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 № 4

Кластеризация

по дисциплине «Математические методы интеллектуальных технологий»

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

Проверил
<*подпись*> А.В.Бахшиев

«23» декабря 2019 г.

Санкт-Петербург

2019

Скрипт приведен в репозитории
https://github.com/makaryb/ml_1sem_5kurs/blob/master/lab4/src/clustering.py

Часть 1 – работа с данными:
Опишите используемую базу данных (или несколько, если хотите). Постройте диаграммы примерно как в лабе по классификации. Опишите выводы – замеченные закономерности и преобразования с базой, которые вы сделаете для дальнейшей работы.

Как и в предыдущей лаб. работе, почистим наш датасет.
Удалим столбец `last_review`, заполним пустые ячейки в нужных столбцах некоторыми значениями.

```
def transformationAndCleaning():
    # проверка на пустые ячейки
    data.isnull().sum().sort_values(ascending = False) # по убыванию

    print(data.tail())
    # для визуального анализа данных
    print(data.dropna())

    REPLACE_NAME_VALUE = "$"
    REPLACE_HOST_NAME_VALUE = "#"

    # заполним пустые (NaN) ячейки
    data['name'].fillna(REPLACE_NAME_VALUE, inplace = True)
    data['host_name'].fillna(REPLACE_HOST_NAME_VALUE, inplace = True)

    # убираем что не нужно
    data.drop(['last_review'], axis = 1, inplace = True)
    # заполняем невалидные ячейки
    data['reviews_per_month'] = data['reviews_per_month'].fillna(0)

    print("\n")
    print("Length of Airbnb DataFrame that match with Name ="
          " \"{}\": {}".format(REPLACE_NAME_VALUE, len(data[data.name == REPLACE_NAME_VALUE])))
    print("Length of Airbnb DataFrame that match with Host_Name ="
          " \"{}\": {}".format(REPLACE_HOST_NAME_VALUE, len(data[data.host_name == REPLACE_HOST_NAME_VALUE])))
    print("\n")

    print(data.head())
    print("\n")
```

Результат:

Другого
```
Length of Airbnb DataFrame that match with Name = "$": 16
Length of Airbnb DataFrame that match with Host_Name = "#": 21
```
результата не ожидали. В лаб. работе 2 было показано какие ячейки невалидны в данном датасете.

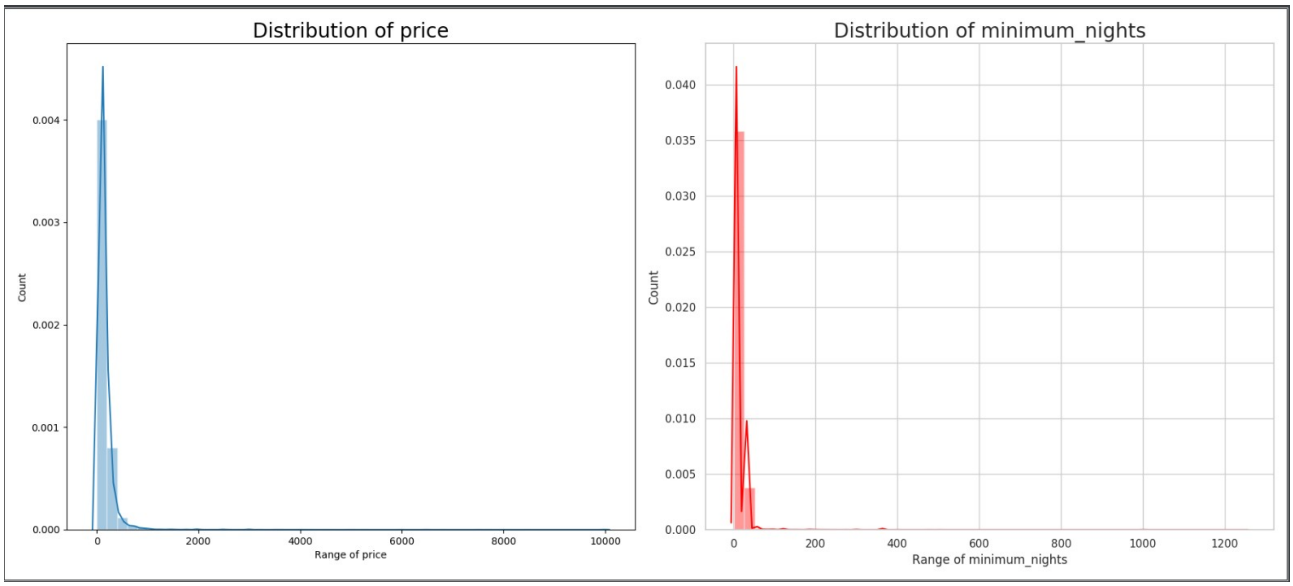
Посмотрим на хорошо знакомый датасет вновь:

```
plt.rcParams['figure.figsize'] = (18, 8)

plt.subplot(1, 2, 1)
sns.set(style='whitegrid')
sns.distplot(data['price'])
plt.title('Distribution of price', fontsize=20)
plt.xlabel('Range of price')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
sns.set(style='whitegrid')
sns.distplot(data['minimum_nights'], color='red')
plt.title('Distribution of minimum_nights', fontsize=20)
plt.xlabel('Range of minimum_nights')
plt.ylabel('Count')
plt.show()
```

