wasm-js-energy-study-analysis

September 28, 2023

[1]:
```
pip install pandas matplotlib seaborn numpy scipy statsmodels scikits.bootstrap
```

```
Collecting pandas
  Downloading
pandas-2.1.1-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (14.9
MB)
                            14.9/14.9 MB
31.6 MB/s eta 0:00:0000:0100:01
Collecting matplotlib
  Downloading
matplotlib-3.8.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
(11.4 MB)
                            11.4/11.4 MB
34.2 MB/s eta 0:00:0000:0100:01
Collecting seaborn
  Downloading seaborn-0.12.2-py3-none-any.whl (293 kB)
                           293.3/293.3 KB
43.1 MB/s eta 0:00:00
Collecting numpy
  Downloading
numpy-1.26.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (14.2
MB)
                            14.2/14.2 MB
33.0 MB/s eta 0:00:0000:0100:01
Collecting scipy
  Downloading
scipy-1.11.3-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (33.0
MB)
                            33.0/33.0 MB
25.7 MB/s eta 0:00:0000:0100:01
Collecting statsmodels
  Downloading
statsmodels-0.14.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
(9.8 MB)
                            9.8/9.8 MB
5.6 MB/s eta 0:00:0000:0100:01m
Collecting scikits.bootstrap
  Downloading scikits.bootstrap-1.1.0-py2.py3-none-any.whl (14 kB)
```

```
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.9/site-packages (from pandas) (2.8.2)
Collecting tzdata>=2022.1
  Downloading tzdata-2023.3-py2.py3-none-any.whl (341 kB)
                           341.8/341.8 KB
33.1 MB/s eta 0:00:00
Requirement already satisfied: pytz>=2020.1 in
/opt/conda/lib/python3.9/site-packages (from pandas) (2022.1)
Requirement already satisfied: importlib-resources>=3.2.0 in
/opt/conda/lib/python3.9/site-packages (from matplotlib) (5.6.0)
Collecting cycler>=0.10
  Downloading cycler-0.11.0-py3-none-any.whl (6.4 kB)
Collecting pillow>=6.2.0
  Downloading Pillow-10.0.1-cp39-cp39-manylinux_2_28_aarch64.whl (3.5 MB)
                            3.5/3.5 MB
24.7 MB/s eta 0:00:0000:0100:01
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.9/site-packages (from matplotlib) (21.3)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.9/site-packages (from matplotlib) (3.0.7)
Collecting contourpy>=1.0.1
  Downloading
contourpy-1.1.1-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (286
kB)
                           286.4/286.4 KB
30.2 MB/s eta 0:00:00
Collecting fonttools>=4.22.0
  Downloading
fonttools-4.42.1-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (4.5
MB)
                            4.5/4.5 MB
14.1 MB/s eta 0:00:00a 0:00:01
Collecting kiwisolver>=1.0.1
  Downloading
kiwisolver-1.4.5-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (1.4
MB)
                            1.4/1.4 MB
28.0 MB/s eta 0:00:00a 0:00:01
Collecting patsy>=0.5.2
  Downloading patsy-0.5.3-py2.py3-none-any.whl (233 kB)
                           233.8/233.8 KB
29.3 MB/s eta 0:00:00
Collecting typing-extensions
  Downloading typing_extensions-4.8.0-py3-none-any.whl (31 kB)
Collecting pyerf
  Downloading pyerf-1.0.1-py3-none-any.whl (11 kB)
Requirement already satisfied: zipp>=3.1.0 in /opt/conda/lib/python3.9/site-
packages (from importlib-resources>=3.2.0->matplotlib) (3.7.0)
```

```
Requirement already satisfied: six in /opt/conda/lib/python3.9/site-packages
(from patsy>=0.5.2->statsmodels) (1.16.0)
Installing collected packages: pyerf, tzdata, typing-extensions, pillow, numpy,
kiwisolver, fonttools, cycler, scipy, scikits.bootstrap, patsy, pandas,
contourpy, statsmodels, matplotlib, seaborn
Successfully installed contourpy-1.1.1 cycler-0.11.0 fonttools-4.42.1
kiwisolver-1.4.5 matplotlib-3.8.0 numpy-1.26.0 pandas-2.1.1 patsy-0.5.3
pillow-10.0.1 pyerf-1.0.1 scikits.bootstrap-1.1.0 scipy-1.11.3 seaborn-0.12.2
statsmodels-0.14.0 typing-extensions-4.8.0 tzdata-2023.3
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy
from scipy import stats
import scipy
import helper.cliffs_delta as cliff
import statsmodels.api as sm
import itertools
import scikits.bootstrap as boot
```

1 Table of contents

2 Prepare Data

[3]:
```
sns.set(rc = {'figure.figsize':(10,10)})
confidence_level = 0.95

df = pd.read_csv('./data/energy.csv',sep=';',quotechar='"',names=["browser",␣
↪"language", "algorithm", "device", "energy"])
df['energy_total'] = df['energy'].astype(float)
df['implementation'] = numpy.where(df['language'] == 'js', 'js', 'wasm')

## unique values in columns
browsers = numpy.sort(df['browser'].unique())
browserpairs = list(itertools.combinations(browsers, 2))
languages = df['language'].unique()
implementations = numpy.sort(df['implementation'].unique())
implementationpairs = list(itertools.combinations(implementations, 2))
languagepairs = list(itertools.combinations(languages, 2))
```

```
algorithms = df['algorithm'].unique()
devices = df['device'].unique()

df.head()
```

[3]:

	browser	language	algorithm	device	energy	energy_total	implementation
0	firefox	c	fibonacci	SM-G991B	6.531442	6.531442	wasm
1	chrome	c	humblenumbers	SM-G991B	5.787359	5.787359	wasm
2	chrome	js	matrixmultiplication	SM-G991B	9.739240	9.739240	js
3	chrome	js	seqnonsquares	SM-G991B	4.516185	4.516185	js
4	chrome	js	nqueens	SM-G991B	5.935186	5.935186	js

2.1 Count samples

[4]:
```
data = []

for browser in browsers:
    for language in languages:
        for algorithm in algorithms:
            for device in devices:
                #print(browser,device,numpy.mean(df[(df['browser'] == browser)␣
 ↪& (df['language'] == language) & (df['algorithm'] == algorithm) &␣
 ↪(df['device'] == device)]['energy']))
                data.append(
                    [browser, language, algorithm, device, df[(df['browser'] ==␣
 ↪browser) & (df['language'] == language) & (df['algorithm'] == algorithm) &␣
 ↪(df['device'] == device)]['energy_total'].count()]
                )

# Create the pandas DataFrame
count = pd.DataFrame(data, columns = ['browser', 'language', 'algorithm',␣
 ↪'device', 'count'])
display(count)
```

	browser	language	algorithm	device	count
0	chrome	c	fibonacci	SM-G991B	30
1	chrome	c	fibonacci	Nexus 5	32
2	chrome	c	humblenumbers	SM-G991B	31
3	chrome	c	humblenumbers	Nexus 5	34
4	chrome	c	matrixmultiplication	SM-G991B	31

```
..       …       …                    …        …    …
147  firefox       js          happynumbers   Nexus 5     31
148  firefox       js           pancakesort  SM-G991B     31
149  firefox       js           pancakesort   Nexus 5     33
150  firefox       js             quicksort  SM-G991B     32
151  firefox       js             quicksort   Nexus 5     33

[152 rows x 5 columns]
```

2.2 Calculations

[5]:
```
data = []
for implementation in implementations:
    x = df[(df['implementation'] == implementation)]
    cnt = x.size
    sum = numpy.round(numpy.sum(x['energy']), 2)

    data.append(
        [implementation, cnt, sum]
    )

# Create the pandas DataFrame
stat = pd.DataFrame(data, columns = ['implementation', 'count', 'sum'])
print(stat.to_string())
```

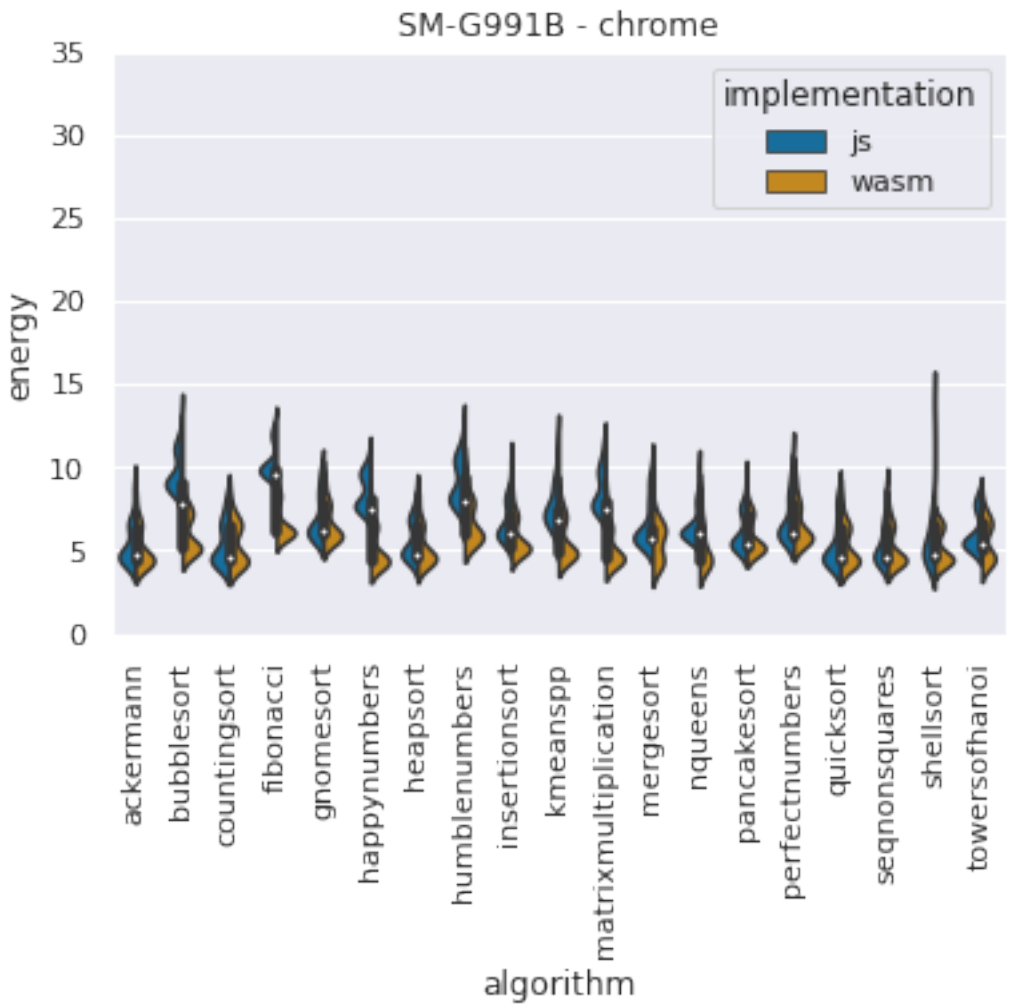
```
  implementation  count       sum
0             js  16877  24490.76
1           wasm  16905  18937.81
```

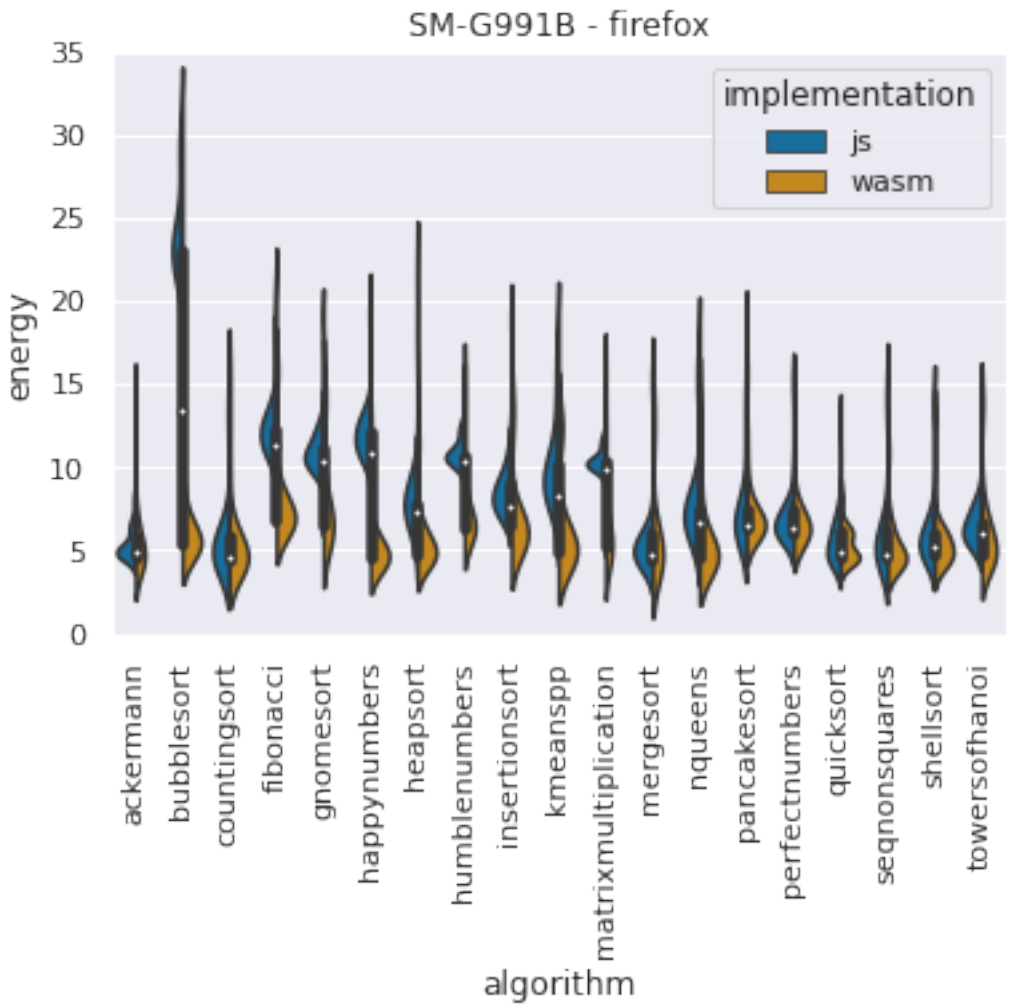
3 Violinplot (Algorithms)

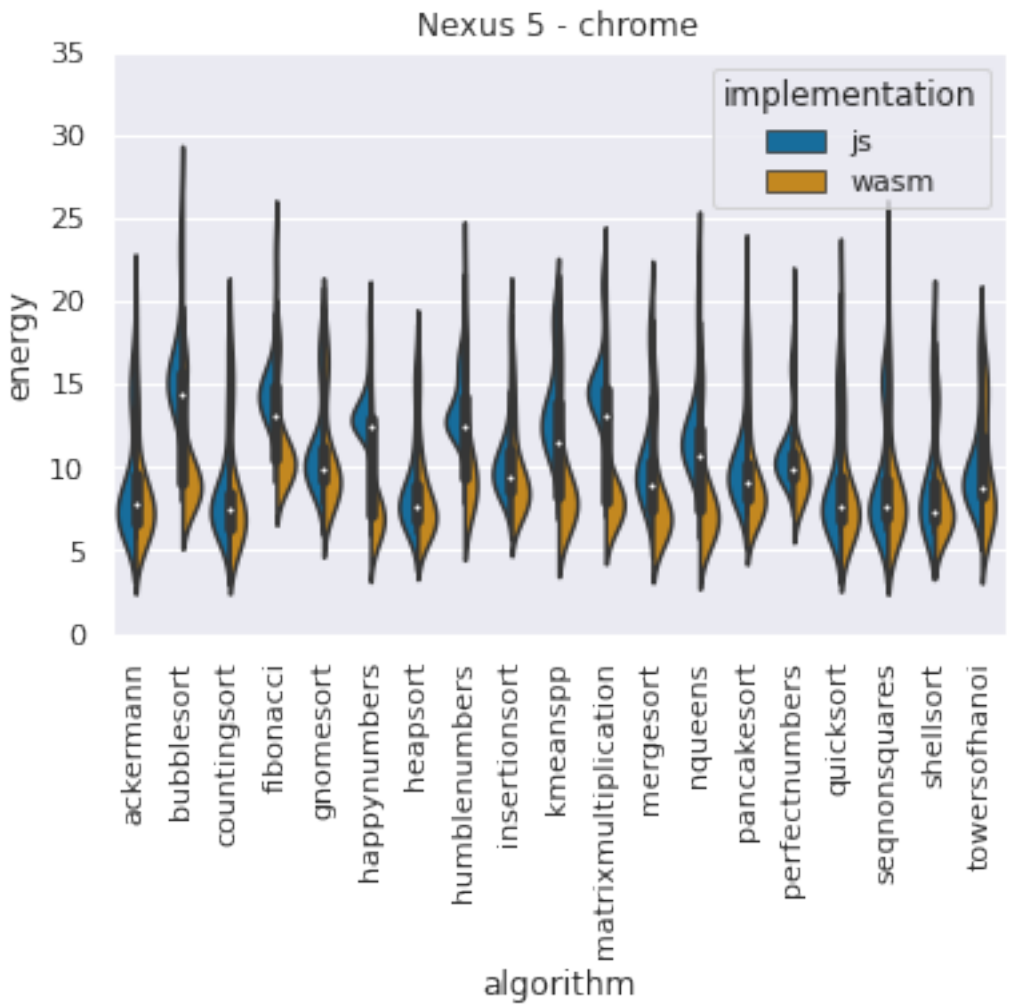
[6]:
```
for device in devices:
    for browser in browsers:
        data = df[(df['device'] == device) & (df['browser'] == browser)].
 ↪sort_values(by=['algorithm'])
        plt.figure()
        plt.xticks(rotation=90)
        plt.ylim(0, 35)
        sns.violinplot(x='algorithm', y='energy', hue='implementation',␣
 ↪hue_order=implementations, data=data, palette='colorblind', split=True).
 ↪set_title(device + " - " + browser)
```

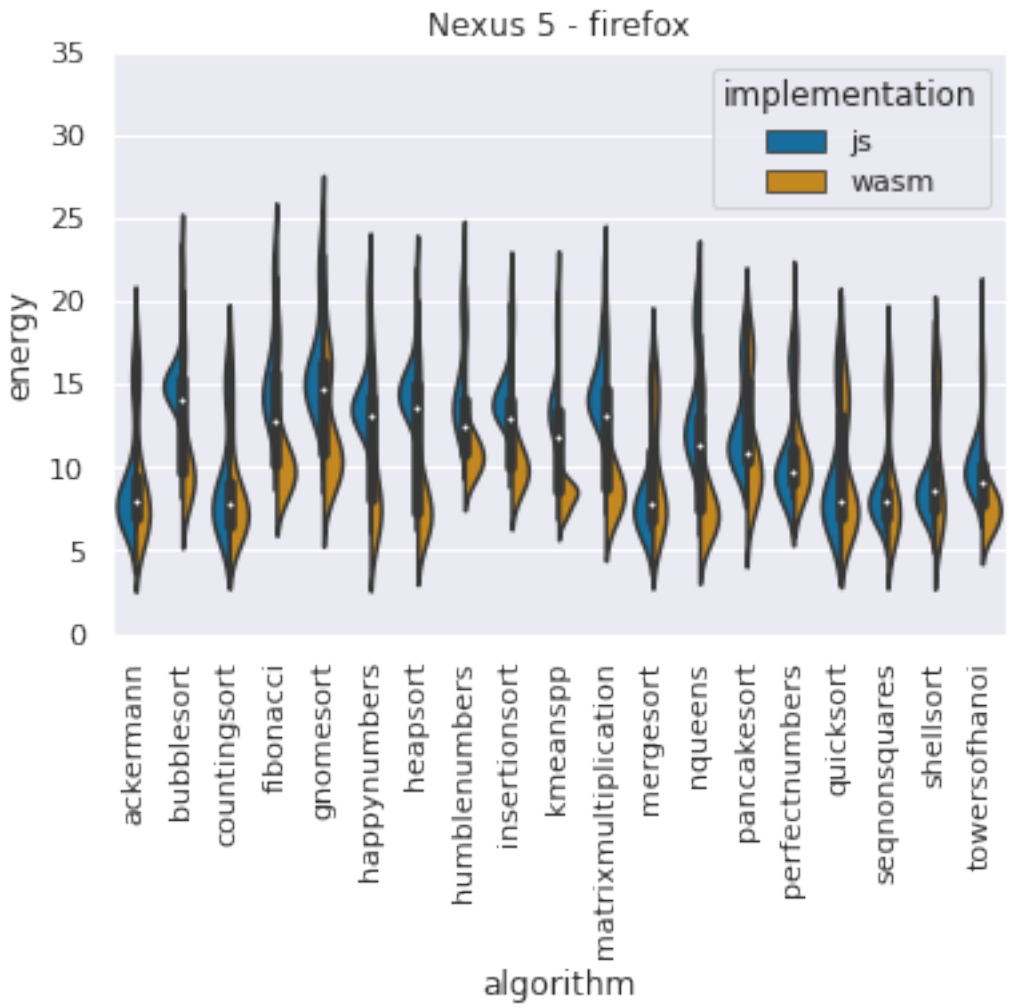
```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
```

```
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```


SM-G991B - chrome
implementation
js
wasm
energy
0
5
10
15
20
25
30
35
ackermann
bubblesort
countingsort
fibonacci
gnomesort
happynumbers
heapsort
humblenumbers
insertionsort
kmeanspp
matrixmultiplication
mergesort
nqueens
pancakesort
perfectnumbers
quicksort
seqnonsquares
shellsort
towersofhanoi
algorithm


SM-G991B - firefox
implementation
js
wasm
energy
0
5
10
15
20
25
30
35
ackermann
bubblesort
countingsort
fibonacci
gnomesort
happynumbers
heapsort
humblenumbers
insertionsort
kmeanspp
matrixmultiplication
mergesort
nqueens
pancakesort
perfectnumbers
quicksort
seqnonsquares
shellsort
towersofhanoi
algorithm


Nexus 5 - chrome
implementation
js
wasm
energy
0
5
10
15
20
25
30
35
ackermann
bubblesort
countingsort
fibonacci
gnomesort
happynumbers
heapsort
humblenumbers
insertionsort
kmeanspp
matrixmultiplication
mergesort
nqueens
pancakesort
perfectnumbers
quicksort
seqnonsquares
shellsort
towersofhanoi
algorithm



4 RQ1: JavaScript vs. WebAssembly

4.1 Shapiro Wilk Test

[7]:
```
data = []
non_normal = 0

for implementation in implementations:
    energy = df[(df['implementation'] == implementation)]['energy']

    if len(energy) >= 3:
        shapiro_test = stats.shapiro(energy)

        non_normal += (1 if shapiro_test.pvalue <= 0.05 else 0)
```

```
        data.append(
            [implementation,
             shapiro_test.statistic,
             shapiro_test.pvalue
            ]
        )

# Create the pandas DataFrame
swt = pd.DataFrame(data, columns = ['implementation', 'w', 'p'])
#print(swt.to_string())
display(swt)

print("\n{} non-normally distributed samples".format(non_normal))
print("{} normally distributed samples".format(len(swt) - non_normal))
print("{:.2f}% non-normally distributed samples".format(non_normal/
 ↪len(swt)*100))
```

	implementation	w	p
0	js	0.927419	1.147179e-32
1	wasm	0.863110	1.110529e-41

