# Report

January 13, 2021

[1]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
```

## 1 Comparing Python and Enso dataframe operations.

This benchmark compares some basic dataframe operations - joining, filling missing values, filtering, mapping vectorized operations.

### 1.1 How it is measured

It uses a very simple methodology - each pipeline is run 20 times and times for each operation of interest are reported. Thus it does *not* measure startup time (as this is not relevant for the comparison) and it does not check how warmup affects efficiency (so that it measures performance as it is right after starting).

The benchmarks were run on a MacBook Pro (2016) with 16GB RAM, connected to AC.

### 1.2 What is measured

[2]:
```
citations = pd.read_csv("../data/Parking_Citations_in_FY_2018.csv",␣
 ↪low_memory=False)
meters = pd.read_csv("../data/LADOT_Metered_Parking_Inventory___Policies.csv")
```

[3]:
```
citations.shape
```

[3]:
```
(2188975, 18)
```

[4]:
```
meters.shape
```

[4]:
```
(33989, 9)
```

[5]:
```
headers = [
    "loading",
```

```
    "filter1",
    "fill_na",
    "joining",
    "filter2",
    "map_and_filter1",
    # "map_and_filter2",
]
```

[6]:
```
citations["Meter Id"] = citations["Meter Id"].fillna("")
citations.join(meters.set_index("SpaceID"), on="Meter Id", how="inner").shape
```

[6]:
```
(341338, 26)
```

We use as input two files from Los Angeles Open Data

- `Parking_Citations_in_FY_2018.csv` - 268MB file consisting of 2188975 rows and 18 columns,
- `LADOT_Metered_Parking_Inventory___Policies.csv` - 3.5MB file consisting of 33989 rows and 18 columns.

We measure the following operations, implemented using analogous constructs in Python and Enso:

- `loading` - loading and parsing the two CSV files into DataFrames/Tables,
- `filter1` - filtering citations whose `Meter Id` is null and returning their count,
- `fill_na` - filling the missing `Meter Id` with empty strings,
- `joining` - joining the two tables over a common index (`Meter Id` and `SpaceID`) where 341338 out of 2188975 citations have common indices with the second table,
- `filter2` - filtering, in the result of the join, which `Meter Id` fields are not equal to an empty string and counting the results,
- `map_and_filter1` - filling missing values in `BlockFace` by an empty string and then selecting from them such entries that end with string `BROADWAY` and counting the results.

### 1.3 Results

[7]:
```
enso_times = pd.read_csv("enso_times.csv")
python_times = pd.read_csv("python_times.csv")
```

[8]:
```
assert(set(enso_times.columns) == set(headers))
```

[9]:
```
assert(set(python_times.columns) == set(headers))
```

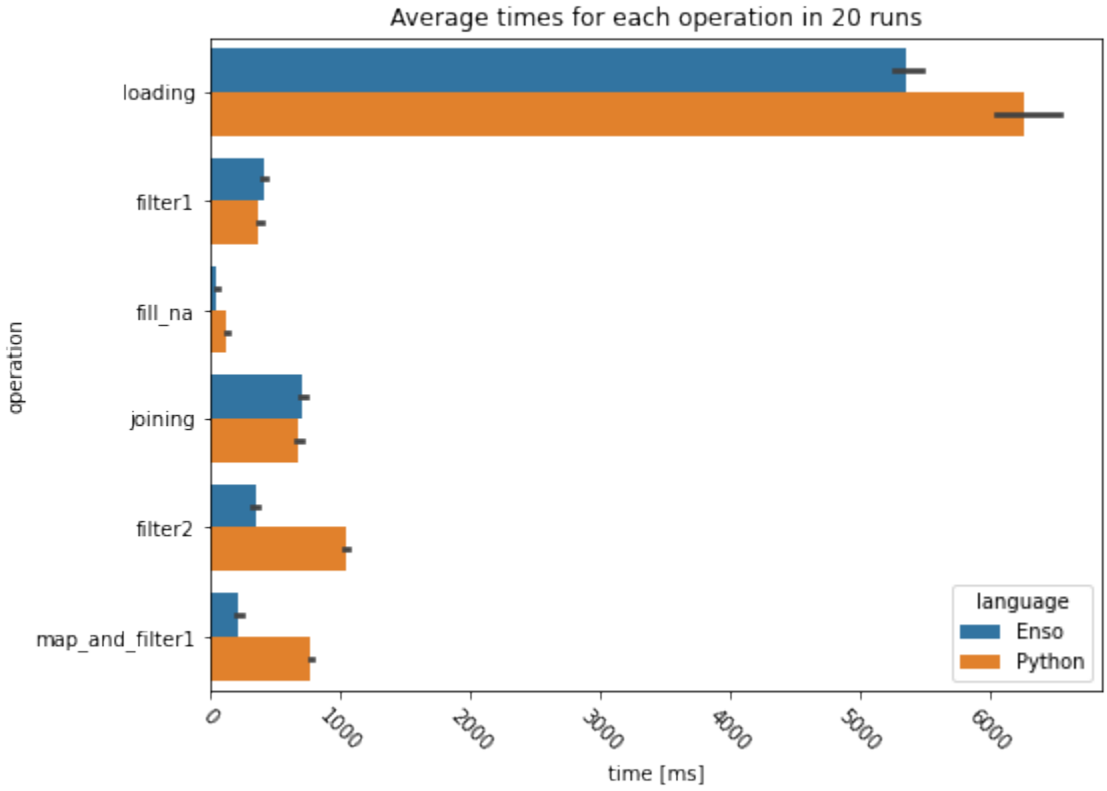
[10]:
```
assert((enso_times.columns == python_times.columns).all())
```

[11]:
```
enso_times["language"] = "Enso"
python_times["language"] = "Python"
```

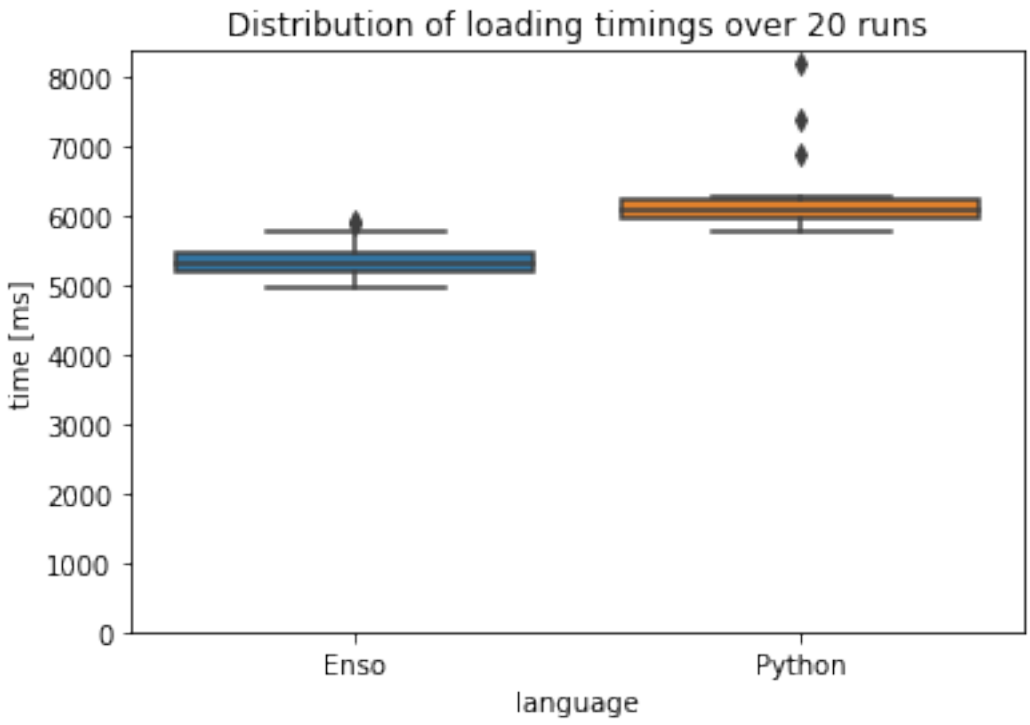
[12]:
```
times = pd.concat([enso_times, python_times])
```

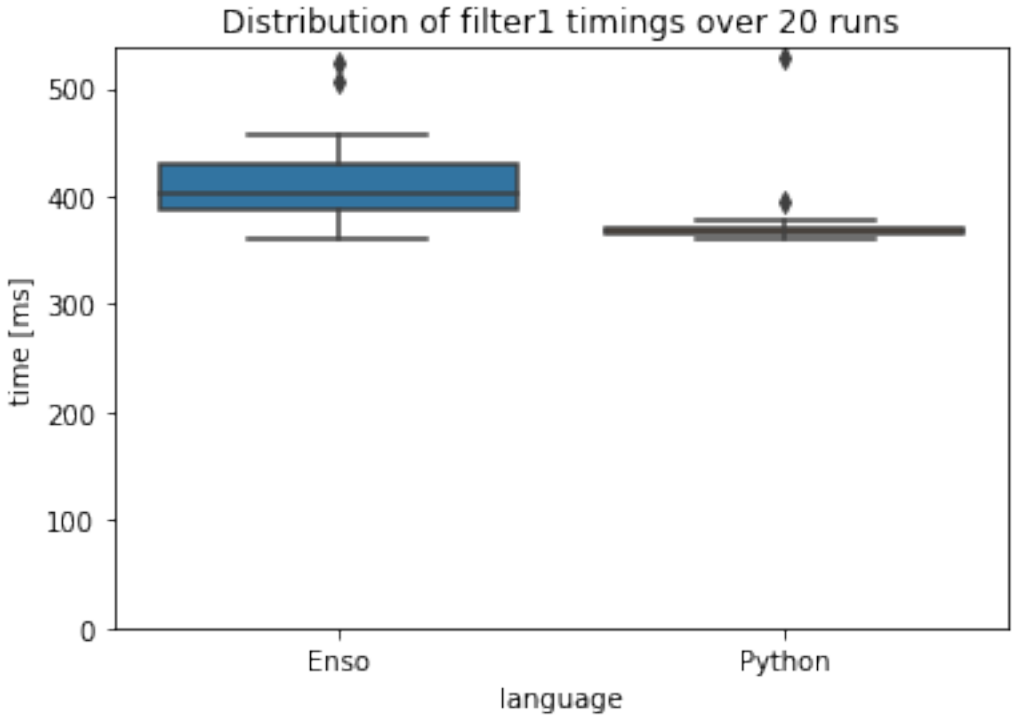
[13]:
```
melted = times.melt(id_vars=["language"], var_name="operation",␣
↪value_name="time")
```

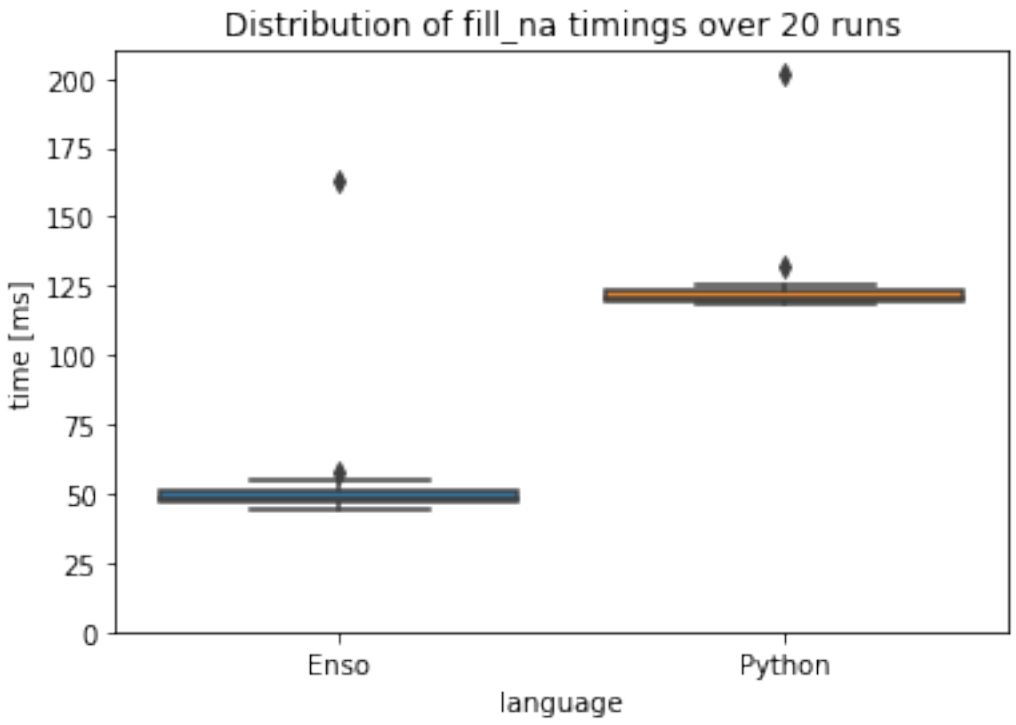
[14]:
```
plt.figure(figsize=(8,6))
sns.barplot(data=melted, y="operation", x="time", hue="language")
plt.xlabel("time [ms]")
plt.title("Average times for each operation in 20 runs")
plt.xticks(rotation=-45)
plt.show()
```

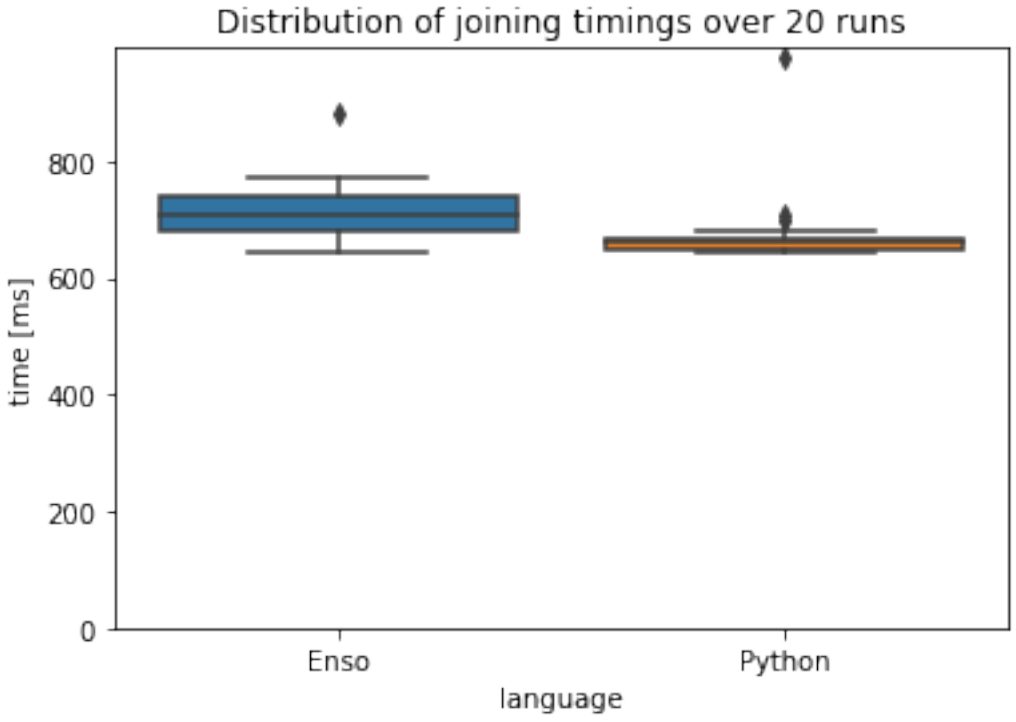


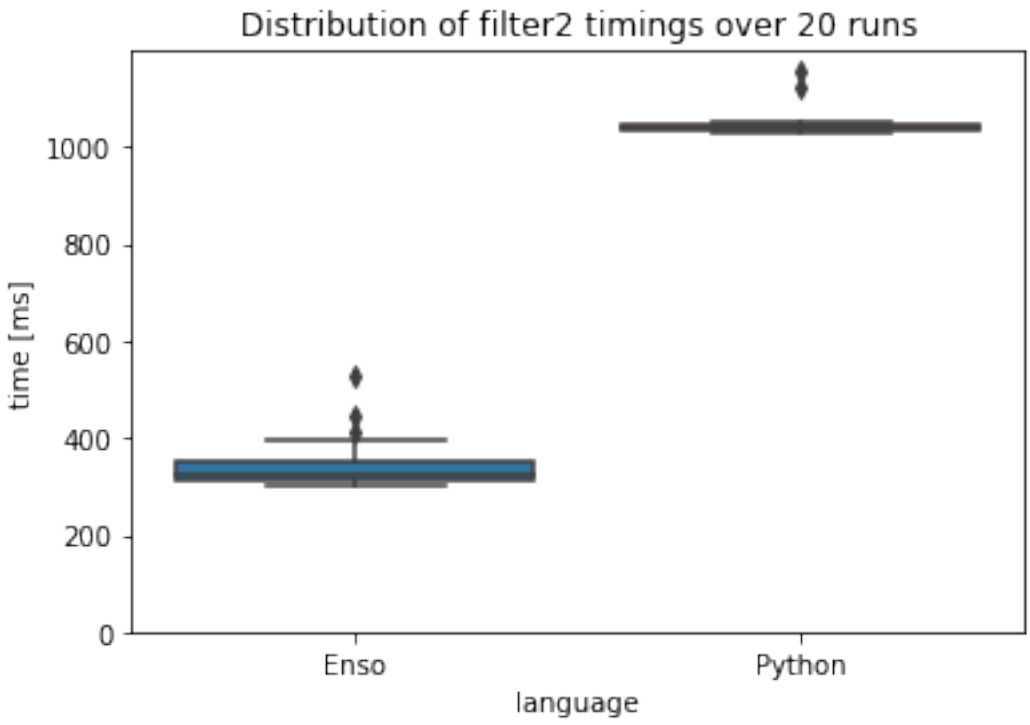

[15]:
```
for op in headers:
    filtered = melted[melted["operation"] == op]
    sns.boxplot(data=filtered, x="language", y="time")
    plt.title(f"Distribution of {op} timings over 20 runs")
    plt.ylim(bottom=0)
    plt.ylabel("time [ms]")
    plt.show()
```

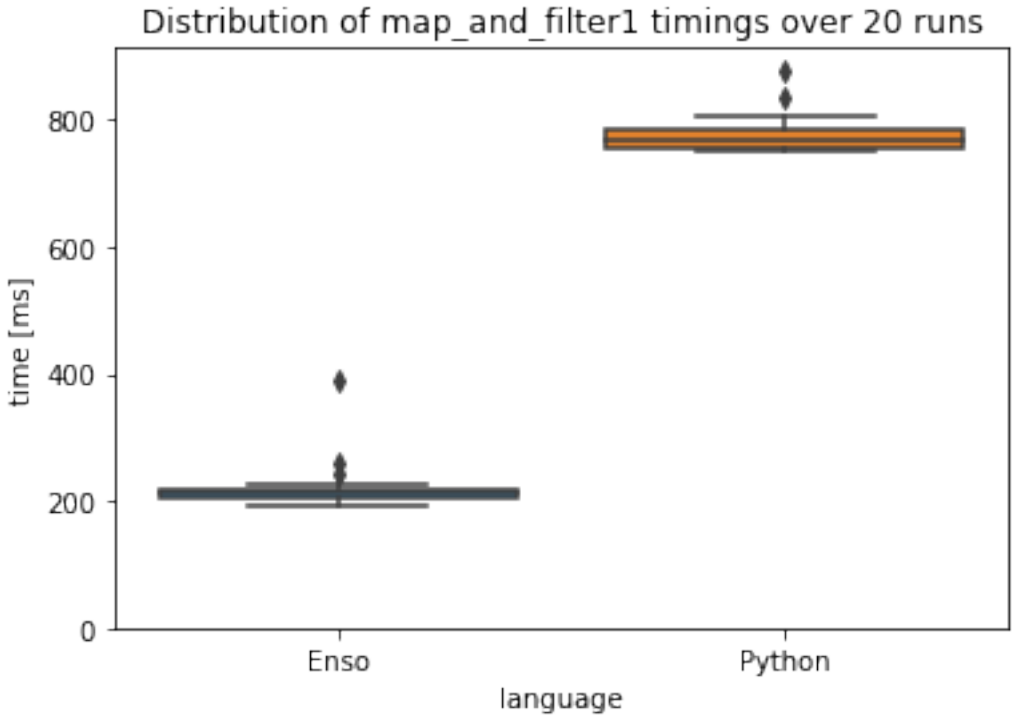

Distribution of loading timings over 20 runs
time [ms]
8000
7000
6000
5000
4000
3000
2000
1000
0
Enso
Python
language


Distribution of filter1 timings over 20 runs
time [ms]
500
400
300
200
100
0
Enso
Python
language


Distribution of fill_na timings over 20 runs
time [ms]
0
25
50
75
100
125
150
175
200
Enso
Python
language


Distribution of joining timings over 20 runs
time [ms]
0
200
400
600
800
Enso
Python
language


Distribution of filter2 timings over 20 runs
time [ms]
0
200
400
600
800
1000
Enso
Python
language

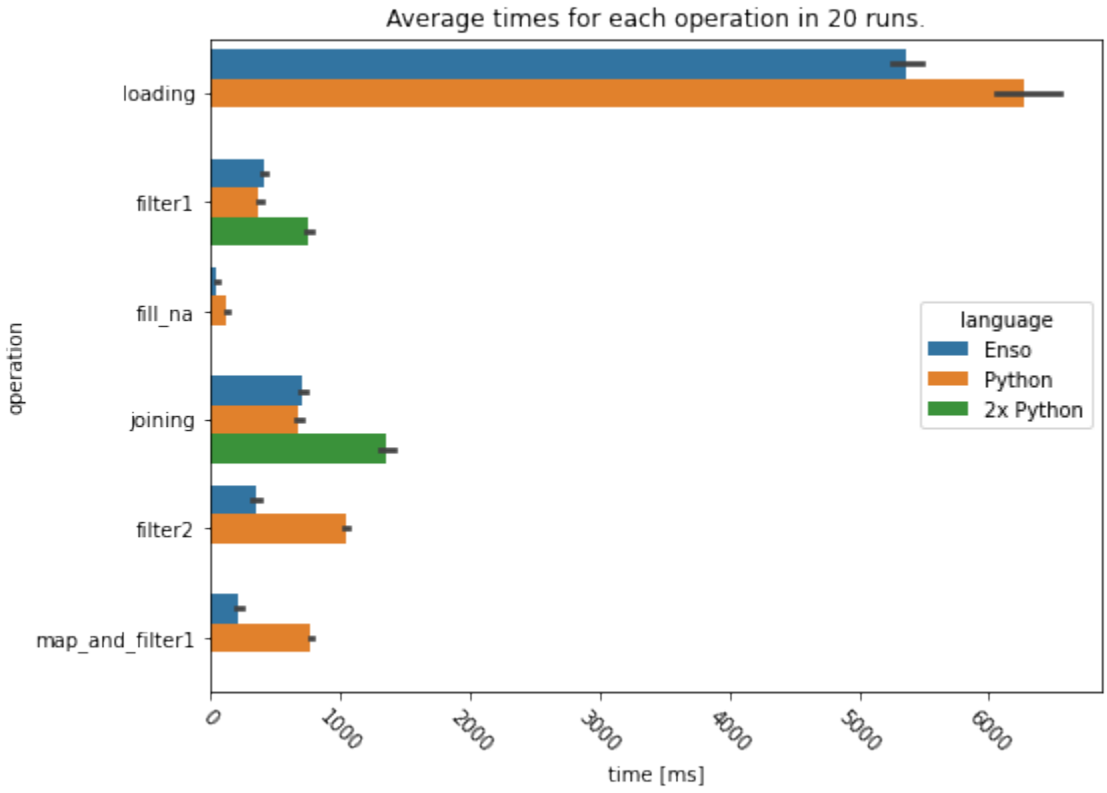

Distribution of map_and_filter1 timings over 20 runs
time [ms]
0
200
400
600
800
Enso
Python
language

[16]:
```
multipliers = []
for op in headers:
    means = melted[melted["operation"] == op].groupby(["language"]).mean()
    enso_mean = means.loc["Enso", "time"]
    python_mean = means.loc["Python", "time"]
    if enso_mean < python_mean:
        mult = 1
    elif enso_mean < 2 * python_mean:
        mult = 2
    elif enso_mean < 3 * python_mean:
        mult = 3
    else:
        mult = 5

    print(f"{op} fits within {mult}x of Python")

    if mult > 1:
        multipliers.append((op, mult))
```

```
loading fits within 1x of Python
filter1 fits within 2x of Python
fill_na fits within 1x of Python
joining fits within 2x of Python
filter2 fits within 1x of Python
map_and_filter1 fits within 1x of Python
```

[17]:
```
multiplied_times = []
for (op, multiplier) in multipliers:
    multiplied = melted[(melted["language"] == "Python") & (melted["operation"]␣
 ↪== op)].copy()
    multiplied["time"] = multiplied["time"] * multiplier
    multiplied["language"] = f"{multiplier}x Python"
    multiplied_times.append(multiplied)
```

[18]:
```
plt.figure(figsize=(8,6))
with2x = pd.concat([melted] + multiplied_times)
sns.barplot(data=with2x, y="operation", x="time", hue="language")
plt.xlabel("time [ms]")
plt.title("Average times for each operation in 20 runs.")
plt.xticks(rotation=-45)
plt.show()
```




We can see that all of the tested operations are strictly no more than 5x slower than their Python counterparts. In fact, most of them have comparable (or for some, even better) performance or are about 2x slower at worst.