1. Bitcoin. Cryptocurrencies. So hot right now.

Since the launch of Bitcoin in 2008 (https://newfronttest.bitcoin.com/bitcoin.pdf), hundreds of similar projects based on the blockchain technology have emerged. We call these cryptocurrencies (also coins or cryptos in the Internet slang). Some are extremely valuable nowadays, and others may have the potential to become extremely valuable in the future[1]. In fact, the 6th of December of 2017 Bitcoin has a market capitalization (https://en.wikipedia.org/wiki/Market_capitalization) above $200 billion.

The astonishing increase of Bitcoin market capitalization in 2017.

*[1]- **WARNING**: The cryptocurrency market is exceptionally volatile and any money you put in might disappear into thin air. Cryptocurrencies mentioned here **might be scams** similar to Ponzi Schemes (https://en.wikipedia.org/wiki/Ponzi_scheme) or have many other issues (overvaluation, technical, etc.). **Please do not mistake this for investment advice**. *

That said, let's get to business. As a first task, we will load the current data from the coinmarketcap API (https://api.coinmarketcap.com) and display it in the output.

In [2]:

```
# Importing pandas
import pandas as pd

# Importing matplotlib and setting aesthetics for plotting later.
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
plt.style.use('fivethirtyeight')

# Reading in current data from coinmarketcap.com

current = pd.read_json("https://api.coinmarketcap.com/v1/ticker/")
# Printing out the first few lines
print(current.head())
```

```
   24h_volume_usd  available_supply           id  last_updated  \
0      9327610000          16876462      bitcoin    1519159767
1      2395970000          97738733     ethereum    1519159751
2       703389000       39009215838       ripple    1519159741
3       735627000          16978250  bitcoin-cash    1519159752
4      1606890000          55305908      litecoin    1519159741

   market_cap_usd    max_supply         name  percent_change_1h  \
0    200308415124  2.100000e+07      Bitcoin               0.82
1     90777584609           NaN     Ethereum              -0.24
2     43920476112  1.000000e+11       Ripple              -0.17
3     25769078503  2.100000e+07  Bitcoin Cash               0.49
4     13715201551  8.400000e+07      Litecoin               1.23

   percent_change_24h  percent_change_7d  price_btc   price_usd  ran
k symbol  \
0                5.91              37.13   1.000000  11869.1000
1    BTC
1               -1.77               9.62   0.079391    928.7780
2    ETH
2               -1.74               9.18   0.000096      1.1259
3    XRP
3               -1.11              22.15   0.129737   1517.7700
4    BCH
4               10.13              54.50   0.021198    247.9880
5    LTC

   total_supply
0      16876462
1      97738733
2   99992725510
3      16978250
4      55305908
```

In [3]:

```
%%nose

import inspect
import pandas as pd

def test_current_is_a_data_frame():
    assert type(current) == pd.core.frame.DataFrame, \
    'The variable current should contain the DataFrame produced by read_json()'

def test_pandas_imported():
    assert inspect.ismodule(pd), 'Do not delete the "from pandas import pd" impo
rt'

def test_plt_imported():
    assert inspect.ismodule(plt), 'Do not delete the "import matplotlib.pyplot a
s plt "import'
```

Out[3]:

```
3/3 tests passed
```

2. Full dataset, filtering, and reproducibility

The previous API call returns only the first 100 coins, and we want to explore as many coins as possible. Moreover, we can't produce reproducible analysis with live online data. To solve these problems, we will load a CSV we conveniently saved on the 6th of December of 2017 using the API call `https://api.coinmarketcap.com/v1/ticker/?limit=0` named `datasets/coinmarketcap_06122017.csv`.

In [4]:

```
# Reading datasets/coinmarketcap_06122017.csv into pandas
dec6 = pd.read_csv('datasets/coinmarketcap_06122017.csv')

# Selecting the 'id' and the 'market_cap_usd' columns
market_cap_raw = dec6[['id','market_cap_usd']]

# Counting the number of values
print(market_cap_raw.count())
```

```
id               1326
market_cap_usd    1031
dtype: int64
```

In [5]:

```
%%nose

def test_dec6_is_dataframe():
    assert type(dec6) == pd.core.frame.DataFrame, \
    'The variable dec6 should contain the DataFrame produced by read_csv()'

def test_market_cap_raw():
    assert list(market_cap_raw.columns) == ['id', 'market_cap_usd'], \
    'The variable market_cap_raw should contain the "id" and "market_cap_usd" co
lumns exclusively'
```

Out[5]:

```
2/2 tests passed
```

3. Discard the cryptocurrencies without a market capitalization

Why do the `count()` for `id` and `market_cap_usd` differ above? It is because some cryptocurrencies listed in coinmarketcap.com have no known market capitalization, this is represented by `NaN` in the data, and `NaN`s are not counted by `count()`. These cryptocurrencies are of little interest to us in this analysis, so they are safe to remove.