```
2 non-normally distributed samples
0 normally distributed samples
100.00% non-normally distributed samples
```

4.2 Shapiro Wilk Test (By Device)

[8]:
```
data = []
non_normal = 0

for device in devices:
    for implementation in implementations:
        energy = df[(df['implementation'] == implementation) & (df['device'] ==␣
 ↪device)]['energy']

        if len(energy) >= 3:
            shapiro_test = stats.shapiro(energy)

            non_normal += (1 if shapiro_test.pvalue <= 0.05 else 0)

            data.append(
                [
                 device,
                 implementation,
                 shapiro_test.statistic,
                 shapiro_test.pvalue
```

```
                ]
            )

# Create the pandas DataFrame
swt = pd.DataFrame(data, columns = ['device', 'implementation', 'w', 'p'])
#print(swt.to_string())
display(swt)

print("\n{} non-normally distributed samples".format(non_normal))
print("{} normally distributed samples".format(len(swt) - non_normal))
print("{:.2f}% non-normally distributed samples".format(non_normal/
 ↪len(swt)*100))
```

	device	implementation	w	p
0	SM-G991B	js	0.771749	1.073380e-37
1	SM-G991B	wasm	0.747010	3.835847e-39
2	Nexus 5	js	0.960188	8.760830e-18
3	Nexus 5	wasm	0.857640	6.419627e-32

```
4 non-normally distributed samples
0 normally distributed samples
100.00% non-normally distributed samples
```

4.3 Mann-Whitney-U-Test

[9]:
```
data = []

for implementationpair in implementationpairs:
    impl1_energy = df[(df['implementation'] == implementationpair[0])]['energy']
    impl2_energy = df[(df['implementation'] == implementationpair[1])]['energy']
    eff = cliff.cliffs_delta(impl1_energy, impl2_energy)

    u = stats.mannwhitneyu(impl1_energy, impl2_energy, alternative='two-sided')

    data.append(
        [
         implementationpair[0] + ' vs. ' + implementationpair[1],
         u.statistic,
         u.pvalue,
         eff[0],
         eff[1]
        ]
    )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns = ['implementation', 'u', 'p', 'eff', 'interp'])
display(ut)
```

```
interp = ut['interp'].value_counts()
interp = pd.DataFrame(interp, columns = ['interp', 'percent'])
interp['percent'] = (interp['interp'] / interp['interp'].sum()) * 100

display(interp)
```

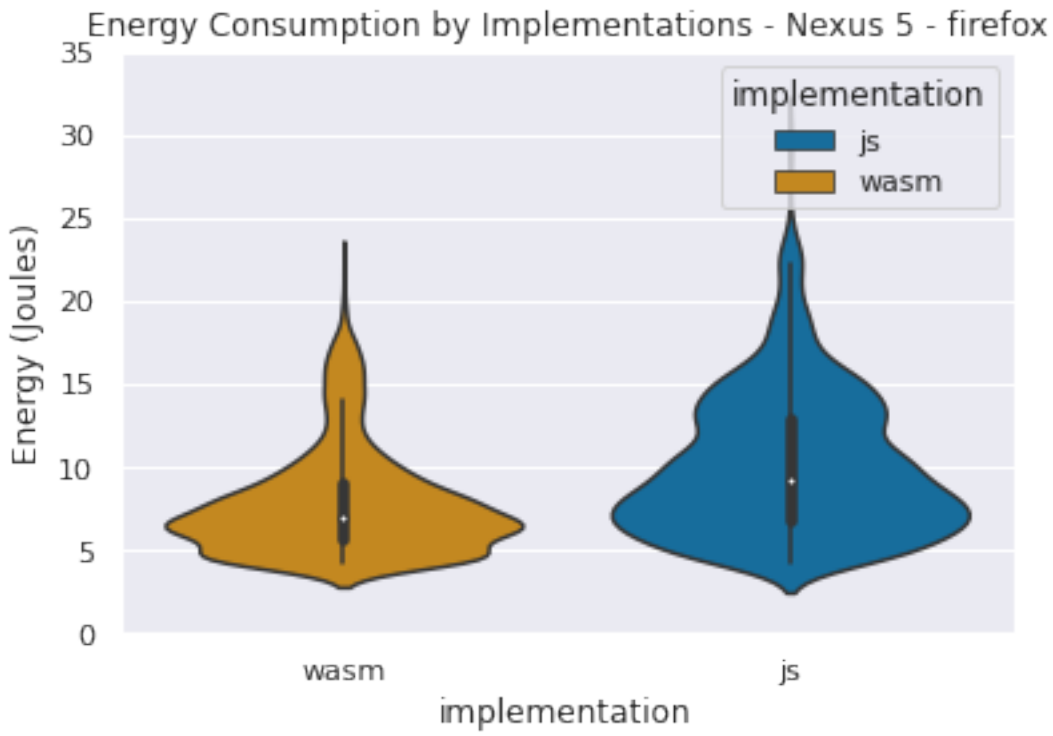
```
  implementation          u             p       eff  interp
0    js vs. wasm  3904820.0  1.173223e-93  0.341271  medium
```

```
Empty DataFrame
Columns: [interp, percent]
Index: []
```

4.4 Violinplot

[10]:
```
data = df
plt.ylim(0, 35)
vp = sns.violinplot(x='implementation', y='energy', hue='implementation',␣
 ↪hue_order=implementations, data=data, palette='colorblind' ,dodge=False)
vp.set_title("Energy Consumption by Implementations - " + device + " - " +␣
 ↪browser)
vp.set_ylabel("Energy (Joules)")
plt.show()
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

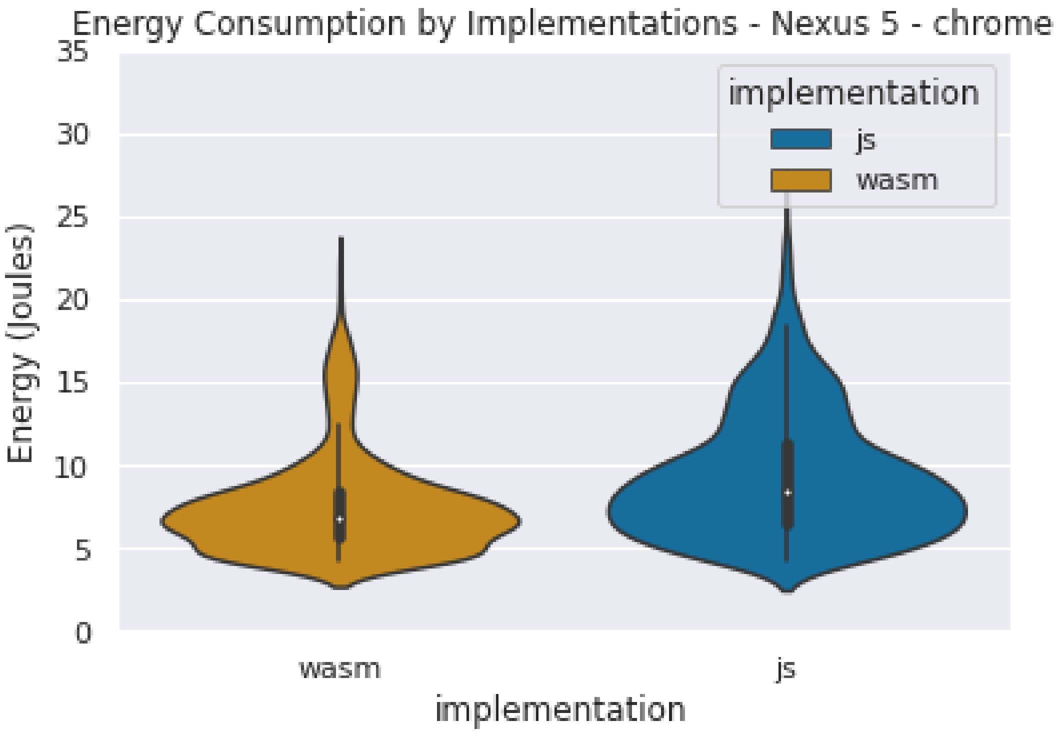


4.5 Violinplot (By Browser)

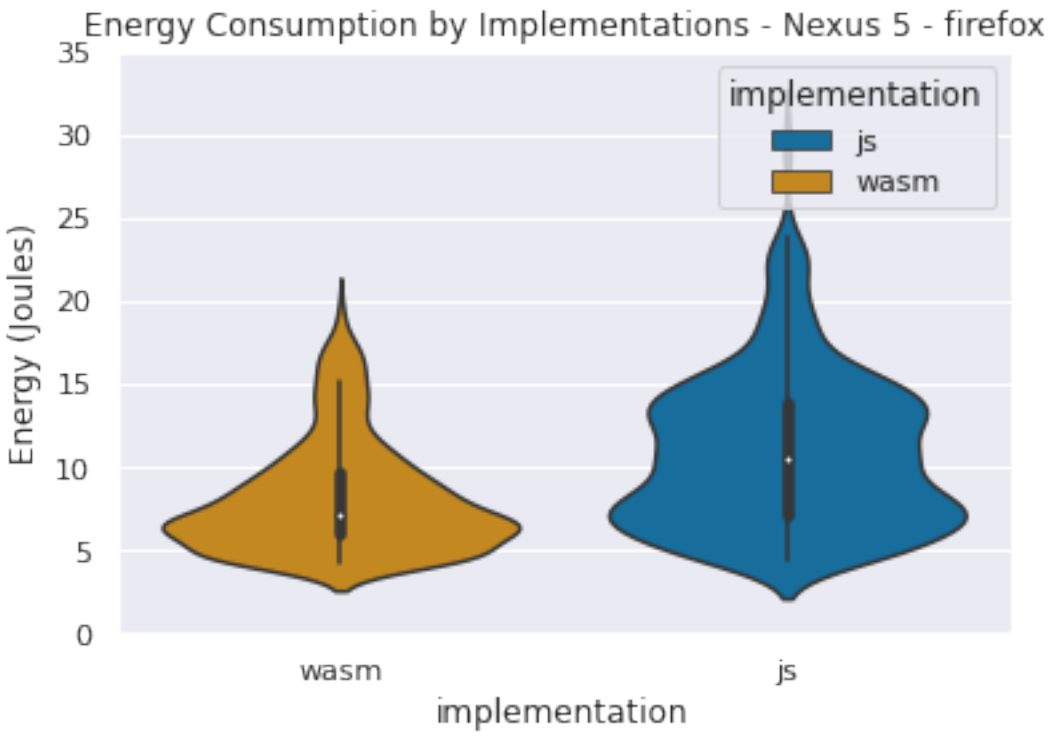
[11]:
```
data = []
for browser in browsers:
    data = df[(df['browser'] == browser)]
    plt.ylim(0, 35)
    vp = sns.violinplot(x='implementation', y='energy', hue='implementation',␣
 ↪hue_order=implementations, data=data, palette='colorblind' ,dodge=False)
    vp.set_title("Energy Consumption by Implementations - " + device + " - " +␣
 ↪browser)
    vp.set_ylabel("Energy (Joules)")
    plt.show()
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
```

```
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```




```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

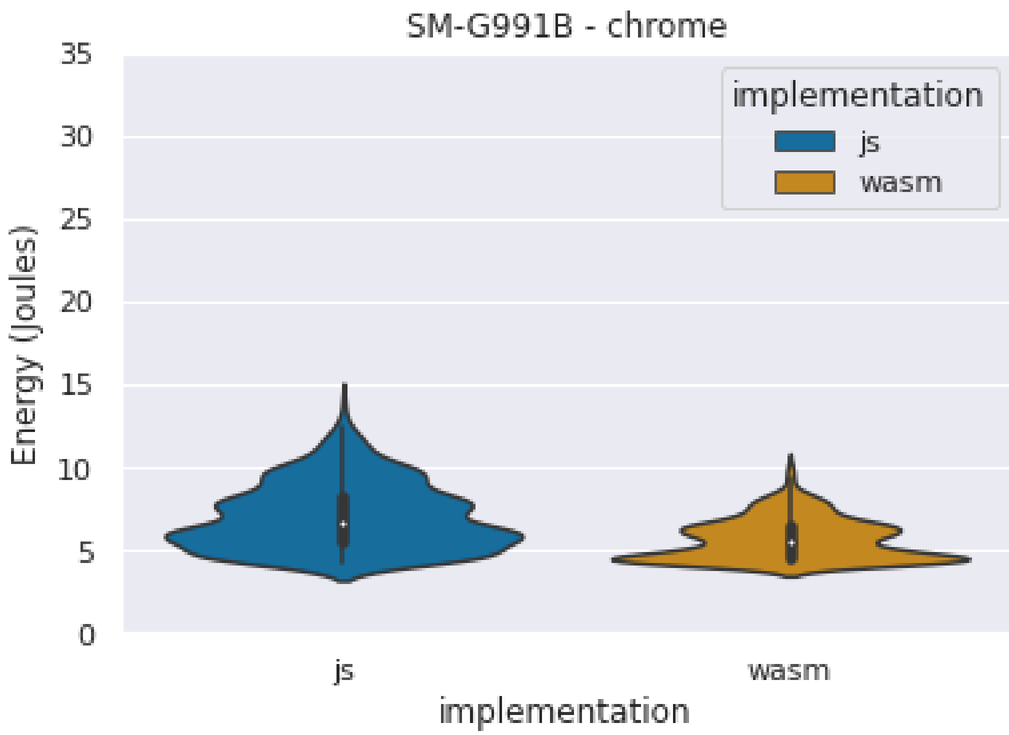


4.6 Violinplot (By Browser & By Device)

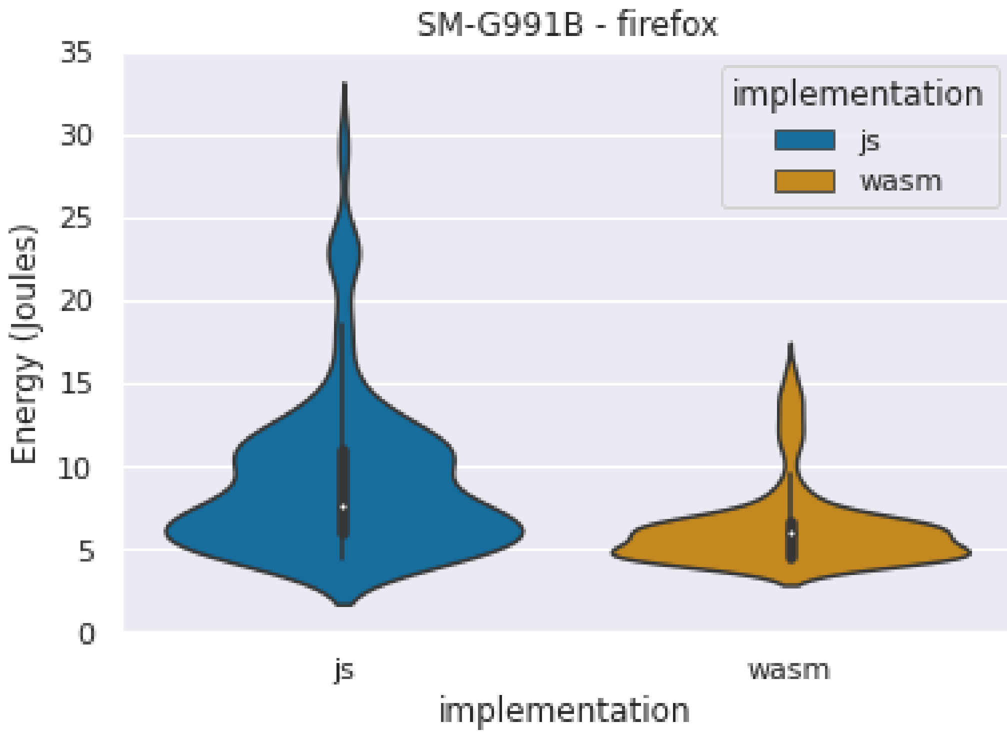
[12]:
```
data = []
for device in devices:
    for browser in browsers:
        data = df[(df['browser'] == browser) & (df['device'] == device)].
 ↪sort_values(by=['implementation'])
        plt.ylim(0, 35)
        vp = sns.violinplot(x='implementation', y='energy',␣
 ↪hue='implementation', hue_order=implementations, data=data,␣
 ↪palette='colorblind', dodge=False)
        vp.set_title(device + " - " + browser)
        vp.set_ylabel("Energy (Joules)")
        plt.show()
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

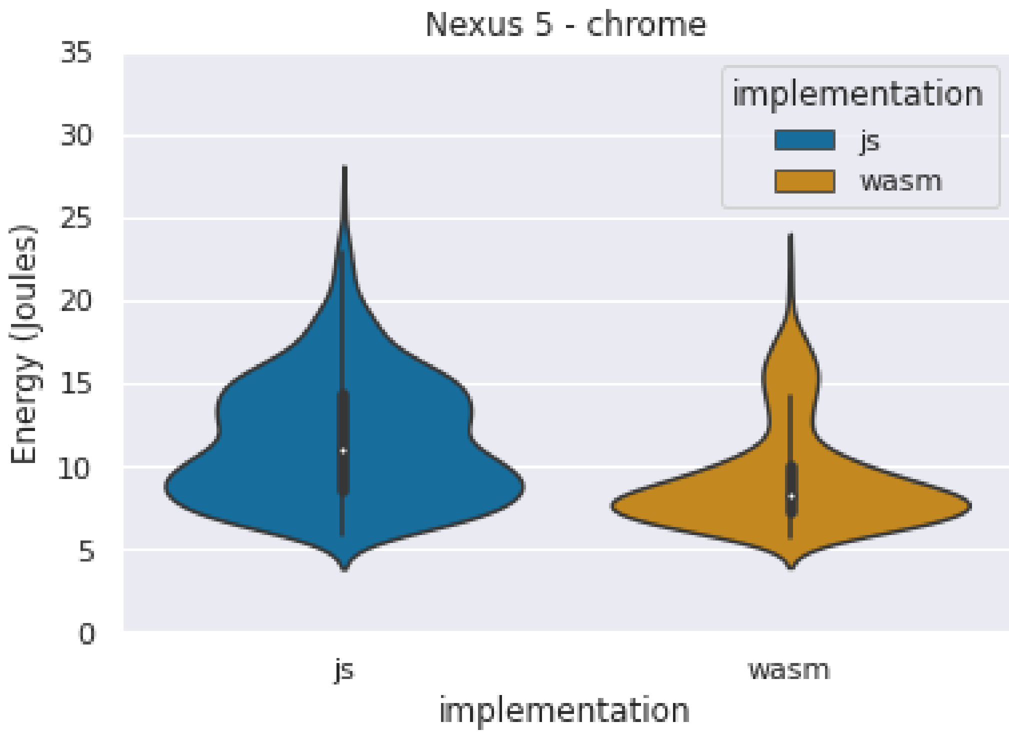
```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



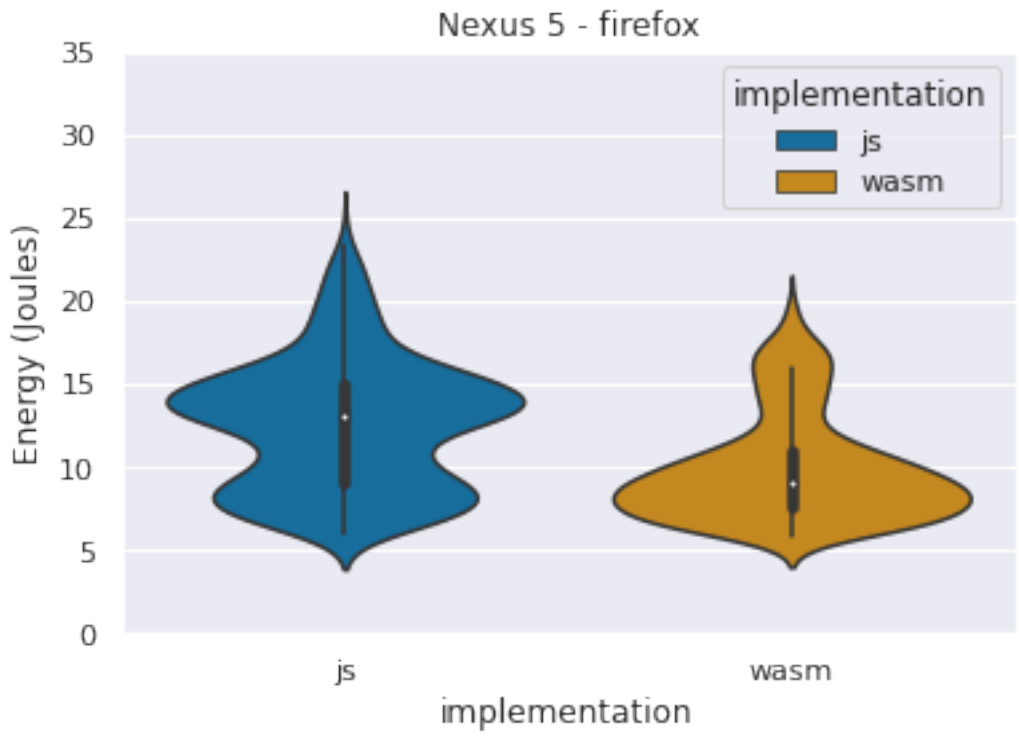

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```




```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



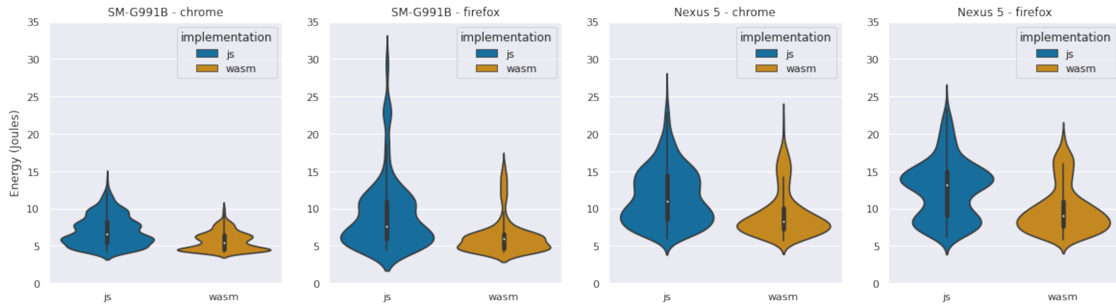

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



[13]:
```
data = []
index=0
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

for device in devices:
    for browser in browsers:
        data = df[(df['browser'] == browser) & (df['device'] == device)].
 ↪sort_values(by=['implementation'])
        vp = sns.violinplot(x='implementation', y='energy',␣
 ↪hue='implementation', hue_order=implementations, data=data,␣
 ↪palette='colorblind', dodge=False, ax=axes[index])
        vp.set_title(device + " - " + browser)
        axes[index].set_ylim(0, 35)
        axes[index].set_xlabel("")
        if index == 0:
            axes[0].set_ylabel("Energy (Joules)")
        else:
            axes[index].set_ylabel("")
        index+=1
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
```

