https://naiveskill.com/mongodb-with-python/ db=database name coll=collection name whichCollection.find() db.coll.insert_one(recordJSON) db.coll.insert_many(recordJSON) db.coll.delete_one(...) db.coll.delete_many(...) db.coll.find({ attr: { $operator: value} }) db.coll.find({ attr: { operator: value} [, { attr: { $operator: value}] })
db.coll.update({attr:value},{$set:{attr:value}})
db.coll.updateMany({ attr: {$op: val} } , { $inc: { attr: val} }
db.coll.updateMany({ attr: {$op: val} } , { $inc: { attr: val} }
aggreg=db.coll.aggregate([
{ '$group': { '_id': attr, 'total': { '$op': "$attr" } } }, { '$sort': { 'total': -1 } }])

```
cur=db.coll.aggregate([
   { "$lookup":
      {
      from: <collection to join>,
      localField: <field from the input documents>,
      foreignField: <field from the documents of the "from" collection>,
      as: <output array field>
       }
   }
])
```

SOME OPERATORS:
Name Description
eq Matches values that are equal to a specified value. $gt Matches values that are greater than a specified value. $gte Matches values that are greater than or equal to a specified value. $in Matches any of the values specified in an array. $lt Matches values that are less than a specified value. $lte Matches values that are less than or equal to a specified value. $ne Matches all values that are not equal to a specified value. $nin Matches none of the values specified in an array. Logical Name Description $and Joins query clauses with a logical AND returns all documents that match the conditions of both clauses. $not Inverts the effect of a query expression and returns documents that do not match the query expression. $nor Joins query clauses with a logical NOR returns all documents that fail to match both clauses. $or Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

In [2]:
```
import pymongo
```

launch the daemon

inside the bin dir of mongo mkdir aulaMONGODB Macs-MBP-4:bin pedro$ mongod --dbpath aulaMONGODB

Connect the client

In [2]:
```
client = pymongo.MongoClient("mongodb://host.docker.internal:32768/")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 client = pymongo.MongoClient("mongodb://host.docker.internal:32768/")

NameError: name 'pymongo' is not defined
```

In [4]:
```
client.list_database_names()
```

Out[4]:
```
['admin', 'config', 'local', 'med_data']
```

import restaurants and neighbourhoods JSON

In [6]:
```
!wget https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/prime
```

```
--2023-03-21 20:01:28--  https://raw.githubusercontent.com/mongodb/docs-assets/
primer-dataset/primer-dataset.json
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.
133, 185.199.108.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.11
1.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11874761 (11M) [text/plain]
Saving to: ‘primer-dataset.json’

primer-dataset.json 100%[===================>]  11.32M  8.57MB/s    in 1.3s

2023-03-21 20:01:31 (8.57 MB/s) - ‘primer-dataset.json’ saved [11874761/1187476
1]
```

See databases

In [ ]:
```
client.list_database_names()
```

Connect to restaurants, reference restaurants

In [12]:
```
db = client['restaurants']
```

In [14]:
```
client['restaurants'].list_collection_names()
```

Out[14]:
```
[]
```

In [15]:
```
restaurants=client['restaurants']['restaurants']
```

In [17]:
```
import json
with open('./work/restaurants.json') as f:
  data = f.readlines()
  file_data = json.loads("[" + ",".join(data) + "]")

restaurants.insert_many(file_data)
```