In [6]:

```
# Filtering out rows without a market capitalization
cap = market_cap_raw.query('market_cap_usd > 0')

# Counting the number of values again
print(market_cap_raw.count())
```

```
id                1326
market_cap_usd    1031
dtype: int64
```

In [7]:

```
%%nose

def test_cap_filtered():
    assert cap.id.count() == cap.market_cap_usd.count(), 'id and market_cap_usd
should have the same count'

def test_cap_small():
    assert cap.id.count() == 1031, 'The resulting amount of cryptos should be 10
31'
```

Out[7]:

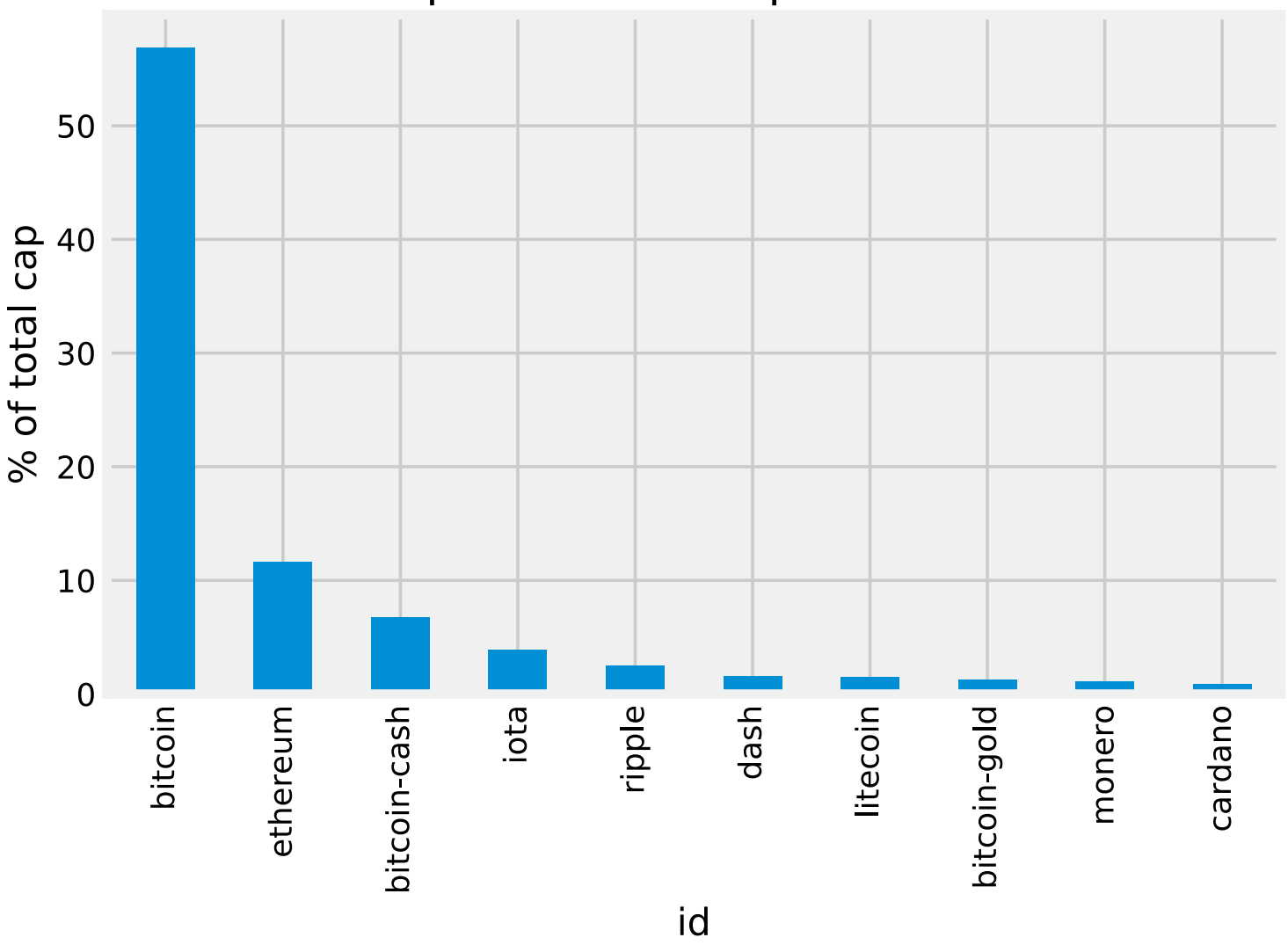
```
2/2 tests passed
```

4. How big is Bitcoin compared with the rest of the cryptocurrencies?

At the time of writing, Bitcoin is under serious competition from other projects, but it is still dominant in market capitalization. Let's plot the market capitalization for the top 10 coins as a barplot to better visualize this.