```
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

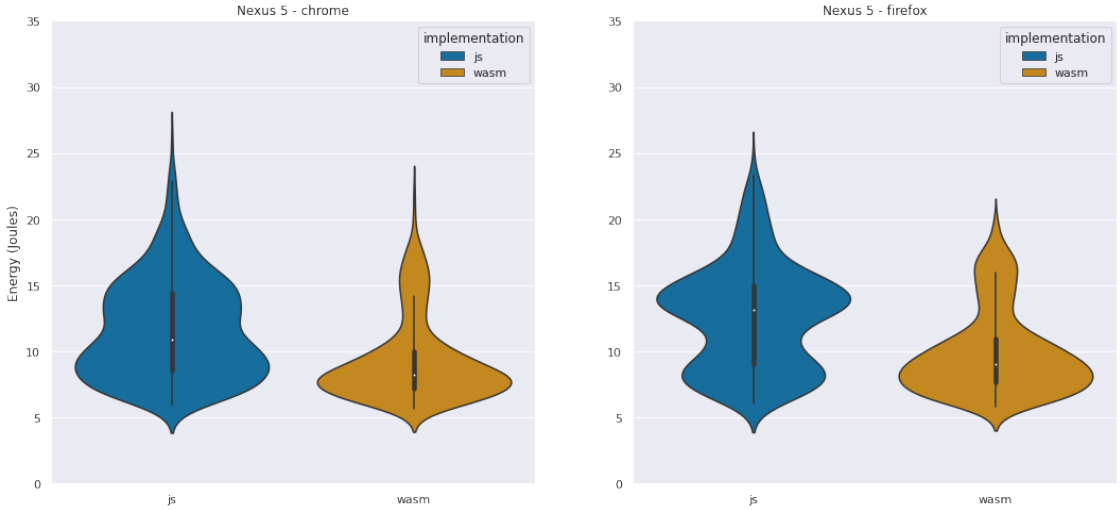


4.7 Violinplot (By Browser & Low End Device)

[14]:
```
data = []

index=0
fig, axes = plt.subplots(1, 2, figsize=(18, 8))
for browser in browsers:
    data = df[(df['browser'] == browser) & (df['device'] == 'Nexus 5')].
 ↪sort_values(by=['implementation'])
    vp = sns.violinplot(x='implementation', y='energy', hue='implementation',␣
 ↪hue_order=implementations, data=data, palette='colorblind', dodge=False,␣
 ↪ax=axes[index])
    vp.set_title("Nexus 5 - " + browser)
    axes[index].set_ylim(0, 35)
    axes[index].set_xlabel("")
    if index == 0:
        axes[0].set_ylabel("Energy (Joules)")
    else:
        axes[index].set_ylabel("")
    index+=1
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
```

```
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

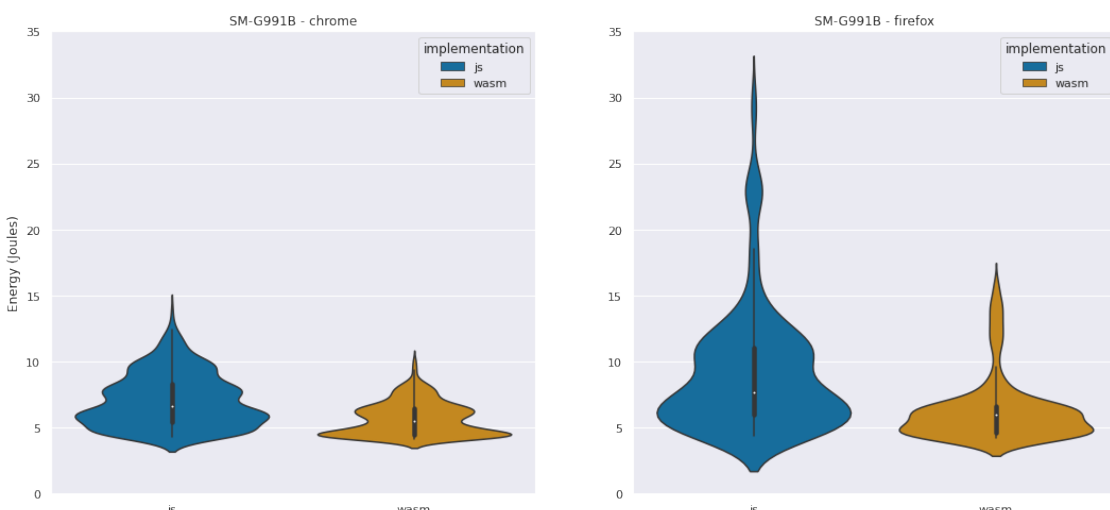


4.8 Violinplot (By Browser & High End Device)

[15]:
```
data = []

index=0
fig, axes = plt.subplots(1, 2, figsize=(18, 8))
for browser in browsers:
    data = df[(df['browser'] == browser) & (df['device'] == 'SM-G991B')].
 ↪sort_values(by=['implementation'])
    vp = sns.violinplot(x='implementation', y='energy', hue='implementation',␣
 ↪hue_order=implementations, data=data, palette='colorblind', dodge=False,␣
 ↪ax=axes[index])
    vp.set_title("SM-G991B - " + browser)
    axes[index].set_ylim(0, 35)
    axes[index].set_xlabel("")
    if index == 0:
        axes[0].set_ylabel("Energy (Joules)")
    else:
        axes[index].set_ylabel("")
```

```python
    index+=1
        #plt.show()
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



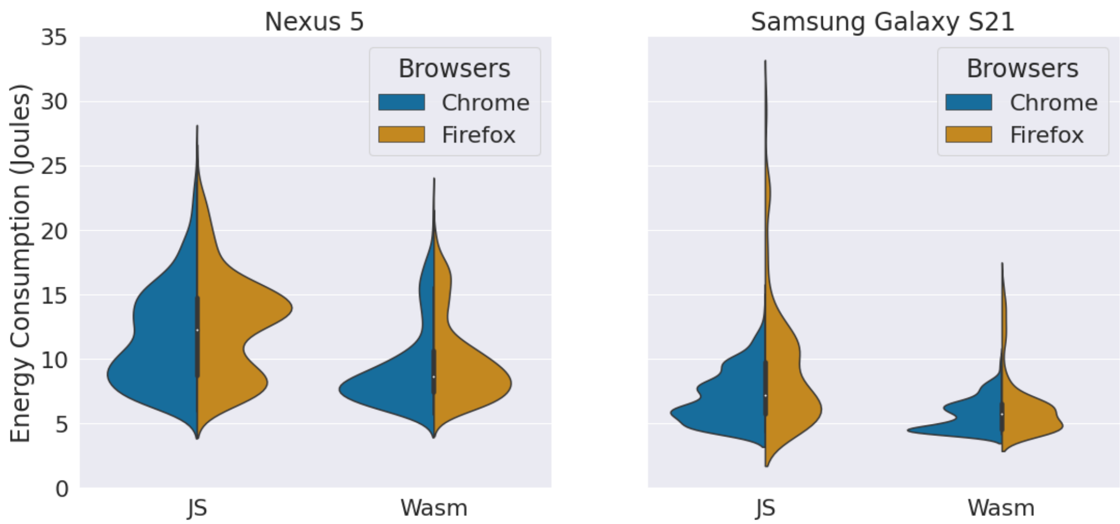
4.9 Violinplot (By Browser & Device Types)

[16]:
```
## Prepare naming of values for final violinplot
df_violin = df.copy()
df_violin['implementation'] = df_violin['implementation'].replace(['wasm'],␣
 ↪'Wasm')
df_violin['implementation'] = df_violin['implementation'].replace(['js'], 'JS')
df_violin['browser'] = df_violin['browser'].replace(['chrome'], 'Chrome')
df_violin['browser'] = df_violin['browser'].replace(['firefox'], 'Firefox')
df_violin['device'] = df_violin['device'].replace(['SM-G991B'], 'Samsung Galaxy␣
 ↪S21')
browsers_violin = numpy.sort(df_violin['browser'].unique())
devices_violin = numpy.sort(df_violin['device'].unique())
```

[17]:
```
sns.set(font_scale=2)
data = []
index=0
fig, axes = plt.subplots(1, 2, figsize=(18, 8))
for device in devices_violin:
    data = df_violin[(df_violin['device'] == device)].
 ↪sort_values(by=['implementation'])
    vp = sns.violinplot(x='implementation', y='energy', hue='browser',␣
 ↪hue_order=browsers_violin, data=data, palette='colorblind', dodge=False,␣
 ↪ax=axes[index], split=True)
    vp.set_title(device)
    axes[index].legend(title="Browsers")
    axes[index].set_ylim(0, 35)
    axes[index].set_xlabel("")
    axes[index].set_ylabel("Energy Consumption (Joules)")
    if index == 0:
        axes[0].set_ylabel("Energy Consumption (Joules)")
    else:
        axes[index].set_ylabel("")
        axes[index].set_yticklabels([])
    index+=1
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
```

```
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



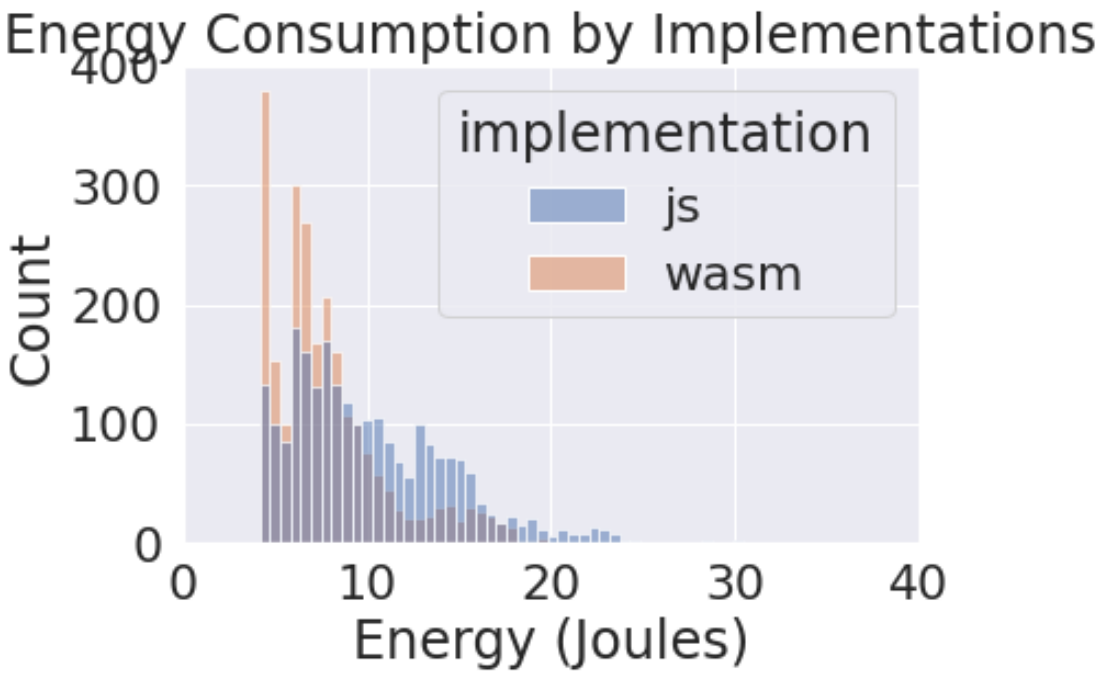
4.10 Histogram

[18]:
```python
sns.histplot(data=df, x="energy", hue="implementation",␣
  ↪hue_order=implementations).set_title("Energy Consumption by Implementations")
plt.xlabel("Energy (Joules)")
plt.ylim(0, 400)
plt.xlim(0, 40)
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

[18]: (0.0, 40.0)

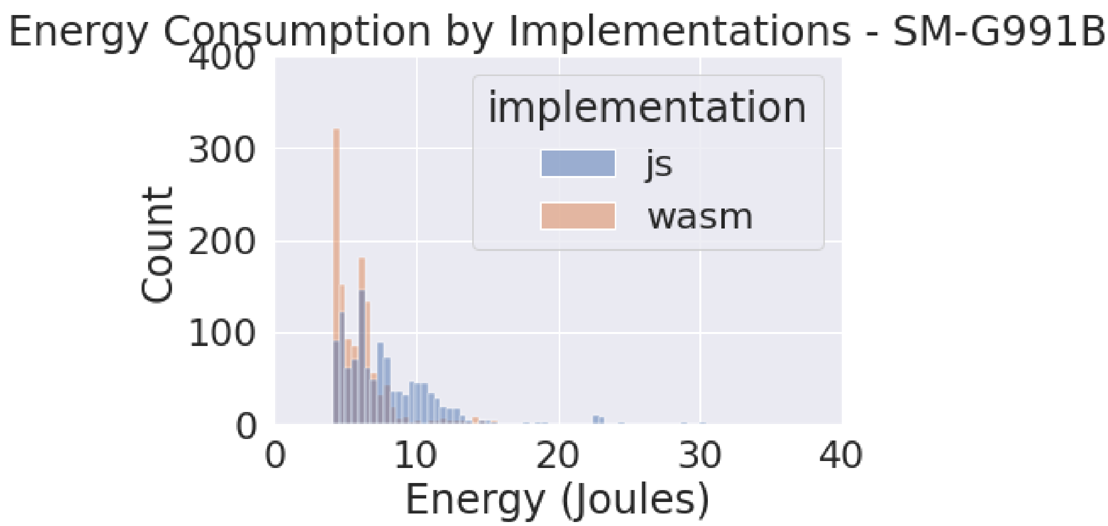


4.11 Histogram (By Device)

[19]:
```
data = []
for device in devices:
    data = df[(df['device'] == device)]
    plt.xlabel("Energy (Joules)")
    plt.ylim(0, 400)
    plt.xlim(0, 40)
    sns.histplot(data=data, x="energy", hue="implementation",␣
 ↪hue_order=implementations).set_title("Energy Consumption by Implementations␣
 ↪- " + device)
    plt.show()
```

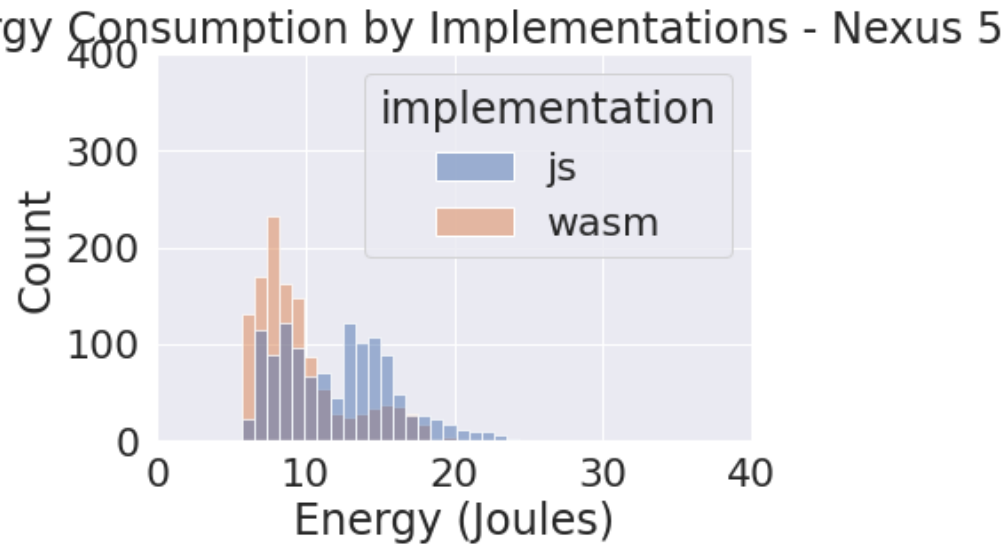
```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
```

```
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



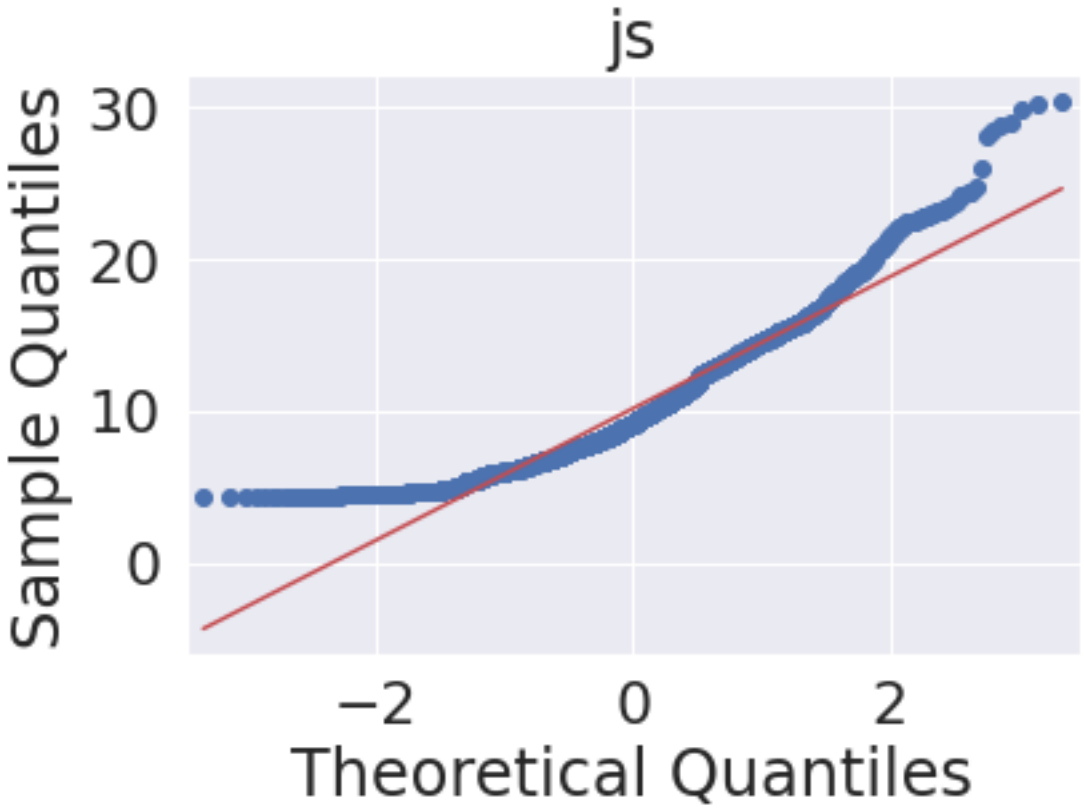

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
```

