Санкт-Петербургский государственный политехнический университет
Институт машиностроения, материалов и транспорта
Кафедра «Мехатроника и Роботостроение» при ЦНИИ РТК

О Т Ч Е Т П О Л А Б О Р А Т О Р Н О Й Р А Б О Т Е № 2

Классификация

по дисциплине «Математические методы интеллектуальных технологий»

Выполнил
студент гр.3341506/90401 <*подпись*> М.А.Борискин

Проверил
<*подпись*> С.Р.Орлова

«24» ноября 2019 г.

Санкт-Петербург
2019

Скрипт приведен в репозитории https://github.com/makaryb/ml_1sem_5kurs/blob/master/lab2/classification/src/firstTask.py

Часть 1 – работа с данными:

1. Сколько записей в базе?

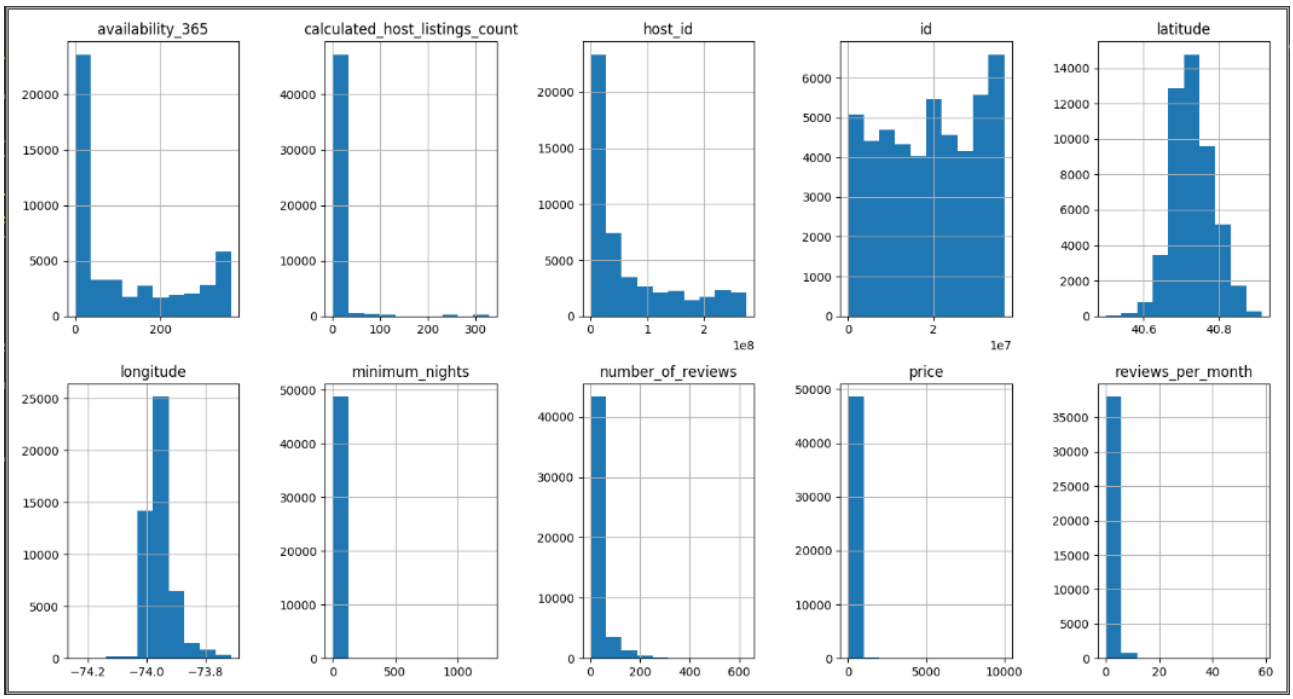
```
def totalABNYC2019():
    print('Количество записей в базе:', len(set(data['id'])), '\n')
```

```
Количество записей в базе: 48895
```

2. Постройте гистограммы всех признаков.

```
def histogram():
    data.info()
    print('\n')

    data.hist(figsize = (15, 8), layout = (2, 5))

    plt.show()
```

```
<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
```



3. Есть ли значения, которые есть смысл отбросить? Есть ли некорректные значения? Если да – укажите, какие, сколько, и что вы с ними собираетесь делать.

```
def areThereAnyMissingItems():
    print(data.isnull().sum())
    print('\n')
    print('Broken items in', 'столбцы \"дата последнего отзыва\" и \"количество отзывов в месяц\"', 'are the same?',
          (data['last_review'].isnull() == data['reviews_per_month'].isnull()).all())
    print('\n')


def fillMissingItems():
    data.loc[data['reviews_per_month'].isnull(), 'reviews_per_month'] = 0


def checkNumericVariables():
    print(data.describe())
    print('\n')


def removeUnnecessaryColumns():
    data.drop(['id', 'name', 'host_name', 'last_review'], axis = 1, inplace = True)


def cleanAbnormalities():
    dataNYC = data[data['price'] > 0]
    dataNYC = data[data['minimum_nights'] < 366]
    dataNYC.reset_index(drop = True, inplace = True)
    dataNYC.info()
    print('\n')
    return dataNYC
```

Некорректные значения: где пустые ячейки. Выведем количество:

```
id                                  0
name                               16
host_id                             0
host_name                          21
neighbourhood_group                 0
neighbourhood                       0
latitude                            0
longitude                           0
room_type                           0
price                               0
minimum_nights                      0
number_of_reviews                   0
last_review                     10052
reviews_per_month               10052
calculated_host_listings_count      0
availability_365                    0
dtype: int64


Broken items in столбцы "дата последнего отзыва" и "количество отзывов в месяц" are the same? True
```

