Logscale

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
import scipy.stats as ss
import vega_datasets
```

Ratio and logarithm

If you use linear scale to visualize ratios, it can be quite misleading.

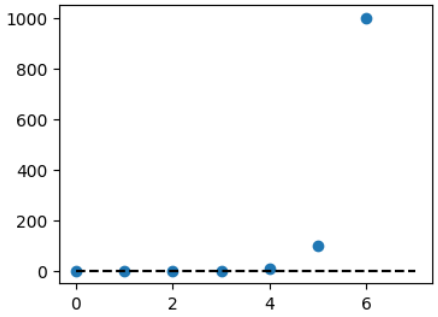
Let's create some ratios to ilustrate this.

In [2]:
```
x = np.array([1,    1,   1,  1, 10, 100, 1000])
y = np.array([1000, 100, 10, 1, 1,  1,   1   ])
ratio = x/y
print(ratio)
```

```
[1.e-03 1.e-02 1.e-01 1.e+00 1.e+01 1.e+02 1.e+03]
```

Q: Plot on the linear scale using the `scatter()` (http://matplotlib.org/examples/shapes_and_collections/scatter_demo.html) function. Also draw a horizontal line at ratio=1 for a reference. The x-axis will be simply the data ID that refers to each ratio data point. Y-axis will be the ratio values.

In [10]:
```
plt.figure(figsize=(4,3))
plt.scatter(np.arange(len(ratio)), ratio)
plt.plot([0, len(ratio)],[1,1], 'k--')
plt.show()
```

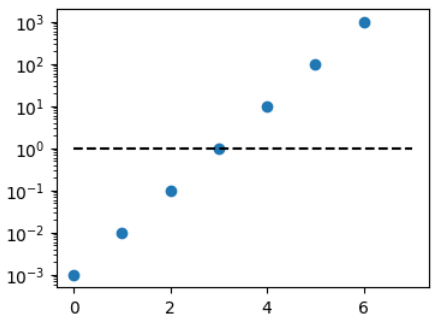


Q: Is this a good visualization of the ratio data? Why? Why not? Explain.

In [ ]:
```
# The downside of this visualization is that we can't observe the difference in scale between the smaller values (up to 10^4 in this example).
# This makes the plot indicate an exponential like trend, but finer details are hard to deduce (the evolution might be as well 0,0,0,0, 100, 1000).
```

Q: Can you fix it?

In [ ]:
```
# log scale adaptively represents diferences between points at different scales
plt.figure(figsize=(4,3))
plt.scatter(np.arange(len(ratio)), ratio)
plt.plot([0, len(ratio)],[1,1], 'k--')
plt.yscale('log')
plt.show()
```



Log-binning