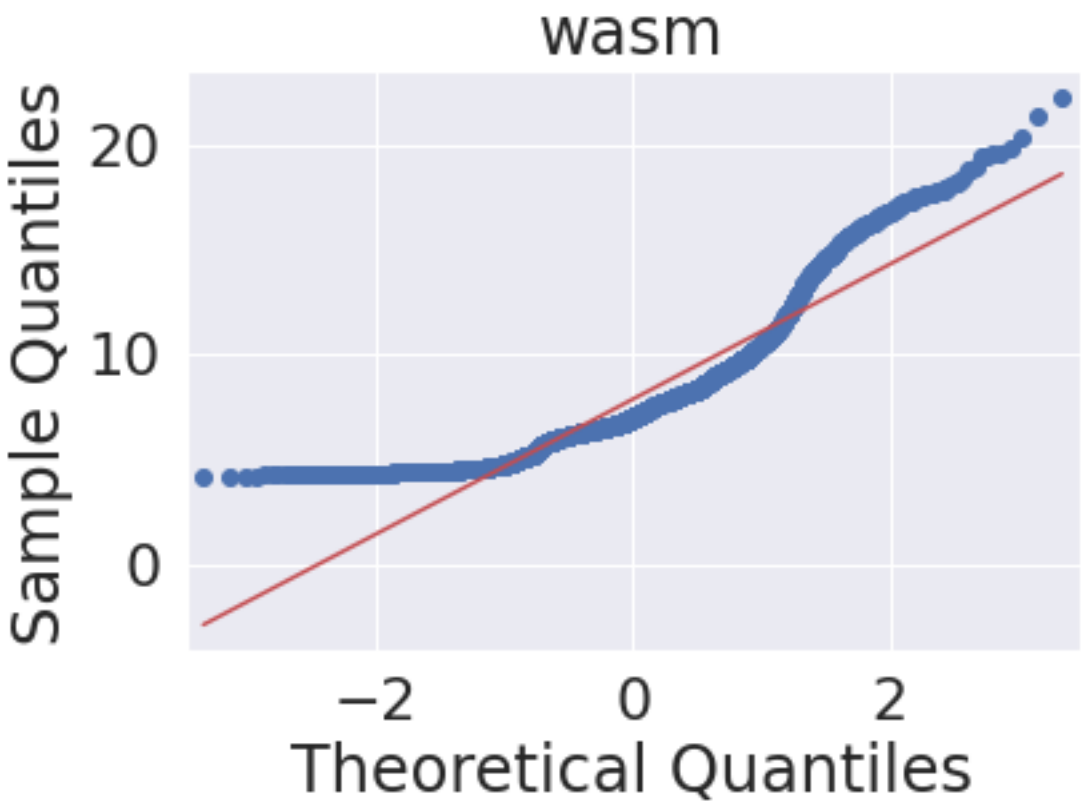
```
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



4.12 Q-Q-Plot

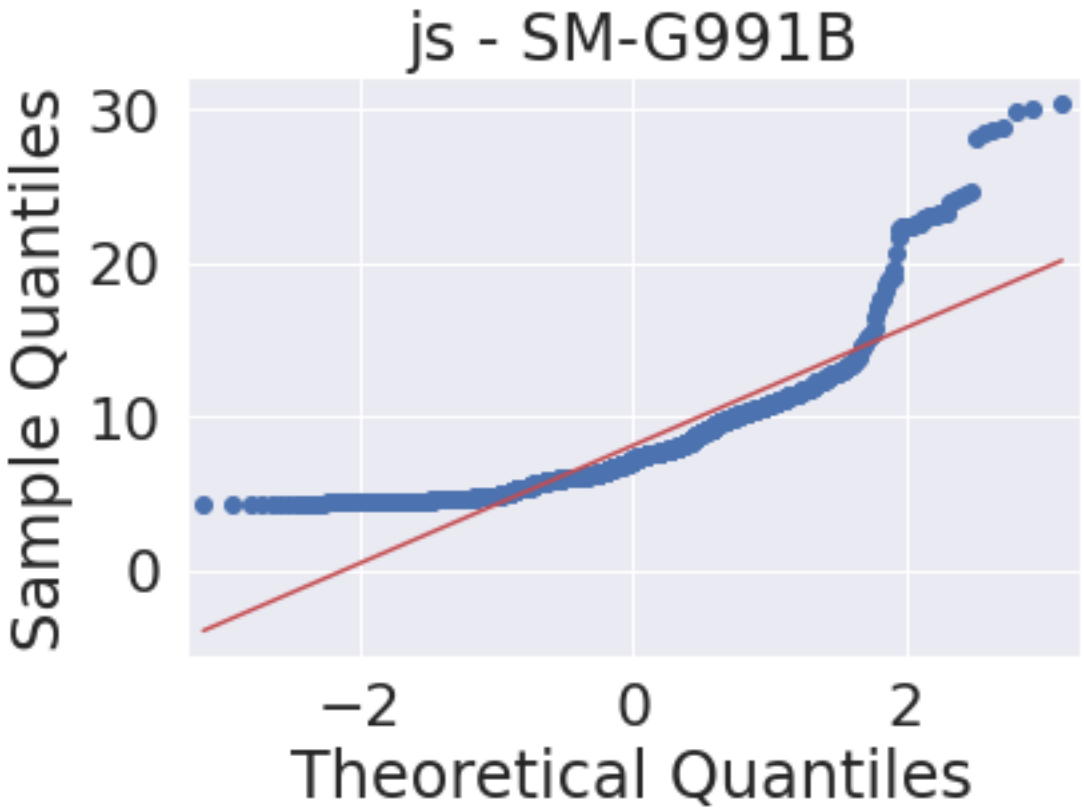
[20]:
```
data = []
for implementation in implementations:
    data = df[(df['implementation'] == implementation)]
    qq = sm.qqplot(data.energy, line='s')
    h = plt.title(implementation)
```

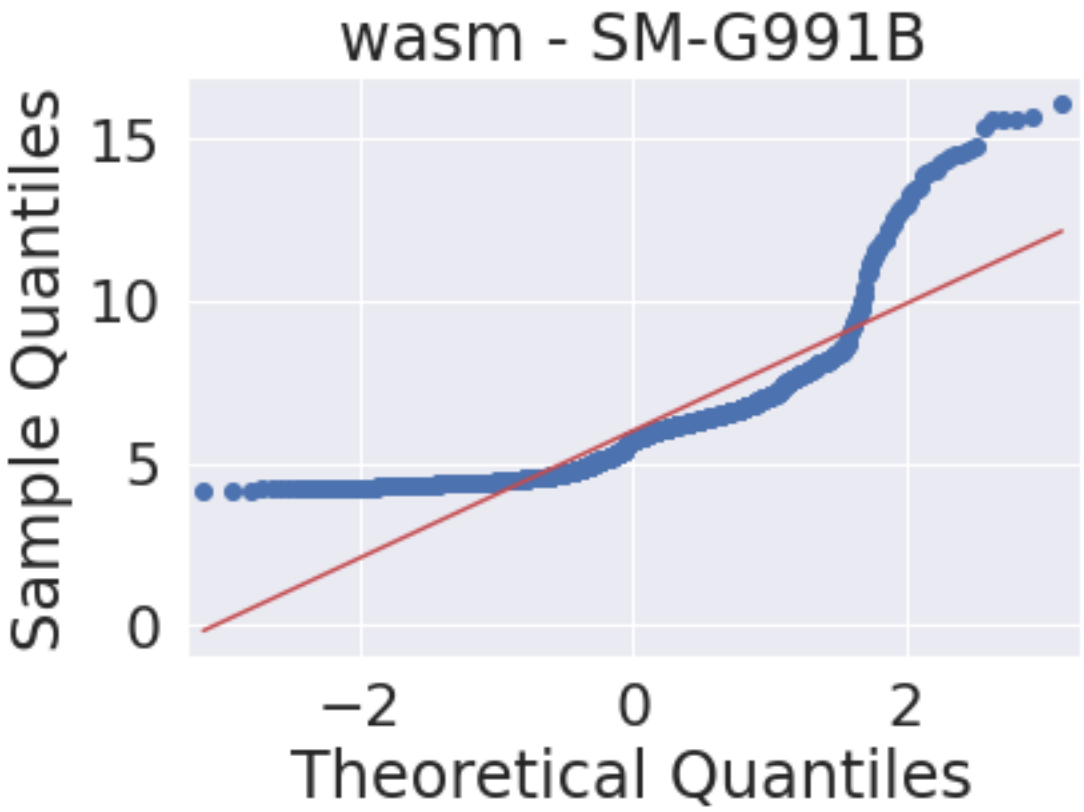

js
Sample Quantiles
30
20
10
0
−2
0
2
Theoretical Quantiles

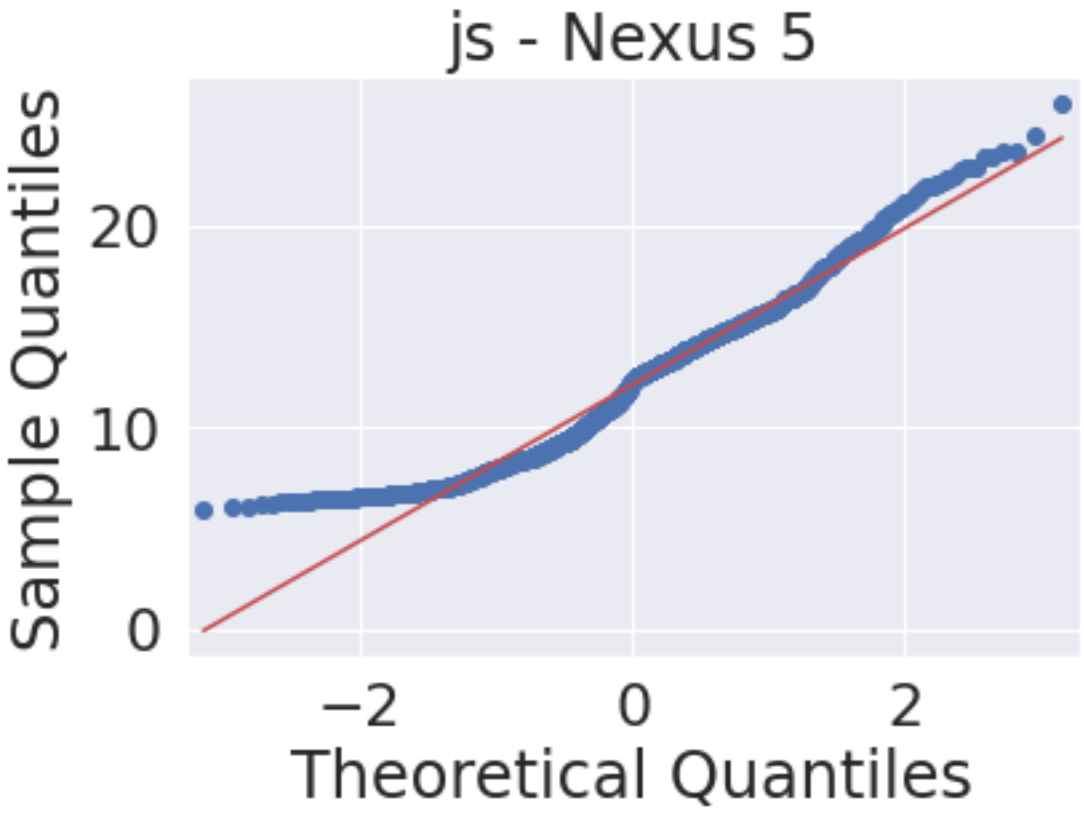


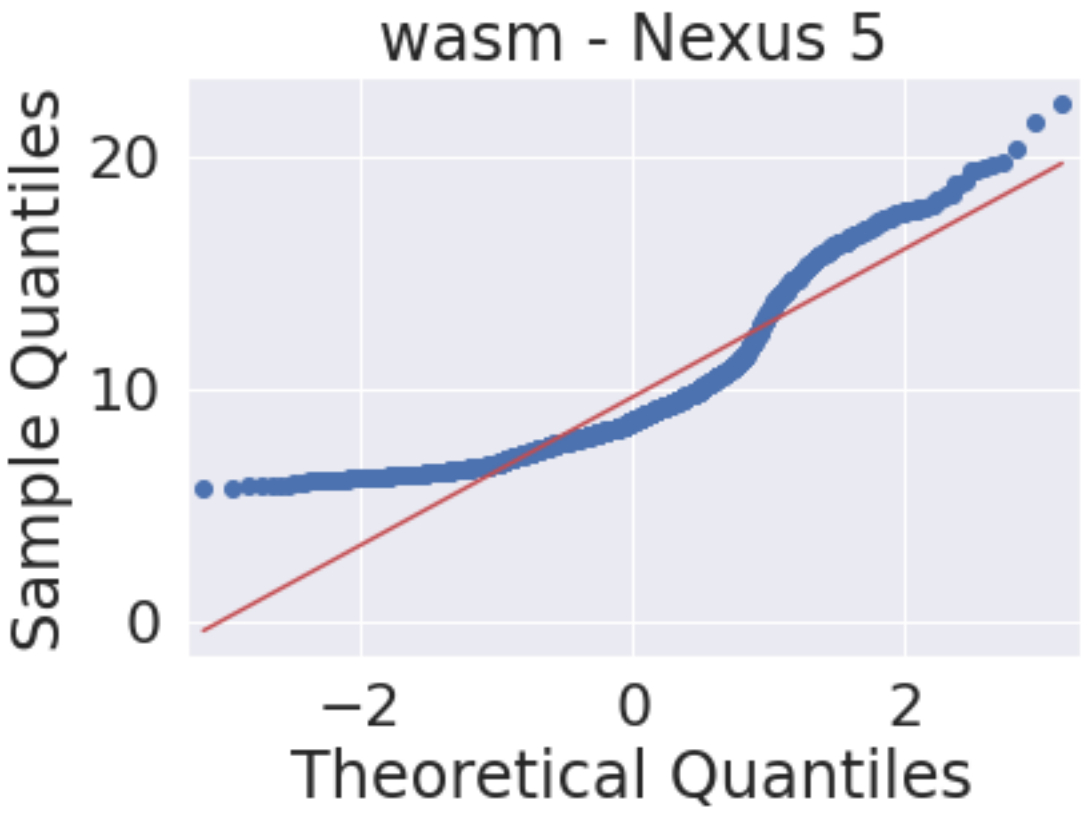
4.13 Q-Q-plot (By Device)

[21]:
```
data = []
for device in devices:
    for implementation in implementations:
        data = df[(df['implementation'] == implementation) & (df['device'] ==␣
 ↪device)]
        qq = sm.qqplot(data.energy, line='s')
        h = plt.title(implementation + " - " + device)
```


js - SM-G991B
Sample Quantiles
30
20
10
0
−2
0
2
Theoretical Quantiles


wasm - SM-G991B
Sample Quantiles
15
10
5
0
−2
0
2
Theoretical Quantiles


js - Nexus 5
Sample Quantiles
20
10
0
−2
0
2
Theoretical Quantiles



4.14 Mann Whitney U Test (same Browsers)

[22]:
```
data = []

for browser in browsers:
    for implementationpair in implementationpairs:
        impl1_energy = df[(df['implementation'] == implementationpair[0]) &␣
↪(df['browser'] == browser)]['energy']
        impl2_energy = df[(df['implementation'] == implementationpair[1]) &␣
↪(df['browser'] == browser)]['energy']
        eff = cliff.cliffs_delta(impl1_energy, impl2_energy)

        u = stats.mannwhitneyu(impl1_energy, impl2_energy,␣
↪alternative='two-sided')

        data.append(
            [
             browser,
             implementationpair[0] + ' vs. ' + implementationpair[1],
             u.statistic,
```

```
            u.pvalue,
            eff[0],
            eff[1]
           ]
        )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns = ['browser', 'implementation', 'u', 'p',␣
 ↪'eff', 'interp'])
display(ut)

interp = ut['interp'].value_counts()
interp = pd.DataFrame(interp, columns = ['interp', 'percent'])
interp['percent'] = (interp['interp'] / interp['interp'].sum()) * 100

display(interp)
```

	browser	implementation	u	p	eff	interp
0	chrome	js vs. wasm	960697.0	1.168028e-41	0.317780	small
1	firefox	js vs. wasm	998597.0	5.082446e-57	0.374314	medium

```
Empty DataFrame
Columns: [interp, percent]
Index: []
```

4.15 Mann Whitney U Test (same Browsers - by Device)

[23]:
```
data = []

for device in devices:
    for browser in browsers:
        for implementationpair in implementationpairs:
            impl1_energy = df[(df['implementation'] == implementationpair[0]) &␣
 ↪(df['browser'] == browser) & (df['device'] == device)]['energy']
            impl2_energy = df[(df['implementation'] == implementationpair[1]) &␣
 ↪(df['browser'] == browser) & (df['device'] == device)]['energy']
            eff = cliff.cliffs_delta(impl1_energy, impl2_energy)

            u = stats.mannwhitneyu(impl1_energy, impl2_energy,␣
 ↪alternative='two-sided')

            data.append(
                [
                 device,
                 browser,
                 implementationpair[0] + ' vs. ' + implementationpair[1],
                 u.statistic,
```

```
                u.pvalue,
                eff[0],
                eff[1]
               ]
            )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns = ['device', 'browser', 'implementation', 'u',␣
 ↪'p', 'eff', 'interp'])
display(ut)

interp = ut['interp'].value_counts()
interp = pd.DataFrame(interp, columns = ['interp', 'percent'])
interp['percent'] = (interp['interp'] / interp['interp'].sum()) * 100

display(interp)
```

	device	browser	implementation	u	p	eff	interp
0	SM-G991B	chrome	js vs. wasm	250266.0	1.591990e-36	0.423401	medium
1	SM-G991B	firefox	js vs. wasm	256565.0	1.160505e-42	0.459227	medium
2	Nexus 5	chrome	js vs. wasm	263410.0	3.872448e-33	0.395187	medium
3	Nexus 5	firefox	js vs. wasm	261860.0	3.617754e-33	0.396013	medium

```
Empty DataFrame
Columns: [interp, percent]
Index: []
```

4.16 Mann Whitney U Test (Cross Browsers)

[24]:
```
data = []

for pairswitch in [[0,1],[1,0]]:
    for implementationpair in implementationpairs:
        for browserpair in browserpairs:
            browser1_energy = df[(df['browser'] == browserpair[0]) &␣
 ↪(df['implementation'] == implementationpair[pairswitch[0]])]['energy']
            browser2_energy = df[(df['browser'] == browserpair[1]) &␣
 ↪(df['implementation'] == implementationpair[pairswitch[1]])]['energy']
            eff = cliff.cliffs_delta(browser1_energy, browser2_energy)

            u = stats.mannwhitneyu(browser1_energy, browser2_energy,␣
 ↪alternative='two-sided')

            data.append(
                [
                 browserpair[0] + ' vs. ' + browserpair[1],
```

```
                implementationpair[pairswitch[0]] + ' vs. ' +␣
 ↪implementationpair[pairswitch[1]],
                u.statistic,
                u.pvalue,
                eff[0],
                eff[1]
               ]
            )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns = ['browser', 'implementation', 'u', 'p',␣
 ↪'eff', 'interp'])
display(ut)

interp = ut['interp'].value_counts()
interp = pd.DataFrame(interp, columns = ['interp', 'percent'])
interp['percent'] = (interp['interp'] / interp['interp'].sum()) * 100

display(interp)
```

	browser	implementation	u	p	eff	interp
0	chrome vs. firefox	js vs. wasm	880131.0	3.841190e-19	0.210271	small
1	chrome vs. firefox	wasm vs. js	391450.0	3.216940e-86	-0.462606	medium

```
Empty DataFrame
Columns: [interp, percent]
Index: []
```

4.17 Mann Whitney U Test (Cross Browsers - By Device)

[25]:
```
data = []

for device in devices:
    for pairswitch in [[0,1],[1,0]]:
        for implementationpair in implementationpairs:
            for browserpair in browserpairs:
                browser1_energy = df[(df['browser'] == browserpair[0]) &␣
 ↪(df['implementation'] == implementationpair[pairswitch[0]]) & (df['device']␣
 ↪== device)]['energy']
                browser2_energy = df[(df['browser'] == browserpair[1]) &␣
 ↪(df['implementation'] == implementationpair[pairswitch[1]]) & (df['device']␣
 ↪== device)]['energy']
                eff = cliff.cliffs_delta(browser1_energy, browser2_energy)

                u = stats.mannwhitneyu(browser1_energy, browser2_energy,␣
 ↪alternative='two-sided')
```

```
                data.append(
                    [
                     device,
                     browserpair[0] + ' vs. ' + browserpair[1],
                     implementationpair[pairswitch[0]] + ' vs. ' +␣
 ↪implementationpair[pairswitch[1]],
                     u.statistic,
                     u.pvalue,
                     eff[0],
                     eff[1]
                    ]
                )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns = ['device', 'browser', 'implementation', 'u',␣
 ↪'p', 'eff', 'interp'])
display(ut)

interp = ut['interp'].value_counts()
interp = pd.DataFrame(interp, columns = ['interp', 'percent'])
interp['percent'] = (interp['interp'] / interp['interp'].sum()) * 100

display(interp)
```

```
     device             browser implementation         u             p  \
0  SM-G991B  chrome vs. firefox    js vs. wasm  227749.0  1.136373e-17
1  SM-G991B  chrome vs. firefox    wasm vs. js   74651.0  4.165707e-65
2   Nexus 5  chrome vs. firefox    js vs. wasm  239728.0  7.751432e-18
3   Nexus 5  chrome vs. firefox    wasm vs. js   97397.0  1.789286e-49

        eff interp
0  0.286627  small
1 -0.572545  large
2  0.284300  small
3 -0.486644  large
```

```
Empty DataFrame
Columns: [interp, percent]
Index: []
```

4.18 Descriptive Statistics

[26]:
```
data = []
for implementation in implementations:
    x = df[(df['implementation'] == implementation)]
    mean = numpy.round(numpy.mean(x['energy']), 2)
    median = numpy.round(numpy.median(x['energy']), 2)
```

```
    min = numpy.round(numpy.amin(x['energy']), 2)
    max = numpy.round(numpy.amax(x['energy']), 2)
    std = numpy.round(numpy.std(x['energy']), 2)
    sem = numpy.round(stats.sem(x['energy']), 2)
    q1 = numpy.round(numpy.quantile(x['energy'], 0.25), 2)
    q3 = numpy.round(numpy.quantile(x['energy'], 0.75), 2)

    data.append(
        [implementation, mean, std, min, q1, median, q3, max, sem]
    )

# Create the pandas DataFrame
stat = pd.DataFrame(data, columns = ['implementation', 'mean', 'std', 'min',␣
 ↪'q1', 'median', 'q3', 'max', 'sem'])
# display(stat)
print(stat.to_string())

# Alternative of pandas: x['energy'].describe()
```

```
  implementation   mean   std   min    q1  median     q3    max   sem
0             js  10.16  4.34  4.30  6.77    9.19  12.96  30.46  0.09
1           wasm   7.84  3.22  4.16  5.75    6.97   9.04  22.29  0.07
```

4.19 Descriptive Statistics Difference

[27]:
```
data = []

for implementationpair in implementationpairs:
    implementation1 = stat[(stat['implementation'] == implementationpair[1])]
    implementation2 = stat[(stat['implementation'] == implementationpair[0])]

    mean_diff = implementation1.iloc[0]['mean']-implementation2.iloc[0]['mean']
    median_diff = implementation1.iloc[0]['median']-implementation2.
 ↪iloc[0]['median']
    min_diff = implementation1.iloc[0]['min']-implementation2.iloc[0]['min']
    max_diff = implementation1.iloc[0]['max']-implementation2.iloc[0]['max']
    std_diff = implementation1.iloc[0]['std']-implementation2.iloc[0]['std']
    sem_diff = implementation1.iloc[0]['sem']-implementation2.iloc[0]['sem']
    q1_diff = implementation1.iloc[0]['q1']-implementation2.iloc[0]['q1']
    q3_diff = implementation1.iloc[0]['q3']-implementation2.iloc[0]['q3']

    data.append(
        [implementationpair[1] + ' vs. ' + implementationpair[0],
         numpy.round(mean_diff, 2),
         numpy.round(mean_diff/implementation2.iloc[0]['mean']*100, 2),
         numpy.round(median_diff, 2),
         numpy.round(median_diff/implementation2.iloc[0]['median']*100, 2),
```

```
            numpy.round(min_diff, 2),
            numpy.round(min_diff/implementation2.iloc[0]['min']*100, 2),
            numpy.round(max_diff, 2),
            numpy.round(max_diff/implementation2.iloc[0]['max']*100, 2),
            numpy.round(std_diff, 2),
            numpy.round(std_diff/implementation2.iloc[0]['std']*100, 2),
            numpy.round(sem_diff, 2),
            numpy.round(sem_diff/implementation2.iloc[0]['sem']*100, 2),
            numpy.round(q1_diff, 2),
            numpy.round(q1_diff/implementation2.iloc[0]['q1']*100, 2),
            numpy.round(q3_diff, 2),
            numpy.round(q3_diff/implementation2.iloc[0]['q3']*100, 2),
           ]
        )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns =␣
 ↪['rq','mean_diff','mean_diff%','median_diff','median_diff%','min_diff','min_diff%','max_dif
display(ut)
#print(ut.to_string())
```

```
           rq  mean_diff  mean_diff%  median_diff  median_diff%  min_diff  \
0  wasm vs. js      -2.32      -22.83        -2.22        -24.16     -0.14

   min_diff%  max_diff  max_diff%  std_diff  std_diff%  sem_diff  sem_diff%  \
0      -3.26     -8.17     -26.82     -1.12     -25.81     -0.02     -22.22

   q1_diff  q1_diff%  q3_diff  q3_diff%
0    -1.02    -15.07    -3.92    -30.25
```

4.20 Descriptive Statistics (By Browser)

[28]:
```
data = []
for implementation in implementations:
    for browser in browsers:
        x = df[(df['implementation'] == implementation) & (df['browser'] ==␣
 ↪browser)]
        mean = numpy.round(numpy.mean(x['energy']), 2)
        median = numpy.round(numpy.median(x['energy']), 2)
        min = numpy.round(numpy.amin(x['energy']), 2)
        max = numpy.round(numpy.amax(x['energy']), 2)
        std = numpy.round(numpy.std(x['energy']), 2)
        sem = numpy.round(stats.sem(x['energy']), 2)
        q1 = numpy.round(numpy.quantile(x['energy'], 0.25), 2)
        q3 = numpy.round(numpy.quantile(x['energy'], 0.75), 2)

        data.append(
```

```
            [implementation, browser, mean, std, min, q1, median,  q3, max, sem]
        )

# Create the pandas DataFrame
stat = pd.DataFrame(data, columns = ['implementation', 'browser', 'mean',␣
 ↪'std', 'min', 'q1', 'median', 'q3', 'max', 'sem'])
# display(stat)
print(stat.to_string())

# Alternative of pandas: x['energy'].describe()
```

	implementation	browser	mean	std	min	q1	median	q3	max	sem
0	js	chrome	9.37	3.79	4.30	6.52	8.47	11.28	26.04	0.11
1	js	firefox	10.94	4.70	4.39	7.11	10.43	13.89	30.46	0.14
2	wasm	chrome	7.55	3.04	4.16	5.64	6.85	8.37	22.29	0.09
3	wasm	firefox	8.14	3.36	4.22	5.96	7.09	9.67	19.78	0.10

4.21 Descriptive Statistics Difference (By Browser)

[29]:
```
data = []

for implementationpair in implementationpairs:
    for browser in browsers:
        implementation1 = stat[(stat['implementation'] ==␣
 ↪implementationpair[1]) & (stat['browser'] == browser)]
        implementation2 = stat[(stat['implementation'] ==␣
 ↪implementationpair[0]) & (stat['browser'] == browser)]

        mean_diff = implementation1.iloc[0]['mean']-implementation2.
 ↪iloc[0]['mean']
        median_diff = implementation1.iloc[0]['median']-implementation2.
 ↪iloc[0]['median']
        min_diff = implementation1.iloc[0]['min']-implementation2.iloc[0]['min']
        max_diff = implementation1.iloc[0]['max']-implementation2.iloc[0]['max']
        std_diff = implementation1.iloc[0]['std']-implementation2.iloc[0]['std']
        sem_diff = implementation1.iloc[0]['sem']-implementation2.iloc[0]['sem']
        q1_diff = implementation1.iloc[0]['q1']-implementation2.iloc[0]['q1']
        q3_diff = implementation1.iloc[0]['q3']-implementation2.iloc[0]['q3']

        data.append(
            [implementationpair[1] + ' vs. ' + implementationpair[0] + ' ' +␣
 ↪browser,
             numpy.round(mean_diff, 2),
             numpy.round(mean_diff/implementation2.iloc[0]['mean']*100, 2),
             numpy.round(median_diff, 2),
             numpy.round(median_diff/implementation2.iloc[0]['median']*100, 2),
             numpy.round(min_diff, 2),
```