Out[17]: <pymongo.results.InsertManyResult at 0xffff741eb940>

In [52]:
```
import pprint
for restaurant in restaurants.find()[:5]:
    pprint.pprint(restaurant)
```

```
{'_id': ObjectId('641a0e7d5e7639d44270ac41'),
 'address': {'building': '1007',
             'coord': [-73.856077, 40.848447],
             'street': 'Morris Park Ave',
             'zipcode': '10462'},
 'borough': 'Bronx',
 'cuisine': 'Bakery',
 'grades': [{'date': {'$date': 1393804800000}, 'grade': 'A', 'score': 2},
            {'date': {'$date': 1378857600000}, 'grade': 'A', 'score': 6},
            {'date': {'$date': 1358985600000}, 'grade': 'A', 'score': 10},
            {'date': {'$date': 1322006400000}, 'grade': 'A', 'score': 9},
            {'date': {'$date': 1299715200000}, 'grade': 'B', 'score': 14}],
 'name': 'Morris Park Bake Shop',
 'restaurant_id': '30075445'}
{'_id': ObjectId('641a0e7d5e7639d44270ac42'),
 'address': {'building': '469',
             'coord': [-73.961704, 40.662942],
             'street': 'Flatbush Avenue',
             'zipcode': '11225'},
 'borough': 'Brooklyn',
 'cuisine': 'Hamburgers',
 'grades': [{'date': {'$date': 1419897600000}, 'grade': 'A', 'score': 8},
            {'date': {'$date': 1404172800000}, 'grade': 'B', 'score': 23},
            {'date': {'$date': 1367280000000}, 'grade': 'A', 'score': 12},
            {'date': {'$date': 1336435200000}, 'grade': 'A', 'score': 12}],
 'name': "Wendy'S",
 'restaurant_id': '30112340'}
{'_id': ObjectId('641a0e7d5e7639d44270ac43'),
 'address': {'building': '351',
             'coord': [-73.98513559999999, 40.7676919],
             'street': 'West   57 Street',
             'zipcode': '10019'},
 'borough': 'Manhattan',
 'cuisine': 'Irish',
 'grades': [{'date': {'$date': 1409961600000}, 'grade': 'A', 'score': 2},
            {'date': {'$date': 1374451200000}, 'grade': 'A', 'score': 11},
            {'date': {'$date': 1343692800000}, 'grade': 'A', 'score': 12},
            {'date': {'$date': 1325116800000}, 'grade': 'A', 'score': 12}],
 'name': 'Dj Reynolds Pub And Restaurant',
 'restaurant_id': '30191841'}
{'_id': ObjectId('641a0e7d5e7639d44270ac44'),
 'address': {'building': '2780',
             'coord': [-73.98241999999999, 40.579505],
             'street': 'Stillwell Avenue',
             'zipcode': '11224'},
 'borough': 'Brooklyn',
 'cuisine': 'American',
 'grades': [{'date': {'$date': 1402358400000}, 'grade': 'A', 'score': 5},
            {'date': {'$date': 1370390400000}, 'grade': 'A', 'score': 7},
            {'date': {'$date': 1334275200000}, 'grade': 'A', 'score': 12},
            {'date': {'$date': 1318377600000}, 'grade': 'A', 'score': 12}],
 'name': 'Riviera Caterer',
 'restaurant_id': '40356018'}
{'_id': ObjectId('641a0e7d5e7639d44270ac45'),
 'address': {'building': '97-22',
             'coord': [-73.8601152, 40.7311739],
             'street': '63 Road',
             'zipcode': '11374'},
```

```
 'borough': 'Queens',
 'cuisine': 'Jewish/Kosher',
 'grades': [{'date': {'$date': 1416787200000}, 'grade': 'Z', 'score': 20},
            {'date': {'$date': 1358380800000}, 'grade': 'A', 'score': 13},
            {'date': {'$date': 1343865600000}, 'grade': 'A', 'score': 13},
            {'date': {'$date': 1323907200000}, 'grade': 'B', 'score': 25}],
 'name': 'Tov Kosher Kitchen',
 'restaurant_id': '40356068'}
```

Compute the average scores of the restaurants.

We pass an array to the aggregate function.

The $unwind parameter is used to deconstruct the grades array in order to output a document for each element.

Next we use the $match parameter including everything (by using open and closing braces). We could filter further here by providing additional criteria.

Next we use the $group parameter to group the data that we want to apply the computation to.

Finally we create new key called “Avg grade” and apply the $avg (average) parameter to the grades scores of individual restaurants referencing grades followed by a dot and the score key.

There are many other parameters that can be used for common computations such as sum,min, $max etc.