Не будем чистить датасет от непонятных нам ранее значений: оставим интревал до 10000 $ цены, а также минимальный съем до 1200 ночей, как и было в оригинальых данных.
Распределение цены жилья и минимальных ночей для аренды выглядит следующим образом:

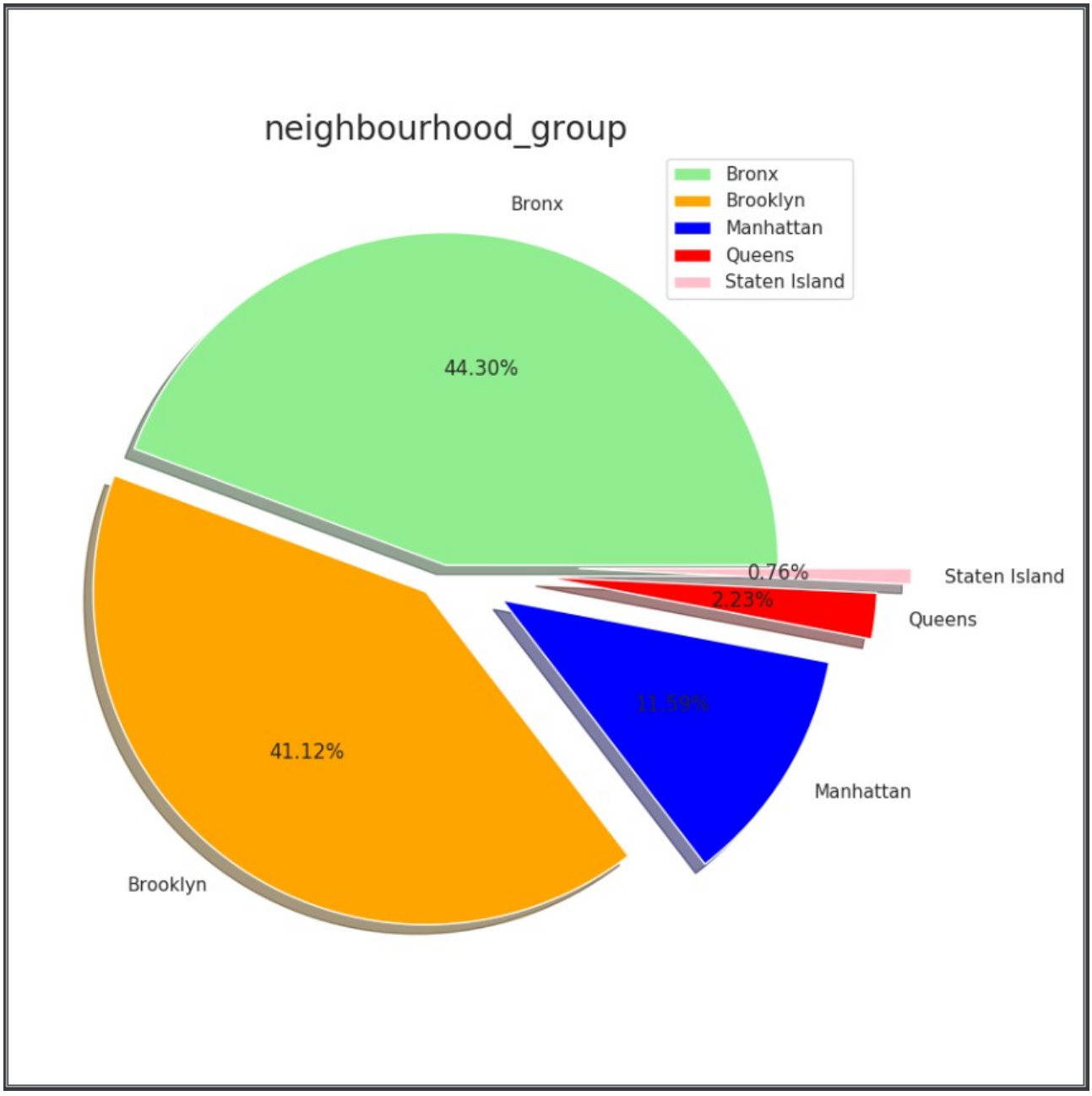


Понятно, что цена в основном ниже 1000$ за ночь и минимальный съем жилья меньше 50 ночей, однако есть и вполне реалистичные выбросы и по цене, и по минимальным ночам.