```
            numpy.round(min_diff/implementation2.iloc[0]['min']*100, 2),
            numpy.round(max_diff, 2),
            numpy.round(max_diff/implementation2.iloc[0]['max']*100, 2),
            numpy.round(std_diff, 2),
            numpy.round(std_diff/implementation2.iloc[0]['std']*100, 2),
            numpy.round(sem_diff, 2),
            numpy.round(sem_diff/implementation2.iloc[0]['sem']*100, 2),
            numpy.round(q1_diff, 2),
            numpy.round(q1_diff/implementation2.iloc[0]['q1']*100, 2),
            numpy.round(q3_diff, 2),
            numpy.round(q3_diff/implementation2.iloc[0]['q3']*100, 2),
           ]
       )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns =␣
 ↪['rq','mean_diff','mean_diff%','median_diff','median_diff%','min_diff','min_diff%','max_dif
display(ut)
#print(ut.to_string())
```

	rq	mean_diff	mean_diff%	median_diff	median_diff%	\
0	wasm vs. js chrome	-1.82	-19.42	-1.62	-19.13	
1	wasm vs. js firefox	-2.80	-25.59	-3.34	-32.02	

	min_diff	min_diff%	max_diff	max_diff%	std_diff	std_diff%	sem_diff	\
0	-0.14	-3.26	-3.75	-14.40	-0.75	-19.79	-0.02	
1	-0.17	-3.87	-10.68	-35.06	-1.34	-28.51	-0.04	

	sem_diff%	q1_diff	q1_diff%	q3_diff	q3_diff%
0	-18.18	-0.88	-13.50	-2.91	-25.80
1	-28.57	-1.15	-16.17	-4.22	-30.38

4.22 Descriptive Statistics Difference (Cross Browser)

[30]:
```
data = []

for pairswitch in [[0,1],[1,0]]:
    for implementationpair in implementationpairs:
        for browserpair in browserpairs:
            implementation1 = stat[(stat['browser'] ==␣
 ↪browserpair[pairswitch[1]]) & (stat['implementation'] ==␣
 ↪implementationpair[1])]
            implementation2 = stat[(stat['browser'] ==␣
 ↪browserpair[pairswitch[0]]) & (stat['implementation'] ==␣
 ↪implementationpair[0])]

#for implementationpair in implementationpairs:
```

```
#    for browser in browsers:
#        implementation1 = stat[(stat['implementation'] ==␣
 ↪implementationpair[0]) & (stat['browser'] == browser)]
#        implementation2 = stat[(stat['implementation'] ==␣
 ↪implementationpair[1]) & (stat['browser'] == browser)]

        mean_diff = implementation1.iloc[0]['mean']-implementation2.
 ↪iloc[0]['mean']
        median_diff = implementation1.iloc[0]['median']-implementation2.
 ↪iloc[0]['median']
        min_diff = implementation1.iloc[0]['min']-implementation2.iloc[0]['min']
        max_diff = implementation1.iloc[0]['max']-implementation2.iloc[0]['max']
        std_diff = implementation1.iloc[0]['std']-implementation2.iloc[0]['std']
        sem_diff = implementation1.iloc[0]['sem']-implementation2.iloc[0]['sem']
        q1_diff = implementation1.iloc[0]['q1']-implementation2.iloc[0]['q1']
        q3_diff = implementation1.iloc[0]['q3']-implementation2.iloc[0]['q3']

        data.append(
            [
             browserpair[pairswitch[1]] + ' ' + implementationpair[1] + ' vs. '␣
 ↪+ browserpair[pairswitch[0]] + ' ' + implementationpair[0],
             numpy.round(mean_diff, 2),
             numpy.round(mean_diff/implementation2.iloc[0]['mean']*100, 2),
             numpy.round(median_diff, 2),
             numpy.round(median_diff/implementation2.iloc[0]['median']*100, 2),
             numpy.round(min_diff, 2),
             numpy.round(min_diff/implementation2.iloc[0]['min']*100, 2),
             numpy.round(max_diff, 2),
             numpy.round(max_diff/implementation2.iloc[0]['max']*100, 2),
             numpy.round(std_diff, 2),
             numpy.round(std_diff/implementation2.iloc[0]['std']*100, 2),
             numpy.round(sem_diff, 2),
             numpy.round(sem_diff/implementation2.iloc[0]['sem']*100, 2),
             numpy.round(q1_diff, 2),
             numpy.round(q1_diff/implementation2.iloc[0]['q1']*100, 2),
             numpy.round(q3_diff, 2),
             numpy.round(q3_diff/implementation2.iloc[0]['q3']*100, 2),
            ]
        )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns =␣
 ↪['rq','mean_diff','mean_diff%','median_diff','median_diff%','min_diff','min_diff%','max_dif
display(ut)
#print(ut.to_string())
```

```
                          rq  mean_diff  mean_diff%  median_diff  \
```

```
0  firefox wasm vs. chrome js      -1.23      -13.13        -1.38
1  chrome wasm vs. firefox js      -3.39      -30.99        -3.58

   median_diff%  min_diff  min_diff%  max_diff  max_diff%  std_diff  \
0        -16.29     -0.08      -1.86     -6.26     -24.04     -0.43
1        -34.32     -0.23      -5.24     -8.17     -26.82     -1.66

   std_diff%  sem_diff  sem_diff%  q1_diff  q1_diff%  q3_diff  q3_diff%
0     -11.35     -0.01      -9.09    -0.56     -8.59    -1.61    -14.27
1     -35.32     -0.05     -35.71    -1.47    -20.68    -5.52    -39.74
```

4.23 Descriptive Statistics (By Browser & By Device)

[31]:
```
data = []
for device in devices:
    for implementation in implementations:
        for browser in browsers:
            x = df[(df['implementation'] == implementation) & (df['browser'] ==␣
 ↪browser) & (df['device'] == device)]
            mean = numpy.round(numpy.mean(x['energy']), 2)
            median = numpy.round(numpy.median(x['energy']), 2)
            min = numpy.round(numpy.amin(x['energy']), 2)
            max = numpy.round(numpy.amax(x['energy']), 2)
            std = numpy.round(numpy.std(x['energy']), 2)
            sem = numpy.round(stats.sem(x['energy']), 2)
            q1 = numpy.round(numpy.quantile(x['energy'], 0.25), 2)
            q3 = numpy.round(numpy.quantile(x['energy'], 0.75), 2)

            data.append(
                [implementation, device, browser, mean, std, min, q1, median,␣
 ↪q3, max, sem]
            )

# Create the pandas DataFrame
stat = pd.DataFrame(data, columns = ['implementation', 'device', 'browser',␣
 ↪'mean', 'std', 'min', 'q1', 'median', 'q3', 'max', 'sem'])
# display(stat)
print(stat.to_string())

# Alternative of pandas: x['energy'].describe()
```

```
  implementation    device  browser   mean   std   min    q1  median     q3
max   sem
0             js  SM-G991B   chrome   7.05  2.00  4.30  5.46    6.62   8.29
13.96  0.08
1             js  SM-G991B  firefox   9.21  4.82  4.39  5.99    7.67  11.03
30.46  0.20
```

```
2           wasm  SM-G991B   chrome   5.67  1.25  4.16  4.49    5.47   6.46
10.14  0.05
3           wasm  SM-G991B  firefox   6.32  2.43  4.22  4.63    5.96   6.62
16.11  0.10
4             js   Nexus 5   chrome  11.64  3.75  5.92  8.53   10.93  14.44
26.04  0.15
5             js   Nexus 5  firefox  12.61  3.91  6.06  9.03   13.13  15.01
24.44  0.16
6           wasm   Nexus 5   chrome   9.34  3.16  5.67  7.23    8.24  10.02
22.29  0.13
7           wasm   Nexus 5  firefox   9.91  3.19  5.79  7.63    9.04  10.99
19.78  0.13
```

4.24 Descriptive Statistics Difference (By Browser & By Device)

[32]:
```
data = []

for implementationpair in implementationpairs:
    for device in devices:
        for browser in browsers:
            implementation1 = stat[(stat['implementation'] ==␣
 ↪implementationpair[1]) & (stat['browser'] == browser) & (stat['device'] ==␣
 ↪device)]
            implementation2 = stat[(stat['implementation'] ==␣
 ↪implementationpair[0]) & (stat['browser'] == browser) & (stat['device'] ==␣
 ↪device)]

            mean_diff = implementation1.iloc[0]['mean']-implementation2.
 ↪iloc[0]['mean']
            median_diff = implementation1.iloc[0]['median']-implementation2.
 ↪iloc[0]['median']
            min_diff = implementation1.iloc[0]['min']-implementation2.
 ↪iloc[0]['min']
            max_diff = implementation1.iloc[0]['max']-implementation2.
 ↪iloc[0]['max']
            std_diff = implementation1.iloc[0]['std']-implementation2.
 ↪iloc[0]['std']
            sem_diff = implementation1.iloc[0]['sem']-implementation2.
 ↪iloc[0]['sem']
            q1_diff = implementation1.iloc[0]['q1']-implementation2.
 ↪iloc[0]['q1']
            q3_diff = implementation1.iloc[0]['q3']-implementation2.
 ↪iloc[0]['q3']

            data.append(
                [implementationpair[1] + ' vs. ' + implementationpair[0] + ' '␣
 ↪+ browser,
```

```
                device,
                numpy.round(mean_diff, 2),
                numpy.round(mean_diff/implementation2.iloc[0]['mean']*100, 2),
                numpy.round(median_diff, 2),
                numpy.round(median_diff/implementation2.iloc[0]['median']*100,␣
 ↪2),
                numpy.round(min_diff, 2),
                numpy.round(min_diff/implementation2.iloc[0]['min']*100, 2),
                numpy.round(max_diff, 2),
                numpy.round(max_diff/implementation2.iloc[0]['max']*100, 2),
                numpy.round(std_diff, 2),
                numpy.round(std_diff/implementation2.iloc[0]['std']*100, 2),
                numpy.round(sem_diff, 2),
                numpy.round(sem_diff/implementation2.iloc[0]['sem']*100, 2),
                numpy.round(q1_diff, 2),
                numpy.round(q1_diff/implementation2.iloc[0]['q1']*100, 2),
                numpy.round(q3_diff, 2),
                numpy.round(q3_diff/implementation2.iloc[0]['q3']*100, 2),
               ]
            )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns =␣
 ↪['rq','device','mean_diff','mean_diff%','median_diff','median_diff%','min_diff','min_diff%'
display(ut)
#print(ut.to_string())
```

```
                  rq    device  mean_diff  mean_diff%  median_diff  \
0   wasm vs. js chrome  SM-G991B      -1.38      -19.57        -1.15
1  wasm vs. js firefox  SM-G991B      -2.89      -31.38        -1.71
2   wasm vs. js chrome   Nexus 5      -2.30      -19.76        -2.69
3  wasm vs. js firefox   Nexus 5      -2.70      -21.41        -4.09

   median_diff%  min_diff  min_diff%  max_diff  max_diff%  std_diff  \
0        -17.37     -0.14      -3.26     -3.82     -27.36     -0.75
1        -22.29     -0.17      -3.87    -14.35     -47.11     -2.39
2        -24.61     -0.25      -4.22     -3.75     -14.40     -0.59
3        -31.15     -0.27      -4.46     -4.66     -19.07     -0.72

   std_diff%  sem_diff  sem_diff%  q1_diff  q1_diff%  q3_diff  q3_diff%
0     -37.50     -0.03     -37.50    -0.97    -17.77    -1.83    -22.07
1     -49.59     -0.10     -50.00    -1.36    -22.70    -4.41    -39.98
2     -15.73     -0.02     -13.33    -1.30    -15.24    -4.42    -30.61
3     -18.41     -0.03     -18.75    -1.40    -15.50    -4.02    -26.78
```

4.25 Descriptive Statistics Difference (Cross Browser & By Device)

[33]:
```
data = []

for pairswitch in [[0,1],[1,0]]:
    for device in devices:
        for implementationpair in implementationpairs:
            for browserpair in browserpairs:
                implementation1 = stat[(stat['browser'] ==␣
↪browserpair[pairswitch[1]]) & (stat['implementation'] ==␣
↪implementationpair[1]) & (stat['device'] == device)]
                implementation2 = stat[(stat['browser'] ==␣
↪browserpair[pairswitch[0]]) & (stat['implementation'] ==␣
↪implementationpair[0]) & (stat['device'] == device)]

    #for implementationpair in implementationpairs:
    #    for browser in browsers:
    #        implementation1 = stat[(stat['implementation'] ==␣
↪implementationpair[0]) & (stat['browser'] == browser)]
    #        implementation2 = stat[(stat['implementation'] ==␣
↪implementationpair[1]) & (stat['browser'] == browser)]

            mean_diff = implementation1.iloc[0]['mean']-implementation2.
↪iloc[0]['mean']
            median_diff = implementation1.iloc[0]['median']-implementation2.
↪iloc[0]['median']
            min_diff = implementation1.iloc[0]['min']-implementation2.
↪iloc[0]['min']
            max_diff = implementation1.iloc[0]['max']-implementation2.
↪iloc[0]['max']
            std_diff = implementation1.iloc[0]['std']-implementation2.
↪iloc[0]['std']
            sem_diff = implementation1.iloc[0]['sem']-implementation2.
↪iloc[0]['sem']
            q1_diff = implementation1.iloc[0]['q1']-implementation2.
↪iloc[0]['q1']
            q3_diff = implementation1.iloc[0]['q3']-implementation2.
↪iloc[0]['q3']

            data.append(
                [
                 browserpair[pairswitch[1]] + ' ' + implementationpair[1] + '␣
↪vs. ' + browserpair[pairswitch[0]] + ' ' + implementationpair[0],
                 device,
                 numpy.round(mean_diff, 2),
                 numpy.round(mean_diff/implementation2.iloc[0]['mean']*100, 2),
```

```
                numpy.round(median_diff, 2),
                numpy.round(median_diff/implementation2.iloc[0]['median']*100,␣
 ↪2),
                numpy.round(min_diff, 2),
                numpy.round(min_diff/implementation2.iloc[0]['min']*100, 2),
                numpy.round(max_diff, 2),
                numpy.round(max_diff/implementation2.iloc[0]['max']*100, 2),
                numpy.round(std_diff, 2),
                numpy.round(std_diff/implementation2.iloc[0]['std']*100, 2),
                numpy.round(sem_diff, 2),
                numpy.round(sem_diff/implementation2.iloc[0]['sem']*100, 2),
                numpy.round(q1_diff, 2),
                numpy.round(q1_diff/implementation2.iloc[0]['q1']*100, 2),
                numpy.round(q3_diff, 2),
                numpy.round(q3_diff/implementation2.iloc[0]['q3']*100, 2),
               ]
            )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns =␣
 ↪['rq','device','mean_diff','mean_diff%','median_diff','median_diff%','min_diff','min_diff%'
display(ut)
#print(ut.to_string())
```

	rq	device	mean_diff	mean_diff%	median_diff	median_diff%	min_diff	min_diff%	max_diff	max_diff%	std_diff	std_diff%	sem_diff	sem_diff%	q1_diff	q1_diff%	q3_diff	q3_diff%
0	firefox wasm vs. chrome js	SM-G991B	-0.73	-10.35	-0.66	-9.97	-0.08	-1.86	2.15	15.40	0.43	21.50	0.02	25.00	-0.83	-15.20	-1.67	-20.14
1	firefox wasm vs. chrome js	Nexus 5	-1.73	-14.86	-1.89	-17.29	-0.13	-2.20	-6.26	-24.04	-0.56	-14.93	-0.02	-13.33	-0.90	-10.55	-3.45	-23.89
2	chrome wasm vs. firefox js	SM-G991B	-3.54	-38.44	-2.20	-28.68	-0.23	-5.24	-20.32	-66.71	-3.57	-74.07	-0.15	-75.00	-1.50	-25.04	-4.57	-41.43
3	chrome wasm vs. firefox js	Nexus 5	-3.27	-25.93	-4.89	-37.24	-0.39	-6.44	-2.15	-8.80	-0.75	-19.18	-0.03	-18.75	-1.80	-19.93	-4.99	-33.24

5 RQ2: JS Energy Browser

5.1 Shapiro Wilk Test

[34]:
```
data = []
non_normal = 0

for browser in browsers:
    energy = df[(df['browser'] == browser) & (df['implementation'] ==␣
 ↪'js')]['energy']

    if len(energy) >= 3:
        shapiro_test = stats.shapiro(energy)

        non_normal += (1 if shapiro_test.pvalue <= 0.05 else 0)

        data.append(
            [browser, 'js',
             shapiro_test.statistic,
             shapiro_test.pvalue
            ]
        )

# Create the pandas DataFrame
swt = pd.DataFrame(data, columns = ['browser', 'implementation', 'w', 'p'])
#print(swt.to_string())
display(swt)

print("\n{} non-normally distributed samples".format(non_normal))
print("{} normally distributed samples".format(len(swt) - non_normal))
print("{:.2f}% non-normally distributed samples".format(non_normal/
 ↪len(swt)*100))
```

	browser	implementation	w	p
0	chrome	js	0.923120	2.485691e-24
1	firefox	js	0.939191	8.412692e-22

```
2 non-normally distributed samples
0 normally distributed samples
100.00% non-normally distributed samples
```

5.2 Shapiro Wilk Test (By Device)

[35]:
```
data = []
non_normal = 0

for device in devices:
```

```
    for browser in browsers:
        energy = df[(df['browser'] == browser) & (df['implementation'] == 'js')␣
 ↪& (df['device'] == device)]['energy']

        if len(energy) >= 3:
            shapiro_test = stats.shapiro(energy)

            non_normal += (1 if shapiro_test.pvalue <= 0.05 else 0)

            data.append(
                [
                 device,
                 browser, 'js',
                 shapiro_test.statistic,
                 shapiro_test.pvalue
                ]
            )

# Create the pandas DataFrame
swt = pd.DataFrame(data, columns = ['device', 'browser', 'implementation', 'w',␣
 ↪'p'])
#print(swt.to_string())
display(swt)

print("\n{} non-normally distributed samples".format(non_normal))
print("{} normally distributed samples".format(len(swt) - non_normal))
print("{:.2f}% non-normally distributed samples".format(non_normal/
 ↪len(swt)*100))
```

	device	browser	implementation	w	p
0	SM-G991B	chrome	js	0.943418	2.788160e-14
1	SM-G991B	firefox	js	0.808114	6.720060e-26
2	Nexus 5	chrome	js	0.949549	1.371225e-13
3	Nexus 5	firefox	js	0.963830	3.718362e-11

```
4 non-normally distributed samples
0 normally distributed samples
100.00% non-normally distributed samples
```

5.3 Mann Whitney U Test

[36]:
```
data = []

for browserpair in browserpairs:
    browser1_energy = df[(df['browser'] == browserpair[0]) &␣
 ↪(df['implementation'] == 'js')]['energy']
```

```
    browser2_energy = df[(df['browser'] == browserpair[1]) &␣
 ↪(df['implementation'] == 'js')]['energy']
    eff = cliff.cliffs_delta(browser1_energy, browser2_energy)

    u = stats.mannwhitneyu(browser1_energy, browser2_energy,␣
 ↪alternative='two-sided')

    data.append(
        [
         browserpair[0] + ' vs. ' + browserpair[1],
         u.statistic,
         u.pvalue,
         eff[0],
         eff[1]
        ]
    )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns = ['rq', 'u', 'p', 'eff', 'interp'])
display(ut)

interp = ut['interp'].value_counts()
interp = pd.DataFrame(interp, columns = ['interp', 'percent'])
interp['percent'] = (interp['interp'] / interp['interp'].sum()) * 100

display(interp)
```

```
                  rq         u             p       eff interp
0  chrome vs. firefox  586526.0  2.471443e-16 -0.192797  small
```

```
Empty DataFrame
Columns: [interp, percent]
Index: []
```

5.4 Mann Whitney U Test (By Device)

[37]:
```
data = []

for device in devices:
    for browserpair in browserpairs:
        browser1_energy = df[(df['browser'] == browserpair[0]) &␣
 ↪(df['implementation'] == 'js') & (df['device'] == device)]['energy']
        browser2_energy = df[(df['browser'] == browserpair[1]) &␣
 ↪(df['implementation'] == 'js') & (df['device'] == device)]['energy']
        eff = cliff.cliffs_delta(browser1_energy, browser2_energy)
```

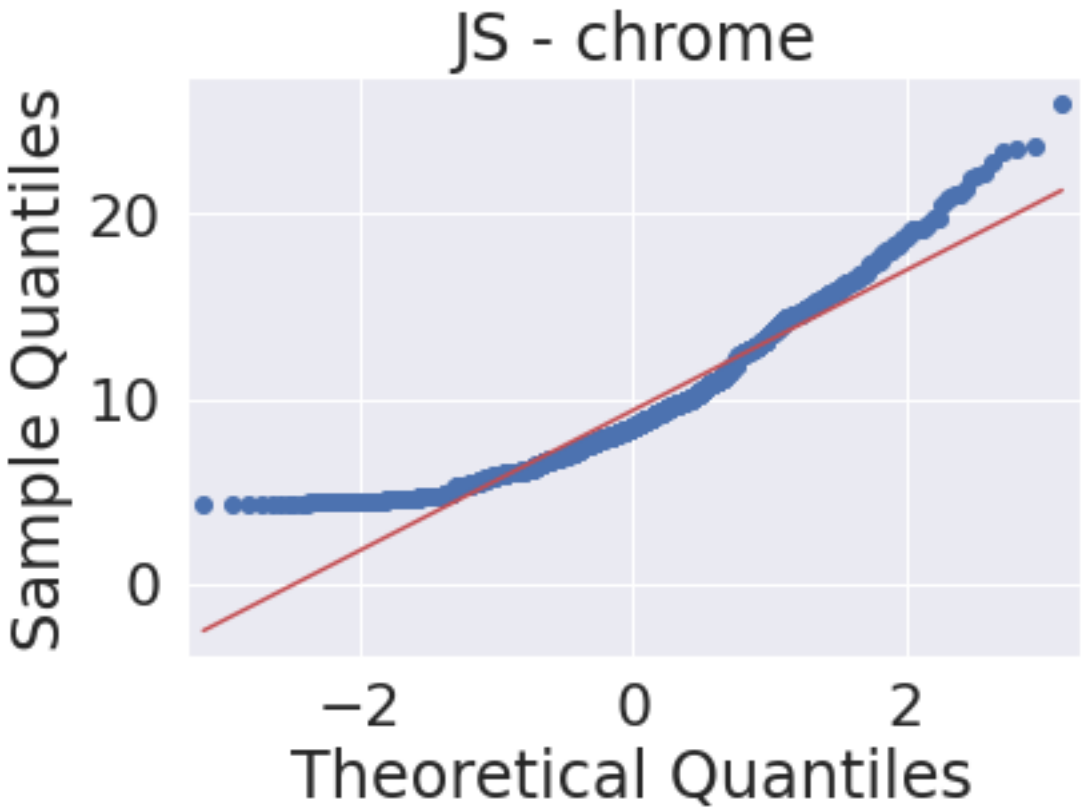
```
        u = stats.mannwhitneyu(browser1_energy, browser2_energy,␣
 ↪alternative='two-sided')

        data.append(
            [
             device,
             browserpair[0] + ' vs. ' + browserpair[1],
             u.statistic,
             u.pvalue,
             eff[0],
             eff[1]
            ]
        )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns = ['device', 'rq', 'u', 'p', 'eff', 'interp'])
display(ut)

interp = ut['interp'].value_counts()
interp = pd.DataFrame(interp, columns = ['interp', 'percent'])
interp['percent'] = (interp['interp'] / interp['interp'].sum()) * 100