In [172…
```
#complete the next code to do it...see the result...WHY DOES IT NOT WORK? go bac
#on restaurants data

result = restaurants.aggregate(
  [
     {"$match": {}},
     {"$group": {"_id": "$restaurant_id", "Avg grade": {"$avg": "$grades.score"
      }
      }
  ]
)
```

In [173…
```
for item in list(result)[:20]:
    print(item)
```

```
{'_id': '50001415', 'Avg grade': None}
{'_id': '41294917', 'Avg grade': None}
{'_id': '41058103', 'Avg grade': None}
{'_id': '41329496', 'Avg grade': None}
{'_id': '41461464', 'Avg grade': None}
{'_id': '41538914', 'Avg grade': None}
{'_id': '50002354', 'Avg grade': None}
{'_id': '50005630', 'Avg grade': None}
{'_id': '41313420', 'Avg grade': None}
{'_id': '40889676', 'Avg grade': None}
{'_id': '50011120', 'Avg grade': None}
{'_id': '50015072', 'Avg grade': None}
{'_id': '41278303', 'Avg grade': None}
{'_id': '40387086', 'Avg grade': None}
{'_id': '41637060', 'Avg grade': None}
{'_id': '41500477', 'Avg grade': None}
{'_id': '41519339', 'Avg grade': None}
{'_id': '41699005', 'Avg grade': None}
{'_id': '50010813', 'Avg grade': None}
{'_id': '50015399', 'Avg grade': None}
```

In [174…

```
#do it again, but now add unwind. See the result table....What does unwind do, w
#work now?

result = restaurants.aggregate(
  [
     {"$unwind": "$grades"},
     {"$match": {}},
     {"$group": {"_id": "$restaurant_id", "Avg grade": {"$avg": "$grades.score"
  ]
)
```

In [175…

```
#redo query but now also sort the returned in ascending or descending order.
#We could simply add another line with the sort parameter specifying which field
#1 (ascending) or -1 (descending).

#do it again, but now add unwind. See the result table....What does unwind do, w
#work now?

result = restaurants.aggregate(
  [
     {"$unwind": "$grades"},
     {"$match": {}},
     {"$group": {"_id": "$restaurant_id", "Avg grade": {"$avg": "$grades.score"
     {"$sort": {"Avg grade": -1}}
  ]
)
```

In [176…

```
#print result, see if ot is sorted correctly:
for item in list(result)[:20]:
    print(item)
```

```
{'_id': '50015959', 'Avg grade': 75.0}
{'_id': '50017374', 'Avg grade': 73.0}
{'_id': '50018457', 'Avg grade': 69.0}
{'_id': '50015245', 'Avg grade': 69.0}
{'_id': '41703965', 'Avg grade': 65.0}
{'_id': '50017252', 'Avg grade': 61.0}
{'_id': '50011285', 'Avg grade': 61.0}
{'_id': '50013507', 'Avg grade': 60.0}
{'_id': '50011359', 'Avg grade': 60.0}
{'_id': '50018376', 'Avg grade': 58.0}
{'_id': '50016721', 'Avg grade': 56.0}
{'_id': '50016506', 'Avg grade': 56.0}
{'_id': '50015482', 'Avg grade': 56.0}
{'_id': '50018079', 'Avg grade': 56.0}
{'_id': '50018468', 'Avg grade': 56.0}
{'_id': '50016149', 'Avg grade': 54.0}
{'_id': '50000607', 'Avg grade': 53.0}
{'_id': '41720129', 'Avg grade': 53.0}
{'_id': '50018202', 'Avg grade': 53.0}
{'_id': '50016844', 'Avg grade': 52.0}
```

Convert data from a Mongo database into tabular form as a Panda's dataframe object

In [42]:
```
import pandas as pd

extracted_data = restaurants.find({},{"borough": "Bronx", "cuisine": "Bakery", '
bronx_bakeries = list(extracted_data)

bakeries=pd.DataFrame.from_dict(bronx_bakeries)
bakeries = bakeries.set_index("_id")
```

In [43]:
```
bakeries
```

Out[43]:

_id	name	borough	cuisine
641a0e7d5e7639d44270ac41	Morris Park Bake Shop	Bronx	Bakery
641a0e7d5e7639d44270ac42	Wendy'S	Bronx	Bakery
641a0e7d5e7639d44270ac43	Dj Reynolds Pub And Restaurant	Bronx	Bakery
641a0e7d5e7639d44270ac44	Riviera Caterer	Bronx	Bakery
641a0e7d5e7639d44270ac45	Tov Kosher Kitchen	Bronx	Bakery
...	...	...	...
641a0e7d5e7639d442710f4b	Subway	Bronx	Bakery
641a0e7d5e7639d442710f4c	Fairfield Inn Suites Penn Station	Bronx	Bakery
641a0e7d5e7639d442710f4d		Bronx	Bakery
641a0e7d5e7639d442710f4e	Indian Oven	Bronx	Bakery
641a0e7d5e7639d442710f4f	Cold Press'D	Bronx	Bakery

25359 rows × 3 columns

Using the mongodb query and a python dataframe:

Show a table and a chart with the average ratings per restarurant, the top 5 and the tail 5.

Since the tail 5 will show no valid info, remove nulls and also remove those with avg rating 0, show the remianing ones

In [162…

```
#complete the code:

result = restaurants.aggregate(
  [
     {"$match": {}},
     {"$unwind": "$grades"},
     {"$group": {"_id": "$restaurant_id", "Avg grade": {"$avg": "$grades.score"
  ]
)
bakeriesAVGPD=pd.DataFrame(list(result))
bakeriesAVGPD=bakeriesAVGPD.dropna().sort_values("Avg grade",ascending=False)
```

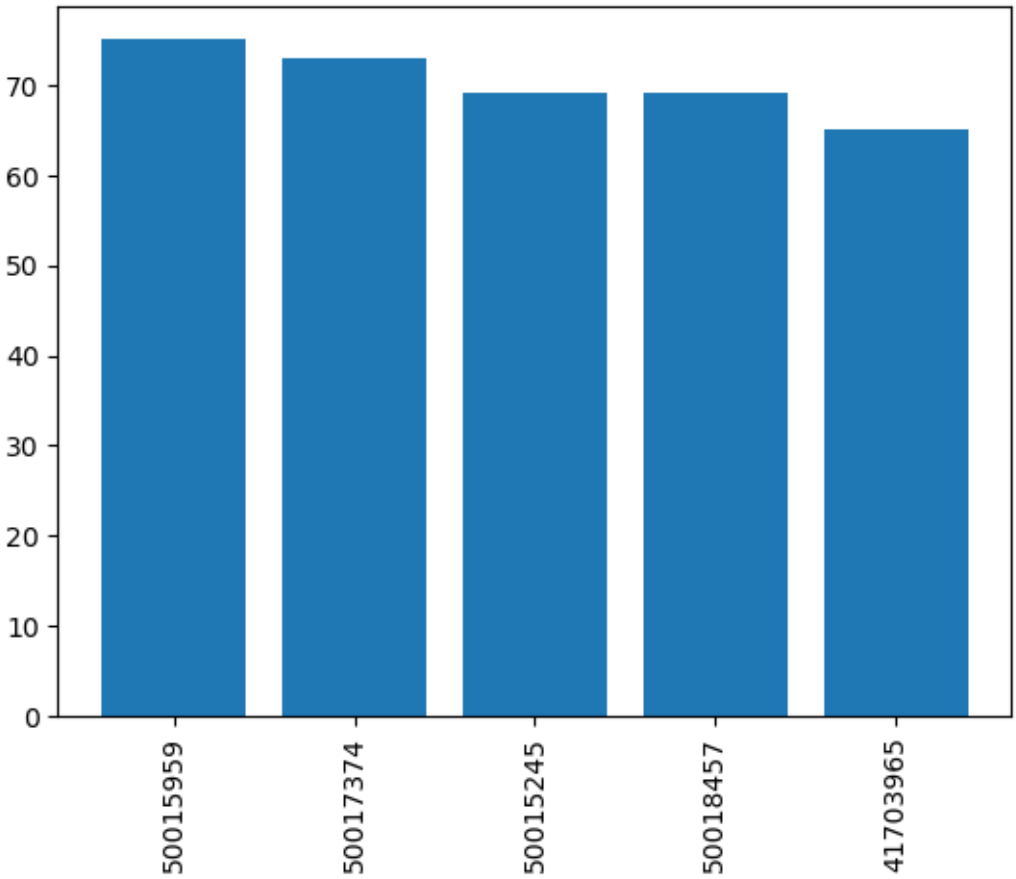
In [74]:

```
pd.concat([bakeriesAVGPD.head(5),bakeriesAVGPD.tail(5)])
```