In [8]:

```
#Declaring these now for later use in the plots
TOP_CAP_TITLE = 'Top 10 market capitalization'
TOP_CAP_YLABEL = '% of total cap'

# Selecting the first 10 rows and setting the index
cap10 = market_cap_raw.set_index('id').head(n=10)

# Calculating market_cap_perc
cap10 = cap10 = cap10.assign(market_cap_perc =
    lambda x: (x.market_cap_usd / cap.market_cap_usd.sum()) * 100)

# Plotting the barplot with the title defined above
ax = cap10.market_cap_perc.plot(kind='bar',title='Top 10 market capitalization')
plt.ylabel('% of total cap')
# Annotating the y axis with the label defined above

plt.show()
```



In [9]:

```
%%nose

def test_len_cap10():
    assert len(cap10) == 10, 'cap10 needs to contain 10 rows'

def test_index():
    assert cap10.index.name == 'id', 'The index should be the "id" column'

def test_perc_correct():
    assert round(cap10.market_cap_perc.iloc[0], 2) == 56.92, 'the "market_cap_pe
rc" formula is incorrect'

def test_title():
    assert ax.get_title() == TOP_CAP_TITLE, 'The title of the plot should be {}'
.format(TOP_CAP_TITLE)

def test_ylabel():
    assert ax.get_ylabel() == TOP_CAP_YLABEL, 'The y-axis should be named {}'.fo
rmat(TOP_CAP_YLABEL)
```

Out[9]:

```
5/5 tests passed
```

5. Making the plot easier to read and more informative

While the plot above is informative enough, it can be improved. Bitcoin is too big, and the other coins are hard to distinguish because of this. Instead of the percentage, let's use a $\log^{10}$ scale of the "raw" capitalization. Plus, let's use color to group similar coins and make the plot more informative[1].

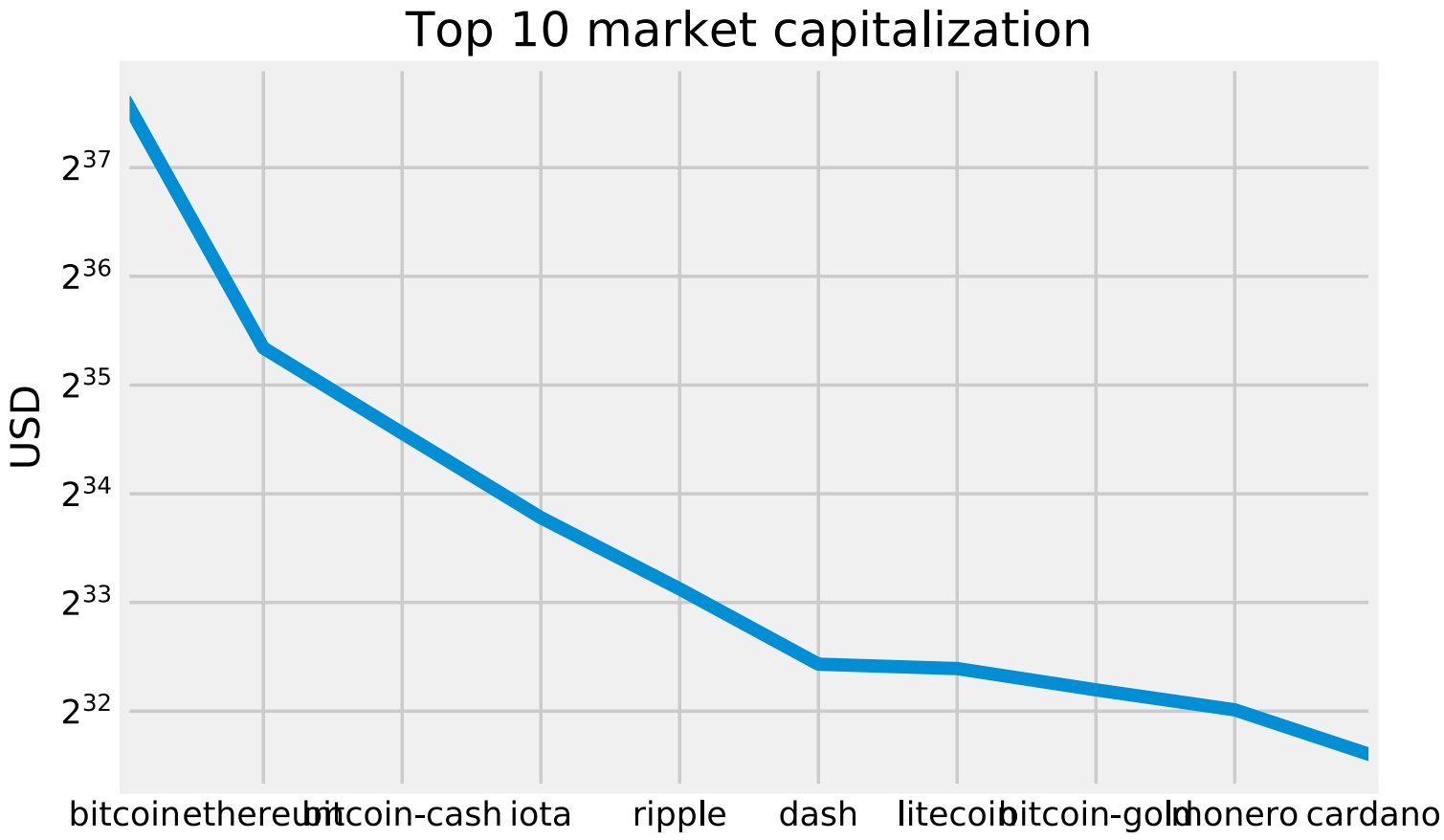
For the colors rationale: bitcoin-cash and bitcoin-gold are forks of the bitcoin blockchain (https://en.wikipedia.org/wiki/Blockchain)[2]. Ethereum and Cardano both offer Turing Complete smart contracts (https://en.wikipedia.org/wiki/Smart_contract). Iota and Ripple are not minable. Dash, Litecoin, and Monero get their own color.