display(interp)
```

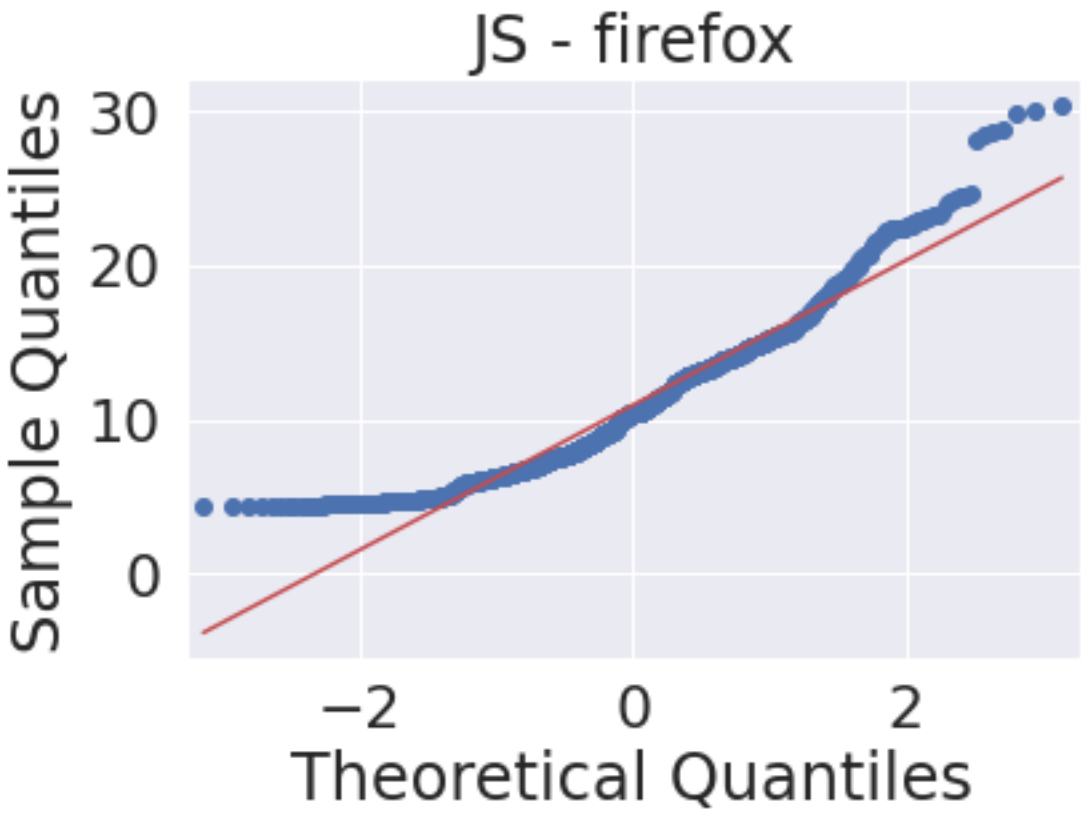
	device	rq	u	p	eff	interp
0	SM-G991B	chrome vs. firefox	130114.0	9.184663e-15	-0.259970	small
1	Nexus 5	chrome vs. firefox	160175.0	9.598055e-06	-0.146084	negligible

Empty DataFrame
Columns: [interp, percent]
Index: []

5.5 Q-Q-Plot

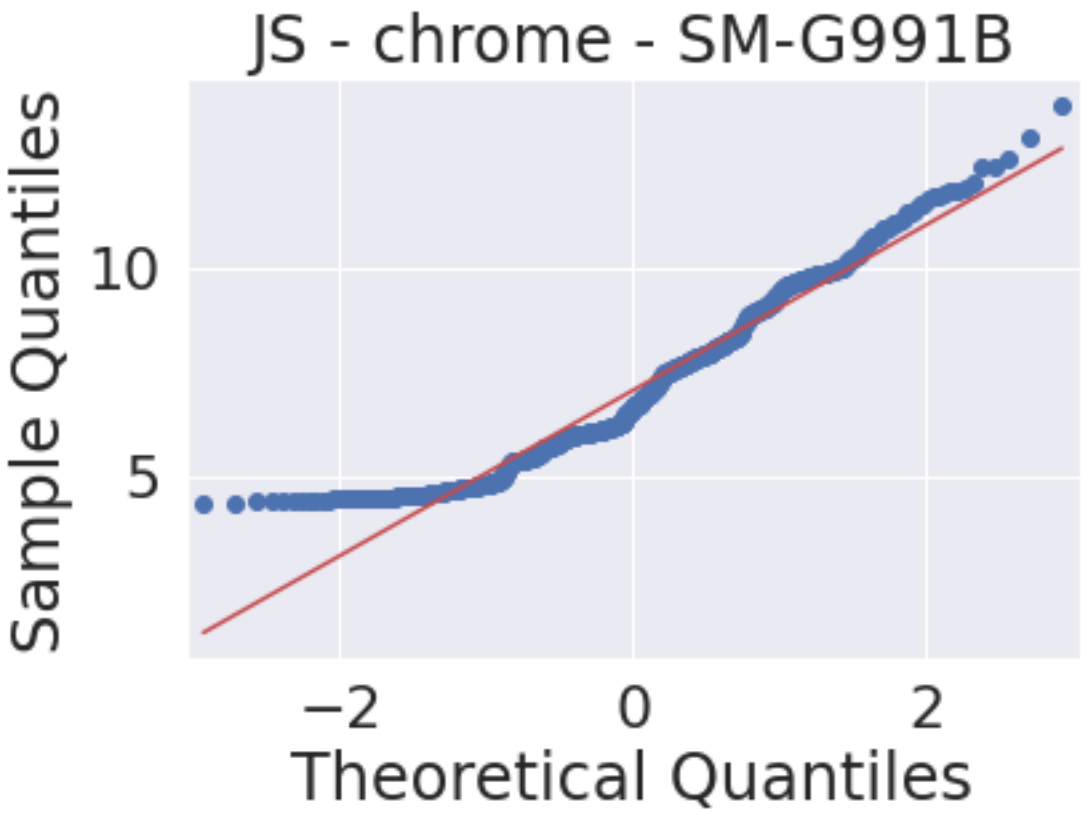
[38]:
```
for browser in browsers:
    data = df[(df['implementation'] == 'js') & (df['browser'] == browser)]
    qq = sm.qqplot(data.energy, line='s')
    h = plt.title('JS - ' + browser)
```

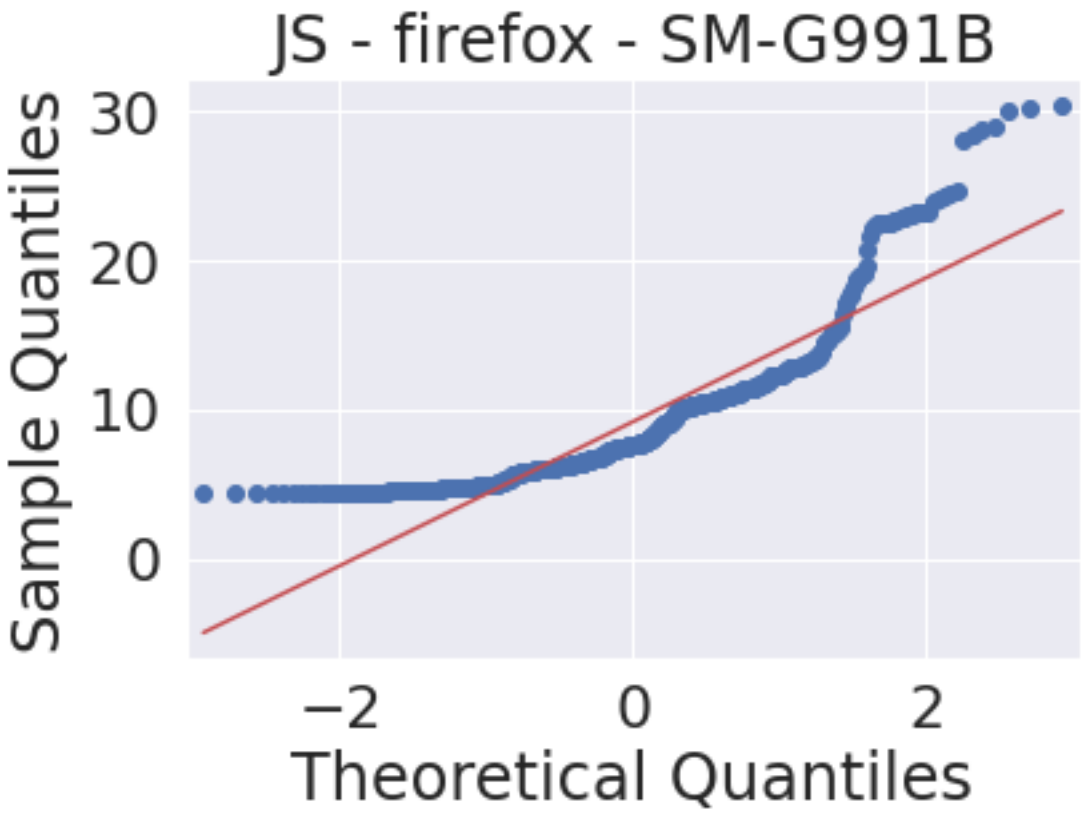

JS - chrome
Sample Quantiles
20
10
0
−2
0
2
Theoretical Quantiles

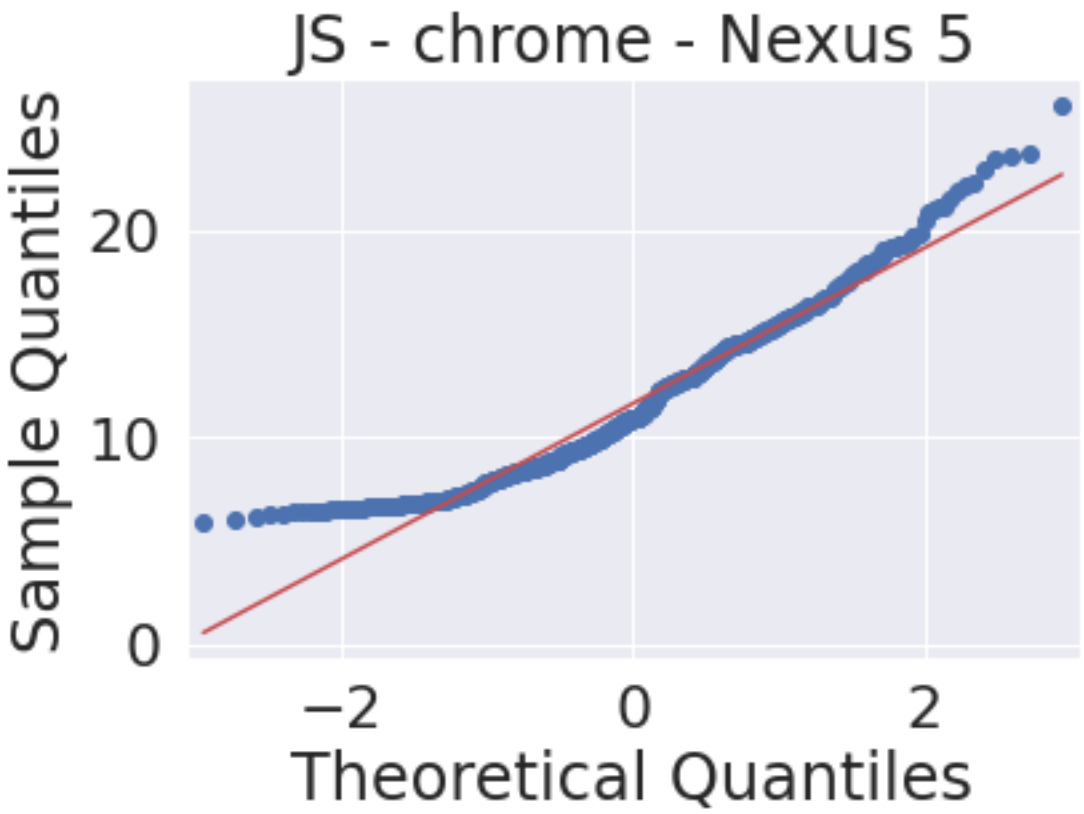


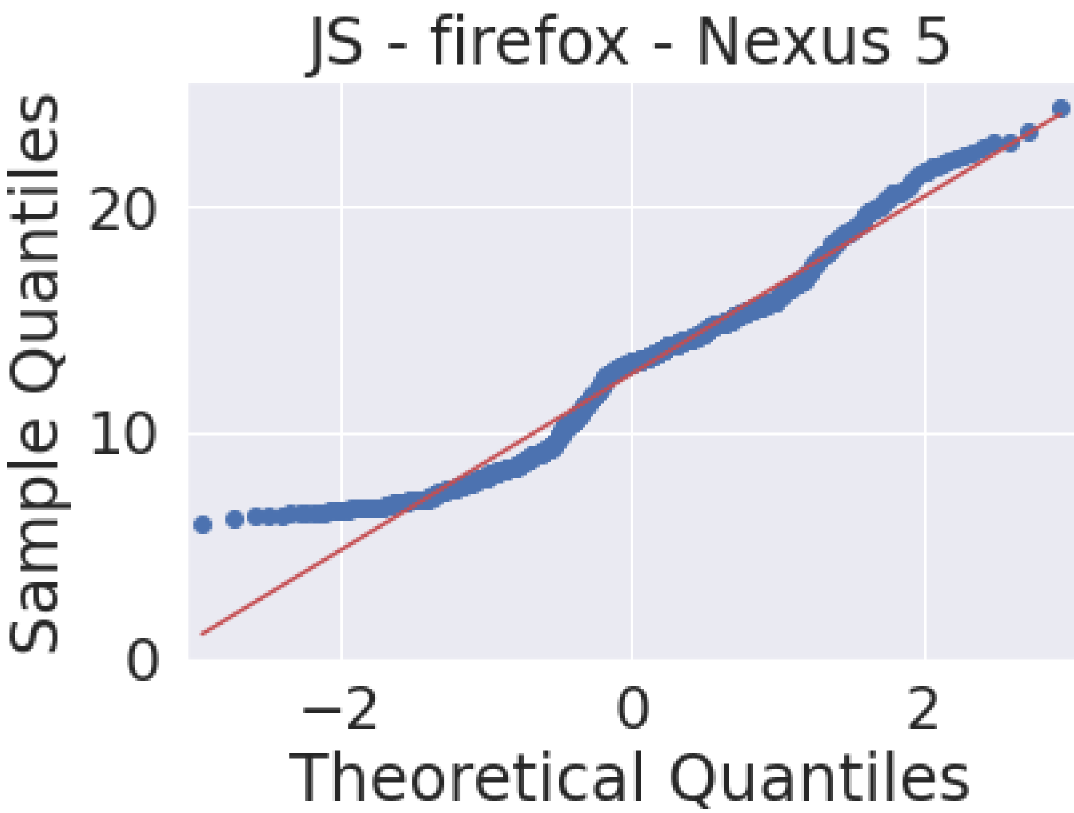
5.6 Q-Q-Plot (By Device)

[39]:
```
for device in devices:
    for browser in browsers:
        data = df[(df['implementation'] == 'js') & (df['browser'] == browser) &␣
↪(df['device'] == device)]
        qq = sm.qqplot(data.energy, line='s')
        h = plt.title('JS - ' + browser + ' - ' + device)
```


JS - chrome - SM-G991B
Sample Quantiles
10
5
−2
0
2
Theoretical Quantiles


JS - firefox - SM-G991B
Sample Quantiles
30
20
10
0
−2
0
2
Theoretical Quantiles


JS - chrome - Nexus 5
Sample Quantiles
20
10
0
−2
0
2
Theoretical Quantiles



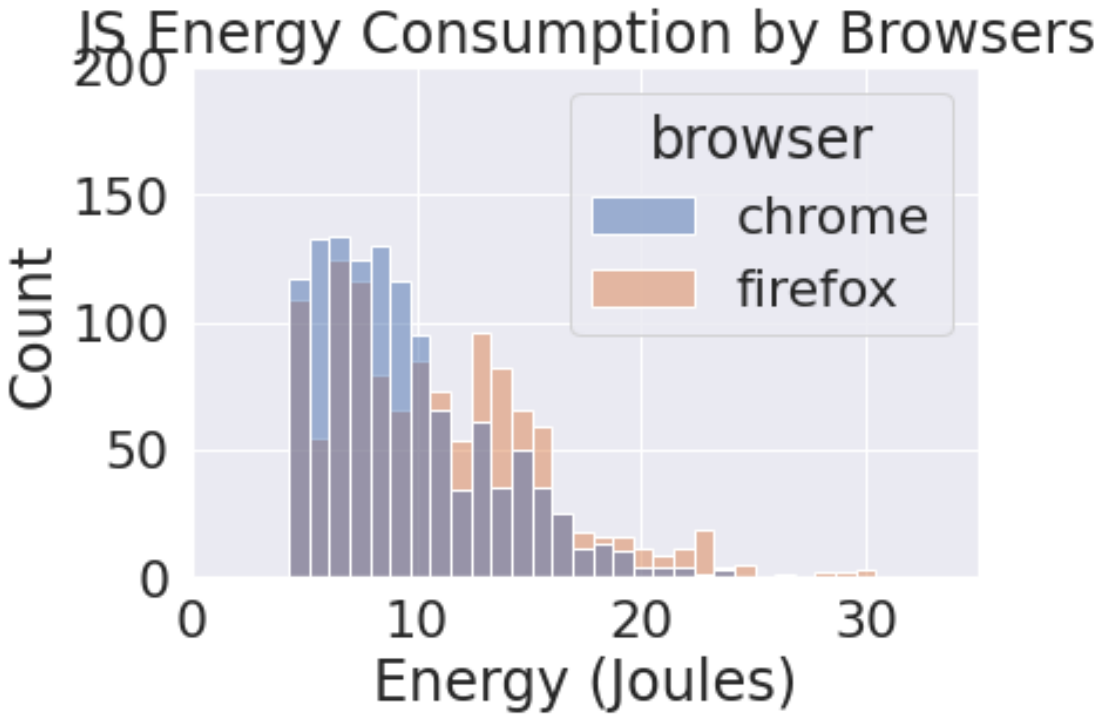
5.7 Histogram

[40]:
```
data = df[(df['implementation'] == 'js')]
sns.histplot(data=data, x="energy", hue="browser", hue_order=browsers).
  ↪set_title("JS Energy Consumption by Browsers")
plt.xlabel("Energy (Joules)")
plt.ylim(0, 200)
plt.xlim(0, 35)
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

[40]: (0.0, 35.0)

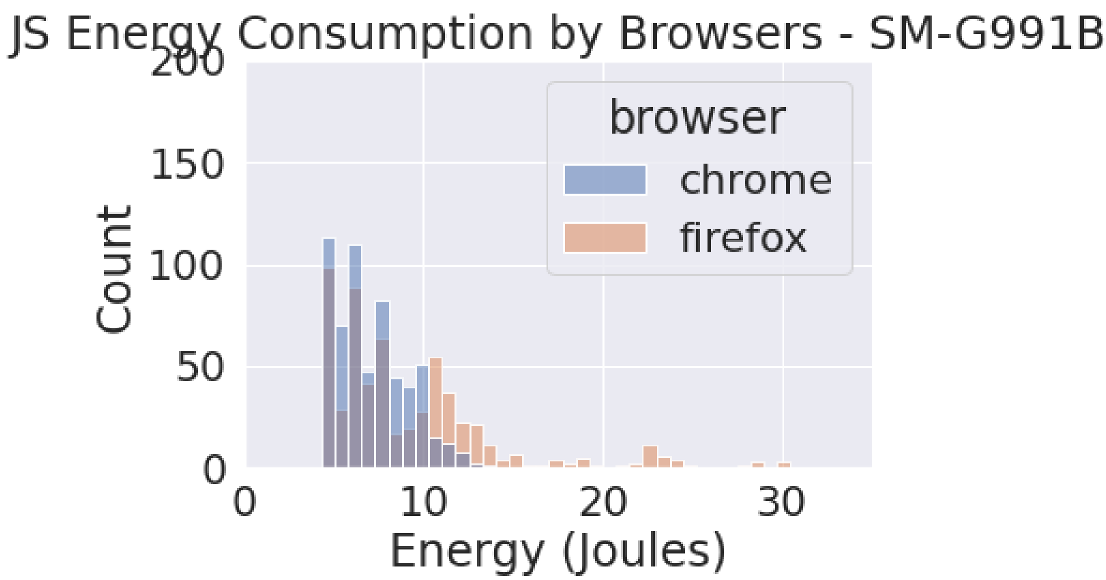


5.8 Histogramm (By Device)

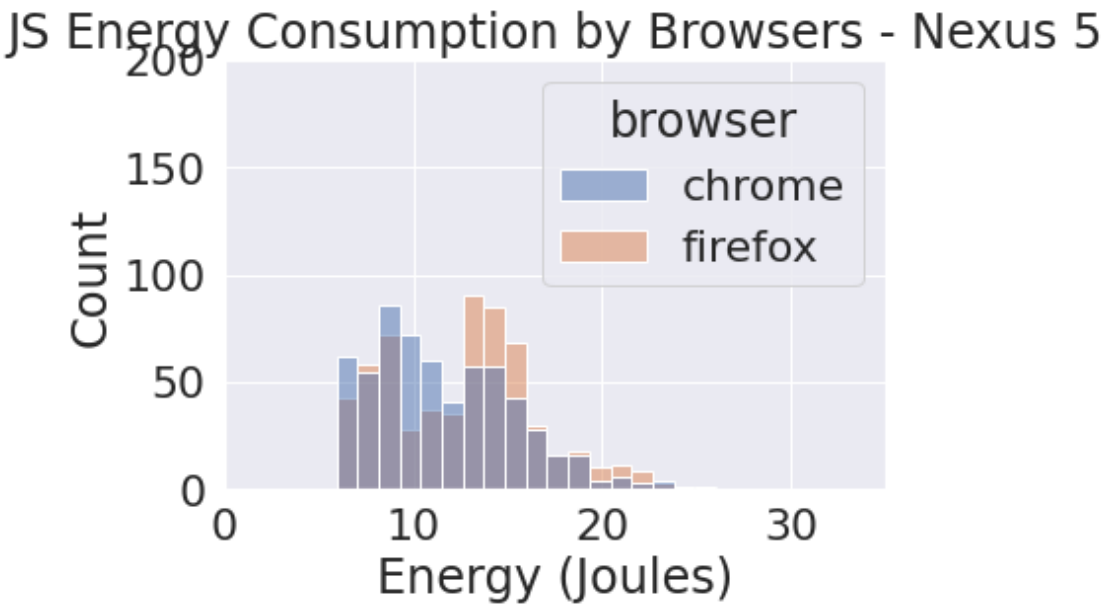
[41]:
```
data = []
for device in devices:
    data = df[(df['implementation'] == 'js') & (df['device'] == device)]
    sns.histplot(data=data, x="energy", hue="browser", hue_order=browsers).
 ↪set_title("JS Energy Consumption by Browsers" + " - " + device)
    plt.xlabel("Energy (Joules)")
    plt.ylim(0, 200)
    plt.xlim(0, 35)
    plt.show()
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
```

```
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



5.9 Descriptive Statistics

[42]:
```
data = []
for browser in browsers:
    x = df[(df['browser'] == browser) & (df['implementation'] == 'js')]
    mean = numpy.round(numpy.mean(x['energy']), 2)
    median = numpy.round(numpy.median(x['energy']), 2)
    min = numpy.round(numpy.amin(x['energy']), 2)
    max = numpy.round(numpy.amax(x['energy']), 2)
    std = numpy.round(numpy.std(x['energy']), 2)
    sem = numpy.round(stats.sem(x['energy']), 2)
    q1 = numpy.round(numpy.quantile(x['energy'], 0.25), 2)
    q3 = numpy.round(numpy.quantile(x['energy'], 0.75), 2)

    data.append(
        [browser, mean, std, min, q1, median, q3, max, sem]
    )

# Create the pandas DataFrame
stat = pd.DataFrame(data, columns = ['browser', 'mean', 'std', 'min', 'q1',␣
 ↪'median', 'q3', 'max', 'sem'])
display(stat)
#print(stat.to_string())