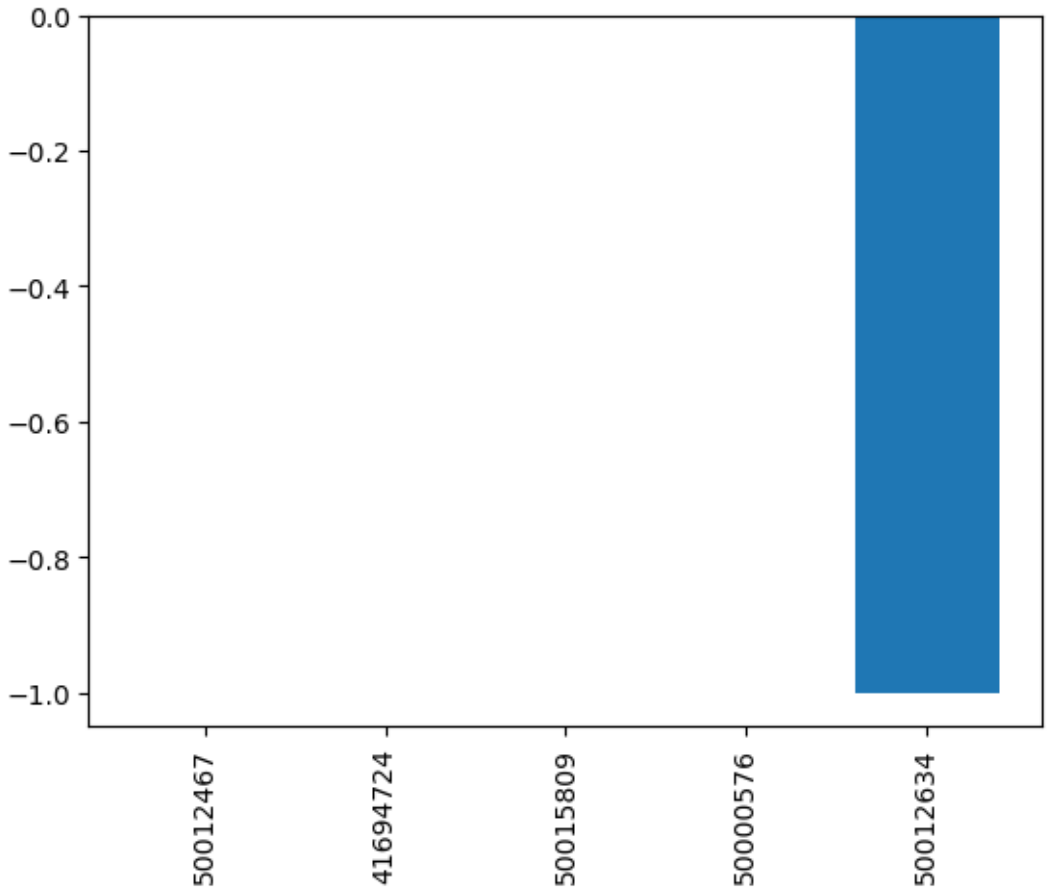
Out[74]:

	_id	Avg grade
19669	50015959	75.0
17078	50017374	73.0
2141	50015245	69.0
13859	50018457	69.0
12270	41703965	65.0
19573	50012467	0.0
5010	41694724	0.0
9653	50015809	0.0
6559	50000576	0.0
13451	50012634	-1.0

In [75]:
```
#complete to plot the top 5 chart:

import matplotlib.pyplot as plt

plt.bar(bakeriesAVGPD.head(5)['_id'],bakeriesAVGPD.head(5)["Avg grade"])
plt.xticks(rotation=90)
plt.show()
```



In [77]:
```
#complete to plot the top 5 chart:

import matplotlib.pyplot as plt

plt.bar(bakeriesAVGPD.tail(5)['_id'],bakeriesAVGPD.tail(5)["Avg grade"])
plt.xticks(rotation=90)
plt.show()
```



In [80]:
```
#drop rows with null values and complete the following code to work well

filter = bakeriesAVGPD["Avg grade"]>0

# filtering data
bakeriesAVGPD1=bakeriesAVGPD.where(filter, inplace = False)

# display
bakeriesAVGPD1.dropna()
```

Out[80]:

	_id	Avg grade
19669	50015959	75.000000
17078	50017374	73.000000
2141	50015245	69.000000
13859	50018457	69.000000
12270	41703965	65.000000
...	...	...
23399	41618402	0.666667
1974	41618440	0.666667
3577	41618385	0.666667
5296	41611748	0.500000
12711	41611720	0.500000

24454 rows × 2 columns

Next we want to see avergae grades per LOCATION (borough=neighborhood) instead of per restaurant

