In [67]:
```
import pandas as pd
import numpy as np
from sklearn import linear_model

df = pd.read_csv('./AB_NYC_2019.csv')
```

In [68]:
```
df.head()
```

Out[68]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	lon
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73

In [48]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
id                                48895 non-null int64
name                              48879 non-null object
host_id                           48895 non-null int64
host_name                         48874 non-null object
neighbourhood_group               48895 non-null object
neighbourhood                     48895 non-null object
latitude                          48895 non-null float64
longitude                         48895 non-null float64
room_type                         48895 non-null object
price                             48895 non-null int64
minimum_nights                    48895 non-null int64
number_of_reviews                 48895 non-null int64
last_review                       38843 non-null object
reviews_per_month                 38843 non-null float64
calculated_host_listings_count    48895 non-null int64
availability_365                  48895 non-null int64
dtypes: float64(3), int64(7), object(6)
memory usage: 6.0+ MB
```

In [49]:
```
df.shape
```

Out[49]:
```
(48895, 16)
```

In [50]:
```
ost_id', 'host_name', 'neighbourhood', 'last_review', 'reviews_per_month'], a
```

In [51]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 9 columns):
neighbourhood_group               48895 non-null object
latitude                          48895 non-null float64
longitude                         48895 non-null float64
room_type                         48895 non-null object
price                             48895 non-null int64
minimum_nights                    48895 non-null int64
number_of_reviews                 48895 non-null int64
calculated_host_listings_count    48895 non-null int64
availability_365                  48895 non-null int64
dtypes: float64(2), int64(5), object(2)
memory usage: 3.4+ MB
```

In [52]:
```
print("Dimension of the data: ", df.shape)

no_of_rows = df.shape[0]
no_of_columns = df.shape[1]

print("No. of Rows: %d" % no_of_rows)
print("No. of Columns: %d" % no_of_columns)
```

```
Dimension of the data:  (48895, 9)
No. of Rows: 48895
No. of Columns: 9
```

In [ ]:
```

```

In [53]:
```
df = df.dropna()
allData = df


y = df['price'] # 1D targer vector
X = df.drop(['price'], axis=1)  # Data Matrix containing all features exclu
```

In [54]:
```
df.shape
```

Out[54]:
```
(48895, 9)
```

In [55]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df['neighbourhood_group'] = le.fit_transform(df['neighbourhood_group'])
df['room_type'] = le.fit_transform(df['room_type'])
```

In [56]:
```
df.head()
```

Out[56]:

	neighbourhood_group	latitude	longitude	room_type	price	minimum_nights	number_of_reviews
0	1	40.64749	-73.97237	1	149	1	9
1	2	40.75362	-73.98377	0	225	1	45
2	2	40.80902	-73.94190	1	150	3	0
3	1	40.68514	-73.95976	0	89	1	270
4	2	40.79851	-73.94399	0	80	10	9

In [57]:
```
df.describe()
```

Out[57]:

	neighbourhood_group	latitude	longitude	room_type	price	minimum_nig
count	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	48895.000
mean	1.675345	40.728949	-73.952170	0.504060	152.720687	7.029
std	0.735816	0.054530	0.046157	0.545379	240.154170	20.510
min	0.000000	40.499790	-74.244420	0.000000	0.000000	1.000
25%	1.000000	40.690100	-73.983070	0.000000	69.000000	1.000
50%	2.000000	40.723070	-73.955680	0.000000	106.000000	3.000
75%	2.000000	40.763115	-73.936275	1.000000	175.000000	5.000
max	4.000000	40.913060	-73.712990	2.000000	10000.000000	1250.000

In [58]:
```
from sklearn.model_selection import train_test_split

X_df = X;
y_df = y;
Y = df['price']
X = df#[['neighbourhood_group', 'longitude', 'room_type', 'availability_365
```

In [ ]:

In [ ]:

In [59]:
```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

X = preprocessing.normalize(X)
X = np.hstack((np.ones( (len(df['price']) ,1)), X))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(39116, 10)
(9779, 10)
(39116,)
(9779,)
```

In [ ]:
```

```

In [60]:
```
# Calculate mae and rmse
def evaluate_predictions(predictions, true):
    mae = np.mean(abs(predictions - true))
    rmse = np.sqrt(np.mean((predictions - true) ** 2))

    return mae, rmse
```

Naive baseline

In [61]:
```
# Naive baseline is the median

import numpy as np


median_pred = y_df.median()

print(median_pred)

median_preds = [median_pred for _ in range(len(y_df))]
true = y_df


# Display the naive baseline metrics
mb_mae, mb_rmse = evaluate_predictions(median_preds, true)
print('Median Baseline  MAE: {:.4f}'.format(mb_mae))
print('Median Baseline RMSE: {:.4f}'.format(mb_rmse))
```

```
106.0
Median Baseline  MAE: 84.0945
Median Baseline RMSE: 244.6542
```

Linear Regression

In [62]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Create linear regression object
model = LinearRegression()

# Train the model using the traing data and label
model.fit(X_train, y_train)

# The intercept
# b = model.intercept_
# print('Intercept: \n',b )

# # The feature weights (coefficients) in an array
# m = model.coef_
# print('Coefficients: \n', m)


# # Show the prediction formula with the intercept and feature weight values
# print("\nPrediction Formula: ")
# print(' y(predicted) = {0} +  x_1 * {1} +  x_2 * {2} + x_3 * {3} + x_4 * {
#       .format(b, m[0], m[1], m[2], m[3], m[4]))


# # Make predictions using the test data
y_predicted = model.predict(X_test)

mb_mae, mb_rmse = evaluate_predictions(y_test, y_predicted)
print('  MAE: {:.4f}'.format(mb_mae))
print(' RMSE: {:.4f}'.format(mb_rmse))
```

```
  MAE: 63.9448
 RMSE: 154.0035
```

In [ ]:
```

```

Logistic Regression

In [63]:
```
X_train = X_train#[:10000]
y_train = y_train#[:10000]
```

In [64]:
```
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Create linear regression object
model = SGDRegressor(max_iter=10000, penalty='l1', verbose=False)

# Train the model using the traing data and label
model.fit(X_train, y_train)



# Make predictions using the test data
y_predicted = model.predict(X_test)

mb_mae, mb_rmse = evaluate_predictions(y_test, y_predicted)
print('  MAE: {:.4f}'.format(mb_mae))
print(' RMSE: {:.4f}'.format(mb_rmse))
```

```
  MAE: 62.7219
 RMSE: 155.6362
```

In [ ]:
```

```

Lasso

In [69]:
```
model = linear_model.Lasso(alpha=0.1)

model.fit(X_train, y_train)



# Make predictions using the test data
y_predicted = model.predict(X_test)

mb_mae, mb_rmse = evaluate_predictions(y_test, y_predicted)
print('  MAE: {:.4f}'.format(mb_mae))
print(' RMSE: {:.4f}'.format(mb_rmse))
```

```
  MAE: 63.6070
 RMSE: 156.9551
```

Ridge

In [82]:
```
model = linear_model.Ridge(alpha=0.01)

model.fit(X_train, y_train)



# Make predictions using the test data
y_predicted = model.predict(X_test)

mb_mae, mb_rmse = evaluate_predictions(y_test, y_predicted)
print('  MAE: {:.4f}'.format(mb_mae))
print(' RMSE: {:.4f}'.format(mb_rmse))
```

```
  MAE: 63.6774
 RMSE: 153.9679
```

Random forest regression

In [87]:
```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(max_depth=30, random_state=0)

model.fit(X_train, y_train)



# Make predictions using the test data
y_predicted = model.predict(X_test)

mb_mae, mb_rmse = evaluate_predictions(y_test, y_predicted)
print('  MAE: {:.4f}'.format(mb_mae))
print(' RMSE: {:.4f}'.format(mb_rmse))
```

```
  MAE: 0.7193
 RMSE: 3.4502
```

In [ ]:
```

```

In [21]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
```

In [22]:
```
import pickle
filename='gridsearch_res.pkl';
```

In [29]:
```
filehandler = open(filename, 'rb')
res = pickle.load(filehandler)
clf = res['clf']
##or below
```

In [31]:
```
%%time
param_grid = {
   ## 'alpha': 10.0 ** -np.arange(1, 7),
    'loss': ['squared_loss', 'log'],#, 'huber', 'epsilon_insensitive'],
   ## 'penalty': ['l2', 'l1'],#, 'elasticnet'],
    ##'learning_rate': ['constant', 'optimal']#, 'invscaling'],
}
clf = GridSearchCV(model, param_grid)
clf.fit(X_train, y_train)
print("Best score: " + str(clf.best_score_))
```

```
---------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
<timed exec> in <module>()

/util/opt/anaconda/4.3.14/envs/jupyterhub-root/lib/python3.6/site-package
s/sklearn/model_selection/_search.py in fit(self, X, y, groups, **fit_par
ams)
    638                                   error_score=self.error_score)
    639           for parameters, (train, test) in product(candidate_para
ms,
--> 640                                                  cv.split(X, y,
groups)))
    641
    642         # if one choose to see train score, "out" will contain tr
ain score info

/util/opt/anaconda/4.3.14/envs/jupyterhub-root/lib/python3.6/site-package
s/sklearn/externals/joblib/parallel.py in __call__(self, iterable)
    777             # was dispatched. In particular this covers the edge
    778             # case of Parallel used with an exhausted iterator.
--> 779             while self.dispatch_one_batch(iterator):
    780                 self._iterating = True
    781             else:

/util/opt/anaconda/4.3.14/envs/jupyterhub-root/lib/python3.6/site-package
s/sklearn/externals/joblib/parallel.py in dispatch_one_batch(self, iterat
or)
    623                 return False
    624             else:
--> 625                 self._dispatch(tasks)
    626                 return True
    627

/util/opt/anaconda/4.3.14/envs/jupyterhub-root/lib/python3.6/site-package
s/sklearn/externals/joblib/parallel.py in _dispatch(self, batch)
    586         dispatch_timestamp = time.time()
    587         cb = BatchCompletionCallBack(dispatch_timestamp, len(batc
h), self)
--> 588         job = self._backend.apply_async(batch, callback=cb)
    589         self._jobs.append(job)
    590

/util/opt/anaconda/4.3.14/envs/jupyterhub-root/lib/python3.6/site-package
s/sklearn/externals/joblib/_parallel_backends.py in apply_async(self, fun
```

```
c, callback)
    109     def apply_async(self, func, callback=None):
    110         """Schedule a func to be run"""
--> 111         result = ImmediateResult(func)
    112         if callback:
    113             callback(result)

/util/opt/anaconda/4.3.14/envs/jupyterhub-root/lib/python3.6/site-package
s/sklearn/externals/joblib/_parallel_backends.py in __init__(self, batch)
    330         # Don't delay the application, to avoid keeping the input
    331         # arguments in memory
--> 332         self.results = batch()
    333
    334     def get(self):

/util/opt/anaconda/4.3.14/envs/jupyterhub-root/lib/python3.6/site-package
s/sklearn/externals/joblib/parallel.py in __call__(self)
    129
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in s
elf.items]
    132
    133     def __len__(self):

/util/opt/anaconda/4.3.14/envs/jupyterhub-root/lib/python3.6/site-package
s/sklearn/externals/joblib/parallel.py in <listcomp>(.0)
    129
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in s
elf.items]
    132
    133     def __len__(self):

/util/opt/anaconda/4.3.14/envs/jupyterhub-root/lib/python3.6/site-package
s/sklearn/model_selection/_validation.py in _fit_and_score(estimator, X,
 y, scorer, train, test, verbose, parameters, fit_params, return_train_sc
ore, return_parameters, return_n_test_samples, return_times, error_score)
    442     train_scores = {}
    443     if parameters is not None:
--> 444         estimator.set_params(**parameters)
    445
    446     start_time = time.time()

/util/opt/anaconda/4.3.14/envs/jupyterhub-root/lib/python3.6/site-package
s/sklearn/linear_model/stochastic_gradient.py in set_params(self, *args,
 **kwargs)
     76     def set_params(self, *args, **kwargs):
     77         super(BaseSGD, self).set_params(*args, **kwargs)
---> 78         self._validate_params(set_max_iter=False)
     79         return self
     80

/util/opt/anaconda/4.3.14/envs/jupyterhub-root/lib/python3.6/site-package
s/sklearn/linear_model/stochastic_gradient.py in _validate_params(self, s
et_max_iter)
    106
    107         if self.loss not in self.loss_functions:
```

```
--> 108             raise ValueError("The loss %s is not supported. " % s
elf.loss)
    109
    110         if not set_max_iter:

ValueError: The loss log is not supported.
```

In [25]:
```
#save result
filehandler = open(filename, 'wb')
pickle.dump({'clf':clf, 'param_grid': param_grid}, filehandler)
```

In [26]:
```
best_model = clf.estimator
```

In [27]:
```
best_model.fit(X_train, y_train)
```

Out[27]:
```
SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.01,
       fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling',
       loss='squared_loss', max_iter=10000, n_iter=None, penalty='l1',
       power_t=0.25, random_state=None, shuffle=True, tol=None,
       verbose=False, warm_start=False)
```

In [28]:
```
# Make predictions using the test data
y_predicted = best_model.predict(X_test)

mb_mae, mb_rmse = evaluate_predictions(y_test, y_predicted)
print('  MAE: {:.4f}'.format(mb_mae))
print(' RMSE: {:.4f}'.format(mb_rmse))
```

```
  MAE: 66.3470
 RMSE: 203.4349
```

Read write example

```
import pickle
filehandler = open(filename, 'wb')
pickle.dump(t, filehandler)

## read
filehandler = open(filename, 'rb')
loaded = pickle.load(filehandler)
```

In [ ]:
```
```