# Alternative of pandas: x['energy'].describe()
```

browser	mean	std	min	q1	median	q3	max	sem

```
0   chrome   9.37  3.79  4.30  6.52    8.47  11.28  26.04  0.11
1  firefox  10.94  4.70  4.39  7.11   10.43  13.89  30.46  0.14
```

5.10 Descriptive Statistics Difference

[43]:
```
data = []

for browserpair in browserpairs:
    browser1 = stat[(stat['browser'] == browserpair[0])]
    browser2 = stat[(stat['browser'] == browserpair[1])]

    mean_diff = browser1.iloc[0]['mean']-browser2.iloc[0]['mean']
    median_diff = browser1.iloc[0]['median']-browser2.iloc[0]['median']
    min_diff = browser1.iloc[0]['min']-browser2.iloc[0]['min']
    max_diff = browser1.iloc[0]['max']-browser2.iloc[0]['max']
    std_diff = browser1.iloc[0]['std']-browser2.iloc[0]['std']
    sem_diff = browser1.iloc[0]['sem']-browser2.iloc[0]['sem']
    q1_diff = browser1.iloc[0]['q1']-browser2.iloc[0]['q1']
    q3_diff = browser1.iloc[0]['q3']-browser2.iloc[0]['q3']

    data.append(
        [browserpair[0] + ' vs. ' + browserpair[1],
         numpy.round(mean_diff, 2),
         numpy.round(mean_diff/browser2.iloc[0]['mean']*100, 2),
         numpy.round(median_diff, 2),
         numpy.round(median_diff/browser2.iloc[0]['median']*100, 2),
         numpy.round(min_diff, 2),
         numpy.round(min_diff/browser2.iloc[0]['min']*100, 2),
         numpy.round(max_diff, 2),
         numpy.round(max_diff/browser2.iloc[0]['max']*100, 2),
         numpy.round(std_diff, 2),
         numpy.round(std_diff/browser2.iloc[0]['std']*100, 2),
         numpy.round(sem_diff, 2),
         numpy.round(sem_diff/browser2.iloc[0]['sem']*100, 2),
         numpy.round(q1_diff, 2),
         numpy.round(q1_diff/browser2.iloc[0]['q1']*100, 2),
         numpy.round(q3_diff, 2),
         numpy.round(q3_diff/browser2.iloc[0]['q3']*100, 2),
        ]
    )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns =␣
 ↪['rq','mean_diff','mean_diff%','median_diff','median_diff%','min_diff','min_diff%','max_dif
display(ut)
#print(ut.to_string())
```

```
                rq  mean_diff  mean_diff%  median_diff  median_diff%  \
```

```
0  chrome vs. firefox      -1.57      -14.35        -1.96        -18.79

   min_diff  min_diff%  max_diff  max_diff%  std_diff  std_diff%  sem_diff  \
0     -0.09      -2.05     -4.42     -14.51     -0.91     -19.36     -0.03

   sem_diff%  q1_diff  q1_diff%  q3_diff  q3_diff%
0     -21.43    -0.59      -8.3    -2.61    -18.79
```

5.11 Descriptive Statistics (By Device)

[44]:
```
data = []
for device in devices:
    for browser in browsers:
        x = df[(df['browser'] == browser) & (df['implementation'] == 'js') &␣
 ↪(df['device'] == device)]
        mean = numpy.round(numpy.mean(x['energy']), 2)
        median = numpy.round(numpy.median(x['energy']), 2)
        min = numpy.round(numpy.amin(x['energy']), 2)
        max = numpy.round(numpy.amax(x['energy']), 2)
        std = numpy.round(numpy.std(x['energy']), 2)
        sem = numpy.round(stats.sem(x['energy']), 2)
        q1 = numpy.round(numpy.quantile(x['energy'], 0.25), 2)
        q3 = numpy.round(numpy.quantile(x['energy'], 0.75), 2)

        data.append(
            [device, browser, mean, std, min, q1, median, q3, max, sem]
        )

# Create the pandas DataFrame
stat = pd.DataFrame(data, columns = ['device', 'browser', 'mean', 'std', 'min',␣
 ↪'q1', 'median', 'q3', 'max', 'sem'])
display(stat)
#print(stat.to_string())

# Alternative of pandas: x['energy'].describe()
```

```
     device  browser   mean   std   min    q1  median     q3    max   sem
0  SM-G991B   chrome   7.05  2.00  4.30  5.46    6.62   8.29  13.96  0.08
1  SM-G991B  firefox   9.21  4.82  4.39  5.99    7.67  11.03  30.46  0.20
2   Nexus 5   chrome  11.64  3.75  5.92  8.53   10.93  14.44  26.04  0.15
3   Nexus 5  firefox  12.61  3.91  6.06  9.03   13.13  15.01  24.44  0.16
```

5.12 Descriptive Statistics Difference (By Device)

[45]:
```
data = []

for device in devices:
    for browserpair in browserpairs:
        browser1 = stat[(stat['browser'] == browserpair[0]) & (stat['device']␣
 ↪== device)]
        browser2 = stat[(stat['browser'] == browserpair[1]) & (stat['device']␣
 ↪== device)]

        mean_diff = browser1.iloc[0]['mean']-browser2.iloc[0]['mean']
        median_diff = browser1.iloc[0]['median']-browser2.iloc[0]['median']
        min_diff = browser1.iloc[0]['min']-browser2.iloc[0]['min']
        max_diff = browser1.iloc[0]['max']-browser2.iloc[0]['max']
        std_diff = browser1.iloc[0]['std']-browser2.iloc[0]['std']
        sem_diff = browser1.iloc[0]['sem']-browser2.iloc[0]['sem']
        q1_diff = browser1.iloc[0]['q1']-browser2.iloc[0]['q1']
        q3_diff = browser1.iloc[0]['q3']-browser2.iloc[0]['q3']

        data.append(
            [
             device,
             browserpair[0] + ' vs. ' + browserpair[1],
             numpy.round(mean_diff, 2),
             numpy.round(mean_diff/browser2.iloc[0]['mean']*100, 2),
             numpy.round(median_diff, 2),
             numpy.round(median_diff/browser2.iloc[0]['median']*100, 2),
             numpy.round(min_diff, 2),
             numpy.round(min_diff/browser2.iloc[0]['min']*100, 2),
             numpy.round(max_diff, 2),
             numpy.round(max_diff/browser2.iloc[0]['max']*100, 2),
             numpy.round(std_diff, 2),
             numpy.round(std_diff/browser2.iloc[0]['std']*100, 2),
             numpy.round(sem_diff, 2),
             numpy.round(sem_diff/browser2.iloc[0]['sem']*100, 2),
             numpy.round(q1_diff, 2),
             numpy.round(q1_diff/browser2.iloc[0]['q1']*100, 2),
             numpy.round(q3_diff, 2),
             numpy.round(q3_diff/browser2.iloc[0]['q3']*100, 2),
            ]
        )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns = ['device',␣
 ↪'rq','mean_diff','mean_diff%','median_diff','median_diff%','min_diff','min_diff%','max_diff
display(ut)
```

```
#print(ut.to_string())
```

```
     device                  rq  mean_diff  mean_diff%  median_diff  \
0  SM-G991B  chrome vs. firefox      -2.16      -23.45        -1.05
1   Nexus 5  chrome vs. firefox      -0.97       -7.69        -2.20

   median_diff%  min_diff  min_diff%  max_diff  max_diff%  std_diff  \
0        -13.69     -0.09      -2.05     -16.5     -54.17     -2.82
1        -16.76     -0.14      -2.31       1.6       6.55     -0.16

   std_diff%  sem_diff  sem_diff%  q1_diff  q1_diff%  q3_diff  q3_diff%
0     -58.51     -0.12     -60.00    -0.53     -8.85    -2.74    -24.84
1      -4.09     -0.01      -6.25    -0.50     -5.54    -0.57     -3.80
```

6 RQ2: WASM Energy Browser

6.1 Shapiro Wilk Test

[46]:
```
data = []
non_normal = 0

for browser in browsers:
    energy = df[(df['browser'] == browser) & (df['implementation'] ==␣
 ↪'wasm')]['energy']

    if len(energy) >= 3:
        shapiro_test = stats.shapiro(energy)

        non_normal += (1 if shapiro_test.pvalue <= 0.05 else 0)

        data.append(
            [browser, 'wasm',
             shapiro_test.statistic,
             shapiro_test.pvalue
            ]
        )

# Create the pandas DataFrame
swt = pd.DataFrame(data, columns = ['browser', 'implementation', 'w', 'p'])
#print(swt.to_string())
display(swt)

print("\n{} non-normally distributed samples".format(non_normal))
print("{} normally distributed samples".format(len(swt) - non_normal))
print("{:.2f}% non-normally distributed samples".format(non_normal/
 ↪len(swt)*100))
```

```
   browser implementation         w             p
0   chrome           wasm  0.837385  2.093366e-33
1  firefox           wasm  0.882936  3.217827e-29


2 non-normally distributed samples
0 normally distributed samples
100.00% non-normally distributed samples
```

6.2 Shapiro Wilk Test (By Device)

[47]:
```
data = []
non_normal = 0

for device in devices:
    for browser in browsers:
        energy = df[(df['browser'] == browser) & (df['implementation'] ==␣
 ↪'wasm') & (df['device'] == device)]['energy']

        if len(energy) >= 3:
            shapiro_test = stats.shapiro(energy)

            non_normal += (1 if shapiro_test.pvalue <= 0.05 else 0)

            data.append(
                [
                 device,
                 browser, 'wasm',
                 shapiro_test.statistic,
                 shapiro_test.pvalue
                ]
            )

# Create the pandas DataFrame
swt = pd.DataFrame(data, columns = ['device', 'browser', 'implementation', 'w',␣
 ↪'p'])
#print(swt.to_string())
display(swt)

print("\n{} non-normally distributed samples".format(non_normal))
print("{} normally distributed samples".format(len(swt) - non_normal))
print("{:.2f}% non-normally distributed samples".format(non_normal/
 ↪len(swt)*100))
```

```
     device  browser implementation         w             p
0  SM-G991B   chrome           wasm  0.911131  4.198296e-18
1  SM-G991B  firefox           wasm  0.723657  3.320376e-30
2   Nexus 5   chrome           wasm  0.828244  3.247344e-25
```

```
3   Nexus 5  firefox           wasm  0.879442  2.271143e-21


4 non-normally distributed samples
0 normally distributed samples
100.00% non-normally distributed samples
```

6.3 Mann Whitney U Test

[48]:
```
data = []

for browserpair in browserpairs:
    browser1_energy = df[(df['browser'] == browserpair[0]) &␣
 ↪(df['implementation'] == 'wasm')]['energy']
    browser2_energy = df[(df['browser'] == browserpair[1]) &␣
 ↪(df['implementation'] == 'wasm')]['energy']
    eff = cliff.cliffs_delta(browser1_energy, browser2_energy)

    u = stats.mannwhitneyu(browser1_energy, browser2_energy,␣
 ↪alternative='two-sided')

    data.append(
        [browserpair[0] + ' vs. ' + browserpair[1],
         u.statistic,
         u.pvalue,
         eff[0],
         eff[1]
        ]
    )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns = ['rq','u', 'p', 'eff', 'interp'])
display(ut)

interp = ut['interp'].value_counts()
interp = pd.DataFrame(interp, columns = ['interp', 'percent'])
interp['percent'] = (interp['interp'] / interp['interp'].sum()) * 100

display(interp)
```

```
                  rq         u         p       eff      interp
0  chrome vs. firefox  655827.0  0.000019 -0.100408  negligible
```

```
Empty DataFrame
Columns: [interp, percent]
Index: []
```

6.4 Mann Whitney U Test (By Device)

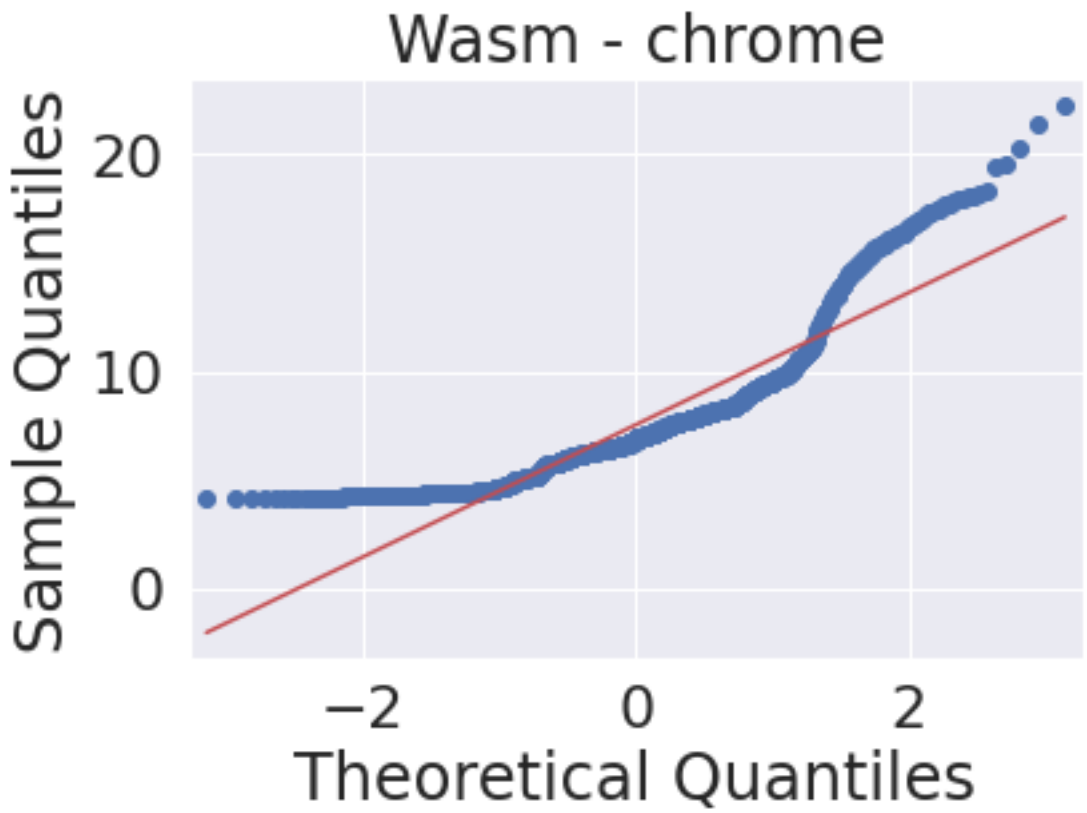
[49]:
```
data = []

for device in devices:
    for browserpair in browserpairs:
        browser1_energy = df[(df['browser'] == browserpair[0]) &␣
 ↪(df['implementation'] == 'wasm') & (df['device'] == device)]['energy']
        browser2_energy = df[(df['browser'] == browserpair[1]) &␣
 ↪(df['implementation'] == 'wasm') & (df['device'] == device)]['energy']
        eff = cliff.cliffs_delta(browser1_energy, browser2_energy)

        u = stats.mannwhitneyu(browser1_energy, browser2_energy,␣
 ↪alternative='two-sided')

        data.append(
            [
             device,
             browserpair[0] + ' vs. ' + browserpair[1],
             u.statistic,
             u.pvalue,
             eff[0],
             eff[1]
            ]
        )

# Create the pandas DataFrame
ut = pd.DataFrame(data, columns = ['device', 'rq','u', 'p', 'eff', 'interp'])
display(ut)

interp = ut['interp'].value_counts()
interp = pd.DataFrame(interp, columns = ['interp', 'percent'])
interp['percent'] = (interp['interp'] / interp['interp'].sum()) * 100