One way to draw a histogram in log-scale, with a broadly distributed data, is by using log-binning.

First, see what happens if we do not use the log scale for a dataset with a heavy tail.

Q: Load the movie dataset from `vega_datasets` and remove the NaN rows based on the following three columns: `IMDB_Rating`, `IMDB_Votes`, `Rotten_Tomatoes_Rating`.

In [19]:
```
movies = vega_datasets.data.movies().dropna(subset=["IMDB_Rating", "IMDB_Votes", "Rotten_Tomatoes_Rating"])
movies.sample(5)
```

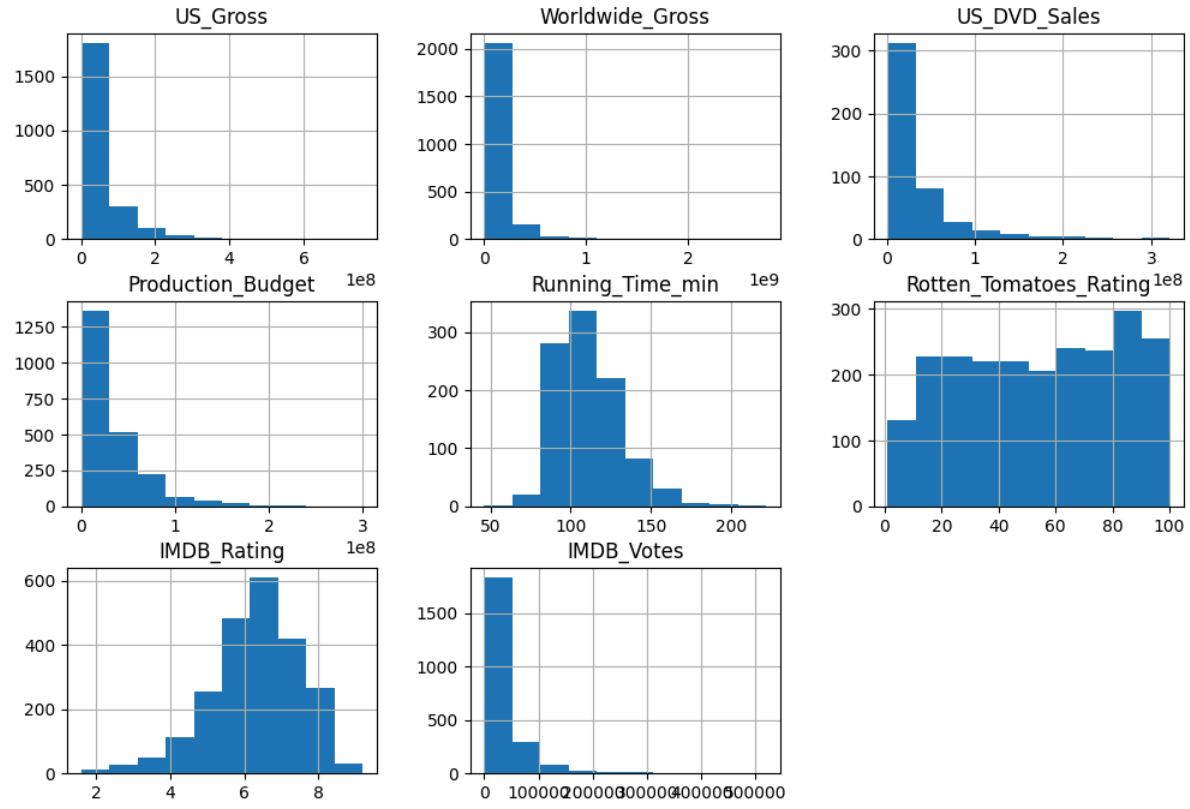
Out[19]:

	Title	US_Gross	Worldwide_Gross	US_DVD_Sales	Production_Budget	Release_Date	MPAA_Rating	Running_Time_min	Distributor	Source	Major_Genr
2805	The Sorcerer's Apprentice	62492818.0	200092818.0	NaN	160000000.0	Jul 14 2010	PG	110.0	Walt Disney Pictures	Based on Short Film	Adventur
3141	A Walk to Remember	41227069.0	46060915.0	NaN	11000000.0	Jan 25 2002	PG	102.0	Warner Bros.	Based on Book/Short Story	Dram
776	The Right Stuff	21500000.0	21500000.0	NaN	27000000.0	Oct 21 1983	None	NaN	Warner Bros.	Based on Book/Short Story	Actio
1885	Ghost World	6217849.0	8764007.0	NaN	5500000.0	Jul 20 2001	R	NaN	MGM	Based on Comic/Graphic Novel	Comec
595	The Mirror Has Two Faces	41267469.0	41267469.0	NaN	42000000.0	Nov 15 1996	PG-13	127.0	Sony Pictures	Original Screenplay	Romant Comec

If you simply call `hist()` method with a dataframe object, it identifies all the numeric columns and draws a histogram for each column.

Q: draw all possible histograms of the movie dataframe. Adjust the size of the plots if needed.

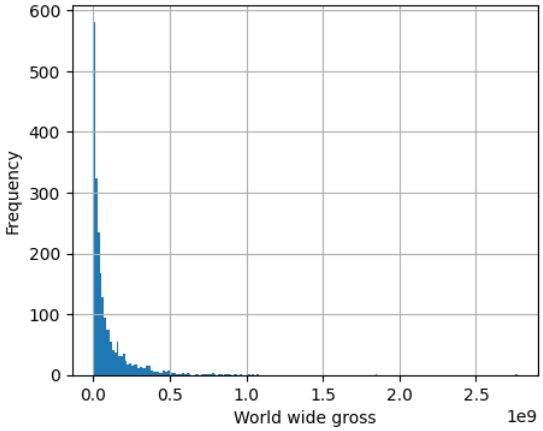
In [38]:
```
movies.hist(figsize=(12,8))
plt.show()
```