Заполним нулями пустые ячейки в `reviews_per_month`.
Посмотрим насчет некорректных значений:

```
                 id       host_id      latitude     longitude         price  \
count  4.889500e+04  4.889500e+04  48895.000000  48895.000000  48895.000000   
mean   1.901714e+07  6.762001e+07     40.728949    -73.952170    152.720687   
std    1.098311e+07  7.861097e+07      0.054530      0.046157    240.154170   
min    2.539000e+03  2.438000e+03     40.499790    -74.244420      0.000000   
25%    9.471945e+06  7.822033e+06     40.690100    -73.983070     69.000000   
50%    1.967728e+07  3.079382e+07     40.723070    -73.955680    106.000000   
75%    2.915218e+07  1.074344e+08     40.763115    -73.936275    175.000000   
max    3.648724e+07  2.743213e+08     40.913060    -73.712990  10000.000000   

       minimum_nights  number_of_reviews  reviews_per_month  \
count    48895.000000       48895.000000       48895.000000   
mean         7.029962          23.274466           1.090910   
std         20.510550          44.550582           1.597283   
min          1.000000           0.000000           0.000000   
25%          1.000000           1.000000           0.040000   
50%          3.000000           5.000000           0.370000   
75%          5.000000          24.000000           1.580000   
max       1250.000000         629.000000          58.500000   

       calculated_host_listings_count  availability_365  
count                    48895.000000      48895.000000  
mean                         7.143982        112.781327  
std                         32.952519        131.622289  
min                          1.000000          0.000000  
25%                          1.000000          0.000000  
50%                          1.000000         45.000000  
75%                          2.000000        227.000000  
max                        327.000000        365.000000  
```

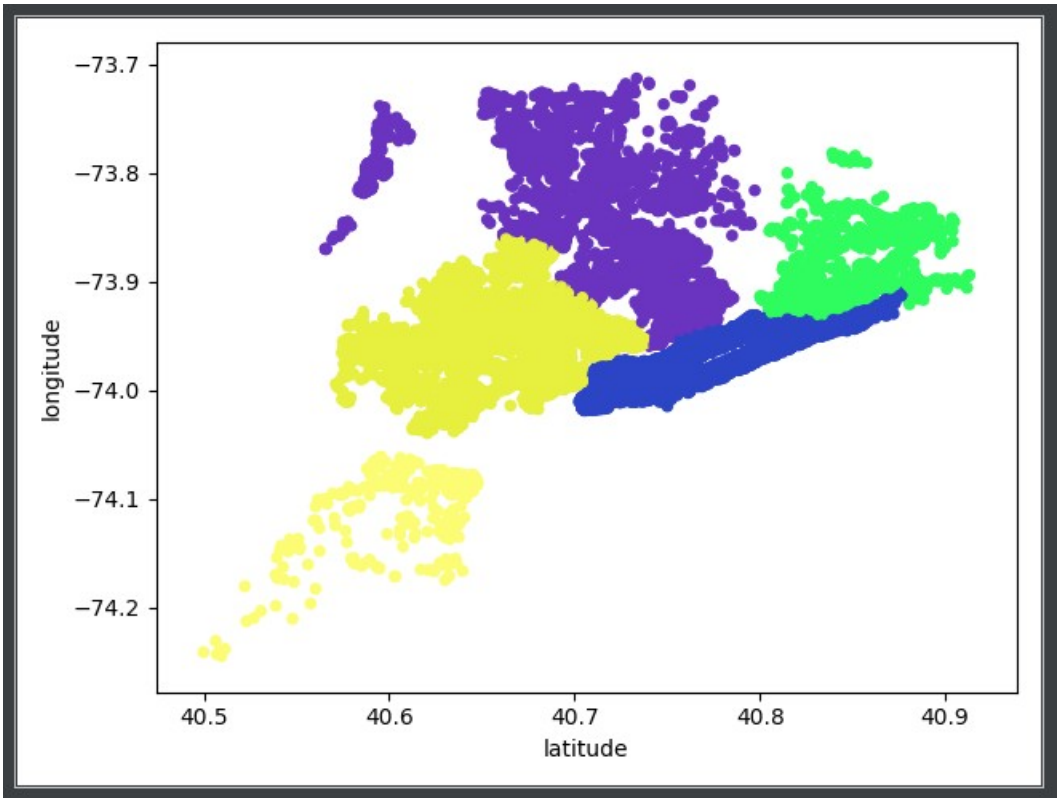
Уберем строки, где в ячейках столбца `price` нули. А также заменим ячейки в столбце `minimum_nights`, где больше 365, на 365.
Также есть смысл отбросить столбцы `id`, `name`, `host_name`, `last_review`, так как смысловой нагрузки для нашего анализа они не несут.
В итоге получается ровный датасет из 12 признаков:

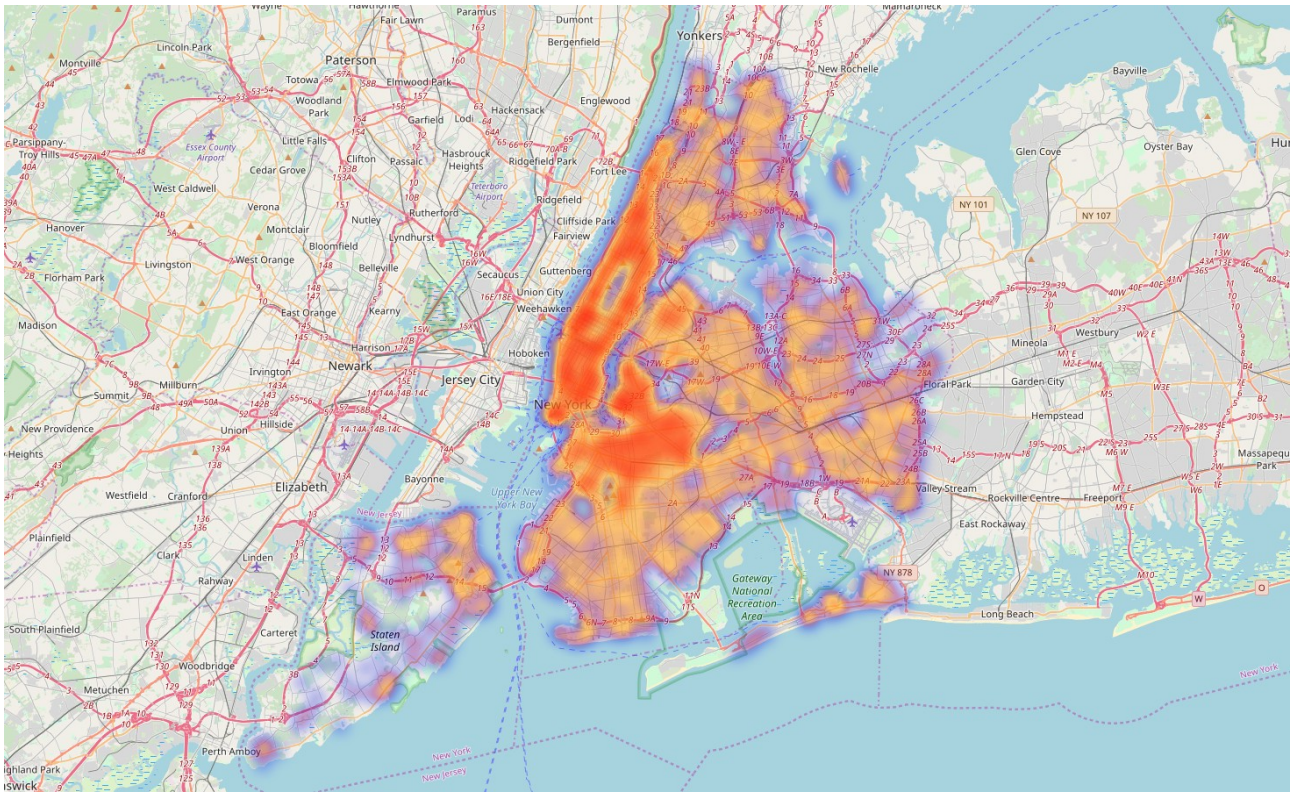
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48881 entries, 0 to 48880
Data columns (total 12 columns):
host_id                             48881 non-null int64
neighbourhood_group                 48881 non-null object
neighbourhood                       48881 non-null object
latitude                            48881 non-null float64
longitude                           48881 non-null float64
room_type                           48881 non-null object
price                               48881 non-null int64
minimum_nights                      48881 non-null int64
number_of_reviews                   48881 non-null int64
reviews_per_month                   48881 non-null float64
calculated_host_listings_count      48881 non-null int64
availability_365                    48881 non-null int64
dtypes: float64(3), int64(6), object(3)
memory usage: 4.5+ MB
```

4. Постройте график, где по осям x и y будут широта и долгота, а цветом помечены районы. Это задание дано, чтобы научиться строить такие графики, ну и красиво получается :) Особые энтузиасты могут нарисовать маску плотности предложений на карте.

```
def latlonNbrhdGr():
    names_neigh = sorted(set(data['neighbourhood_group']))

    colors = []
    for i in range(len(names_neigh)):
        color = "#" + "%06x" % random.randint(0, 0xFFFFFF)
        colors.append(color)

    color_dict = dict()

    for color, name in zip(colors, names_neigh):
        color_dict.update({name: color})

    data_colors = data['neighbourhood_group'].map(color_dict)
    ny_plot = data.plot(kind = 'scatter', x = 'latitude', y = 'longitude', color = data_colors)
    plt.show()


def heatMap():
    m = folium.Map([40.7128, -74.0060], zoom_start = 11)
    data_loc = data[['latitude', 'longitude']].values
    data_loc = data_loc.tolist()
    HeatMap(data[['latitude', 'longitude']].dropna(), radius = 8,
            gradient = {0.2: 'blue', 0.4: 'purple', 0.6: 'orange', 1.0: 'red'}).add_to(m)

    # if you want you can open it manually in your webbrowser
    m.save("heatMap.html")
```

Результат:


−73.7
−73.8
−73.9
−74.0
−74.1
−74.2
longitude
40.5
40.6
40.7
40.8
40.9
latitude

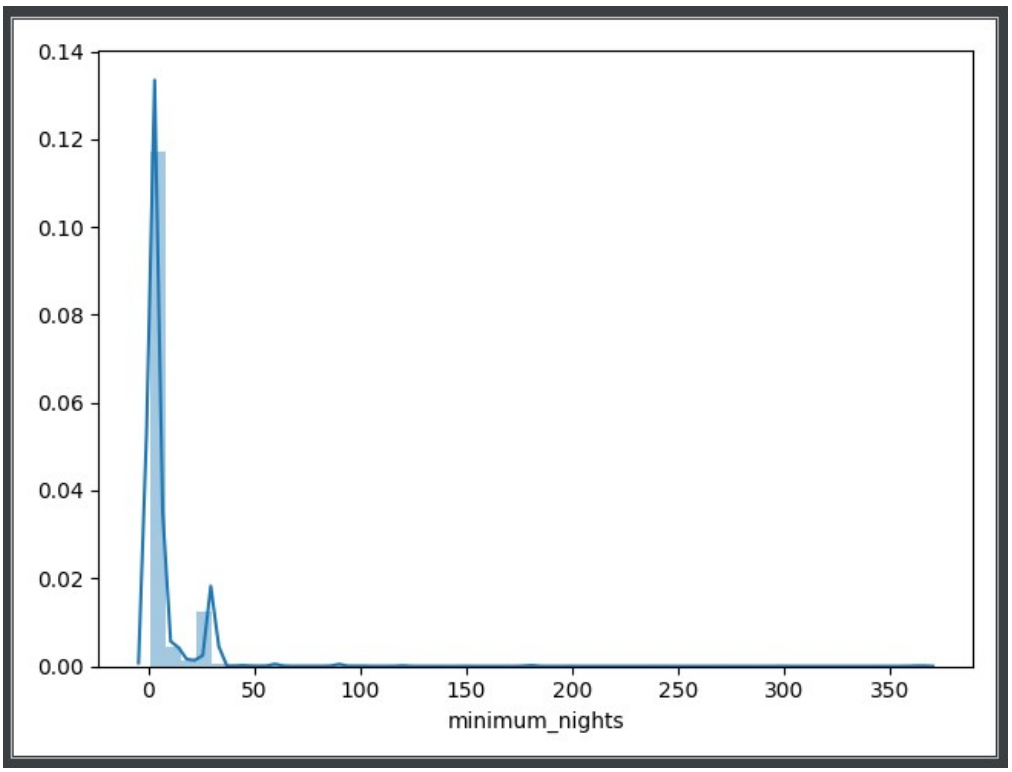

Paterson
Yonkers
Newark
Jersey City
New York
Elizabeth
Staten Island
Hempstead
Long Beach
Gateway National Recreation Area

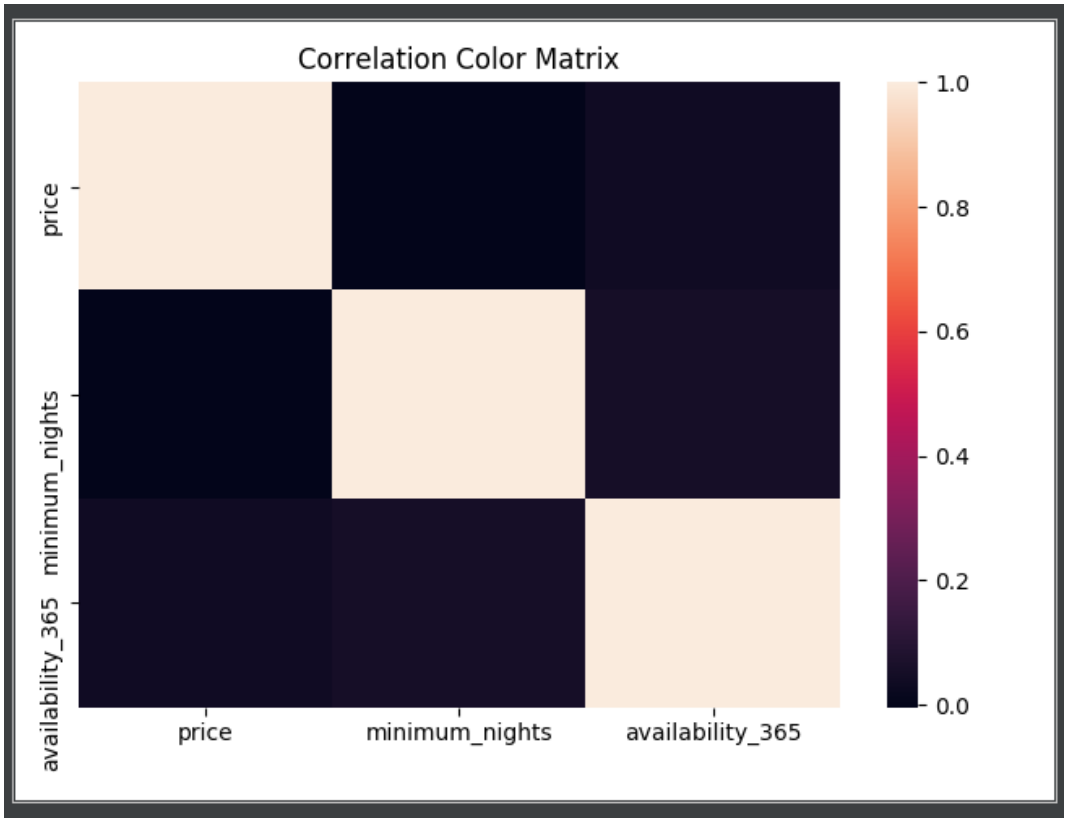
5. Теперь подумайте, могут ли какие-то признаки быть взаимозависимы? Если да, то какие? А после этого постройте матрицу корреляции, но с цветовой индикацией, пожалуйста. Совпало с вашими догадками?

Взаимосвязанными могут быть признаки: `price`, `minimum_nights` и `availability_365`.

```
def correlation():
    print(set(data['neighbourhood_group']))
    print('\n')

    for group in set(data['neighbourhood_group']):
        print(group)
        print(data[data['neighbourhood_group'] == group][['price', 'minimum_nights', 'availability_365']].corr())
        print('\n')
    sns.distplot(data['minimum_nights'])
    plt.show()

    corr = data[data['neighbourhood_group'] == group][['price', 'minimum_nights', 'availability_365']].corr()
    sns.heatmap(corr, xticklabels = corr.columns.values, yticklabels = corr.columns.values)
    plt.title('Correlation Color Matrix')
    plt.show()

    print('\n')
```

```
{'Brooklyn', 'Bronx', 'Queens', 'Manhattan', 'Staten Island'}


Brooklyn
                    price  minimum_nights  availability_365
price             1.000000        0.022414          0.061612
minimum_nights    0.022414        1.000000          0.114106
availability_365  0.061612        0.114106          1.000000


Bronx
                    price  minimum_nights  availability_365
price             1.000000       -0.031339          0.068172
minimum_nights   -0.031339        1.000000          0.069003
availability_365  0.068172        0.069003          1.000000


Queens
                    price  minimum_nights  availability_365
price             1.000000        0.183480          0.039103
minimum_nights    0.183480        1.000000          0.030341
availability_365  0.039103        0.030341          1.000000


Manhattan
                    price  minimum_nights  availability_365
price             1.000000        0.037029          0.118068
minimum_nights    0.037029        1.000000          0.241033
availability_365  0.118068        0.241033          1.000000


Staten Island
                    price  minimum_nights  availability_365
price             1.000000       -0.003898          0.037875
minimum_nights   -0.003898        1.000000          0.053381
availability_365  0.037875        0.053381          1.000000
```



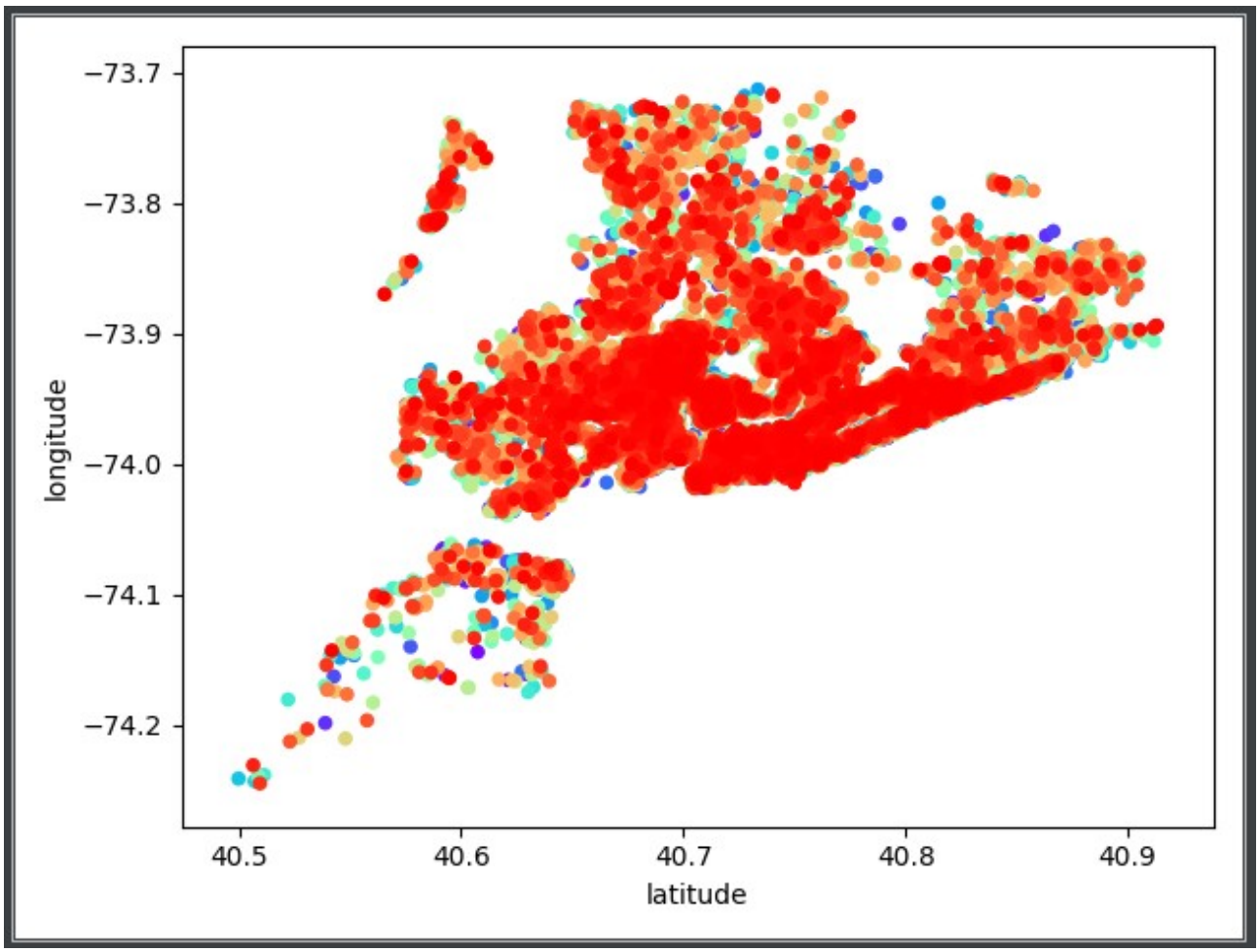


С ожиданиями совпало.

6. Возможно, многие уже задумались о насущном.. Постройте график, где по осям x и y будут широта и долгота, а цветом помечена цена. Те, кто в пункте 3 делал это с seaborn – есть смысл попробовать метод pandas.DataFrame.plot, потому что там цветовая индикация непрерывная, а seaborn разбивает на несколько жалких диапазонов, и при неравномерном распределении ничего не понятно. Подсказка: если какое-то относительно небольшое количество записей сильно выбивается из общей кучи, можно их не рассматривать, нас пока интересуют общие закономерности, а не исключения.

```
def latlonPrice():
    all_prices = data['price'].values

    price_colors = cm.rainbow(np.linspace(0, 1, len(all_prices)))

    price_plot = data.plot(kind = 'scatter', x = 'latitude', y = 'longitude', color = price_colors)
    plt.show()
```

Результат:



7. Составьте рейтинг слов из названий по популярности (частоте появления) и укажите 25 самых популярных с числом их появлений.

```
def wordsCounter():
    name = pd.read_csv('AB_NYC_2019.csv', usecols = [1], squeeze = True)

    dictionary = {}
    dictionary = name.str.lower().str.split(None, expand = True).stack().value_counts()

    print('25 наиболее часто встречаемых слов в колонке name:\n')
    print(list(islice(dictionary.items(), 25)))
    print('\n')
```

```
25 наиболее часто встречаемых слов в колонке name:

[('in', 16733), ('room', 9389), ('bedroom', 7231), ('private', 6985), ('apartment', 6113), ('cozy', 4634), ('the', 3869), ('to', 3827
), ('studio', 3772), ('brooklyn', 3629), ('apt', 3571), ('spacious', 3387), ('1', 3357), ('with', 3092), ('2', 3079), ('of', 2993), (
'east', 2967), ('and', 2869), ('manhattan', 2855), ('&', 2820), ('park', 2632), ('sunny', 2536), ('beautiful', 2320), ('near', 2295),
 ('williamsburg', 2293)]
```

Часть 2 – классификация:

```
def classificationsWorker(price):
    dataset = pd.read_csv('AB_NYC_2019.csv', parse_dates = ['last_review'], index_col = ['id'])

    objects = [c for c in dataset.columns if dataset[c].dtype.name == 'object']

    dataset['reviews_per_month'] = dataset['reviews_per_month'].fillna(0)

    if price:
        dataset['price'] = pd.qcut(dataset['price'], 9, labels = list(map(str, np.arange(9))))
    else:
        del dataset['last_review']

    dataset = dataset.dropna(axis = 0, subset = ['name', 'host_name'])

    dataset_describe = dataset.describe(include = [object])

    for c in objects:
        dataset[c] = dataset[c].fillna(dataset_describe[c]['top'])

    dataset.count(axis = 0)

    binary = [c for c in objects if dataset_describe[c]['unique'] == 2]
    nonBinary = [c for c in objects if dataset_describe[c]['unique'] > 2]
    nonBinary3 = [c for c in objects if dataset_describe[c]['unique'] <= 3]

    dataset_nonBinary = pd.get_dummies(dataset[nonBinary3])

    print("\n")
```

Для начала подготовим данные для классификации:
Как и прежде, удалим не несущие информации для анализа столбцы `name`, `host_name`, `last_review`.
В стобце `reviews_per_month` заполним пустые ячейки нулями.
Пустые ячейки в столбцах, которые отнесены к типу obj, заполним наиоболее часто встречающимися значениями в ячейках по соответствующим столбцам.

```
if price:
    dataset_pr_class = dataset[['price']]
    dataset_geo = dataset[['latitude', 'longitude']]

    dataset_numerical = dataset[['minimum_nights', 'number_of_reviews', 'reviews_per_month', 'calculated_host_listings_count', 'availability_365']]
    dataset_numerical = (dataset_numerical - dataset_numerical.mean()) / dataset_numerical.std()
    dataset_geo = (dataset_geo - dataset_geo.mean()) / dataset_geo.std()

    temp = pd.concat((dataset_numerical, dataset[binary], dataset_nonBinary, dataset_pr_class, dataset_geo), axis = 1)

    dataset = pd.DataFrame(temp)

    Xn = dataset.drop(('price'), axis = 1)
    Yn = dataset['price']
    Yn = Yn.astype('int')

    x_train, x_test, y_train, y_test = train_test_split(Xn, Yn, test_size = 0.2, random_state = 11)
```

```
else:
    dataset_numerical = dataset[
        ['latitude', 'longitude', 'price', 'minimum_nights', 'number_of_reviews', 'reviews_per_month',
         'calculated_host_listings_count', 'availability_365']]
    dataset_numerical = (dataset_numerical - dataset_numerical.mean()) / dataset_numerical.std()

    temp = pd.concat((dataset_numerical, dataset[binary], dataset_nonBinary, dataset[['neighbourhood_group']]),
                     axis=1)

    dataset = pd.DataFrame(temp)

    Xn = dataset.drop(('neighbourhood_group'), axis=1)
    Yn = dataset['neighbourhood_group']

    x_train, x_test, y_train, y_test = train_test_split(Xn, Yn, test_size=0.2, random_state=11)
```

В данном случае предсказывать будем цену и район.
В случае цены делим ее на несколько диапазонов, как указано в разъяснении к заданию.

x – вход, y – (верный) выход.

```
print("Предсказание по цене")

print(kNN(x_train, y_train, x_test, y_test)[0])
print(kNN(x_train, y_train, x_test, y_test)[1])

print("\n")

decisionTree(x_train, y_train, x_test, y_test, price)

print("\n")

print("Наивный Байес по цене:")
print(naiveBayes(x_train, y_train, x_test, y_test))

print("\n")

print("SVC по цене:")
print(supportVectorMachine(x_train, y_train, x_test, y_test))
```

```
print("Предсказание по району")

print(kNN(x_train, y_train, x_test, y_test)[0])
print(kNN(x_train, y_train, x_test, y_test)[1])

print("\n")

decisionTree(x_train, y_train, x_test, y_test, price)

print("\n")

print("Наивный Байес по району:")
print(naiveBayes(x_train, y_train, x_test, y_test))

print("\n")

print("SVC по району:")
print(supportVectorMachine(x_train, y_train, x_test, y_test))
```

kNN:

```
def kNN(x_train, y_train, x_test, y_test):
    knn = neighbors.KNeighborsClassifier(n_neighbors = 5)
    knn.fit(x_train, y_train)

    n_neighbors_array = [1, 3, 5, 7, 10, 15]

    # https://iq.opengenus.org/euclidean-vs-manhattan-vs-chebyshev-distance/
    metrics = ['euclidean', 'manhattan', 'chebyshev']

    err = {'euclidean': [], 'manhattan': [], 'chebyshev': []}
    t = {'euclidean': [], 'manhattan': [], 'chebyshev': []}

    for i in n_neighbors_array:
        for met in metrics:
            t0 = time()

            knn = neighbors.KNeighborsClassifier(n_neighbors = i, metric = met, weights = 'distance')
            knn.fit(x_train, y_train)

            y_test_predict = knn.predict(x_test)

            t1 = time()

            err[met].append(round(np.mean(y_test != y_test_predict) * 100))

            t[met].append(round(t1 - t0, 2))

    knn_time = pd.DataFrame(t, index = n_neighbors_array)
    knn_err = pd.DataFrame(err, index = n_neighbors_array)

    return knn_err, knn_time
```

Для метода k ближайших соседей по метрикам chebyshev, eucledean и manhattan. Евклидово расстояние, манхэттенское расстояние и расстояние Чебышева - это все метрики расстояния, которые вычисляют число на основе двух точек данных.

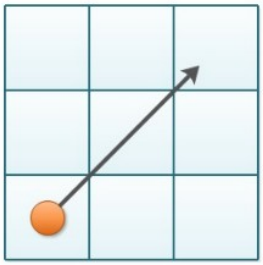
Предсказание по району

	chebyshev	euclidean	manhattan
1	2.0	2.0	2.0
3	2.0	2.0	2.0
5	2.0	2.0	2.0
7	2.0	2.0	2.0
10	2.0	2.0	2.0
15	3.0	2.0	2.0
	chebyshev	euclidean	manhattan
1	0.24	0.47	0.73
3	0.35	0.70	1.03
5	0.40	0.78	1.41
7	0.68	0.97	1.47
10	0.47	1.25	1.57
15	0.54	1.12	1.61

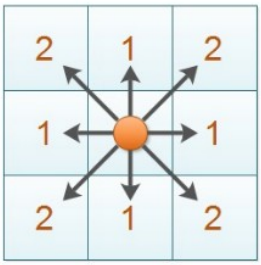
Предсказание по цене

	chebyshev	euclidean	manhattan
1	73.0	72.0	72.0
3	72.0	71.0	70.0
5	70.0	70.0	69.0
7	70.0	69.0	69.0
10	69.0	69.0	68.0
15	69.0	68.0	67.0
	chebyshev	euclidean	manhattan
1	0.17	0.33	0.47
3	0.24	0.43	0.64
5	0.27	0.50	0.75
7	0.36	0.56	0.93
10	0.37	0.74	0.95
15	0.47	0.83	1.23

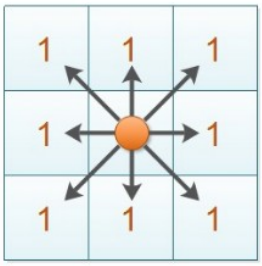
Euclidean Distance



Manhattan Distance



Chebyshev Distance



$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad |x_1 - x_2| + |y_1 - y_2| \quad \max(|x_1 - x_2|, |y_1 - y_2|)$$

DecisionTree:

```
def decisionTree(x_train, y_train, x_test, y_test, integer):
    tree_array = [6, 10, 14, 18, 22, 26]

    crit = 'entropy'

    for i in tree_array:
        t1 = time()
        clf_tree = DecisionTreeClassifier(criterion = crit, max_depth = i, random_state = 20, presort = True)

        clf_tree.fit(X = x_train, y = y_train)

        if integer:
            err_train = round(np.mean(y_train != clf_tree.predict(x_train).astype('int')) * 100, 2)
            err_test = round(np.mean(y_test != clf_tree.predict(x_test).astype('int')) * 100, 2)
        else:
            err_train = round(np.mean(y_train != clf_tree.predict(x_train)) * 100, 2)
            err_test = round(np.mean(y_test != clf_tree.predict(x_test)) * 100, 2)

        t = t1 - time()

        print("Глубина дерева: {}, ошибка на обучающей: {}, ошибка на тестовой: {}, время {}".format(clf_tree.get_depth(), err_train, err_test, t))
```

Для метода, на основе решающих деревьев:

```
Глубина дерева: 6, ошибка на обучающей: 0.35, ошибка на тестовой: 0.51, время -0.14497661590576172
Глубина дерева: 10, ошибка на обучающей: 0.0, ошибка на тестовой: 0.15, время -0.15219807624816895
Глубина дерева: 11, ошибка на обучающей: 0.0, ошибка на тестовой: 0.15, время -0.1527845859527588
Глубина дерева: 11, ошибка на обучающей: 0.0, ошибка на тестовой: 0.15, время -0.1522200107574463
Глубина дерева: 11, ошибка на обучающей: 0.0, ошибка на тестовой: 0.15, время -0.15212726593017578
Глубина дерева: 11, ошибка на обучающей: 0.0, ошибка на тестовой: 0.15, время -0.15246915817260742
```

```
Глубина дерева: 6, ошибка на обучающей: 68.6, ошибка на тестовой: 69.4, время -0.23167943954467773
Глубина дерева: 10, ошибка на обучающей: 61.91, ошибка на тестовой: 68.24, время -0.5850245952606201
Глубина дерева: 14, ошибка на обучающей: 44.5, ошибка на тестовой: 69.55, время -1.7023890018463135
Глубина дерева: 18, ошибка на обучающей: 20.32, ошибка на тестовой: 71.46, время -3.23724627494812
Глубина дерева: 22, ошибка на обучающей: 5.34, ошибка на тестовой: 72.01, время -4.228155851364136
Глубина дерева: 26, ошибка на обучающей: 0.68, ошибка на тестовой: 72.39, время -4.91480565071106
```

```
def naiveBayes(x_train, y_train, x_test, y_test):
    model = GaussianNB()

    model.fit(x_train, y_train)

    return testTrain(x_train, y_train, x_test, y_test, model)


def supportVectorMachine(x_train, y_train, x_test, y_test):
    svc = SVC(gamma = 'scale')

    svc.fit(x_train, y_train)

    return testTrain(x_train, y_train, x_test, y_test, svc)

def testTrain(x_train, y_train, x_test, y_test, model):
    err_train = round(np.mean(y_train != model.predict(x_train)) * 100, 2)
    err_test = round(np.mean(y_test != model.predict(x_test)) * 100, 2)

    return err_train, err_test
```

Для Наивного Байесовского классификатора и метода на основе Машины Опорных Векторов (вывод ошибок классификации):

```
Наивный Байес по району:
(25.059999999999999, 23.760000000000002)


SVC по району:
(1.05, 1.1699999999999999)
```

```
Наивный Байес по цене:
(76.150000000000006, 75.25)


SVC по цене:
(66.780000000000001, 68.079999999999998)
```

Для каждого алгоритма составить список/таблицу настроечных параметров, описать их смысл, в каких случаях что используется и как это влияет (пусть предположительно) на результат.

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Параметр	Описание	На что влияет
n_neighbors	Number of neighbors to use by default for kneighbors queries.	Time && Quality
weights	weight function used in prediction	Quality
algorithm	Algorithm used to compute the nearest neighbors	Time && Quality
leaf_size	Leaf size passed to BallTree or KDTree	Time
p	Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.	Time
metric	the distance metric to use for the tree	Time && Quality
metric_params	Additional keyword arguments for the metric function	Quality
n_jobs	The number of parallel jobs to run for neighbors search	Time

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Параметр	Описание	На что влияет
splitter	The strategy used to choose the split at each node	Time && Quality
max_depth	The maximum depth of the tree	Time && Quality
min_samples_split	The minimum number of samples required to split an internal node	Time && Quality
min_samples_leaf	The minimum number of samples required to be at a leaf node	Time && Quality
min_weight_fraction_leaf	The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node	Time && Quality
max_features	The number of features to consider when looking for the best split	Time && Quality
random_state	If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random	-
max_leaf_nodes	Grow a tree with max_leaf_nodes in best-first fashion	Time && Quality
min_impurity_decrease	A node will be split if this split induces a decrease of the impurity greater than or equal to this value	Time && Quality
class_weight	Weights associated with classes in the form	Time && Quality
presort	Whether to presort the data to speed up the finding of best splits in fitting	Time && Quality

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

Параметр	Описание	На что влияет
priors	Prior probabilities of the classes. If specified the priors are not adjusted according to the data	Quality
var_smoothing	Portion of the largest variance of all features that is added to variances for calculation stability	Time && Quality

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Параметр	Описание	На что влияет
C	Penalty parameter C of the error term	Time && Quality
kernel	Specifies the kernel type to be used in the algorithm	Time && Quality
degree	Degree of the polynomial kernel function (‘poly’). Ignored by all other kernels	Time && Quality
gamma	Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’	Time && Quality
coef0	Independent term in kernel function. It is only significant in ‘poly’ and ‘sigmoid’	Time && Quality
shrinking	Whether to use the shrinking heuristic	Describe
probability	Whether to enable probability estimates	Describe
tol	Tolerance for stopping criterion	Time && Quality
cache_size	Specify the size of the kernel cache (in MB)	Time
class_weight	Set the parameter C of class i to class_weight[i]*C for SVC	Time && Quality
verbose	Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.	Describe
max_iter	Hard limit on iterations within solver, or -1 for no limit	Time && Quality
decision_function_shape	Whether to return a one-vs-rest (‘ovr’) decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one (‘ovo’) decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2).	Time && Quality
random_state	The seed of the pseudo random number generator	-

Выводы:

По приведенным выше результатам работы программы можно сделать вывод о том, что для классификации по району, в отличие от классификации по цене, метод на основе решающих деревьев имеет большую точность (ошибка ~1). Для цены ошибка на тестовой выборке составила в лучшем случае ~69. Видно, что точность обратно пропорциональна количеству классов.
Остальные три методы показали примерно одинаковую точность. По времени выиграл Наивный Байесовский классификатор.
Для метода на основе Машины Опорных Векторов наилучшие показатели получились при использовании параметра `scale`.
На основе приведенных примеров для метода на основе решающих деревьев был взять критерий `entropy`.