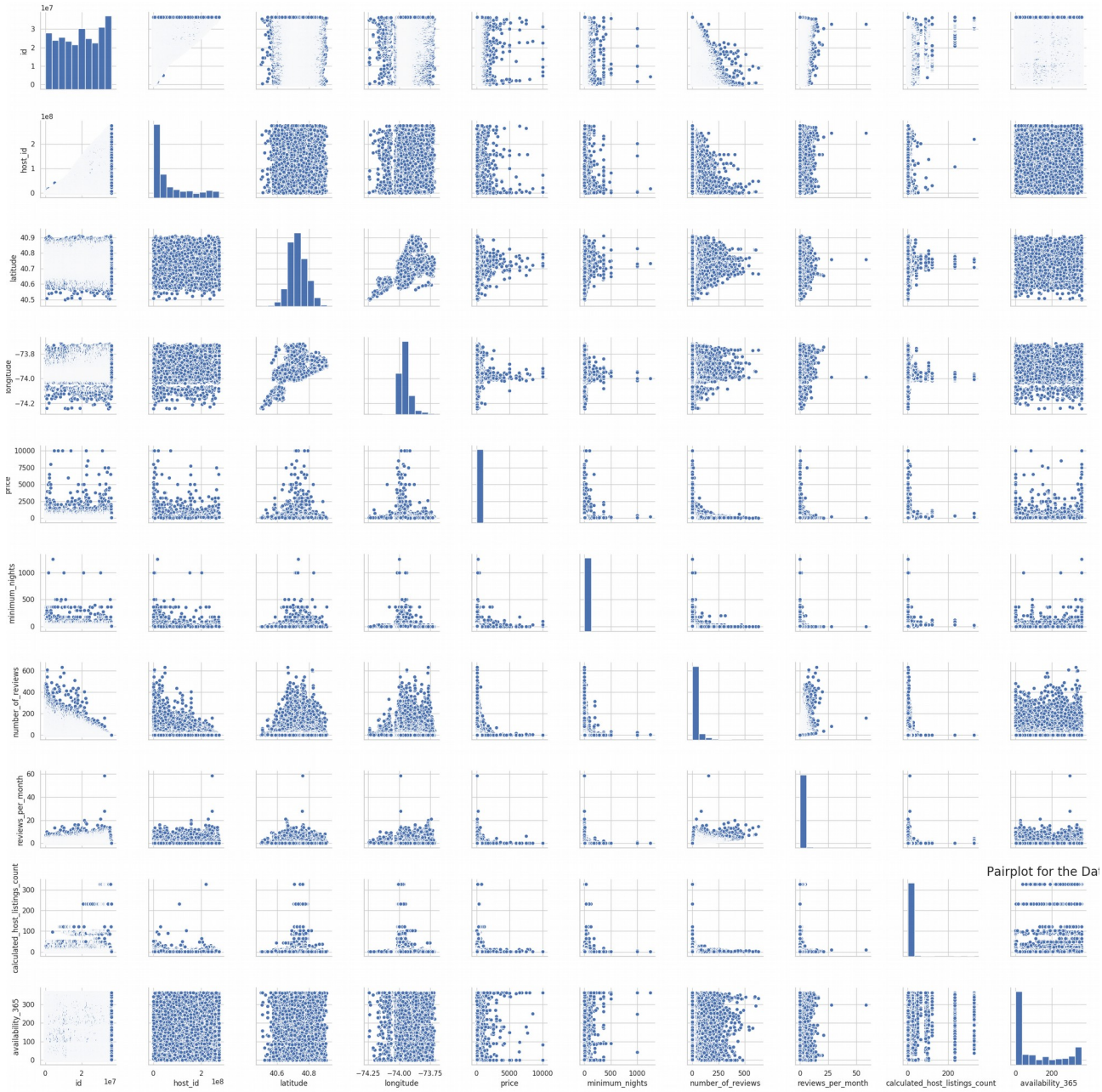
По района все понятно:

```
labels = ['Bronx', 'Brooklyn', 'Manhattan', 'Queens', 'Staten Island']
size = data['neighbourhood_group'].value_counts()
colors = ['lightgreen', 'orange', 'blue', 'red', 'pink']
explode = [0, 0.1, 0.2, 0.3, 0.4]

plt.rcParams['figure.figsize'] = (9, 9)
plt.pie(size, colors=colors, explode=explode, labels=labels, shadow=True, autopct='%.2f%%')
plt.title('neighbourhood_group', fontsize=20)
plt.axis('off')
plt.legend()
plt.show()
```

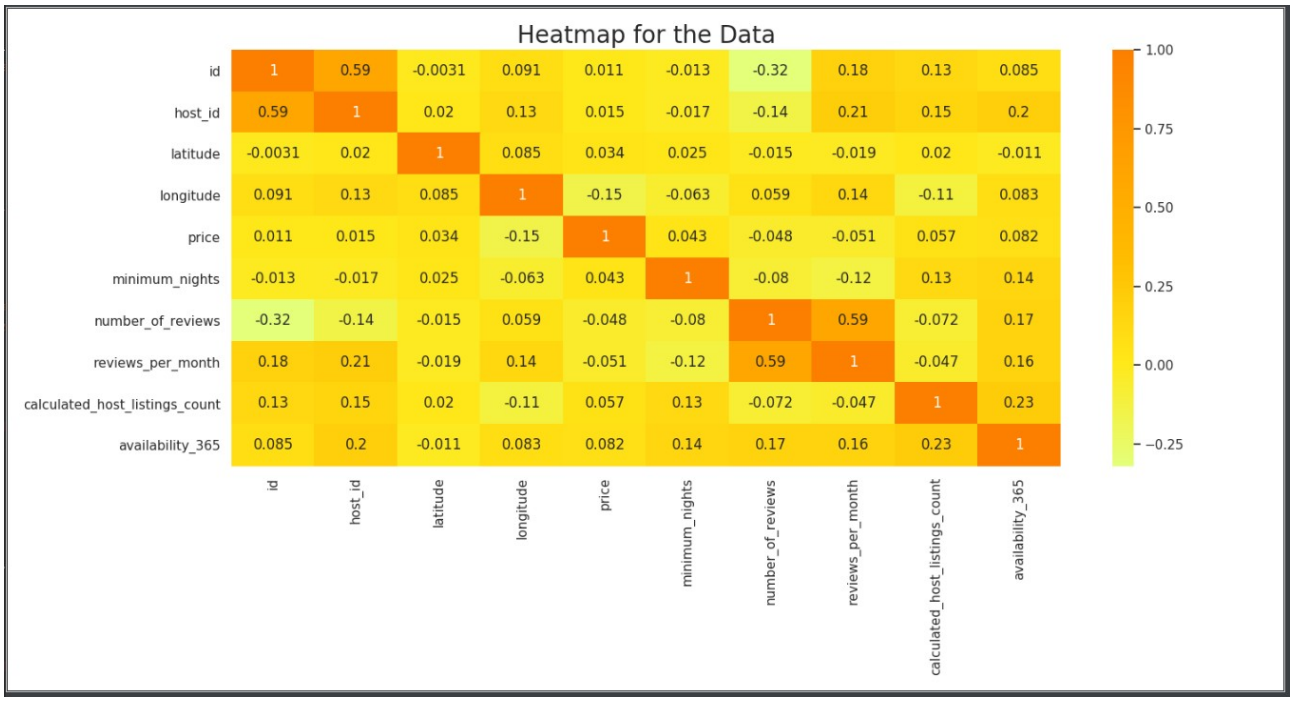


Рассмотрим матрицу диаграмм для всех используемых признаков, относительно всех других признаков. Здесь также все ожидаемо. Для более подробного ознакомления, воспользуйтесь картинкой в репозитории:



Невозможно обойтись без напоминания о матрице корреляции:

```
plt.rcParams['figure.figsize'] = (15, 8)
sns.heatmap(data.corr(), cmap='Wistia', annot=True)
plt.title('Heatmap for the Data', fontsize=20)
plt.show()
```



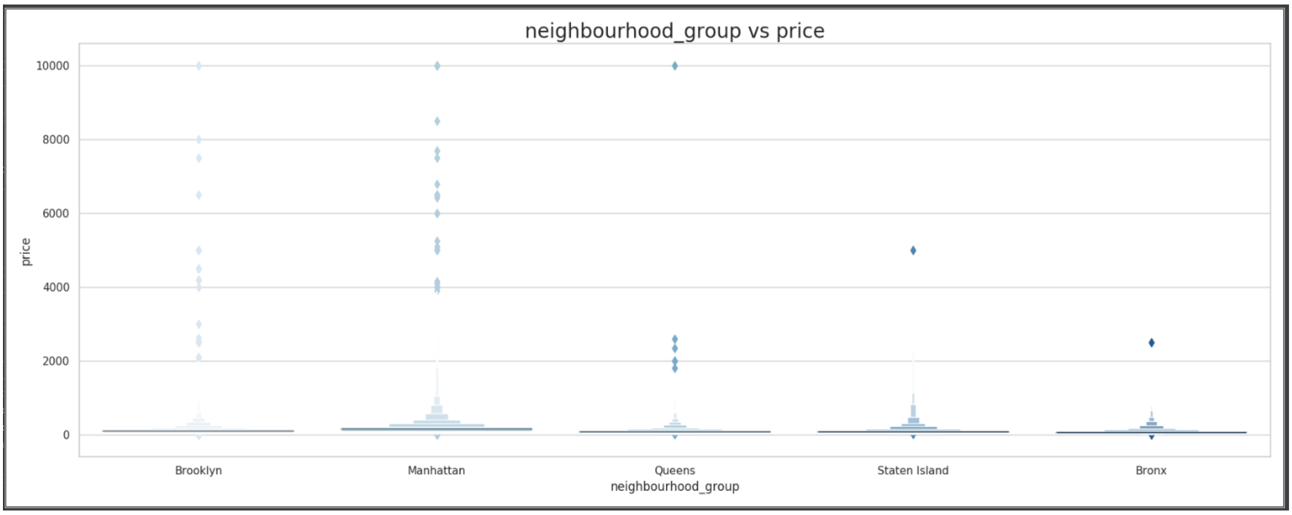
Более подробно остановимся на визуализации распределения цены, так как отображение ее распределения по районам нагляднее всего:

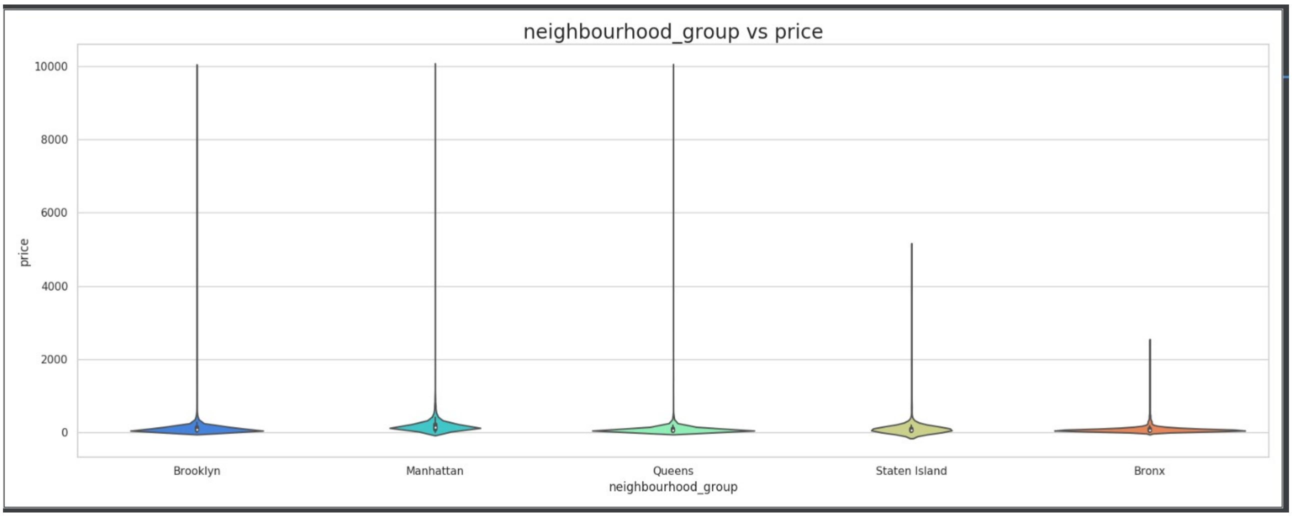
```
plt.rcParams['figure.figsize'] = (18, 7)
sns.boxenplot(data['neighbourhood_group'], data['price'], palette='Blues')
plt.title('neighbourhood_group vs price', fontsize=20)
plt.show()
# plt.savefig("5.png")

plt.rcParams['figure.figsize'] = (18, 7)
sns.violinplot(data['neighbourhood_group'], data['price'], palette='rainbow')
plt.title('neighbourhood_group vs price', fontsize=20)
plt.show()
# plt.savefig("6.png")

plt.rcParams['figure.figsize'] = (18, 7)
sns.stripplot(data['neighbourhood_group'], data['price'], palette='Purples', size=10)
plt.title('neighbourhood_group vs price', fontsize=20)
plt.show()
```

В результате:





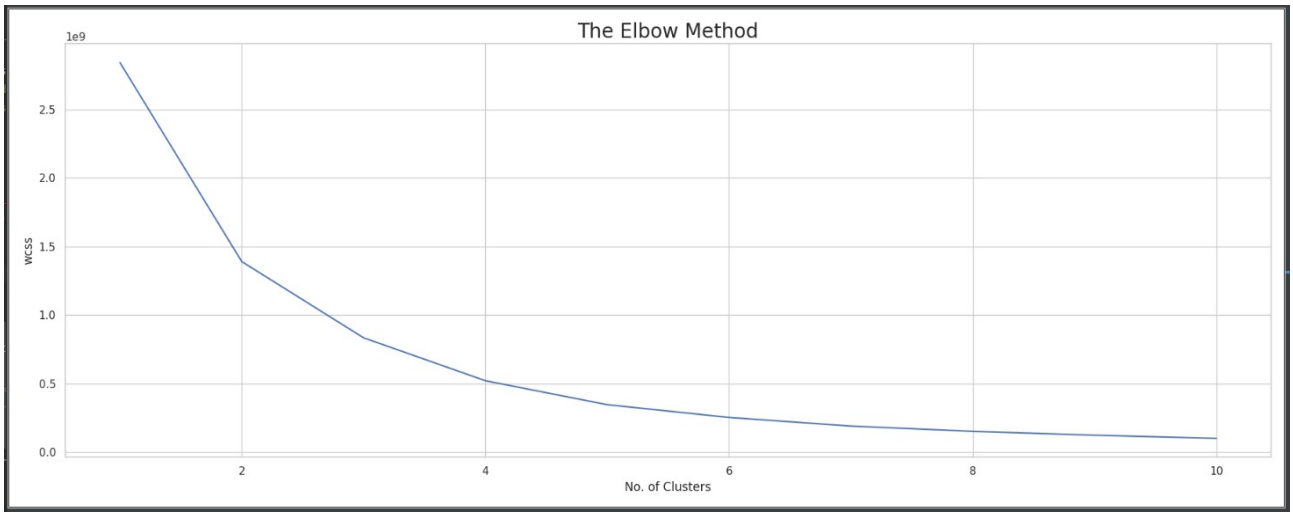
На последнем отображении можно увидеть, что для Манхеттена и Бруклина ценник, больший 2000 $, - не единичный случай.

Кластерный анализ:

Метод Elbow для нахождения оптимального количества кластеров по нашим данным:

```
wcss = []
for i in range(1, 11):
    km = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    km.fit(x)
    wcss.append(km.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method', fontsize=20)
plt.xlabel('No. of Clusters')
plt.ylabel('wcss')
plt.show()
```



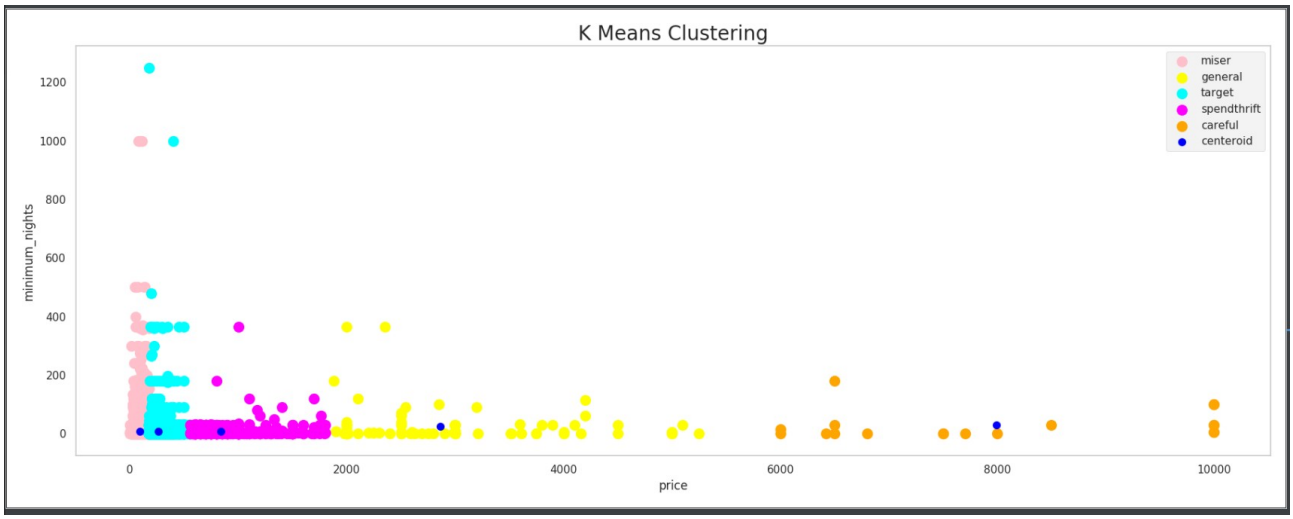
Далее проведем саму K-means кластеризацию, определив цену и минимальное количество ночей, как условие.

```
km = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_means = km.fit_predict(x)

plt.scatter(x[y_means == 0, 0], x[y_means == 0, 1], s=100, c='pink', label='miser')
plt.scatter(x[y_means == 1, 0], x[y_means == 1, 1], s=100, c='yellow', label='general')
plt.scatter(x[y_means == 2, 0], x[y_means == 2, 1], s=100, c='cyan', label='target')
plt.scatter(x[y_means == 3, 0], x[y_means == 3, 1], s=100, c='magenta', label='spendthrift')
plt.scatter(x[y_means == 4, 0], x[y_means == 4, 1], s=100, c='orange', label='careful')
plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], s=50, c='blue', label='centeroid')

plt.style.use('fivethirtyeight')
plt.title('K Means Clustering', fontsize=20)
plt.xlabel('price')
plt.ylabel('minimum_nights')
plt.legend()
plt.grid()
plt.show()
```

Результат разбиения следующий:



Кластерный анализ дает нам четкое представление о различных уровнях предполагаемой состоятельности людей, которые будут бронировать жилье. Здесь отображение разбито на пять сегментов. На картинке можно видеть слева направо:

Можно видеть, что люди, которые арендуют жилье до 1000 $ за ночь могут арендовать его на более длительный срок. Но настолько длительных сроков же для тех, кто арендует за огромную цену, не предусмотрено.
Такие различия вызваны рынком.

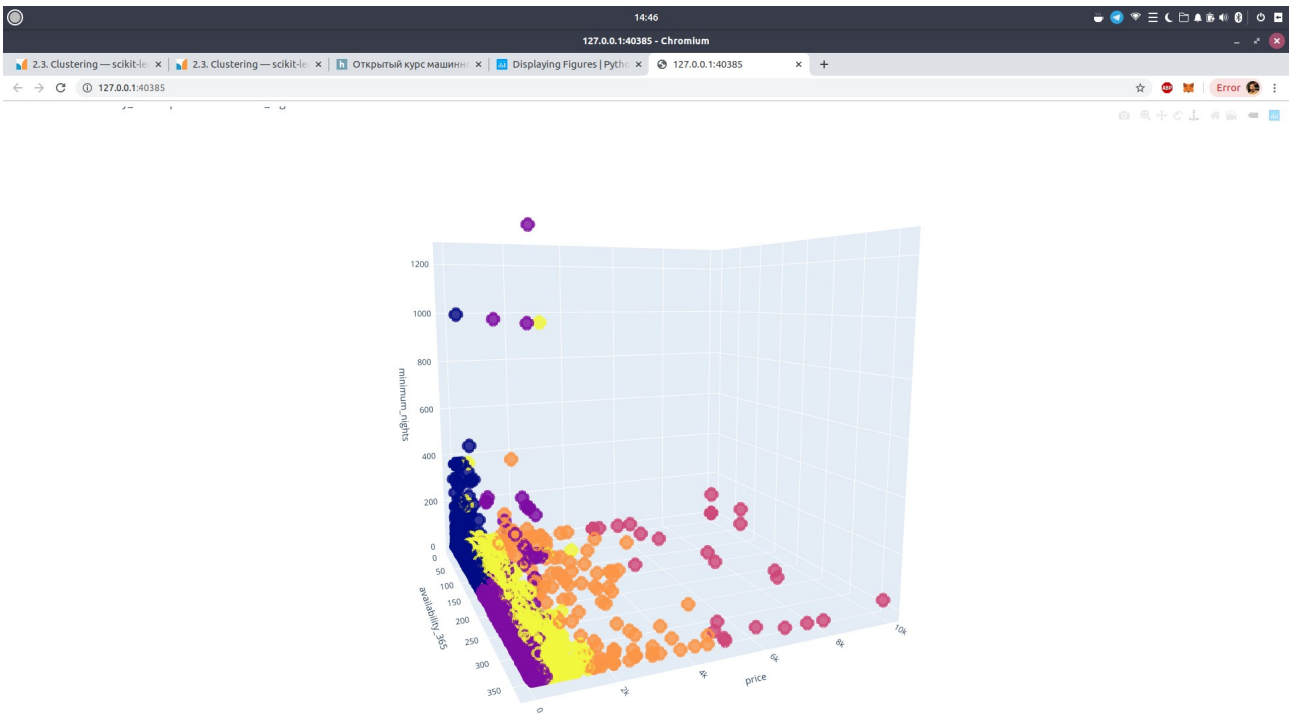
Для большего интереса, проведем кластеризацию с трехмерным результатом вывода:

```
x = data[['availability_365', 'price', 'minimum_nights']].values
km = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10, random_state=0)
km.fit(x)
labels = km.labels_
centroids = km.cluster_centers_

data['labels'] = labels
trace1 = go.Scatter3d(
    x=data['availability_365'],
    y=data['price'],
    z=data['minimum_nights'],
    mode='markers',
    marker=dict(
        color=data['labels'],
        size=10,
        line=dict(
            color=data['labels'],
            width=12
        ),
        opacity=0.8
    )
)
df = [trace1]

layout = go.Layout(
    title='availability_365 vs price vs minimum_nights',
    margin=dict(
        l=0,
        r=0,
        b=0,
        t=0
    ),
    scene=dict(
        xaxis=dict(title='availability_365'),
        yaxis=dict(title='price'),
        zaxis=dict(title='minimum_nights')
    )
)
fig = go.Figure(data=df, layout=layout)
fig.show()
```

Здесь мы будем опираться на три признака объявления: availability_365, minimum_nights, price.



В результате такого анализа можно увидеть, что доступность аренды по представленнмы объявлениям практически никак не связана с ценой или минимальным количеством ночей. Все распределение по доступности примерно одинаково (относительно количества объявлений). Такое положение вещей также продиктовано рынком. Однако видно, что есть намек: для более высокого ценника доступность аренды 365 дней в году выше. Однако судить точно на таком небольшом количестве данных об объявлениях по цене больше 5000 $ за ночь – не получится.

Проведем сравнение алгоритмов кластеризации с нормированными данными:

```
df_knn = data1[['latitude',
                'longitude',
                'minimum_nights',
                'number_of_reviews',
                'reviews_per_month',
                'calculated_host_listings_count',
                'availability_365',
                'price']]
df_knn.apply(pd.to_numeric)

df_knn = shuffle(df_knn)
```

```
df_norm = (df_knn[['latitude',
                   'longitude',
                   'minimum_nights',
                   'number_of_reviews',
                   'reviews_per_month',
                   'calculated_host_listings_count',
                   'availability_365',
                   'price']] -
           df_knn[['latitude',
                   'longitude',
                   'minimum_nights',
                   'number_of_reviews',
                   'reviews_per_month',
                   'calculated_host_listings_count',
                   'availability_365',
                   'price']].min()) / \
          (df_knn[['latitude',
                   'longitude',
                   'minimum_nights',
                   'number_of_reviews',
                   'reviews_per_month',
                   'calculated_host_listings_count',
                   'availability_365',
                   'price']].max() -
           df_knn[['latitude',
                   'longitude',
                   'minimum_nights',
                   'number_of_reviews',
                   'reviews_per_month',
                   'calculated_host_listings_count',
                   'availability_365',
                   'price']].min())
```

```
df_norm = df_norm[(pd.notnull(data1['latitude'])) &
                  (pd.notnull(data1['longitude'])) &
                  (pd.notnull(data1['minimum_nights'])) &
                  (pd.notnull(data1['number_of_reviews'])) &
                  (pd.notnull(data1['reviews_per_month'])) &
                  (pd.notnull(data1['calculated_host_listings_count'])) &
                  (pd.notnull(data1['availability_365'])) &
                  (pd.notnull(data1['price']))]

df_norm = df_norm.round(6)
df_norm = df_norm.dropna()
df_norm.apply(pd.to_numeric)
```

Сравнение подходов проведем по метрике Silhouette Score:

```
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.cluster import Birch
from sklearn.cluster import AgglomerativeClustering

bestSil = -1
for k in range(2, 6):
    clus = [KMeans(n_clusters=k, n_jobs=-1), Birch(n_clusters=k), AgglomerativeClustering(n_clusters=k)]
    for cl in clus:
        res = cl.fit(df_norm)
        sil = metrics.silhouette_score(df_norm, res.labels_)
        print(str(cl)[:10] + ' with k=' + str(k) + ": " + str(round(sil, 4)))
        if sil > bestSil:
            bestSil = sil
            bestCl = cl
            bestK = k
print('***********************************************')
print('Best algorithm is... ' + str(bestCl)[:8] + '     with k=' + str(bestK))
print('**********************')
print('With Silhouette Score ' + str(bestSil))
```

Это лучший показатель для определения количества кластеров, которые будут сформированы на основе данных. Он рассчитывается для каждого экземпляра следующим образом:

$$Silhouette\ Coefficient = (x-y)/\ max(x,y)$$

где, y - среднее расстояние внутри кластера: среднее расстояние до других экземпляров в том же кластере; x - среднее расстояние до ближайшего кластера, то есть среднее расстояние до экземпляров следующего ближайшего кластера.

Коэффициент варьируется от -1 до 1. Значение, близкое к 1, означает, что экземпляр находится вблизи своего кластера. Принимая во внимание, что значение, близкое к -1, означает, что это значение назначено неправильному кластеру.

Результаты для наших данных выглядят следующим образом:

```
KMeans(alg with k=2: 0.3936
Birch(bran with k=2: 0.3916
Agglomerat with k=2: 0.3916
KMeans(alg with k=3: 0.392
Birch(bran with k=3: 0.3743
Agglomerat with k=3: 0.3743
KMeans(alg with k=4: 0.4021
Birch(bran with k=4: 0.3885
Agglomerat with k=4: 0.3885
KMeans(alg with k=5: 0.3796
Birch(bran with k=5: 0.3683
Agglomerat with k=5: 0.3683
**********************************************
Best algorithm is... KMeans(a     with k=4
**********************
With Silhouette Score 0.402128157913392
```

Далее приведен обзор подходов.

class sklearn.cluster.KMeans(n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=None, algorithm='auto')

Кластеризация K-means:
Особенности метода:
* Количество кластеров необходимо определять самостоятельно.
* Теоретически обеспечена сходимость к локальному минимуму.
* На практике локальные минимумы могут не давать логичный результат для кластеризации.
* Результат зависит от выбора начальных центроидов, которых может быть бесконечно много.
* Старается создавать кластеры примерно одного размера / разброса, выделяет эллиптические (шарообразные) кластеры.

n_init влияет на качество, но также и на время работы. На выходе будет лучший из n_init результатов.
max_iter влияет на время и качество, в случае данной работы конкретное наглядное влияние отследить не получилось.

class sklearn.cluster.Birch(threshold=0.5, branching_factor=50, n_clusters=3, compute_labels=True, copy=True)

Особенности метода:
* Каждое решение кластеризации локально и осуществляется без просмотра всех точек данных и существующих на текущий момент кластеров.
* Метод работает на наблюдениях, пространство данных которых обычно не однородно заполнено и не каждая точка данных одинаково важна.
* Метод позволяет использовать всю доступную память для получения наиболее точных возможных подкластеров при минимизации цены ввода/вывода.
* Метод является инкрементальным и не требует наличия полного набора данных сразу.

Первая фаза строит *CF дерево* точек данных, высоко сбалансированную древесную структуру, определённую следующим образом:

- Если дан набор N d-мерных точек данных, *признак кластеризации* (англ. *Clustering Feature*) CF набора определяется как тройка $CF = (N, LS, SS)$, где $\overrightarrow{LS} = \sum_{i=1}^{N} \overrightarrow{X_i}$ является линейной суммой, а $\overrightarrow{SS} = \sum_{i=1}^{N} (\overrightarrow{X_i})^2$ является суммой квадратов точек данных.
- Признаки кластеризации организуются в *CF-дерево*, высоко сбалансированное дерево с двумя параметрами: коэффициентом ветвления B и порогом T. Каждый нелистовой узел состоит максимум из B входов вида $[CF_i, child_i]$, где $child_i$ является указателем на его i-ого потомка, а CF_i является признаком кластеризации, представляющим связанный подкластер. Лист содержит не более L входов, каждый вида $[CF_i]$. Он также имеет два указателя, prev и next, которые используются для соединения в цепь все листы. Размер дерева зависит от параметра T. Требуется, чтобы узел A вмещался на страницу размера P. B и L определяются значением P. Таким образом, P может меняться для настройки производительности[en]. Это очень компактное представление набора данных, поскольку каждый лист не является отдельной точкой данных, а является подкластером.

branching_factor как раз устанавливает аксимальное количество подкластеров CF в каждом узле, что влияет на время и на качетсво кластеризации.

class sklearn.cluster.AgglomerativeClustering(n_clusters=2, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None)

Особенности метода:
* Новые кластеры создаются путем объединения более мелких кластеров и, таким образом, дерево создается от листьев к стволу.

affinity – метрика, на основе которой вычисляются связи, евклидова по умолчанию. Влияет на качество, однако в данном случае наглядной разницы не видно.
compute_full_tree – лучше поставить ‘auto’ - для уменьшения времени вычислений. Тогда все дерево не будет вычислено целиком, если текущих результатов достаточно.

Результаты и выводы по работе:

В ходе данной лабораторной работы были повышены комптенции в области кластеризации с использованием библиотеки sklearn.
Также в ходе работы были получены практические результаты для использовавшихся подходов к решению поставленной задачи:

было реализовано сравнение между подходами KMeans, Birch, AgglomerativeClustering из модуля clusters библиотеки sklearn.

По метрике Silhouette Score лучший результат для использованных данных продемонстрировал подход KMeans, который показал результат: *0.402128* для разбиения на $k=4$ кластеров.