[1] *This coloring is a simplification. There are more differences and similarities that are not being represented here.*

[2] *The bitcoin forks are actually **very** different, but it is out of scope to talk about them here. Please see the warning above and do your own research.*

In [10]:

```
# Colors for the bar plot
COLORS = ['orange', 'green', 'orange', 'cyan', 'cyan', 'blue', 'silver', 'orange
', 'red', 'green']

# Plotting market_cap_usd as before but adding the colors and scaling the y-axis
ax = cap10.market_cap_usd.plot()
ax.set_yscale('log', basey=2)
plt.ylabel("USD")
plt.xlabel('')
plt.title('Top 10 market capitalization')
# Annotating the y axis with 'USD'
plt.show()
```



In [11]:

```
%%nose

def test_title():
    assert ax.get_title() == TOP_CAP_TITLE, 'The title of the plot should be {}'
.format(TOP_CAP_TITLE)

def test_ylabel():
    assert ax.get_ylabel() == 'USD', 'The y-axis should be named {}'.format(TOP_
CAP_YLABEL)

def test_xlabel():
    assert not ax.get_xlabel(), 'The X label should contain an empty string, cur
rently it contains "{}"'.format(ax.get_xlabel())

def test_log_scale():
    assert ax.get_yaxis().get_scale() == 'log', \
    'The y-axis is not on a log10 scale. Do not transform the data yourself, use
the pandas/matplotlib interface'

#def test_colors():
#    assert round(ax.patches[1].get_facecolor()[1], 3) == 0.502, 'The colors of
the bars are not correct'
```

Out[11]:

```
4/4 tests passed
```

6. What is going on?! Volatility in cryptocurrencies

The cryptocurrencies market has been spectacularly volatile since the first exchange opened. This notebook didn't start with a big, bold warning for nothing. Let's explore this volatility a bit more! We will begin by selecting and plotting the 24 hours and 7 days percentage change, which we already have available.

In [12]:

```
# Selecting the id, percent_change_24h and percent_change_7d columns
volatility = dec6[['id', 'percent_change_24h', 'percent_change_7d']]

# Setting the index to 'id' and dropping all NaN rows
volatility = volatility.set_index('id').dropna()

# Sorting the DataFrame by percent_change_24h in ascending order
volatility = volatility.sort_values('percent_change_24h',ascending=True)

# Checking the first few rows
volatility.head()
```

Out[12]:

	percent_change_24h	percent_change_7d
id		
flappycoin	-95.85	-96.61
credence-coin	-94.22	-95.31
coupecoin	-93.93	-61.24
tyrocoin	-79.02	-87.43
petrodollar	-76.55	542.96

In [13]:

```
%%nose

def test_vol():
    assert list(volatility.columns) == ['percent_change_24h', 'percent_change_7d
'], '"volatility" not loaded correctly'

def test_vol_index():
    assert list(volatility.index[:3]) == ['flappycoin', 'credence-coin', 'coupec
oin'], \
    '"volatility" index is not set to "id", or data sorted incorrectly'
```