In [157…

```
# Show tables with the average, max and stdev ratings per location, charts with
# rating

result = restaurants.aggregate(
  [
    {"$match": {"borough":{"$ne": None}}},
    {"$unwind": "$grades"},
     {"$facet" : {
      "Borough" : [{"$group": {"_id": "$borough"}}],
      "Avg" : [{"$group": {"_id": "$borough", "V": {"$avg": "$grades.score" }}}]
      "Max" : [{"$group": {"_id": "$borough", "V": {"$max": "$grades.score" }}}]
      "Stdev" : [{"$group": {"_id": "$borough", "V": {"$stdDevPop": "$grades.scc
      "Bot5" : [{"$group": {"_id": "$borough", "V": {
        "$bottomN": {"n":5 , "output": [ "$grades.score" ],"sortBy": { "grades.s
        ],
      "Top5" : [{"$group": {"_id": "$borough", "V": {
        "$topN": {"n":5 , "output": [ "$grades.score" ],"sortBy": { "grades.scor
        ],
     }
     },
  ]
)

bakeriesAVG = list(result)

bakeriesAVGPD=pd.DataFrame.from_dict(*bakeriesAVG)
bakeriesAVG
```

Out[157]:

```
[{'Borough': [{'_id': 'Manhattan'},
   {'_id': 'Bronx'},
   {'_id': 'Staten Island'},
   {'_id': 'Missing'},
   {'_id': 'Queens'},
   {'_id': 'Brooklyn'}],
  'Avg': [{'_id': 'Brooklyn', 'V': 11.44797595737899},
   {'_id': 'Manhattan', 'V': 11.418151216986018},
   {'_id': 'Staten Island', 'V': 11.370957711442786},
   {'_id': 'Missing', 'V': 9.632911392405063},
   {'_id': 'Queens', 'V': 11.634865110930088},
   {'_id': 'Bronx', 'V': 11.036186099942562}],
  'Max': [{'_id': 'Bronx', 'V': 82},
   {'_id': 'Brooklyn', 'V': 86},
   {'_id': 'Manhattan', 'V': 131},
   {'_id': 'Queens', 'V': 84},
   {'_id': 'Staten Island', 'V': 68},
   {'_id': 'Missing', 'V': 47}],
  'Stdev': [{'_id': 'Bronx', 'V': 7.017117017155606},
   {'_id': 'Manhattan', 'V': 7.433047046784533},
   {'_id': 'Staten Island', 'V': 6.623126963476623},
   {'_id': 'Missing', 'V': 6.610649664908034},
   {'_id': 'Queens', 'V': 7.252487538887249},
   {'_id': 'Brooklyn', 'V': 7.392521214522643}],
  'Bot5': [{'_id': 'Missing', 'V': [[2], [2], [2], [2], [0]]},
   {'_id': 'Queens', 'V': [[None], [None], [None], [None], [None]]},
   {'_id': 'Bronx', 'V': [[0], [0], [0], [0], [None]]},
   {'_id': 'Brooklyn', 'V': [[-1], [-1], [-1], [None], [None]]},
   {'_id': 'Staten Island', 'V': [[0], [0], [0], [-1], [-1]]},
   {'_id': 'Manhattan', 'V': [[-1], [-1], [-1], [None], [None]]}],
  'Top5': [{'_id': 'Queens', 'V': [[84], [78], [73], [73], [70]]},
   {'_id': 'Missing', 'V': [[47], [40], [23], [21], [13]]},
   {'_id': 'Manhattan', 'V': [[131], [98], [98], [92], [90]]},
   {'_id': 'Bronx', 'V': [[82], [76], [75], [73], [73]]},
   {'_id': 'Brooklyn', 'V': [[86], [81], [78], [77], [77]]},
   {'_id': 'Staten Island', 'V': [[68], [65], [60], [58], [58]]}]}]
```


In [ ]:
```
#complete to show the head and tail 5

import matplotlib.pyplot as plt

plt.bar(*bakeriesAVGPD["Top5"],bakeriesAVGPD.head(5)["?"])
plt.xticks(rotation=90)
plt.show()

?
```

In [ ]:
```
#complete to show all values

plt.bar(bakeriesAVGPD['?'],bakeriesAVGPD["?"])
plt.xticks(rotation=90)
plt.show()
```

The end !!!!