display(interp)
```

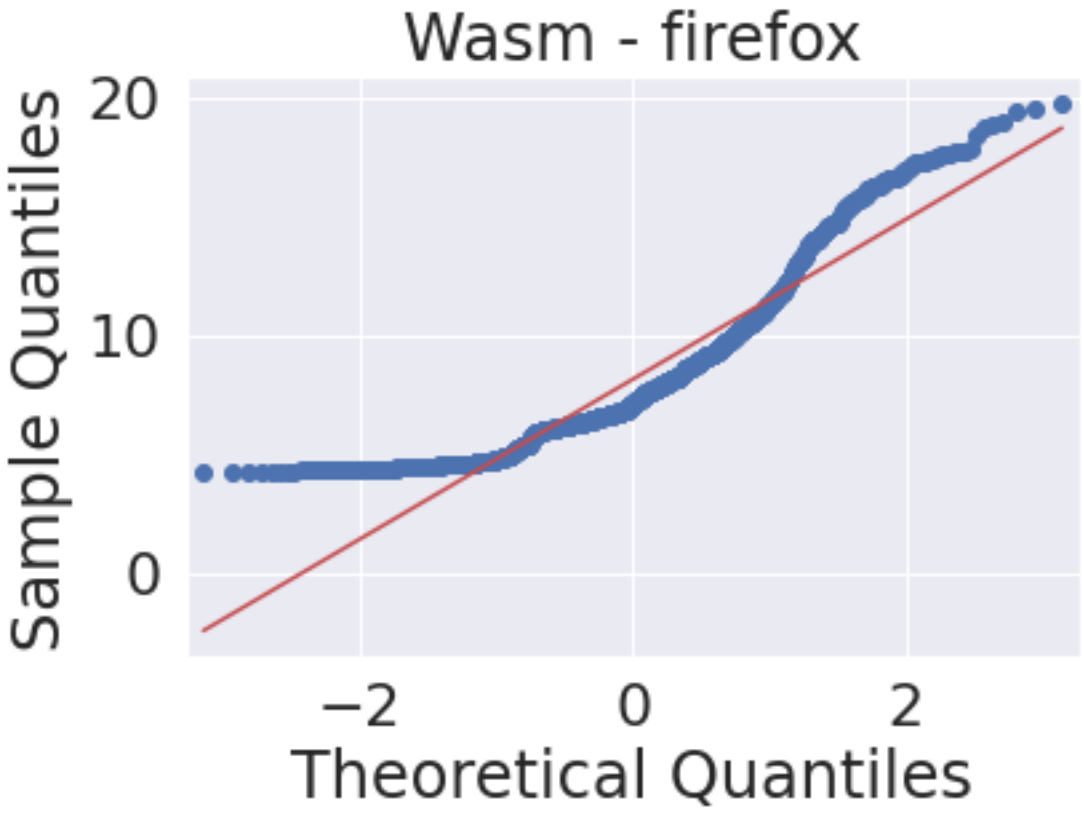
	device	rq	u	p	eff	interp
0	SM-G991B	chrome vs. firefox	154952.0	0.000402	-0.118702	negligible
1	Nexus 5	chrome vs. firefox	162482.0	0.000023	-0.139392	negligible

Empty DataFrame
Columns: [interp, percent]
Index: []

6.5 Q-Q-Plot

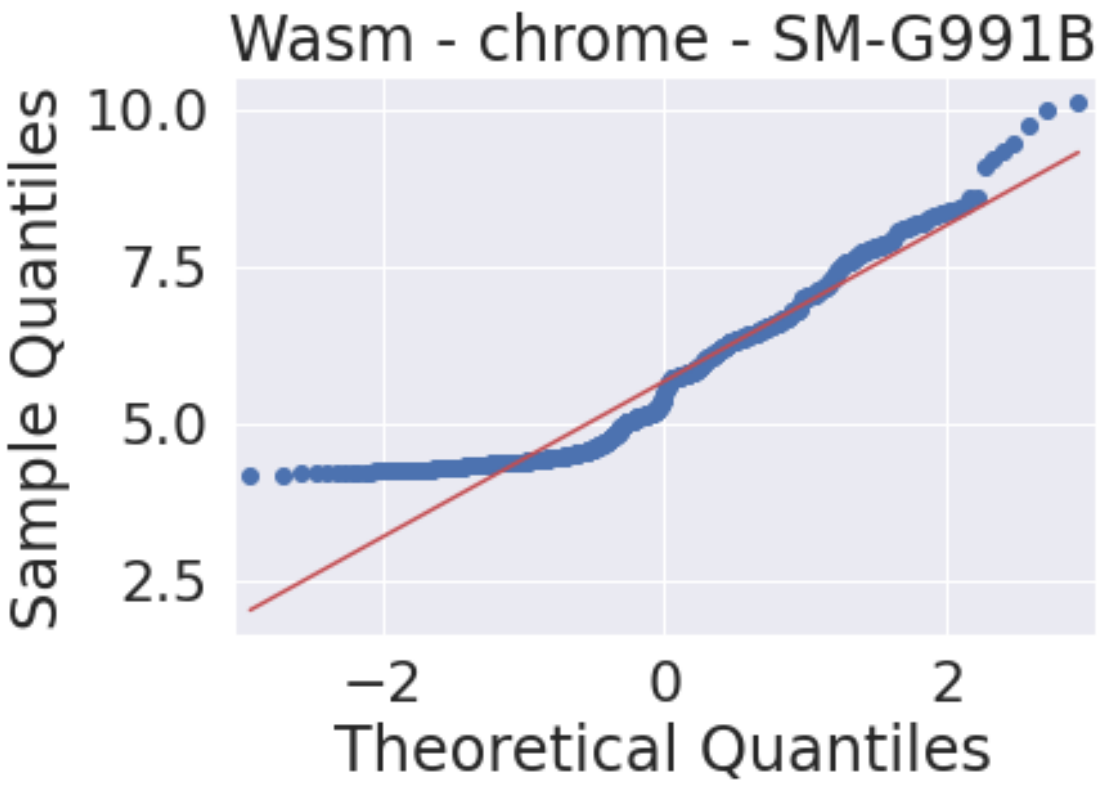
[50]:
```
for browser in browsers:
    data = df[(df['implementation'] == 'wasm') & (df['browser'] == browser)]
    qq = sm.qqplot(data.energy, line='s')
    h = plt.title('Wasm - ' + browser)
```

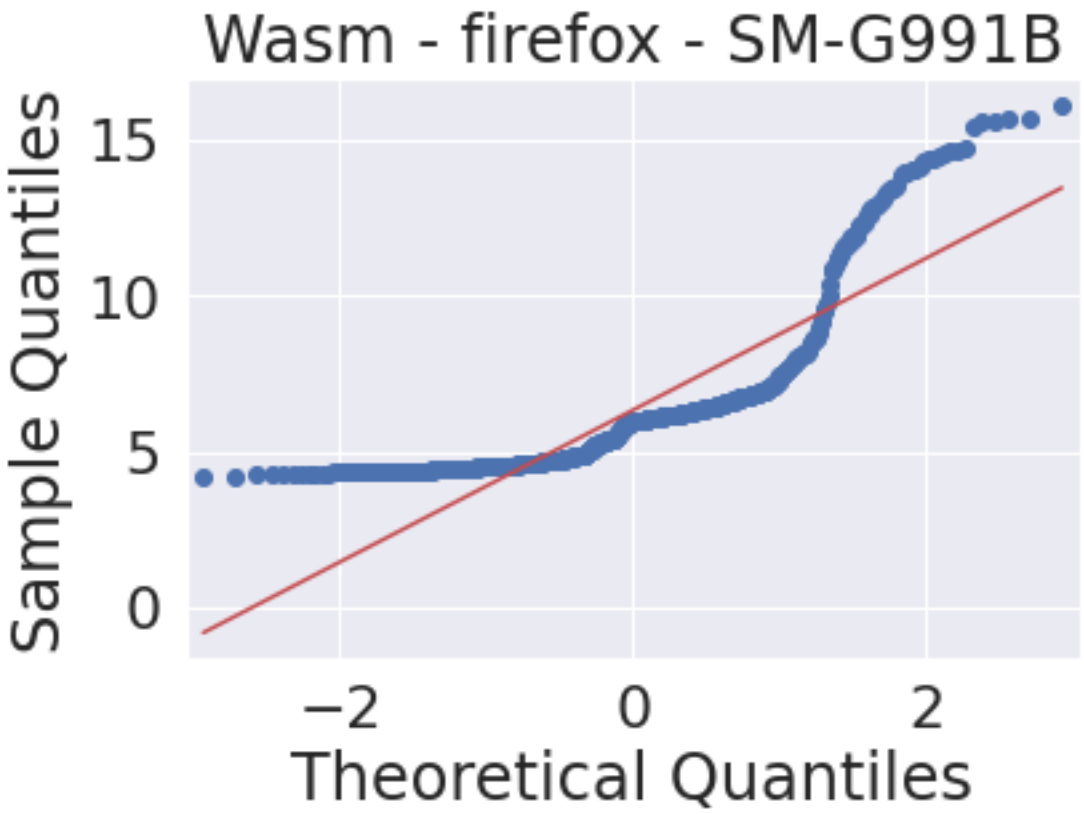


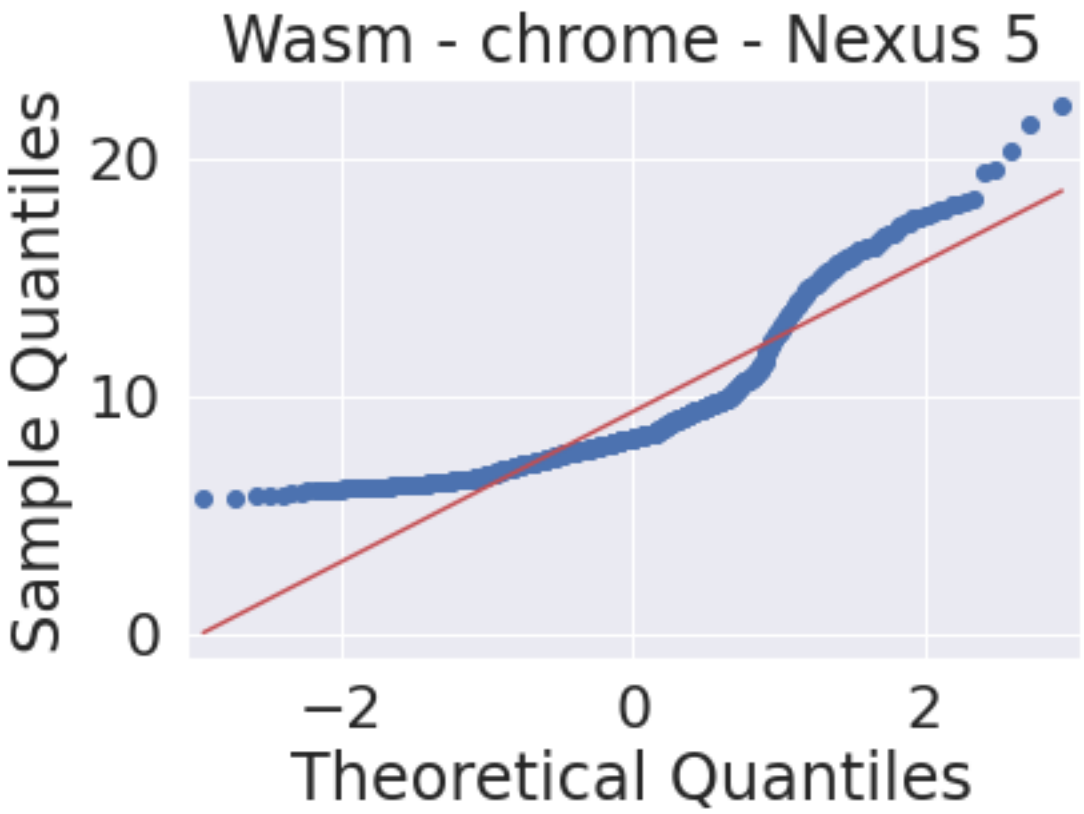


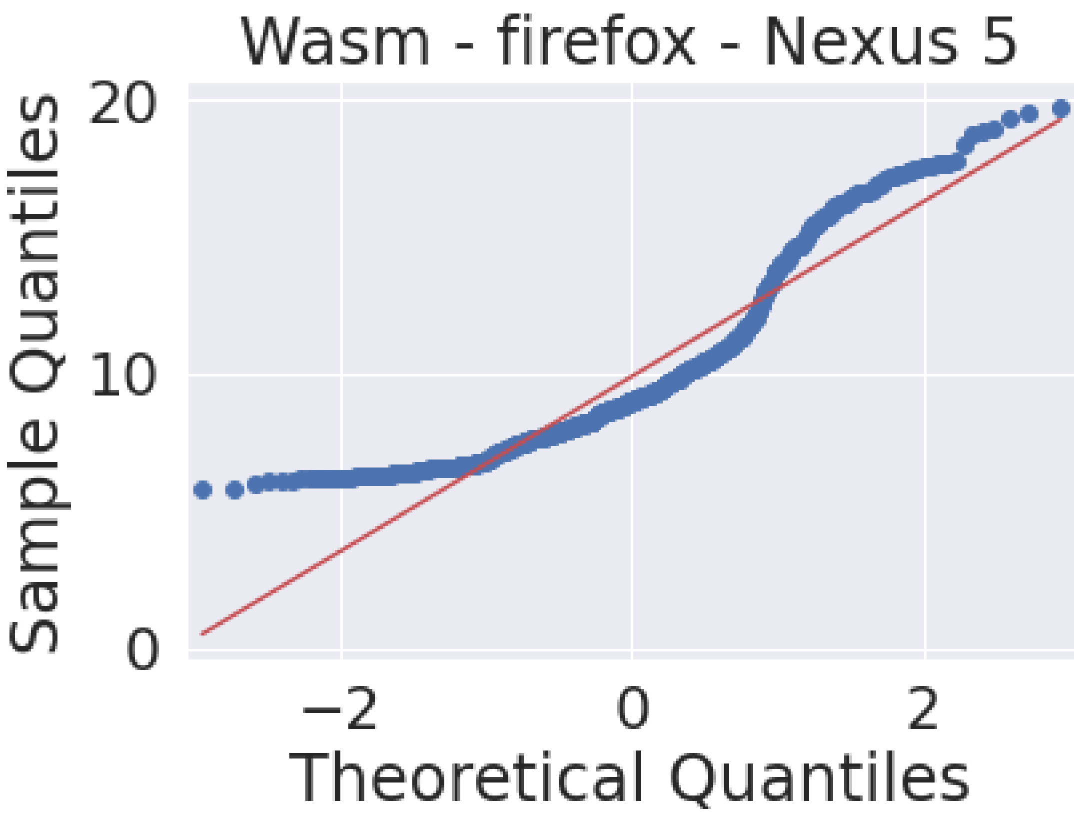
6.6 Q-Q-Plot (By Device)

[51]:
```
for device in devices:
    for browser in browsers:
        data = df[(df['implementation'] == 'wasm') & (df['browser'] == browser)␣
↪& (df['device'] == device)]
        qq = sm.qqplot(data.energy, line='s')
        h = plt.title('Wasm - ' + browser + ' - ' + device)
```


Wasm - chrome - SM-G991B
Sample Quantiles
10.0
7.5
5.0
2.5
−2
0
2
Theoretical Quantiles


Wasm - firefox - SM-G991B
Sample Quantiles
15
10
5
0
−2
0
2
Theoretical Quantiles


Wasm - chrome - Nexus 5
Sample Quantiles
20
10
0
−2
0
2
Theoretical Quantiles



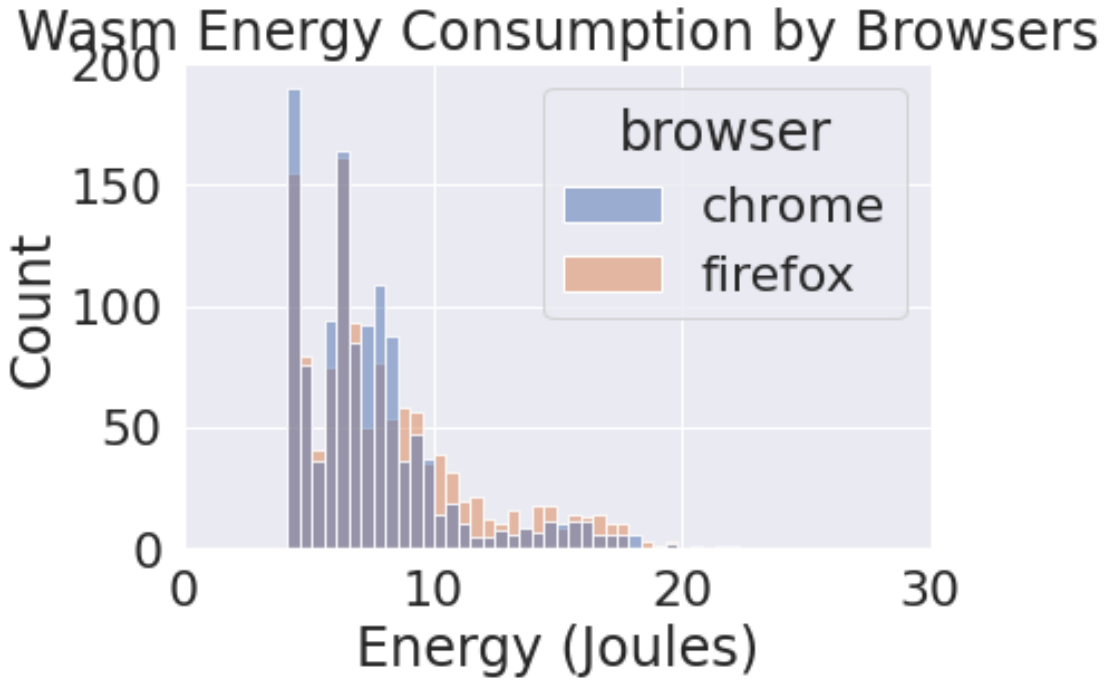
6.7 Histogram

[52]:
```
data = df[(df['implementation'] == 'wasm')]
sns.histplot(data=data, x="energy", hue="browser", hue_order=browsers).
  ↪set_title("Wasm Energy Consumption by Browsers")
plt.xlabel("Energy (Joules)")
plt.ylim(0, 200)
plt.xlim(0, 30)
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

[52]: (0.0, 30.0)

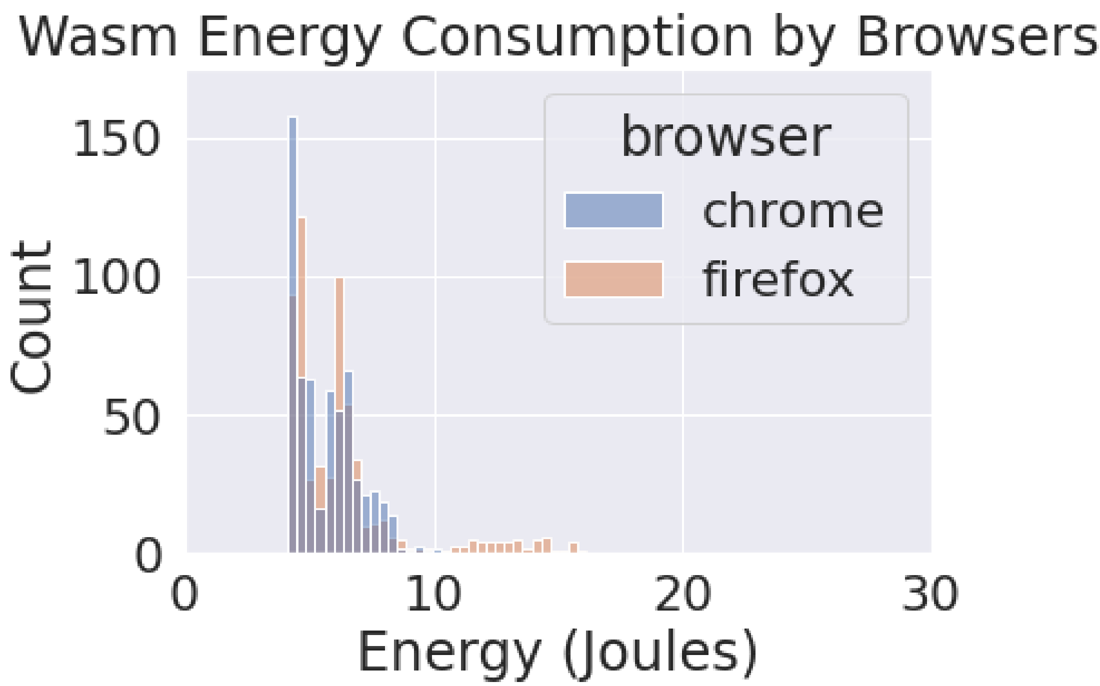


6.8 Histogram (By Device)

[53]:
```
data = []
for device in devices:
    data = df[(df['implementation'] == 'wasm') & (df['device'] == device)]
    sns.histplot(data=data, x="energy", hue="browser", hue_order=browsers).
 ↪set_title("Wasm Energy Consumption by Browsers")
    plt.xlabel("Energy (Joules)")
    plt.ylim(0, 175)
    plt.xlim(0, 30)
    plt.show()
```

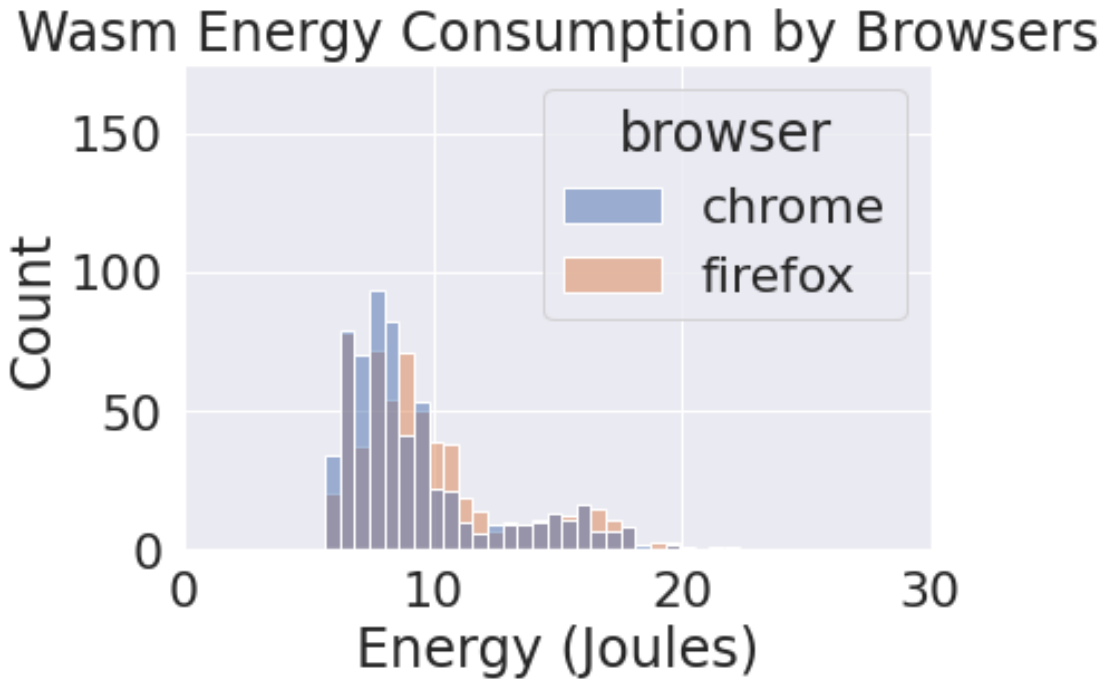
```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
```

```
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
```

```
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



6.9 Descriptive Statistics

[54]:
```
data = []
for browser in browsers:
    x = df[(df['browser'] == browser) & (df['implementation'] == 'wasm')]
    mean = numpy.round(numpy.mean(x['energy']), 2)
    median = numpy.round(numpy.median(x['energy']), 2)
    min = numpy.round(numpy.amin(x['energy']), 2)
    max = numpy.round(numpy.amax(x['energy']), 2)
    std = numpy.round(numpy.std(x['energy']), 2)
    sem = numpy.round(stats.sem(x['energy']), 2)
    q1 = numpy.round(numpy.quantile(x['energy'], 0.25), 2)
    q3 = numpy.round(numpy.quantile(x['energy'], 0.75), 2)

    data.append(
        [browser, mean, std, min, q1, median, q3, max, sem]
    )

# Create the pandas DataFrame
stat = pd.DataFrame(data, columns = ['browser', 'mean', 'std', 'min', 'q1',␣
 ↪'median', 'q3', 'max', 'sem'])
```

```
# display(stat)
print(stat.to_string())

# Alternative of pandas: x['energy'].describe()
```

```
   browser  mean   std   min    q1  median    q3    max   sem
0   chrome  7.55  3.04  4.16  5.64    6.85  8.37  22.29  0.09
1  firefox  8.14  3.36  4.22  5.96    7.09  9.67  19.78  0.10
```

6.10 Descriptive Statistics Difference

[55]:
```
data = []

for browserpair in browserpairs:
    browser1 = stat[(stat['browser'] == browserpair[0])]
    browser2 = stat[(stat['browser'] == browserpair[1])]

    mean_diff = browser1.iloc[0]['mean']-browser2.iloc[0]['mean']
    median_diff = browser1.iloc[0]['median']-browser2.iloc[0]['median']
    min_diff = browser1.iloc[0]['min']-browser2.iloc[0]['min']
    max_diff = browser1.iloc[0]['max']-browser2.iloc[0]['max']
    std_diff = browser1.iloc[0]['std']-browser2.iloc[0]['std']
    sem_diff = browser1.iloc[0]['sem']-browser2.iloc[0]['sem']
    q1_diff = browser1.iloc[0]['q1']-browser2.iloc[0]['q1']
    q3_diff = browser1.iloc[0]['q3']-browser2.iloc[0]['q3']

    data.append(
        [browserpair[0] + ' vs. ' + browserpair[1],
         numpy.round(mean_diff, 2),
         numpy.round(mean_diff/browser2.iloc[0]['mean']*100, 2),
         numpy.round(median_diff, 2),
         numpy.round(median_diff/browser2.iloc[0]['median']*100, 2),
         numpy.round(min_diff, 2),
         numpy.round(min_diff/browser2.iloc[0]['min']*100, 2),
         numpy.round(max_diff, 2),
         numpy.round(max_diff/browser2.iloc[0]['max']*100, 2),
         numpy.round(std_diff, 2),
         numpy.round(std_diff/browser2.iloc[0]['std']*100, 2),
         numpy.round(sem_diff, 2),
         numpy.round(sem_diff/browser2.iloc[0]['sem']*100, 2),
         numpy.round(q1_diff, 2),
         numpy.round(q1_diff/browser2.iloc[0]['q1']*100, 2),
         numpy.round(q3_diff, 2),
         numpy.round(q3_diff/browser2.iloc[0]['q3']*100, 2),
        ]
    )
```

```
# Create the pandas DataFrame
ut = pd.DataFrame(data, columns =␣
 ↪['rq','mean_diff','mean_diff%','median_diff','median_diff%','min_diff','min_diff%','max_dif
display(ut)
#print(ut.to_string())
```

```
                  rq  mean_diff  mean_diff%  median_diff  median_diff%  \
0  chrome vs. firefox      -0.59       -7.25        -0.24         -3.39

   min_diff  min_diff%  max_diff  max_diff%  std_diff  std_diff%  sem_diff  \
0     -0.06      -1.42      2.51      12.69     -0.32      -9.52     -0.01

   sem_diff%  q1_diff  q1_diff%  q3_diff  q3_diff%
0      -10.0    -0.32     -5.37     -1.3    -13.44
```

6.11 Descriptive Statistics (By Device)

[56]:
```
data = []

for device in devices:
    for browser in browsers:
        x = df[(df['browser'] == browser) & (df['implementation'] == 'wasm') &␣
 ↪(df['device'] == device)]
        mean = numpy.round(numpy.mean(x['energy']), 2)
        median = numpy.round(numpy.median(x['energy']), 2)
        min = numpy.round(numpy.amin(x['energy']), 2)
        max = numpy.round(numpy.amax(x['energy']), 2)
        std = numpy.round(numpy.std(x['energy']), 2)
        sem = numpy.round(stats.sem(x['energy']), 2)
        q1 = numpy.round(numpy.quantile(x['energy'], 0.25), 2)
        q3 = numpy.round(numpy.quantile(x['energy'], 0.75), 2)

        data.append(
            [device, browser, mean, std, min, q1, median, q3, max, sem]
        )

# Create the pandas DataFrame
stat = pd.DataFrame(data, columns = ['device', 'browser', 'mean', 'std', 'min',␣
 ↪'q1', 'median', 'q3', 'max', 'sem'])
# display(stat)
print(stat.to_string())

# Alternative of pandas: x['energy'].describe()
```

```
     device  browser  mean   std   min    q1  median     q3     max   sem
0  SM-G991B   chrome  5.67  1.25  4.16  4.49    5.47   6.46  10.14  0.05
1  SM-G991B  firefox  6.32  2.43  4.22  4.63    5.96   6.62  16.11  0.10
```

```
2   Nexus 5   chrome  9.34  3.16  5.67  7.23    8.24  10.02  22.29  0.13
3   Nexus 5  firefox  9.91  3.19  5.79  7.63    9.04  10.99  19.78  0.13
```

6.12 Descriptive Statistics Difference (By Device)

[57]:
```
data = []

for device in devices:
    for browserpair in browserpairs:
        browser1 = stat[(stat['browser'] == browserpair[0]) & (stat['device']␣
 ↪== device)]
        browser2 = stat[(stat['browser'] == browserpair[1]) & (stat['device']␣
 ↪== device)]

        mean_diff = browser1.iloc[0]['mean']-browser2.iloc[0]['mean']
        median_diff = browser1.iloc[0]['median']-browser2.iloc[0]['median']
        min_diff = browser1.iloc[0]['min']-browser2.iloc[0]['min']
        max_diff = browser1.iloc[0]['max']-browser2.iloc[0]['max']
        std_diff = browser1.iloc[0]['std']-browser2.iloc[0]['std']
        sem_diff = browser1.iloc[0]['sem']-browser2.iloc[0]['sem']
        q1_diff = browser1.iloc[0]['q1']-browser2.iloc[0]['q1']
        q3_diff = browser1.iloc[0]['q3']-browser2.iloc[0]['q3']

        data.append(
            [
             device,
             browserpair[0] + ' vs. ' + browserpair[1],
             numpy.round(mean_diff, 2),
             numpy.round(mean_diff/browser2.iloc[0]['mean']*100, 2),
             numpy.round(median_diff, 2),
             numpy.round(median_diff/browser2.iloc[0]['median']*100, 2),
             numpy.round(min_diff, 2),
             numpy.round(min_diff/browser2.iloc[0]['min']*100, 2),
             numpy.round(max_diff, 2),
             numpy.round(max_diff/browser2.iloc[0]['max']*100, 2),
             numpy.round(std_diff, 2),
             numpy.round(std_diff/browser2.iloc[0]['std']*100, 2),
             numpy.round(sem_diff, 2),
             numpy.round(sem_diff/browser2.iloc[0]['sem']*100, 2),
             numpy.round(q1_diff, 2),
             numpy.round(q1_diff/browser2.iloc[0]['q1']*100, 2),
             numpy.round(q3_diff, 2),
             numpy.round(q3_diff/browser2.iloc[0]['q3']*100, 2),
            ]
        )

# Create the pandas DataFrame
```

```
ut = pd.DataFrame(data, columns = ['device',␣
  ↪'rq','mean_diff','mean_diff%','median_diff','median_diff%','min_diff','min_diff%','max_diff
display(ut)
#print(ut.to_string())
```

```
     device                  rq  mean_diff  mean_diff%  median_diff  \
0  SM-G991B  chrome vs. firefox      -0.65      -10.28        -0.49
1   Nexus 5  chrome vs. firefox      -0.57       -5.75        -0.80

   median_diff%  min_diff  min_diff%  max_diff  max_diff%  std_diff  \
0         -8.22     -0.06      -1.42     -5.97     -37.06     -1.18
1         -8.85     -0.12      -2.07      2.51      12.69     -0.03

   std_diff%  sem_diff  sem_diff%  q1_diff  q1_diff%  q3_diff  q3_diff%
0     -48.56     -0.05      -50.0    -0.14     -3.02    -0.16     -2.42
1      -0.94      0.00        0.0    -0.40     -5.24    -0.97     -8.83
```

[ ]: