
```
def rmse(targets, preds):
    return root_mean_squared_error(targets,preds)
```

[68]:
```
train_rmse = rmse(train_targets, train_preds)
train_rmse
```

[68]:
```
9.789782
```

[69]:
```
val_rmse = rmse(val_targets, val_preds)
val_rmse
```

[69]:
```
9.899954
```

BASELINE MODEL:

[70]:
```
from sklearn.linear_model import LinearRegression
```

[72]:
```
linreg_model = LinearRegression()
```

[73]:
```
linreg_model.fit(train_inputs, train_targets)
```

[73]:
```
LinearRegression
LinearRegression()
```

[76]:
```
train_preds = linreg_model.predict(train_inputs)
train_preds
```

[76]:
```
array([11.546237, 11.28461 , 11.28414 , ..., 11.458918, 11.284281,
       11.284448], dtype=float32)
```

[77]:
```
train_targets
```

[77]:
```
353352     6.0
360070     3.7
372609    10.0
550895     8.9
444151     7.3
          ...
110268     9.3
259178    18.5
365838    10.1
131932    10.9
121958     9.5
Name: fare_amount, Length: 441960, dtype: float32
```

[78]:
```
rmse(train_preds, train_targets)
```

[78]:
```
9.788632
```

[80]:
```
val_preds = linreg_model.predict(val_inputs)
val_preds
```

[80]:
```
array([11.284328 , 11.284496 , 11.2847805, ..., 11.8045   , 11.284433 ,
       11.284133 ], dtype=float32)
```

[81]:
```
rmse(val_preds, val_targets)
```

[81]:
```
9.898088
```

MAKE PREDICTIONS AND SUBMIT TO KAGGLE:

[82]:
```
test_preds = linreg_model.predict(test_inputs)
test_preds
```

[82]:
```
array([11.28428  , 11.284634 , 11.284384 , ..., 11.721249 , 11.7207985,
       11.720594 ], dtype=float32)
```

[84]:
```
submission_df = pd.read_csv(data_dir+'/sample_submission.csv')
submission_df
```

[84]:

	key	fare_amount
0	2015-01-27 13:08:24.0000002	11.35
1	2015-01-27 13:08:24.0000003	11.35
2	2011-10-08 11:53:44.0000002	11.35
3	2012-12-01 21:12:12.0000002	11.35
4	2012-12-01 21:12:12.0000003	11.35
...	...	...
9909	2015-05-10 12:37:51.0000002	11.35
9910	2015-01-12 17:05:51.0000001	11.35
9911	2015-04-19 20:44:15.0000001	11.35
9912	2015-01-31 01:05:19.0000005	11.35
9913	2015-01-18 14:06:23.0000006	11.35

9914 rows × 2 columns

[85]:
```
def generate_submission(test_preds, fname):
    sub_df = pd.read_csv(data_dir+'/sample_submission.csv')
    sub_df['fare_amount'] = test_preds
    sub_df.to_csv(fname, index=None)
```

[86]:
```
generate_submission(test_preds, 'linreg_submission.csv')
```

FEATURE ENGINEERING:

EXTRACT PARTS OF DATE:

[87]:
```
def add_dateparts(df, col):
    df[col + '_year'] = df[col].dt.year
    df[col + '_month'] = df[col].dt.month
    df[col + '_day'] = df[col].dt.day
    df[col + '_weekday'] = df[col].dt.weekday
    df[col + '_hour'] = df[col].dt.hour
```

[89]:
```
add_dateparts(train_df,'pickup_datetime')
```

[90]:
```
add_dateparts(val_df,'pickup_datetime')
```

[91]:
```
add_dateparts(test_df,'pickup_datetime')
```

[92]:
```
train_df
```

[92]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_datetime_year	pickup_datetime_mo
353352	6.0	2015-04-12 03:40:38+00:00	-73.993652	40.741543	-73.977974	40.742352	4.0	2015	
360070	3.7	2011-01-26 19:21:00+00:00	-73.993805	40.724579	-73.993805	40.724579	1.0	2011	
372609	10.0	2012-10-03 10:40:17+00:00	-73.959160	40.780750	-73.969116	40.761230	1.0	2012	
550895	8.9	2012-03-14 13:44:27+00:00	-73.952187	40.783951	-73.978645	40.772602	1.0	2012	
444151	7.3	2012-02-05 15:33:00+00:00	-73.977112	40.746834	-73.991104	40.750404	2.0	2012	
...	...	...	...	...	...	...	...	...	
110268	9.3	2009-09-06 16:12:00+00:00	-73.987152	40.750633	-73.979073	40.763168	1.0	2009	
259178	18.5	2009-04-12 09:58:56+00:00	-73.972656	40.764042	-74.013176	40.707840	2.0	2009	
365838	10.1	2012-07-12 19:30:00+00:00	-73.991982	40.749767	-73.989845	40.720551	3.0	2012	
		2011-02-17							

131932	10.9	18:33:00+00:00	-73.969055	40.761398	-73.990814	40.751328	1.0	2011
121958	9.5	2015-01-14 17:35:03+00:00	-73.983040	40.742142	-74.002510	40.727650	1.0	2015

441960 rows × 12 columns

ADD DISTANCE BETWEEN PICKUP AND DROP:

[93]:
```
import numpy as np

def haversine_np(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)

    All args must be of equal length.

    """
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2.0)**2

    c = 2 * np.arcsin(np.sqrt(a))
    km = 6367 * c
    return km
```

[94]:
```
def add_trip_distance (df):
    df['trip_distance'] = haversine_np(df['pickup_longitude'],
                                       df['pickup_latitude'],
                                       df['dropoff_longitude'],
                                       df['dropoff_latitude'])
```

[95]:
```
add_trip_distance(train_df)
```

[96]:
```
add_trip_distance(val_df)
```

[97]:
```
add_trip_distance(test_df)
```

[98]:
```
train_df
```

[98]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_datetime_year	pickup_datetime_mo
353352	6.0	2015-04-12 03:40:38+00:00	-73.993652	40.741543	-73.977974	40.742352	4.0	2015	
360070	3.7	2011-01-26 19:21:00+00:00	-73.993805	40.724579	-73.993805	40.724579	1.0	2011	
372609	10.0	2012-10-03 10:40:17+00:00	-73.959160	40.780750	-73.969116	40.761230	1.0	2012	
550895	8.9	2012-03-14 13:44:27+00:00	-73.952187	40.783951	-73.978645	40.772602	1.0	2012	
444151	7.3	2012-02-05 15:33:00+00:00	-73.977112	40.746834	-73.991104	40.750404	2.0	2012	
...	...	...	...	...	...	...	...	...	
110268	9.3	2009-09-06 16:12:00+00:00	-73.987152	40.750633	-73.979073	40.763168	1.0	2009	
259178	18.5	2009-04-12 09:58:56+00:00	-73.972656	40.764042	-74.013176	40.707840	2.0	2009	
365838	10.1	2012-07-12 19:30:00+00:00	-73.991982	40.749767	-73.989845	40.720551	3.0	2012	
131932	10.9	2011-02-17 18:33:00+00:00	-73.969055	40.761398	-73.990814	40.751328	1.0	2011	
121958	9.5	2015-01-14 17:35:03+00:00	-73.983040	40.742142	-74.002510	40.727650	1.0	2015	

441960 rows × 13 columns

ADD DISTANCE FROM POPULAR LANDMARKS:

[99]:
```
jfk_lonlat = -73.7781, 40.6413
lga_lonlat = -73.8740, 40.7769
ewr_lonlat = -74.1745, 40.6895
met_lonlat = -73.9632, 40.7794
wtc_lonlat = -74.0099, 40.7126
```

[100]:
```
def add_landmark_dropoff_distance(df, landmark_name, landmark_lonlat):
    lon, lat = landmark_lonlat
    df[landmark_name + '_drop_distance'] = haversine_np(lon, lat, df['dropoff_longitude'], df['dropoff_latitude'])
```

[101]:
```
%%time
for a_df in [train_df, val_df, test_df]:
    for name, lonlat in [('jfk', jfk_lonlat), ('lga', lga_lonlat), ('ewr', ewr_lonlat), ('met', met_lonlat), ('wtc', wtc_lonlat)]:
        add_landmark_dropoff_distance(a_df, name, lonlat)
```

```
CPU times: total: 78.1 ms
Wall time: 109 ms
```

[102]:
```
train_df
```

[102]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_datetime_year	pickup_datetime_mo
353352	6.0	2015-04-12 03:40:38+00:00	-73.993652	40.741543	-73.977974	40.742352	4.0	2015	
360070	3.7	2011-01-26 19:21:00+00:00	-73.993805	40.724579	-73.993805	40.724579	1.0	2011	
372609	10.0	2012-10-03 10:40:17+00:00	-73.959160	40.780750	-73.969116	40.761230	1.0	2012	
550895	8.9	2012-03-14 13:44:27+00:00	-73.952187	40.783951	-73.978645	40.772602	1.0	2012	
444151	7.3	2012-02-05 15:33:00+00:00	-73.977112	40.746834	-73.991104	40.750404	2.0	2012	
...	...	...	...	...	...	...	...	...	
110268	9.3	2009-09-06 16:12:00+00:00	-73.987152	40.750633	-73.979073	40.763168	1.0	2009	
259178	18.5	2009-04-12 09:58:56+00:00	-73.972656	40.764042	-74.013176	40.707840	2.0	2009	
365838	10.1	2012-07-12 19:30:00+00:00	-73.991982	40.749767	-73.989845	40.720551	3.0	2012	
131932	10.9	2011-02-17 18:33:00+00:00	-73.969055	40.761398	-73.990814	40.751328	1.0	2011	
121958	9.5	2015-01-14 17:35:03+00:00	-73.983040	40.742142	-74.002510	40.727650	1.0	2015	

441960 rows × 18 columns

REMOVE OUTLIERS AND INVALID DATA:

[103]:
```
def remove_outliers(df):
    return df[(df['fare_amount'] >= 1.) &
              (df['fare_amount'] <= 500.) &
              (df['pickup_longitude'] >= -75) &
              (df['pickup_longitude'] <= -72) &
              (df['dropoff_longitude'] >= -75) &
              (df['dropoff_longitude'] <= -72) &
              (df['pickup_latitude'] >= 40) &
              (df['pickup_latitude'] <= 42) &
              (df['dropoff_latitude'] >=40) &
              (df['dropoff_latitude'] <= 42) &
              (df['passenger_count'] >= 1) &
              (df['passenger_count'] <= 6)]
```

[123]:
```
train_df = remove_outliers(train_df)
train_df
```

[123]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_datetime_year	pickup_datetime_mo
353352	6.0	2015-04-12 03:40:38+00:00	-73.993652	40.741543	-73.977974	40.742352	4.0	2015	
360070	3.7	2011-01-26 19:21:00+00:00	-73.993805	40.724579	-73.993805	40.724579	1.0	2011	
372609	10.0	2012-10-03 10:40:17+00:00	-73.959160	40.780750	-73.969116	40.761230	1.0	2012	
550895	8.9	2012-03-14 13:44:27+00:00	-73.952187	40.783951	-73.978645	40.772602	1.0	2012	
444151	7.3	2012-02-05 15:33:00+00:00	-73.977112	40.746834	-73.991104	40.750404	2.0	2012	
...	...	...	...	...	...	...	...	...	
110268	9.3	2009-09-06 16:12:00+00:00	-73.987152	40.750633	-73.979073	40.763168	1.0	2009	
259178	18.5	2009-04-12 09:58:56+00:00	-73.972656	40.764042	-74.013176	40.707840	2.0	2009	
365838	10.1	2012-07-12 19:30:00+00:00	-73.991982	40.749767	-73.989845	40.720551	3.0	2012	

131932	10.9	2011-02-17 18:33:00+00:00	-73.969055	40.761398	-73.990814	40.751328	1.0	2011
121958	9.5	2015-01-14 17:35:03+00:00	-73.983040	40.742142	-74.002510	40.727650	1.0	2015

431098 rows × 18 columns

[124]:
```
val_df = remove_outliers(val_df)
val_df
```

[124]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_datetime_year	pickup_datetime_mo
15971	14.000000	2015-05-19 09:27:24+00:00	-73.995834	40.759190	-73.973679	40.739086	1.0	2015	
149839	6.500000	2010-04-10 15:07:51+00:00	-73.977386	40.738335	-73.976143	40.751205	1.0	2010	
515867	49.570000	2009-07-25 14:11:00+00:00	-73.983910	40.749470	-73.787170	40.646645	1.0	2009	
90307	49.700001	2011-11-11 19:09:21+00:00	-73.790794	40.643463	-73.972252	40.690182	1.0	2011	
287032	8.500000	2015-03-09 18:06:44+00:00	-73.976593	40.761944	-73.991463	40.750309	2.0	2015	
...	...	...	...	...	...	...	...	...	
506435	3.700000	2010-05-30 10:44:30+00:00	-73.959457	40.774727	-73.966301	40.772179	1.0	2010	
467556	6.100000	2010-04-03 20:16:00+00:00	-73.968567	40.761238	-73.983406	40.750019	3.0	2010	
19482	7.300000	2010-04-26 00:32:00+00:00	-73.986725	40.755920	-73.985855	40.731171	1.0	2010	
382260	32.900002	2011-07-07 16:10:59+00:00	-73.980057	40.760334	-73.872589	40.774300	1.0	2011	
18838	11.500000	2015-02-27 12:10:08+00:00	-73.955406	40.782417	-73.960434	40.767666	1.0	2015	

107742 rows × 18 columns

SAVE INTERMEDIATE DATAFRAMES:

[125]:
```
train_df.to_parquet('train.parquet')
val_df.to_parquet('val.parquet')
test_df.to_parquet('test.parquet')
```

SPLIT INPUTS AND TARGETS:

[126]:
```
train_df.columns
```

[126]:
```
Index(['fare_amount', 'pickup_datetime', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count',
       'pickup_datetime_year', 'pickup_datetime_month', 'pickup_datetime_day',
       'pickup_datetime_weekday', 'pickup_datetime_hour', 'trip_distance',
       'jfk_drop_distance', 'lga_drop_distance', 'ewr_drop_distance',
       'met_drop_distance', 'wtc_drop_distance'],
      dtype='object')
```

[127]:
```
input_cols = ['pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count',
       'pickup_datetime_year', 'pickup_datetime_month', 'pickup_datetime_day',
       'pickup_datetime_weekday', 'pickup_datetime_hour', 'trip_distance',
       'jfk_drop_distance', 'lga_drop_distance', 'ewr_drop_distance',
       'met_drop_distance', 'wtc_drop_distance']
```

[128]:
```
target_col = ['fare_amount']
```

[129]:
```
train_inputs = train_df[input_cols]
train_targets = train_df[target_col]
```

[130]:
```
val_inputs = val_df[input_cols]
val_targets = val_df[target_col]
```

[131]:
```
test_inputs = test_df[input_cols]
```

[132]:
```
def evaluate(model):
    train_preds = model.predict(train_inputs)
    train_rmse = root_mean_squared_error(train_targets, train_preds)
    val_preds = model.predict(val_inputs)
    val_rmse = root_mean_squared_error(val_targets, val_preds)
    return train_rmse, val_rmse, train_preds, val_preds
```

[133]:
```
def predict_and_submit(model, fname):
    test_preds = model.predict(test_inputs)
    sub_df = pd.read_csv(data_dir+'/sample_submission.csv')
    sub_df['fare_amount'] = test_preds
    sub_df.to_csv(fname, index=None)
    return sub_df
```

RIDGE REGRESSION:

[177]:
```
from sklearn.linear_model import Ridge
```

[178]:
```
model1 = Ridge(random_state = 42)
```

[179]:
```
model1.fit(train_inputs,train_targets)
```

[179]:
```
Ridge(random_state=42)
```

[180]:
```
evaluate(model1)
```

[180]:
```
(5.049315152711231,
 5.217865657340075,
 array([[ 8.12925918],
        [ 4.11578439],
        [ 8.75063014],
        ...,
        [10.47234932],
        [ 8.2305928 ],
        [10.58672774]]),
 array([[10.91955339],
        [ 6.20493172],
        [46.21787888],
        ...,
        [ 8.0463052 ],
        [25.56885585],
        [ 8.45342102]]))
```

[181]:
```
predict_and_submit(model1, 'ridge_submission.csv')
```

[181]:

	key	fare_amount
0	2015-01-27 13:08:24.0000002	10.082151
1	2015-01-27 13:08:24.0000003	11.399499
2	2011-10-08 11:53:44.0000002	5.356923
3	2012-12-01 21:12:12.0000002	8.763571
4	2012-12-01 21:12:12.0000003	14.609918
...	...	...
9909	2015-05-10 12:37:51.0000002	9.024995
9910	2015-01-12 17:05:51.0000001	11.218598
9911	2015-04-19 20:44:15.0000001	47.926481
9912	2015-01-31 01:05:19.0000005	22.600022
9913	2015-01-18 14:06:23.0000006	8.981499

9914 rows × 2 columns

RANDOM FOREST REGRESSOR:

[139]:
```
from sklearn.ensemble import RandomForestRegressor
```

[142]:
```
model2 = RandomForestRegressor(random_state = 42, n_jobs = -1, max_depth = 10, n_estimators = 50)
```

[143]:
```
model2.fit(train_inputs, train_targets)
```

```
C:\Users\adity\anaconda3\Lib\site-packages\sklearn\base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
```

[143]:
```
RandomForestRegressor(max_depth=10, n_estimators=50, n_jobs=-1, random_state=42)
```

[144]:
```
evaluate(model2)
```

[144]:
```
(3.6022913838561528,
 4.167257655248953,
 array([ 7.06907907,  9.09651906,  9.09834234, ..., 10.39574834,
         7.73362485, 10.36404777]),
 array([12.53923917,  6.14414213, 47.3274893 , ...,  8.32839336,
        29.20883778,  8.27729971]))
```

[145]:
```
predict_and_submit(model2,'rf_submission.csv')
```

[145]:

	key	fare_amount
0	2015-01-27 13:08:24.0000002	10.522156
1	2015-01-27 13:08:24.0000003	10.499965
2	2011-10-08 11:53:44.0000002	5.049496
3	2012-12-01 21:12:12.0000002	8.486169
4	2012-12-01 21:12:12.0000003	14.316837
...	...	...
9909	2015-05-10 12:37:51.0000002	8.873896
9910	2015-01-12 17:05:51.0000001	12.551152
9911	2015-04-19 20:44:15.0000001	55.266349
9912	2015-01-31 01:05:19.0000005	21.518925
9913	2015-01-18 14:06:23.0000006	6.864720

9914 rows × 2 columns

GRADIENT BOOSTING:

[146]:
```
from xgboost import XGBRegressor
```

[147]:
```
model3 = XGBRegressor(random_state = 42, objective = 'reg:squarederror')
```

[149]:
```
model3.fit(train_inputs,train_targets)
```

[149]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=42, ...)
```

[150]:
```
evaluate(model3)
```

[150]:
```
(3.166428,
 3.9793801,
 array([ 6.653713 ,  8.931815 , 10.276588 , ..., 12.1582575,  9.564414 ,
         9.845059 ], dtype=float32),
 array([14.876896 ,  5.986584 , 47.171738 , ...,  7.5201406, 30.39466  ,
         8.515031 ], dtype=float32))
```

[151]:
```
predict_and_submit(model3, 'xgb_submission.csv')
```

[151]:

	key	fare_amount
0	2015-01-27 13:08:24.0000002	10.819051
1	2015-01-27 13:08:24.0000003	11.724770
2	2011-10-08 11:53:44.0000002	4.443203
3	2012-12-01 21:12:12.0000002	9.096115
4	2012-12-01 21:12:12.0000003	16.050816
...	...	...
9909	2015-05-10 12:37:51.0000002	9.119739
9910	2015-01-12 17:05:51.0000001	11.764132
9911	2015-04-19 20:44:15.0000001	53.424809
9912	2015-01-31 01:05:19.0000005	18.867956
9913	2015-01-18 14:06:23.0000006	6.916947

9914 rows × 2 columns

TUNE HYPERPARAMETERS:

[158]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import root_mean_squared_error

def test_params(ModelClass, **params):
    """Trains a model with the given parameters and returns training & validation RMSE"""
    model = ModelClass(**params).fit(train_inputs, train_targets)
    train_rmse = root_mean_squared_error(model.predict(train_inputs), train_targets)
```

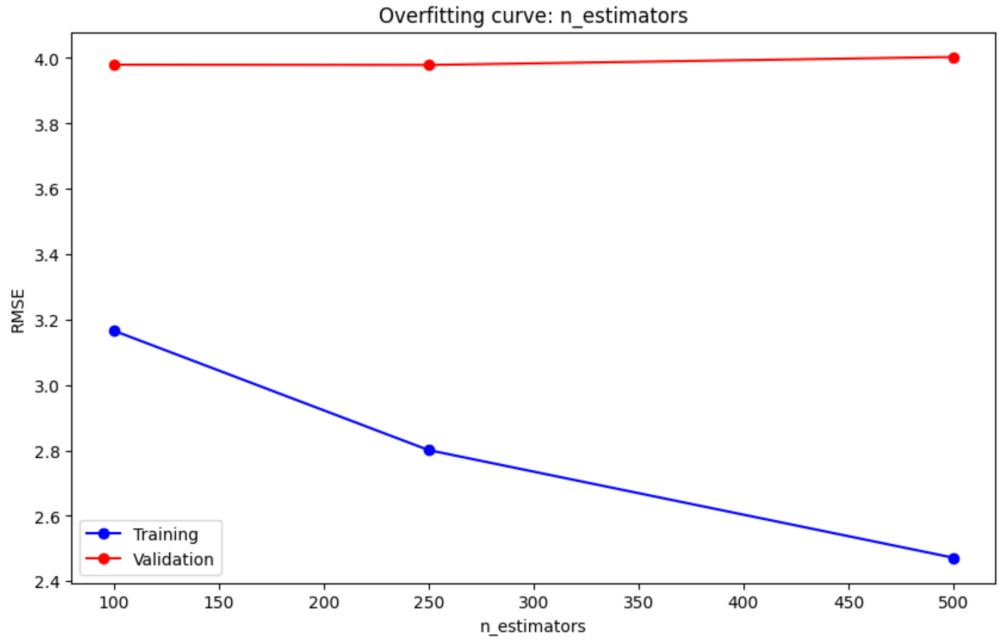
```
    val_rmse = root_mean_squared_error(model.predict(val_inputs), val_targets)
    return train_rmse, val_rmse

def test_param_and_plot(ModelClass, param_name, param_values, **other_params):
    """Trains multiple models by varying the value of param_name according to param_values"""
    train_errors, val_errors = [], []
    for value in param_values:
        params = dict(other_params)
        params[param_name] = value
        train_rmse, val_rmse = test_params(ModelClass, **params)
        train_errors.append(train_rmse)
        val_errors.append(val_rmse)

    plt.figure(figsize=(10,6))
    plt.title('Overfitting curve: ' + param_name)
    plt.plot(param_values, train_errors, 'b-o')
    plt.plot(param_values, val_errors, 'r-o')
    plt.xlabel(param_name)
    plt.ylabel('RMSE')
    plt.legend(['Training', 'Validation'])
```

[159]:
```
best_params = {
    'random_state': 42,
    'n_jobs': -1,
    'objective': 'reg:squarederror'}
```

NO. OF TREES:

[160]:
```
%%time
test_param_and_plot(XGBRegressor, 'n_estimators', [100, 250, 500], **best_params)
```

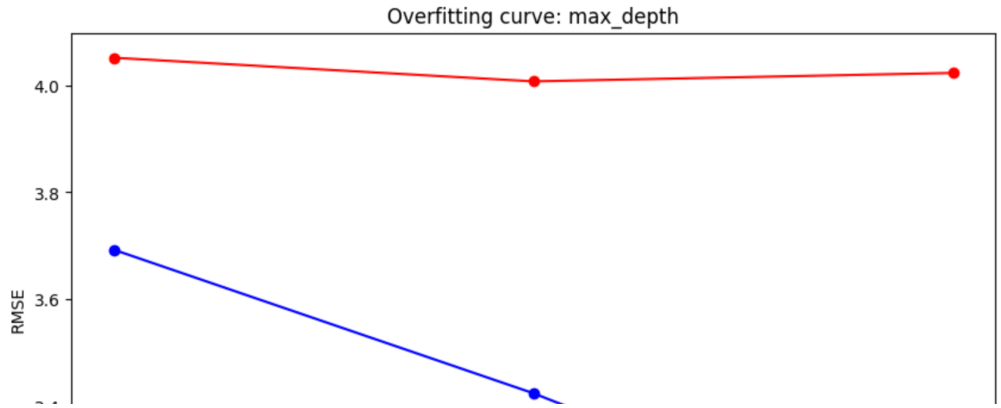
```
CPU times: total: 41.7 s
Wall time: 6.89 s
```

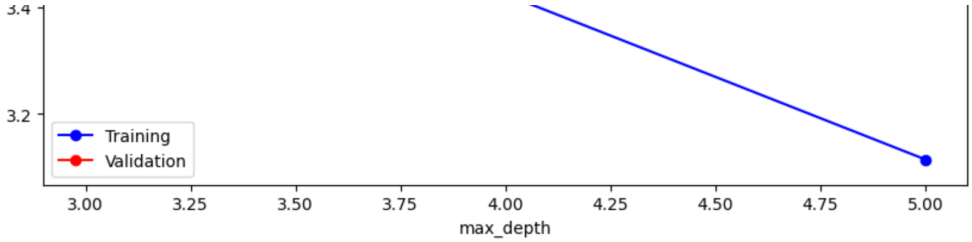


[161]:
```
best_params['n_estimators'] = 250
```

MAX_DEPTH :

[163]:
```
test_param_and_plot(XGBRegressor,'max_depth',[3,4,5], **best_params)
```