Out[13]:

```
2/2 tests passed
```

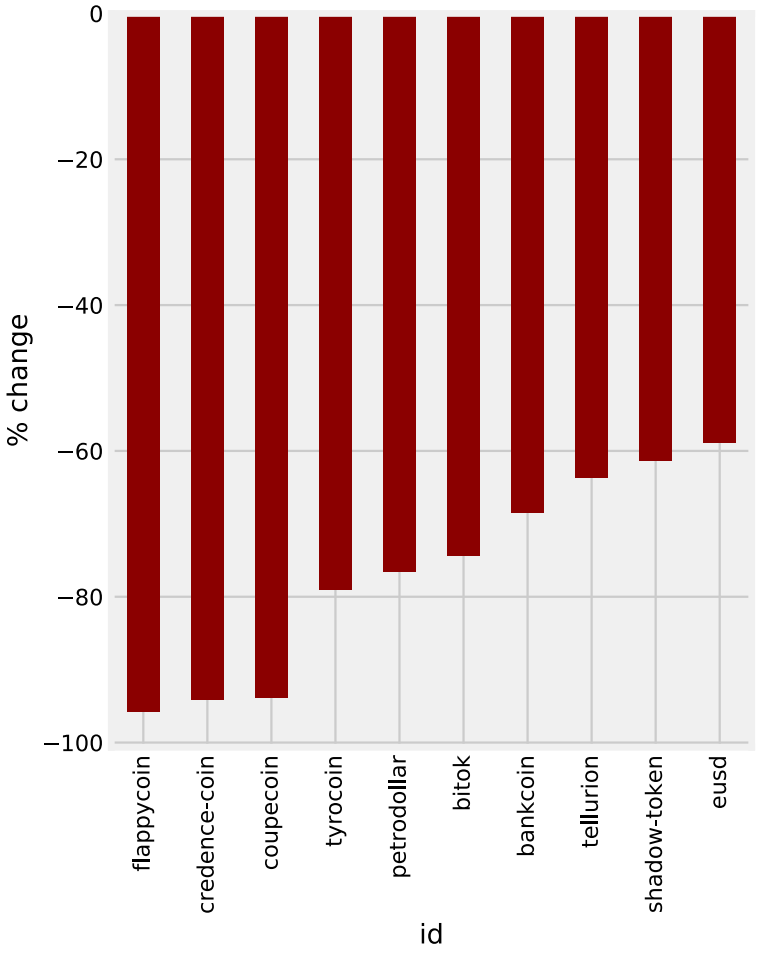
7. Well, we can already see that things are *a bit* crazy

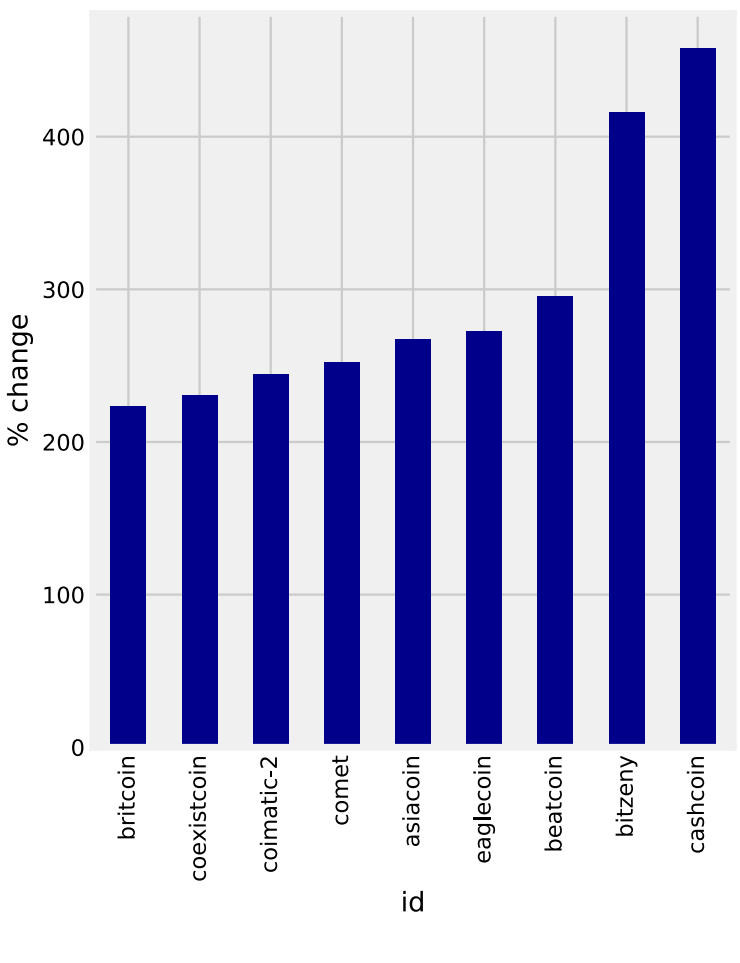
It seems you can lose a lot of money quickly on cryptocurrencies. Let's plot the top 10 biggest gainers and top 10 losers in market capitalization.

In [14]:

```
#Defining a function with 2 parameters, the series to plot and the title
def top10_subplot(volatility_series, title):
    # Making the subplot and the figure for two side by side plots
    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 6))

    # Plotting with pandas the barchart for the top 10 losers
    ax =(volatility_series[:10].plot.bar(color="darkred",ax=axes[0]))

    # Setting the figure's main title to the text passed as parameter
    fig.suptitle(title)

    # Setting the ylabel to '% change'
    ax.set_ylabel('% change')

    # Same as above, but for the top 10 winners
    ax = (volatility_series[-10:-1].plot.bar(color="darkblue",ax=axes[1]))
    fig.suptitle(title)
    ax.set_ylabel('% change')

    # Returning this for good practice, might use later
    return fig, ax

DTITLE = "24 hours top losers and winners"

# Calling the function above with the 24 hours period series and title DTITLE
fig, ax = top10_subplot(volatility.percent_change_24h,DTITLE)
```

24 hours top losers and winners





In [15]:

```
%%nose

DTITLE = "24 hours top losers and winners"

def test_title():
    assert fig.get_children()[-1].get_text() == DTITLE, 'The title of the plot s
hould be {}'.format(DTITLE)

def test_subplots():
    assert len(fig.get_axes()) == 2, 'The plot should have 2 subplots'

def test_ylabel():
    fig.get_axes()[0].get_ylabel() == '% change', 'y axis label should be set to
% change'

def test_comet_coin():
    assert round(fig.get_children()[2].get_children()[3].get_height(), 0) == 252
.0, \
    'The data on the winners plot is incorrect'

def test_tyrocoin():
    assert abs(round(fig.get_children()[1].get_children()[4].get_height(), 0)) =
= 77.0, \
    'The data on the losers plot is incorrect'

#def test_colors():
#    r, g, b, a = fig.get_axes()[0].patches[1].get_facecolor()
#    assert round(r, 1) and not round(g, 1) and not round(b, 1), 'The bars on th
e left plot are not red'

#def test_colors2():
#    r, g, b, a = fig.get_axes()[1].patches[1].get_facecolor()
#    assert not round(r, 1) and not round(g, 1) and round(b, 1), 'The bars on th
e left plot are not blue'
```

Out[15]:

```
5/5 tests passed
```

8. Ok, those are... interesting. Let's check the weekly Series too.

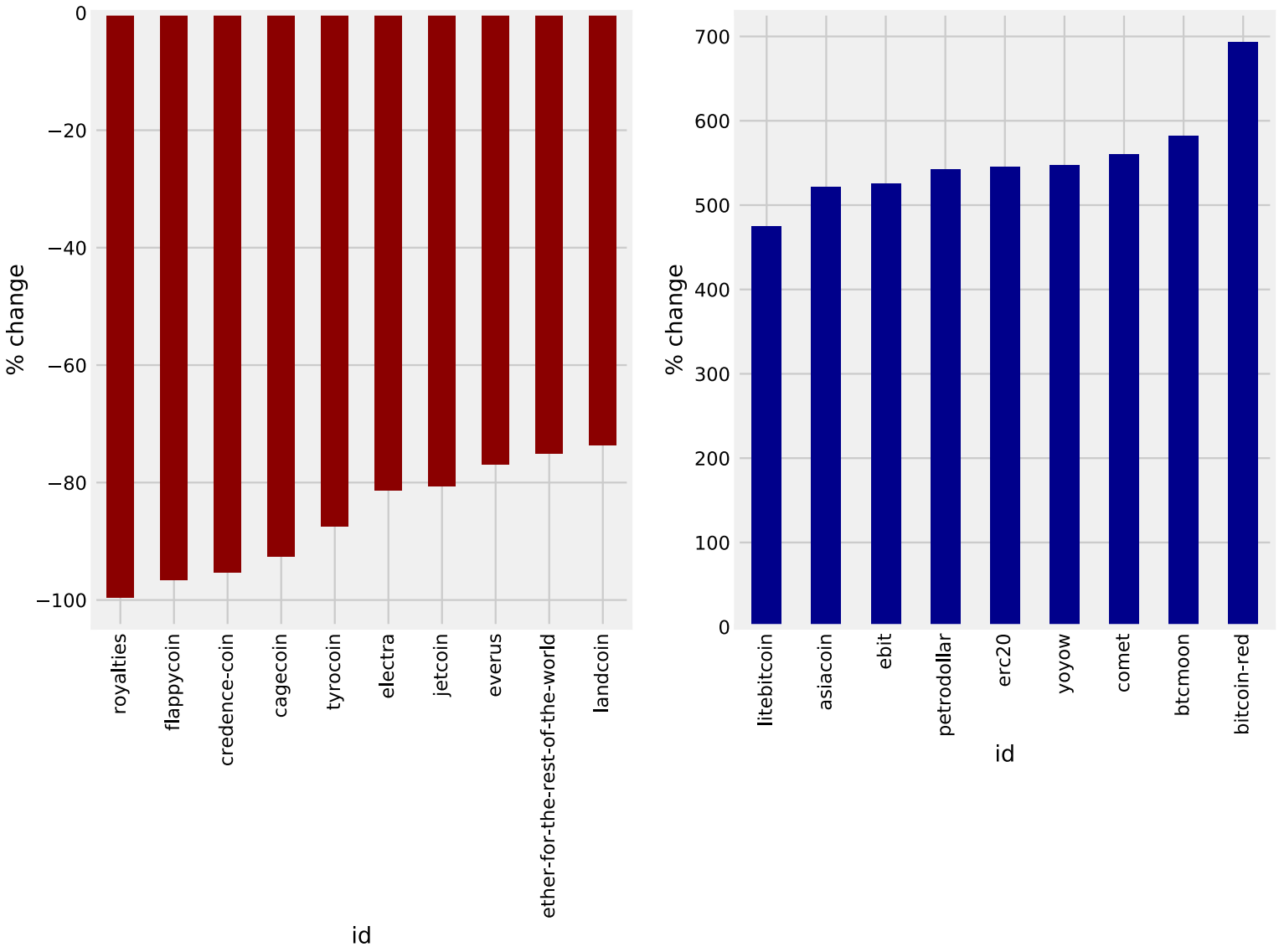
800% daily increase?! Why are we doing this tutorial and not buying random coins?[1]

After calming down, let's reuse the function defined above to see what is going weekly instead of daily.

[1] *Please take a moment to understand the implications of the red plots on how much value some cryptocurrencies lose in such short periods of time*

In [16]:

```
# Sorting in ascending order
volatility7d = volatility.sort_values('percent_change_7d',ascending=True)

WTITLE = "Weekly top losers and winners"

# Calling the top10_subplot function
fig, ax = top10_subplot(volatility7d.percent_change_7d,WTITLE)
```



In [17]:

```
%%nose

WTITLE = "Weekly top losers and winners"

def test_title():
    assert fig.get_children()[-1].get_text() == WTITLE, 'The title of the plot s
hould be {}'.format(WTITLE)

def test_subplots():
    assert len(fig.get_axes()) == 2, 'The plot should have 2 subplots'

def test_ylabel():
    fig.get_axes()[0].get_ylabel() == '% change', "y axis label should be set to
'% change'"

#def test_colors():
#    r, g, b, a = fig.get_axes()[0].patches[1].get_facecolor()
#    assert round(r, 1) and not round(g, 1) and not round(b, 1), 'The bars on th
e left plot are not red'
#
#def test_colors2():
#    r, g, b, a = fig.get_axes()[1].patches[1].get_facecolor()
#    assert not round(r, 1) and not round(g, 1) and round(b, 1), 'The bars on th
e left plot are not blue'

def test_comet_coin():
    assert abs(round(fig.get_children()[2].get_children()[3].get_height(), 0)) =
= 543.0, \
    'The data on the gainers plot is incorrect'

def test_tyrocoin():
    assert abs(round(fig.get_children()[1].get_children()[4].get_height(), 0)) =
= 87.0, \
    'The data on the losers plot is incorrect'
```

Out[17]:

```
5/5 tests passed
```

9. How small is small?

The names of the cryptocurrencies above are quite unknown, and there is a considerable fluctuation between the 1 and 7 days percentage changes. As with stocks, and many other financial products, the smaller the capitalization, the bigger the risk and reward. Smaller cryptocurrencies are less stable projects in general, and therefore even riskier investments than the bigger ones[1]. Let's classify our dataset based on Investopedia's capitalization definitions (https://www.investopedia.com/video/play/large-cap/) for company stocks.

[1] *Cryptocurrencies are a new asset class, so they are not directly comparable to stocks. Furthermore, there are no limits set in stone for what a "small" or "large" stock is. Finally, some investors argue that bitcoin is similar to gold, this would make them more comparable to a commodity (https://www.investopedia.com/terms/c/commodity.asp) instead.*

In [18]:

```
# Selecting everything bigger than 10 billion
largecaps = market_cap_raw.query('market_cap_usd > 10000000000')

# Printing out largecaps
print(largecaps)
```

```
           id  market_cap_usd
0     bitcoin    2.130493e+11
1    ethereum    4.352945e+10
2  bitcoin-cash    2.529585e+10
3        iota    1.475225e+10
```

In [19]:

```
%%nose

def test_large():
    assert not largecaps.market_cap_usd.count() < 4, 'You filtered too much'

def test_small():
    assert not largecaps.market_cap_usd.count() > 4, "You didn't filter enough"

def test_order():
    assert largecaps.iloc[1].id == "ethereum", "The dataset is not in the right
order, no need to manipulate it here"
```

Out[19]:

```
3/3 tests passed
```

10. Most coins are tiny

Note that many coins are not comparable to large companies in market cap, so let's divert from the original Investopedia definition by merging categories.

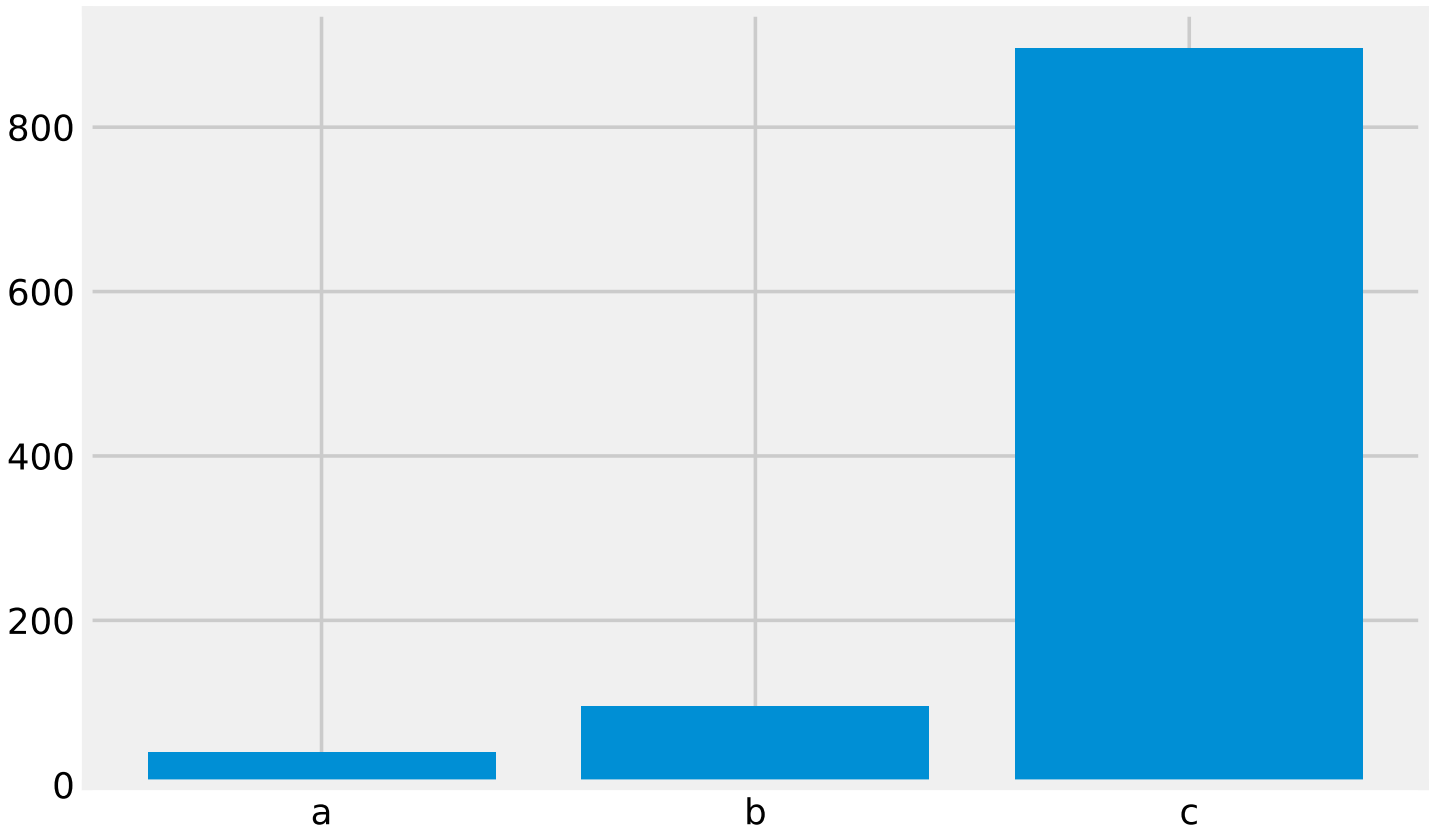
This is all for now. Thanks for completing this project!

In [20]:

```
# Making a nice function for counting different marketcaps from the
# "cap" DataFrame. Returns an int.
# INSTRUCTORS NOTE: Since you made it to the end, consider it a gift :D
def capcount(query_string):
    return cap.query(query_string).count().id

# Labels for the plot
LABELS = ["biggish", "micro", "nano"]

# Using capcount count the biggish cryptos
biggish = capcount('market_cap_usd > 300000000')

# Same as above for micro ...
micro = capcount('market_cap_usd > 50000000 and market_cap_usd < 300000000' )

# ... and for nano
nano =  capcount('market_cap_usd < 50000000')

# Making a list with the 3 counts
values = [biggish,micro,nano]

# Plotting them with matplotlib
import matplotlib.pyplot as plt
labels = ["a", "b", "c"]
plt.bar(range(len(values)), values, tick_label=labels)
```

Out[20]:

```
<Container object of 3 artists>
```



In [21]:

```
%%nose

def test_biggish():
    assert biggish == 39, 'biggish is not correct'

def test_micro():
    assert micro == 96, 'micro is not correct'

def test_nano():
    assert nano == 896, 'nano is not correct'

def test_lenvalues():
    assert len(values) == 3, "values list is not correct"
```

Out[21]:

```
4/4 tests passed
```