As we can see, a majority of the columns are not normally distributed. In particular, if you look at the worldwide gross variable, you only see a couple of meaningful data from the histogram. Is this a problem of resolution? How about increasing the number of bins?

In [40]:
```
ax = movies["Worldwide_Gross"].hist(bins=200, figsize=(5,4))
ax.set_xlabel("World wide gross")
ax.set_ylabel("Frequency")
```

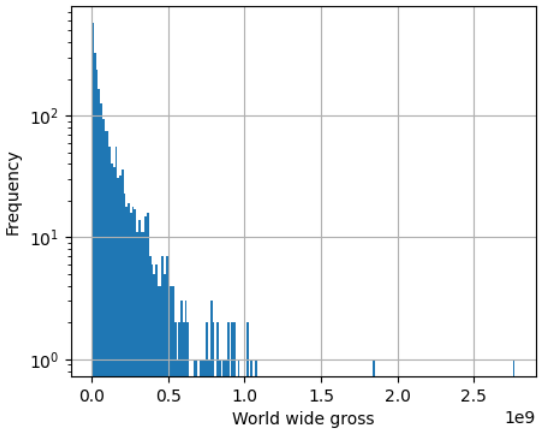
Out[40]: Text(0, 0.5, 'Frequency')



Maybe a bit more useful, but it doesn't tell anything about the data distribution above certain point. How about changing the vertical scale to logarithmic scale?

In [42]:
```
ax = movies["Worldwide_Gross"].hist(bins=200, figsize=(5,4))
ax.set_yscale('log')
ax.set_xlabel("World wide gross")
ax.set_ylabel("Frequency")
```

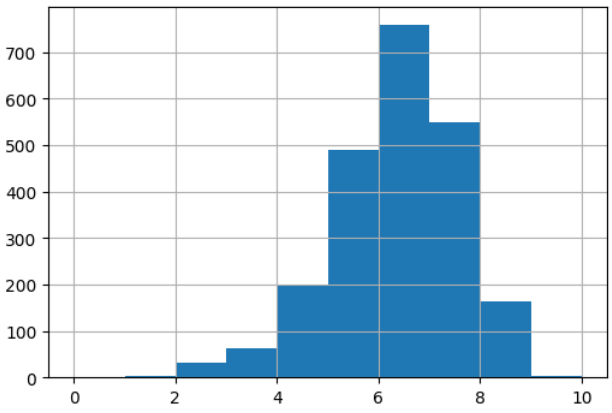
Out[42]: Text(0, 0.5, 'Frequency')



Now, let's try log-bin. Recall that when plotting histgrams we can specify the edges of bins through the `bins` parameter. For example, we can specify the edges of bins to [1, 2, 3, ... , 10] as follows.

In [43]:
```
movies["IMDB_Rating"].hist(bins=range(0,11), figsize=(6,4))
```

Out[43]: <Axes: >



Here, we can specify the edges of bins in a similar way. Instead of specifying on the linear scale, we do it on the log space. Some useful resources:

- numpy.logspace (http://docs.scipy.org/doc/numpy/reference/generated/numpy.logspace.html)
- numpy.linspace vs numpy.logspace (http://stackoverflow.com/questions/31480033/difference-in-output-between-numpy-linspace-and-numpy-logspace)

Hint: since $10^{\text{start}} = \min(\text{Worldwide\_Gross})$, $\text{start} = \log_{10}(\min(\text{Worldwide\_Gross}))$

In [44]:
```
min(movies["Worldwide_Gross"])
```

Out[44]:
```
0.0
```

Because there seems to be movie(s) that made $0, and because log(0) is undefined & log(1) = 0, let's add 1 to the variable.

In [45]:
```
movies["Worldwide_Gross"] = movies["Worldwide_Gross"]+1.0
```

In [52]:
```
max(movies['Worldwide_Gross'])
```

Out[52]:
```
2767891500.0
```

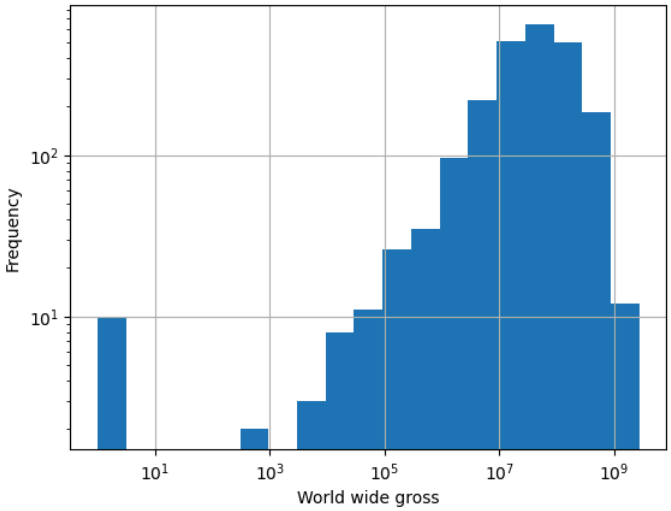
In [57]:
```
bins = np.logspace(np.log(min(movies['Worldwide_Gross'])), np.log(max(movies['Worldwide_Gross'])), num=20, base=np.e)
bins
```

Out[57]:
```
array([1.00000000e+00, 3.14018485e+00, 9.86076088e+00, 3.09646119e+01,
       9.72346052e+01, 3.05334634e+02, 9.58807191e+02, 3.01083182e+03,
       9.45456845e+03, 2.96890926e+04, 9.32292387e+04, 2.92757043e+05,
       9.19311230e+05, 2.88680720e+06, 9.06510822e+06, 2.84661155e+07,
       8.93888645e+07, 2.80697558e+08, 8.81442219e+08, 2.76789150e+09])
```

Now we can plot a histogram with log-bin. Set both axis to be log-scale.

In [58]:
```
ax = (movies["Worldwide_Gross"]+1.0).hist(bins=bins)
ax.set_yscale('log')
ax.set_xscale('log')
ax.set_xlabel("World wide gross")
ax.set_ylabel("Frequency")
```

Out[58]:
```
Text(0, 0.5, 'Frequency')
```



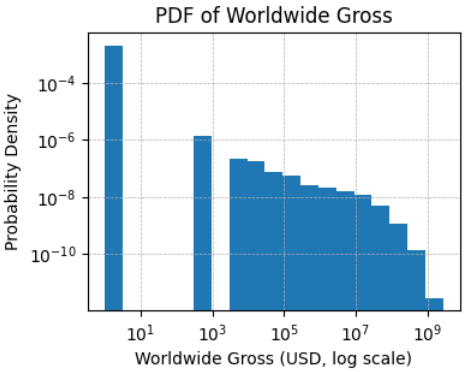
What is going on? Is this the right plot?

Q: explain and fix

In [ ]:
```
# Raw frequency doesn't scale well when data is highly skewed.
# The huge range of values (from a few dollars to billions) means that bins
# on the left can look empty or make spikes due to outliers or noise.
# The spike at low values could just be a lot of zeros or near-zeros, which is not meaningful.

plt.figure(figsize=(4,3))
plt.hist(movies['Worldwide_Gross'], bins=bins, density=True, log=True)  # log=True for y-axis
plt.xscale('log')  # log scale for x-axis

plt.xlabel('Worldwide Gross (USD, log scale)')
plt.ylabel('Probability Density')
plt.title('PDF of Worldwide Gross')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```



Q: Can you explain the plot? Why are there gaps?

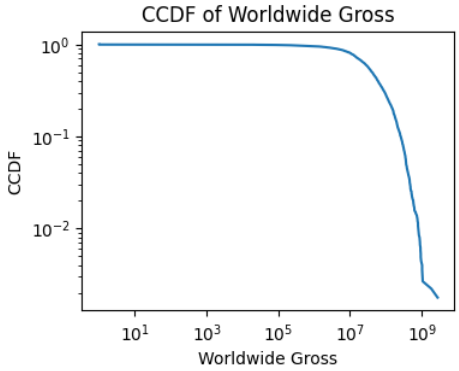
In [ ]:
```
# Logarithmic Scale amplifies sparse data.
# Using a log scale, small differences at the low end can end into big gaps on the graph.
# It stretches out the lower-value range, so even small data voids appear as noticeable spaces.
```

CCDF

CCDF is a nice alternative to examine distributions with heavy tails. The idea is same as CDF, but the direction of aggregation is opposite. For a given value x, CCDF(x) is the number (fraction) of data points that are same or larger than x. To write code to draw CCDF, it'll be helpful to draw it by hand by using a very small, toy dataset. Draw it by hand and then think about how each point in the CCDF plot can be computed.

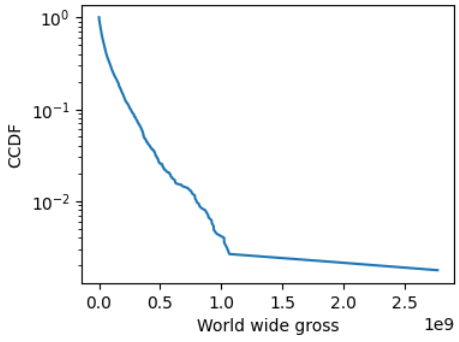
Q: Draw a CCDF of worldwide gross data in log-log scale

In [109]:
```
gross = movies['Worldwide_Gross']
worldgross_sorted = np.sort(gross)
Y = ccdf = 1.0 - np.arange(1, len(worldgross_sorted) + 1) / len(worldgross_sorted)
plt.figure(figsize=(4, 3))
plt.plot(worldgross_sorted, ccdf)
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Worldwide Gross')
plt.ylabel('CCDF')
plt.title('CCDF of Worldwide Gross')
plt.show()
```



We can also try in semilog scale (only one axis is in a log-scale), where the horizontal axis is linear.

In [111]:
```
plt.figure(figsize=(4,3))
plt.xlabel("World wide gross")
plt.ylabel("CCDF")
plt.plot(worldgross_sorted,Y)
plt.yscale('log')
```



A straight line in semilog scale means exponential decay (cf. a straight line in log-log scale means power-law decay). So it seems like the amount of money a movie makes across the world follows *roughly* an exponential distribution, while there are some outliers that make insane amount of money.

Q: Which is the most successful movie in our dataset?

You can use the following

- `idxmax()` : https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.idxmax.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.idxmax.html)
- `loc` : https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html)

In [113]:
```
idx = movies['Worldwide_Gross'].idxmax()
best_movie = movies.loc[idx]
best_movie
```

Out[113]:
```
Title                              Avatar
US_Gross                       760167650.0
Worldwide_Gross               2767891500.0
US_DVD_Sales                   146153933.0
Production_Budget              237000000.0
Release_Date                  Dec 18 2009
MPAA_Rating                         PG-13
Running_Time_min                      NaN
Distributor               20th Century Fox
Source                  Original Screenplay
Major_Genre                        Action
Creative_Type              Science Fiction
Director                     James Cameron
Rotten_Tomatoes_Rating                83.0
IMDB_Rating                            8.3
IMDB_Votes                        261439.0
Name: 1234, dtype: object
```

Paired t-test

- Dependencies: scipy pandas seaborn pingouin

In [115]:
```
import pandas as pd
import scipy.stats as stats
import pingouin as pg
```

Paired Sample T-test

The paired sample t-test is also known as the dependent sample t-test, and paired t-test. This type of t-test compares two averages (means) and will give you information if the difference between these two averages are zero. In a paired sample t-test, each participant is measured twice, which results in pairs of observations. The t-test also tells you whether the differences are statistically significant. In other words it lets you know if those differences could have happened by chance.

Q: When to use this test?

For example, if clinical psychologists want to test whether a treatment for depression will change the quality of life, they might set up an experiment. In this experiment, they will collect information about the participants' quality of life before the intervention. They are conducting a pre- and post-test study. In the pre-test the average quality of life might be 3, while in the post-test the average quality of life might be 5. Numerically, we could think that the treatment is working. However, it could be due to a fluke and, in order to test this, the clinical researchers can use the paired sample t-test.

Hypotheses

Now, when performing dependent sample t-tests you typically have the following two hypotheses:

1. Null hypotheses: the true mean difference is equal to zero (between the observations), i.e. the means are equal;
2. Alternative hypotheses: the true mean difference is not equal to zero (two-tailed), i.e. the means are different.

Note, in some cases we also may have a specific idea, based on theory, about the direction of the measured effect. For example, we may strongly believe (due to previous research and/or theory) that a specific intervention should have a positive effect. In such a case, the alternative hypothesis will be something like: the true mean difference is greater than zero (one-tailed). Note, it can also be smaller than zero, of course.

Assumptions

Besides that the dependent variable is on interval/ratio scale, and is continuous, there are three assumptions that need to be met:

1. Are the two samples independent?
2. Does the data, i.e., the differences for the matched-pairs, follow a normal distribution?
3. Do the two samples have the same variances?

If your data is not following a normal distribution you can transform your dependent variable using square root, log, or Box-Cox in Python. In the next section, we will import data.

data

In [116]:
```
data = 'https://gist.githubusercontent.com/baskaufs/1a7a995c1b25d6e88b45/raw/4bb17ccc5c1e62c27627833a4f25380f27d30b35/t-test.csv'
df = pd.read_csv(data)

df.head()
```

Out[116]:

	grouping	height
0	men	181.5
1	men	187.3
2	men	175.3
3	men	178.3
4	men	169.0

In [117]:
```
# subset the data
male = df.query('grouping == "men"')['height']
female = df.query('grouping == "women"')['height']
```

Descriptive statistics: use the groupby method together with the describe method to calculate summary statistics. As we are interested in the difference between 'A' and 'B', in the dataset, we used 'grouping' as input to the groupby method.

In [118]:
```
df.groupby('grouping').describe()
```

Out[118]:

	height							
	count	mean	std	min	25%	50%	75%	max
grouping								
men	7.0	179.871429	6.216836	169.0	176.80	181.5	183.85	187.3
women	7.0	171.057143	5.697619	165.2	166.65	170.3	173.75	181.1

Two-Sample T-Test with Pingouin

In [119]:
```
pg.ttest(male, female, paired=True)
```

Out[119]:

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	2.85249	6	two-sided	0.029087	[1.25, 16.38]	1.478192	2.973	0.899761

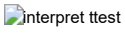
- Interpreting the P-value

Now, the p-value of the test is 0.029087, which is less than the significance level alpha (e.g., 0.05). Furthermore, this means that we can conclude that the men's average height is statistically different from the female's average height.

Specifically, a p-value is a probability of obtaining an effect at least as extreme as the one in the data you have obtained (i.e., your sample), assuming that the null hypothesis is true. Moreover, p-values address only one question which is concerned about how likely your collected data is, assuming a true null hypothesis? Importantly, it cannot be used as support for the alternative hypothesis.

- Interpreting the Effect Size (Cohen's D)

One common way to interpret Cohen's D that is obtained in a t-test is in terms of the relative strength of e.g. the condition. Cohen (1988) suggested that d=0.2 should be considered a 'small' effect size, 0.5 is a 'medium' effect size, and that 0.8 is a 'large' effect size. This means that if two groups' means don't differ by 0.2 standard deviations or more, the difference is trivial, even if it is statistically significant.



Draw the data

In [ ]:
```
import seaborn as sns
# somehow seaborn failed with " to_dict() takes from 1 to 2 positional arguments but 4 were given"
# so I rewrote this
#sns.boxplot(data=[male, female])
plt.boxplot([male, female])
plt.xticks(ticks=[1,2], labels=['male', 'female'])
```

Out[ ]:
```
([<matplotlib.axis.XTick at 0x247dee76350>,
  <matplotlib.axis.XTick at 0x247dee76320>],
 [Text(1, 0, 'male'), Text(2, 0, 'female')])
```