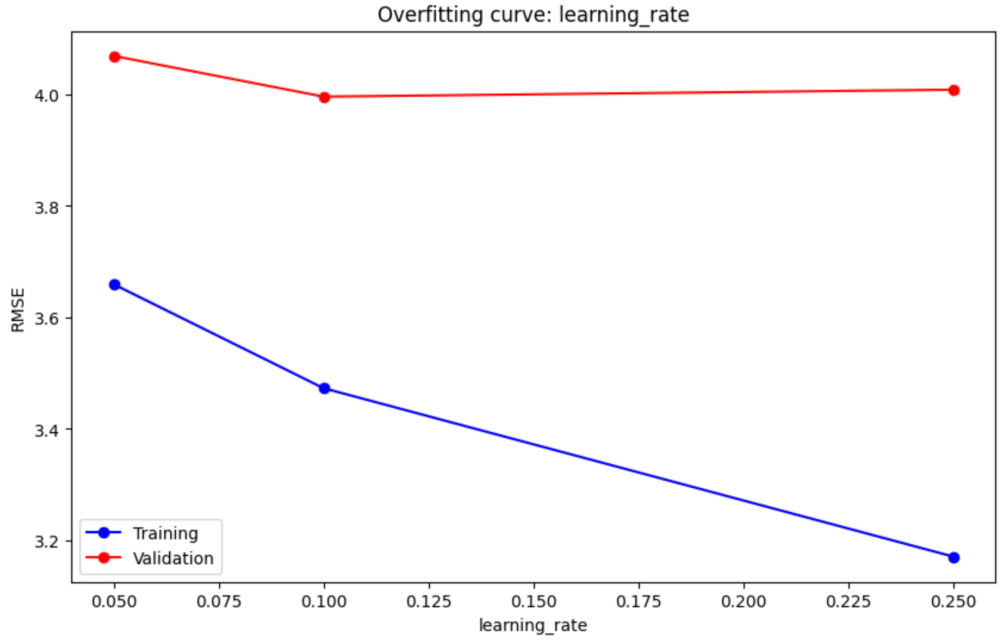


[165]:
```
best_params['max_depth'] = 5
```

LEARNING_RATE:

[167]:
```
%%time
test_param_and_plot(XGBRegressor, 'learning_rate', [0.05, 0.1, 0.25], **best_params)
```

```
CPU times: total: 42.1 s
Wall time: 6.35 s
```



[168]:
```
best_params['learning_rate'] = 0.25
```

[169]:
```
xgb_model_final = XGBRegressor(objective='reg:squarederror', n_jobs=-1, random_state=42,
                               n_estimators=500, max_depth=5, learning_rate=0.1,
                               subsample=0.8, colsample_bytree=0.8)
```

[170]:
```
xgb_model_final.fit(train_inputs,train_targets)
```

[170]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=5, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=500, n_jobs=-1,
             num_parallel_tree=None, random_state=42, ...)
```

[182]:
```
evaluate(xgb_model_final)
```

[182]:
```
(3.2074616,
 3.9877155,
 array([ 6.3942747,  8.156118 ,  9.941316 , ..., 11.766499 ,  9.200898 ,
        10.145953 ], dtype=float32),
 array([14.554565 ,  5.795544 , 48.046738 , ...,  7.6102695, 30.916338 ,
         8.5561   ], dtype=float32))
```

[173]:
```
predict_and_submit(xgb_model_final, 'xgb_tuned_submission.csv')
```

[173]:

	key	fare_amount
0	2015-01-27 13:08:24.0000002	10.849108
1	2015-01-27 13:08:24.0000003	10.850513
2	2011-10-08 11:53:44.0000002	4.986358

3	2012-12-01 21:12:12.0000002	8.618697
4	2012-12-01 21:12:12.0000003	15.675056
...	...	...
9909	2015-05-10 12:37:51.0000002	8.669248
9910	2015-01-12 17:05:51.0000001	11.662894
9911	2015-04-19 20:44:15.0000001	53.927334
9912	2015-01-31 01:05:19.0000005	19.240555
9913	2015-01-18 14:06:23.0000006	6.641398

9914 rows × 2 columns

[174]:
```
from IPython.display import FileLink
```

[175]:
```
FileLink('xgb_tuned_submission.csv')
```

[175]: xgb_tuned_submission.csv

[ ]: