# Sussy Stock Trades

Prabina Pokharel, Atharva Kulkarni

# Summary of Findings

## Introduction

This dataset has been maintained by Timothy Carambat as part of the House Stock Watcher project. Each row in this dataset corresponds to one stock trasanction that was carried out by a member of the U.S. House of Representatives.

There are several features, such as:

`disclosure_date` `transaction_date`
`ticker`
`asset_description`
`amount`
`representative`

and several more...

For reference, the question we are going to be exploring is:

**Are representatives from the state of Michigan more likely to have an average trade proportion that result in capital gains over 200 usd higher than the rest of the population?**

So in order to be able to answer this question, let's identify a few important columns:

`cap_gains_over_200_usd`

- Describes if the particular trade resulted in a capital gain of over $200.00

and

`state`

- we will create in the cleaning section

## Cleaning and EDA

Here's an overview of our cleaning steps:

- Changing `disclosure_year` and `transaction_date` to pandas datetime object
  - This was handy to create, as we could simply subtract the two columns to create a new new `non_disclosure_period(days)` column.
- Removed the 'Hon.' out of representatives' names
  - To make it easier to read.
- Added a `state` column, created from `district`
  - Necessary in order to be able to answer our main question!
  - Also useful to create cool aggregates by state!
- Added an `amount_cleaned` column, created from `amount`
  - This was important in order to be able to do any sort of math calculations with relation to amount of the transaction value.

## Assessment of Missingness

We picked `owner` has the column to analyze missingness for because it contained the most missing values. We ran permutation testing to analyze whether the missigness of `owner` is depended on any columns.

We used ks as test stat for permutation test for two numerical columns `disclosure_year` and `amount_cleaned` and we used TVD as test stat for one categorical column `state` .

For all the three columns, our p-values were very close to 0.0, which suggests that the distribution of all three columns when `owner` is missing and the distribution of all three columns when `owner` is not missing are most likely different. This means that the missingness of `owner` depends on those columns individually.

## Hypothesis Test

- **Question:**

The reps from the state of Michigan have 77% of trades that result in the `cap_gains_over_200_usd` column having a True value.

Is this just by chance, or are the representatives from some states, like *The Great Lakes State*, just that damn much better at trading then representatives from other states?

- **Null hypothesis:** Reps from Michigan *are not* more likely to have an average trade proportion that result in capital gains over $200 higher than the rest of the population.

- **Alternative Hypothesis:** Reps from Michigan *are* more likely to have an average trade proportion that result in capital gains over $200 higher than the rest of the population.

- **Test statistic:** Difference of means between average value of `cap_gains_over_200_usd` column for Michigan reps vs average value of `cap_gains_over_200_usd` column for non-Michigan reps.

    - Difference of means is appropriate here because we have numerical/quantitiative data, and we cn have just one aggregation statistic, for each Michigan only group, and non-Michigan only group.
- **Significance level:** 1%

    - Why 1% vs 5%? No particular reason, we just want to have results that are as statistically significant as possible.

Here's our setup for the hypothesis (permutation) test:

- Calculate the observed test statistic, the difference in means between the average proportion of trades that result in capital gains over $200 for Michigan representatives, vs. non-Michigan representatives.
- Shuffle the `cap_gains_over_200_usd` in a dataframe, and store the resulting test stat in a list. Repeat 500 times.
- Calculate the p-value.

Describe the setup and results of your hypothesis test. Make sure to explicitly state your null and alternative hypotheses, test statistic, the significance level you used, and what conclusions you can draw from the results.

# Code

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

# Cleaning and EDA

## Data Cleaning

A note for this section: we decided to carry out cleaning and EDA before formulating our question, hoping that an interesting question would dawn upon us while doing EDA, so our cleaning will include more steps than necessary than required for eventual hypothesis testing.

In [2]:
```
df = pd.read_csv('data/all_transactions.csv')
# splitting up so dataframe is displayable in the final pdf :)
display(df.iloc[:, :8].head())
display(df.iloc[:, 8:].head())
```

| | disclosure_year | disclosure_date | transaction_date | owner | ticker | asset_description | |
|---|---|---|---|---|---|---|---|
| **0** | 2021 | 10/04/2021 | 2021-09-27 | joint | BP | BP plc | purc |
| **1** | 2021 | 10/04/2021 | 2021-09-13 | joint | XOM | Exxon Mobil Corporation | purc |
| **2** | 2021 | 10/04/2021 | 2021-09-10 | joint | ILPT | Industrial Logistics Properties Trust - Common... | purc |
| **3** | 2021 | 10/04/2021 | 2021-09-28 | joint | PM | Phillip Morris International Inc | purc |
| **4** | 2021 | 10/04/2021 | 2021-09-17 | self | BLK | BlackRock Inc | sale_pa |

| | representative | district | ptr_link | cap_gains_over_200_usd |
|---|---|---|---|---|
| **0** | Hon. Virginia Foxx | NC05 | https://disclosures-clerk.house.gov/public_dis... | False |
| **1** | Hon. Virginia Foxx | NC05 | https://disclosures-clerk.house.gov/public_dis... | False |
| **2** | Hon. Virginia Foxx | NC05 | https://disclosures-clerk.house.gov/public_dis... | False |
| **3** | Hon. Virginia Foxx | NC05 | https://disclosures-clerk.house.gov/public_dis... | False |
| **4** | Hon. Alan S. Lowenthal | CA47 | https://disclosures-clerk.house.gov/public_dis... | False |

Let's look at the types of the columns to see which ones may need to have their types changed.

In [3]:

```
df.dtypes
```

Out[3]:
```
disclosure_year            int64
disclosure_date           object
transaction_date          object
owner                     object
ticker                    object
asset_description         object
type                      object
amount                    object
representative            object
district                  object
ptr_link                  object
cap_gains_over_200_usd      bool
dtype: object
```

`disclosure_data` and `transaction_date` can be represented in date time, but they are normal objects right now.

Let's change them to pandas's `datetime` objects

In [4]:
```
# need to have errors = 'coerce', so some weird data gets set to np.nan
df['disclosure_date'] = pd.to_datetime(df['disclosure_date'], errors = 'coerc
df['transaction_date'] = pd.to_datetime(df['transaction_date'], errors = 'coe
df.dtypes
```

Out[4]:
```
disclosure_year                    int64
disclosure_date           datetime64[ns]
transaction_date          datetime64[ns]
owner                             object
ticker                            object
asset_description                 object
type                              object
amount                            object
representative                    object
district                          object
ptr_link                          object
cap_gains_over_200_usd              bool
dtype: object
```

This next step isn't super important for data analysis, but I'd say it is very important for some readability. If you can't read your data, what will you do??

Let's clean the 'Hon. ' out of every rep's name so it becomes easier to read.

In [5]:
```
# regex=False to get rid of regex depracated warning
df['representative'] = df['representative'].str.replace('Hon. ', '', regex=Fa
```

Let's create a `state` column so we can do groupby state if we need to in the future.

In [6]:
```
df['state'] = df['district'].str[:2]
df[['state']].head()
```

Out[6]:

| | state |
|---|---|
| **0** | NC |
| **1** | NC |
| **2** | NC |
| **3** | NC |
| **4** | CA |

Now let's deal with the ranges in the `amount` column. The ranges are an issue for 2 reasons:

- We can't do math with a range
- It's harder to read

So let's create a new column, `amount_cleaned`, that consists of the mean of the range given in `amount`.

And for values without a range, e.g. '$1,001 -', we'll deal with those after we fill in the ranges with averages.

In [7]:

```
# create a dictionary to map range values to their average value
mapped_values = {
    '$1,001 - $15,000':8_000.5,
    '$15,001 - $50,000':32_500.5,
    '$50,001 - $100,000':75_000.5,
    '$100,001 - $250,000':175_000.5,

    # place holder so we can calculate values by filling in average
    '$1,001 -':-99,

    '$250,001 - $500,000':375_000.5,
    '$500,001 - $1,000,000':375_000.5,
    '$1,000,001 - $5,000,000':3_000_000.5,

    # place holder so we can calculate values by filling in average
    '$1,000,000 +':-100,

    '$5,000,001 - $25,000,000':15_000_000.5,
    '$1,000 - $15,000':8_000,
    '$15,000 - $50,000':32_000,

    # we aren't giving 50,000,000+ a placeholder here, because we think
    # 50,000,000+ is already too much so let's just keep the average as is.
    '$50,000,000 +':50_000_000,
    '$1,000,000 - $5,000,000':3_000_000,
}
```

In [8]:

```
# replace range values with their average value
df['amount_cleaned'] = df['amount'].replace(mapped_values)
```

Now let's deal with the non-ranges, i.e., '$1,001 -' and '$1,000,000 +'

Let's find the average of `amount_cleaned` column when values are above $1,001 and $1,000,000 seperately, and then replace the respective values.

In [9]:

```
avg_above_1001 = round(df.loc[df['amount_cleaned'] >= 1_001]['amount_cleaned'
avg_above_1_mil = round(df.loc[df['amount_cleaned'] >= 1_000_000]['amount_cle

avg_above_1001, avg_above_1_mil
```

Out[9]:

```
(53267.76, 6163265.79)
```

In [10]:
```
# replace -99, which maps to '$1,001 -' in 'amounts' with avg_above_1001
df['amount_cleaned'] = df['amount_cleaned'].replace({-99: avg_above_1001})

# replace -100, which maps to '$1,000,000 +' in 'amounts' with avg_above_1_mi
df['amount_cleaned'] = df['amount_cleaned'].replace({-100: avg_above_1_mil})

# let's make sure the -99 and -100 values are gone!
(-99 or -100) in df['amount_cleaned'].value_counts()
```

Out[10]:
```
False
```

In [11]:
```
# let's make sure our newly added column is of type float
df.dtypes[-1]
```

Out[11]:
```
dtype('float64')
```

What if we wanted to examine how long it takes for reps to disclose their trade? Let's create a new column called `non_disclosure_period(days)` .

Our previous work of converting `disclosure_date` and `transaction_date` to `datetime` objects will also come in handy here, since we can simply subtract the 2 columns.

In [12]:
```
# assign a new column 'non_disclosure_period(days)' to be difference of 'disc
df['non_disclosure_period(days)'] = (df['disclosure_date'] - df['transaction_
df[['non_disclosure_period(days)']].head()
```

Out[12]:

| | **non_disclosure_period(days)** |
|---|---|
| **0** | 7.0 |
| **1** | 21.0 |
| **2** | 24.0 |
| **3** | 6.0 |
| **4** | 17.0 |

Let's see if there are any weird values in this new column.

In [13]:
```
df[df['non_disclosure_period(days)'] < 0].shape[0]
```

Out[13]:
```
13
```

There are 13 transactions where non disclosure days are negative. We will exclude those.

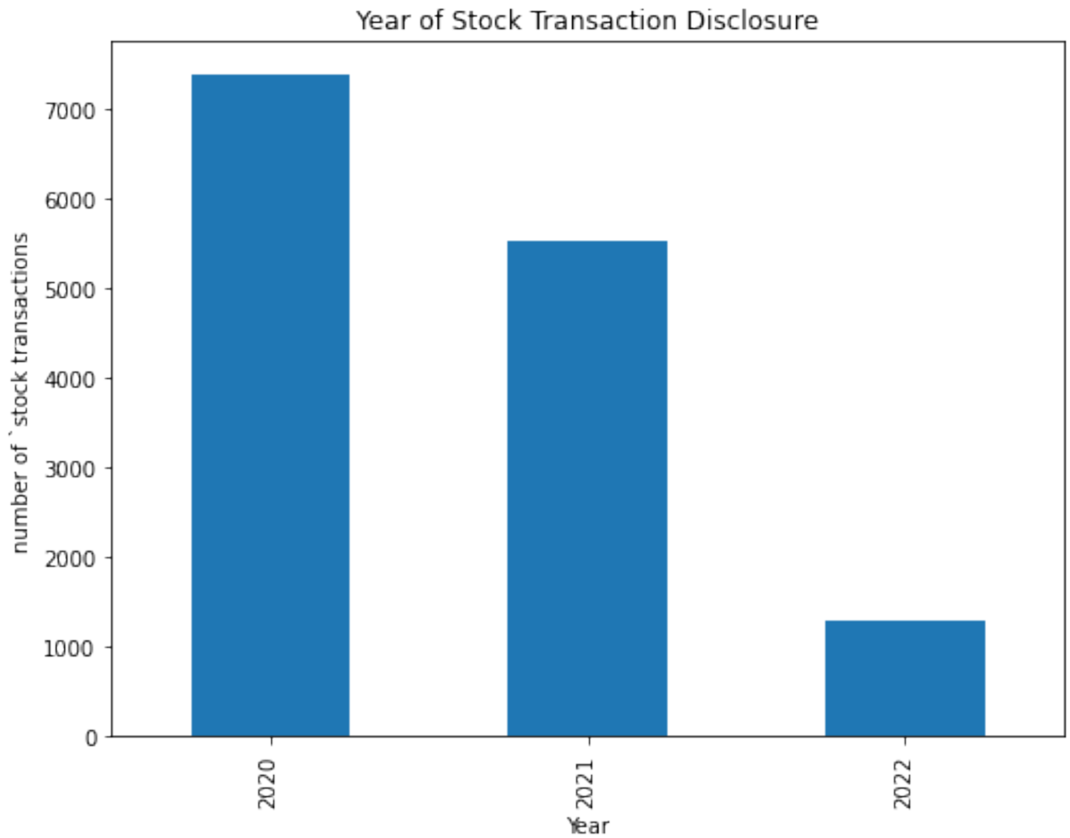
This concludes our data cleaning section.

## EDA - Univariate Analysis

Note: `value_counts()` will help us understand the values of a column by showing us how many times each entry occurs.

Let's explore the `disclosure_year` columm first.

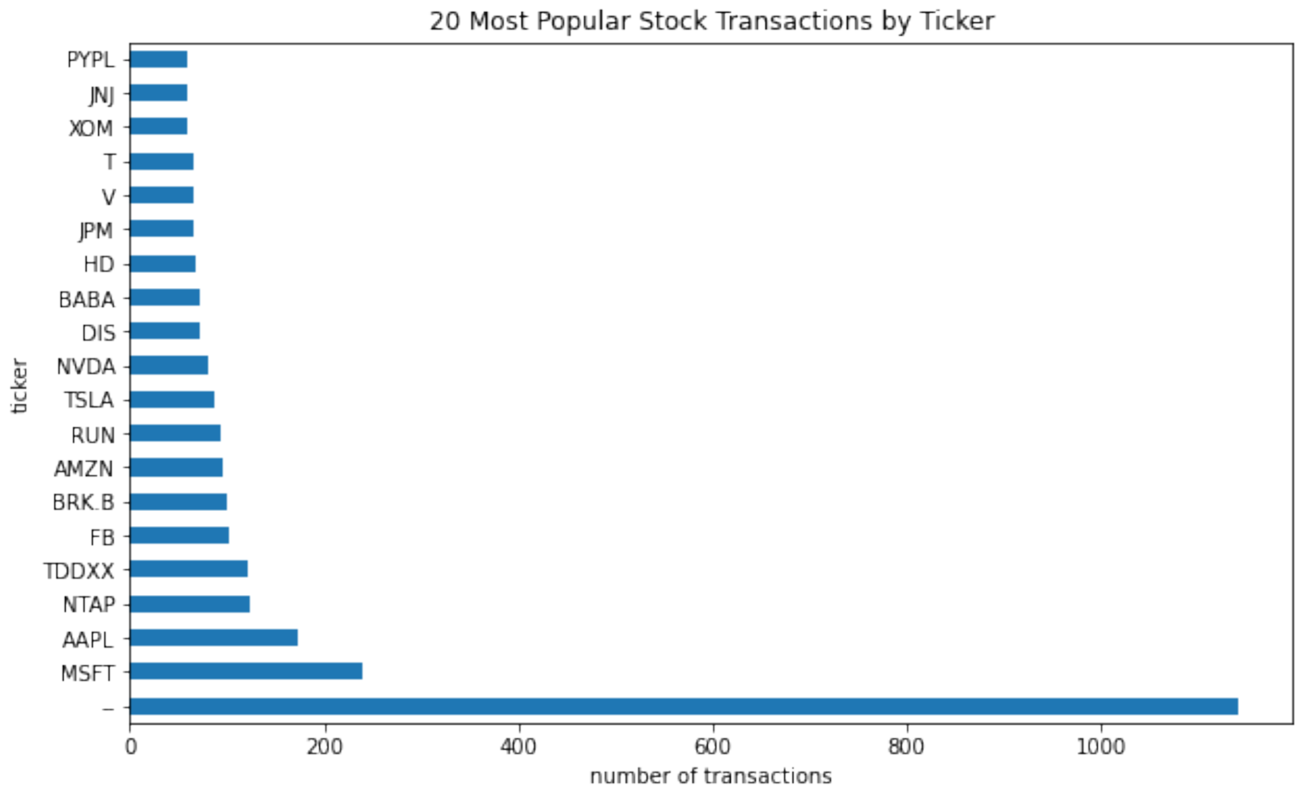
In [14]:
```
# first call .value_counts(), then .plot(), so graph is easier to read
df['disclosure_year'].value_counts().plot(kind = 'bar', \
    title = 'Year of Stock Transaction Disclosure', \
    xlabel = 'Year', ylabel = 'number of `stock transactions', figsize=(8,6))
```




Clearly **2020** was the most popular year in this data set. This doesn't mean that the reps just traded more in 2020 than in 2021 or 2022, they probably just have not disclosed trades from those two years yet.

Now let's look at the `ticker` column. There are too many individiual tickers (abbreviations used to uniquely identify shares of a particular stock), so we will only look at the top 20.

In [15]:
```
df['ticker'].value_counts()[:20].plot(kind = 'barh', figsize=(10,6))
plt.title('20 Most Popular Stock Transactions by Ticker')
plt.ylabel('ticker')
plt.xlabel('number of transactions')
plt.show()
```



Clearly, the most popular stock trades are in tech:

- Microsoft
- Apple
- NetApp
- FaceBook (Meta)
- Amazon
- Tesla
- and so on...

But there is even a Fed Fund:

- BLF FedFund (TDDXX)

**Surprised rich people are investing in fed funds?**

**Hey, maybe the old people in the House of Representatives *really do care* about their retirement funds!**

The biggest surprise, however, is the most popular ticker, '--', which isn't even a stock. What is '--'? Let's clean that up and convert to `np.nan` .

In [16]:
```
df['ticker'] = df['ticker'].replace({'--': np.nan})
```

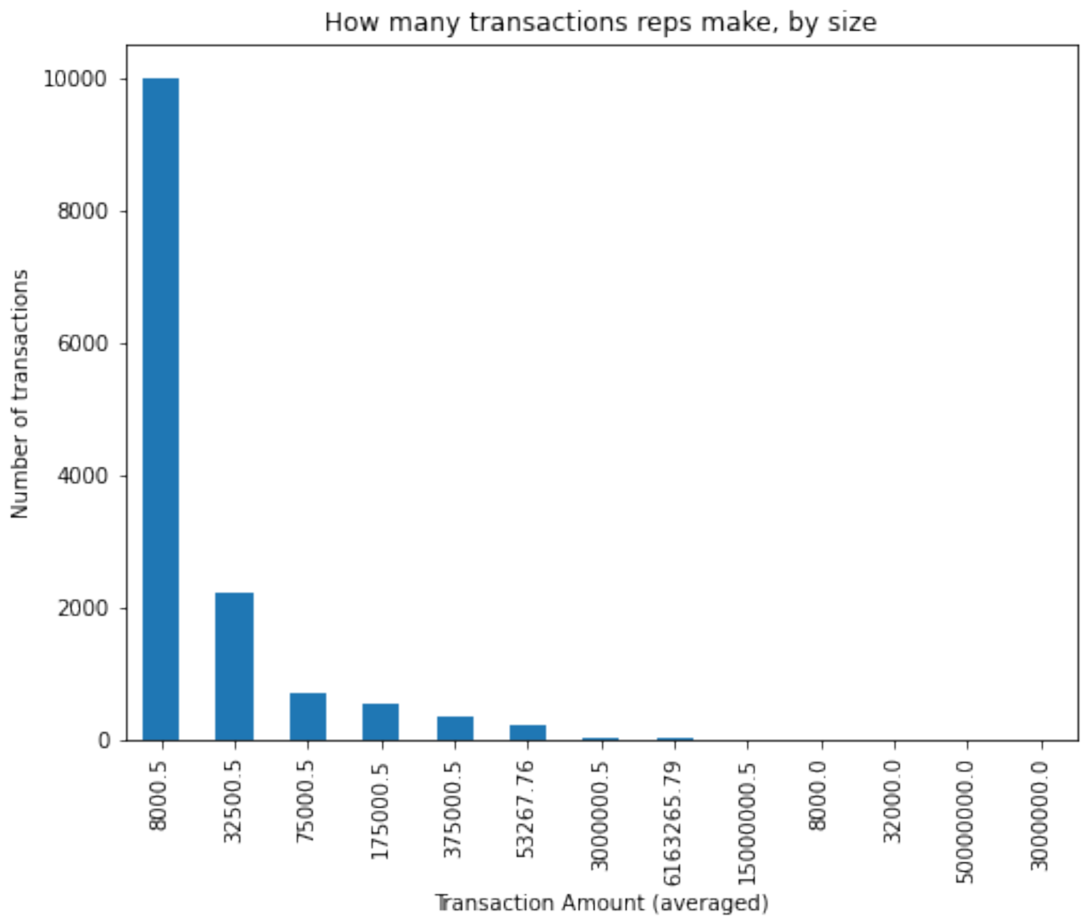
Let's make sure the `--` characters are gone.

In [17]:
```
# resulting dataframe should have 0 rows, .shape[0] should be equal to 0
df[df['ticker'] == '--'].shape[0] == 0
```

Out[17]:
```
True
```

Now let's look at the `amount_cleaned` column, to see how much of that 💰💸🤑 reps bring in.

In [18]:
```
# first call .value_counts(), then .plot(), so graph is easier to read
df['amount_cleaned'].value_counts().plot(kind = 'bar', \
    title = 'How many transactions reps make, by size', ylabel = 'Number of t
    xlabel = 'Transaction Amount (averaged)', figsize=(8,6));
```



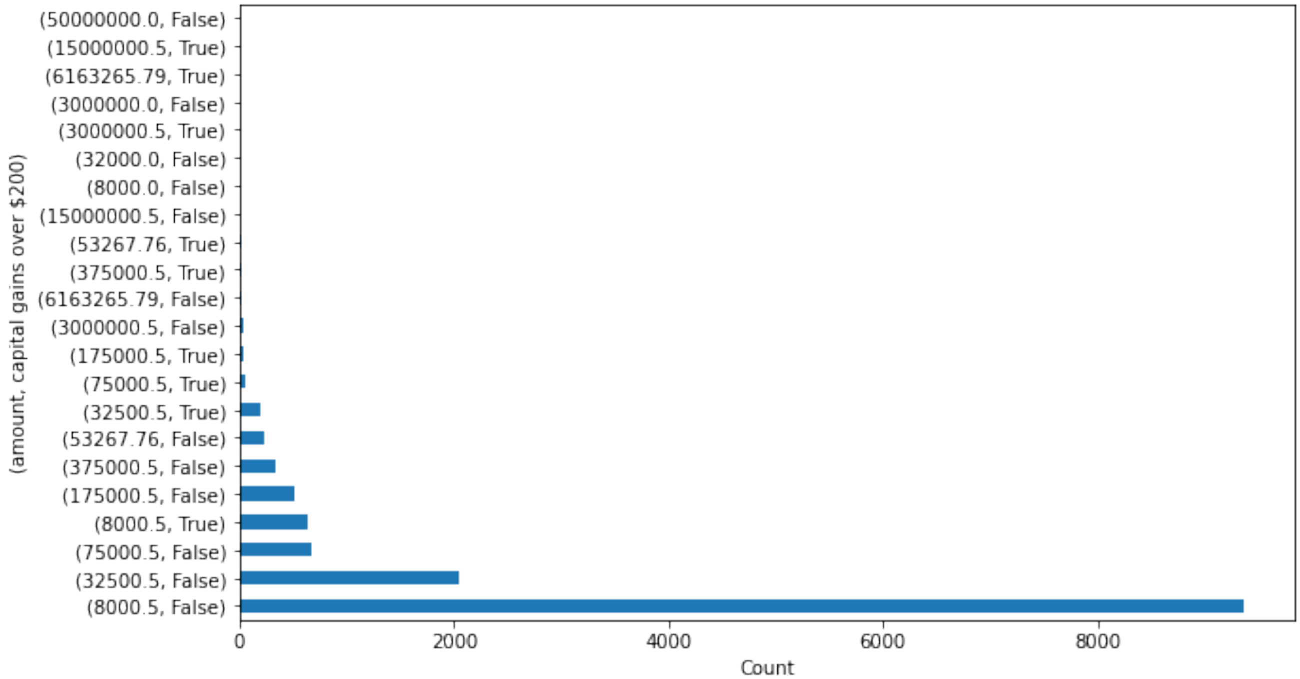

Wow! Such an overwhelming majority of stock trades are averaged out to 8,000 usd that the graph gets stretched vertically so much that we cannot even estimate how many transactions are done for any transaction above 53267.76 usd.

## EDA - Bivariate Analysis

We want to see whether reps have many counts of capital gains of over $200 or few. Let's find that out by grouping `amount_cleaned` and `cap_gains_over_200_usd` .

In [19]:

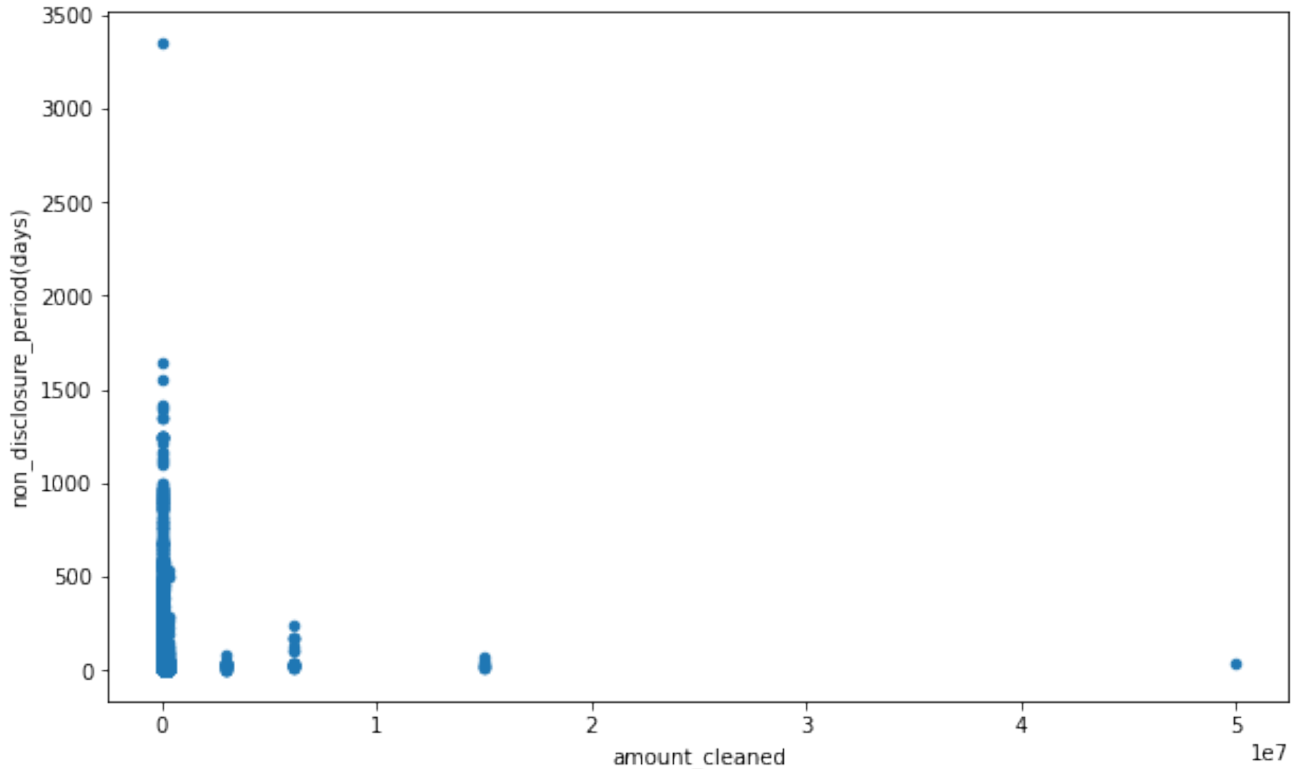
```
df[['amount_cleaned', 'cap_gains_over_200_usd']].value_counts().plot(kind = '
plt.xlabel("Count")
plt.ylabel("(amount, capital gains over $200)")
plt.show()
```




Let's see if the `non_disclosure_period(days)` column has any correlation with `amount_cleaned`.

Does a bigger stock transaction (higher `amount_cleaned`) lead to a longer `non_disclosure_period(days)` period? Let's find out.

In [20]:
```
relevant_non_disclosure_periods = df.loc[(df['non_disclosure_period(days)'] >
    [['amount_cleaned', 'non_disclosure_period(days)']]
relevant_non_disclosure_periods.plot(kind='scatter', x='amount_cleaned', \
    y = 'non_disclosure_period(days)', figsize=(10,6));
```

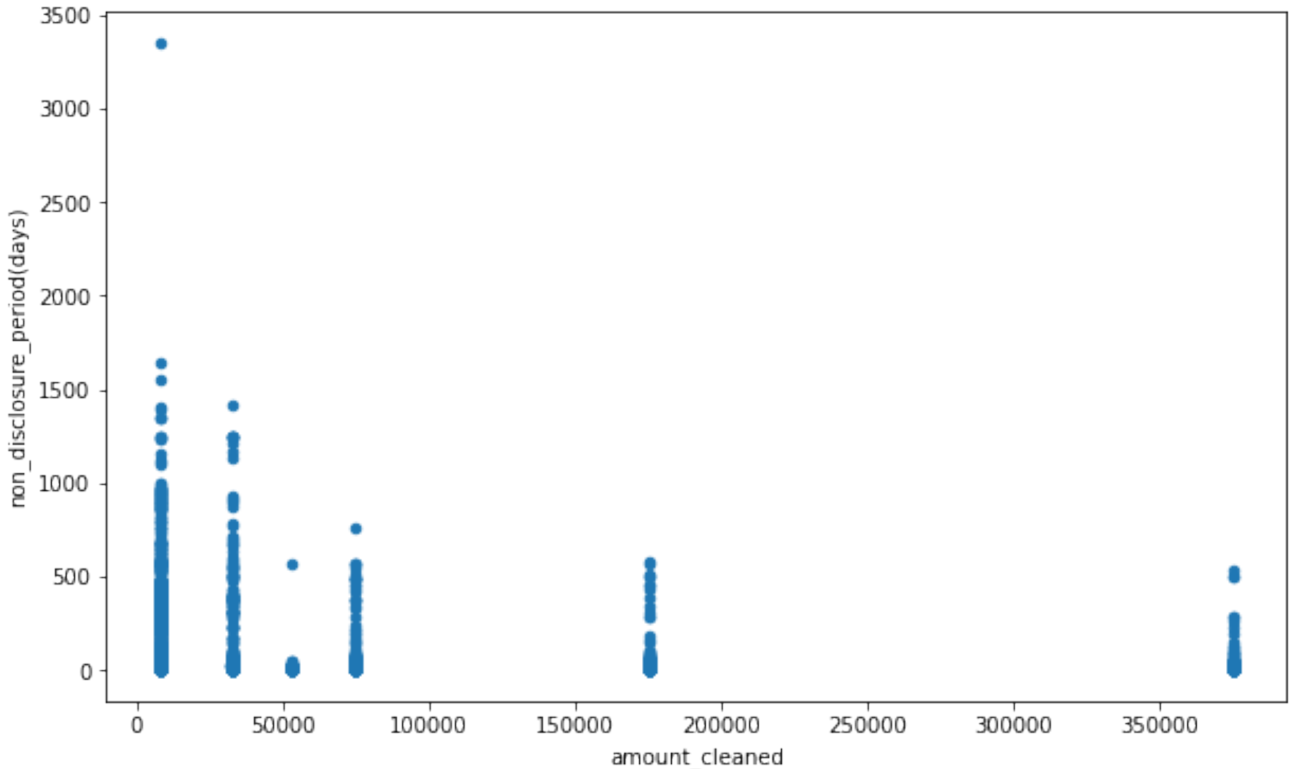



Before analyzing anything, an important point: this graph seems kinda weird right? Why are all the points jumbled around a few x values?

Well, this is the result of replacing a range with the average of the range.

As most of our data is seems to be less than trade transaction amount of 2 million, let's zoom in on its left to see if we notice anything.

In [21]:
```
closer_look = relevant_non_disclosure_periods[relevant_non_disclosure_periods
closer_look.plot(kind='scatter', x='amount_cleaned', y = 'non_disclosure_peri
```




Huh!

We notice that there doesn't seem to be much relationship between transaction amount and the non disclosure period. Our hypothesis was that reps that have invested more in trades delay disclosing it, **but here we see the opposite**, where, if the transaction was high, they seem to be disclosing it sooner than if the transaction was low.

This makes sense, they probably don't want to get into legal trouble. They are politicians after all, and they will do anything to make sure they get reelected. I'm sure the headlines of

*Rep. X. arrested after failing to disclose a trade of $5,000,000.*

won't be great for their political career.

## Interesting Aggregates / Pivot Tables

Let's find out if there are any reps with a 100% `cap_gains_over_200_usd` proportion score.

In [22]:

```
(
    df.pivot_table(index = 'representative', values = 'cap_gains_over_200_usd
    .sort_values(by='cap_gains_over_200_usd', ascending=False)
)
```

Out[22]:

| representative | cap_gains_over_200_usd |
|---|---|
| **Patrick T. McHenry** | 1.000000 |
| **Mr. TJ John (Tj) Cox** | 1.000000 |
| **Tim Burchett** | 1.000000 |
| **Mr. Peter Meijer** | 0.902256 |
| **Bradley S. Schneider** | 0.777778 |
| **...** | ... |
| **Harold Dallas Rogers** | 0.000000 |
| **Harley E. Rouda** | 0.000000 |
| **Gus M. Bilirakis** | 0.000000 |
| **Greg Steube** | 0.000000 |
| **Katherine M. Clark** | 0.000000 |

178 rows × 1 columns

It looks like Reps. Patrick T. McHenry, Mr. TJ John (Tj) Cox, and Tim Burchett are pros, with a 100% `cap_gains_over_200_usd` proportion score. But, are they really?"

In [23]:

```
df[
    (df['representative'] == 'Patrick T. McHenry') |
    (df['representative'] == 'Mr. TJ John (Tj) Cox') |
    (df['representative'] == 'Tim Burchett')
]
```

Out[23]:

| | disclosure_year | disclosure_date | transaction_date | owner | ticker | asset_description | |
|---|---|---|---|---|---|---|---|
| **56** | 2020 | 2020-09-22 | 2020-08-17 | NaN | NaN | Metallic Minerals Corp. | sa |
| **10854** | 2021 | 2021-01-10 | 2020-11-19 | NaN | D | Sale of shares in Dominion Energy Inc. | |
| **13381** | 2020 | 2020-02-26 | 2020-02-12 | NaN | DENN | Denny's Corporation | |

Upon further review, it seems that they each have only traded once so we don't have enough data to conclude that they are really pros at stock trading.

Ok, that was a flop.

Here's a new one to aggregate: **Which state has the highest average number of trades that result in capital gains over $200?**

In [24]:

```
(
    df.pivot_table(index = 'state', values = 'cap_gains_over_200_usd')
    .sort_values(by = 'cap_gains_over_200_usd', ascending=False)
    .head()
)
```

Out[24]:

| state | cap_gains_over_200_usd |
|---|---|
| MI | 0.772871 |
| SC | 0.379310 |
| NY | 0.280702 |
| ID | 0.222222 |
| LA | 0.222222 |

Michigan! With a pretty mind blowing result of 77% 🤯

Does the value of 77% seem *sussy?* Well, we do. (Yes, this is foreshadowing for our Hypothesis Testing)

**Note:** The first that may come to mind is that Michigan may have had a high `cap_gains_over_200_usd` value for relatively small trades, in the $8000.5 category, and low numbers in the rest. This would have induced Simpson's paradox. But as shown below, we checked with another pivot table, and it is really the case that Michigan has high `cap_gains_over_200_usd` values for *all* amount categories.

Let's look at `non_disclosure_period(days)` values indexed by the stock transaction amount itself, for each type of exchange.

In [25]:

```
df.pivot_table(index='amount_cleaned', values='non_disclosure_period(days)', 
```

Out [25]:

| type<br>amount_cleaned | exchange | purchase | sale_full | sale_partial |
|---|---|---|---|---|
| **8000.00** | NaN | NaN | 19.000000 | 19.000000 |
| **8000.50** | 149.081967 | 65.273784 | 73.570919 | 57.616108 |
| **32000.00** | NaN | NaN | 19.000000 | 19.000000 |
| **32500.50** | 35.285714 | 49.260524 | 44.978177 | 80.050562 |
| **53267.76** | 19.000000 | 10.752688 | 39.545455 | 13.031746 |
| **75000.50** | 40.875000 | 42.727506 | 42.405738 | 63.606557 |
| **175000.50** | 40.500000 | 25.249097 | 53.600917 | 40.725000 |
| **375000.50** | 33.333333 | 27.622754 | 37.950311 | 47.933333 |
| **3000000.00** | NaN | NaN | 23.000000 | NaN |
| **3000000.50** | 43.000000 | 29.210526 | 21.000000 | 25.666667 |
| **6163265.79** | NaN | 97.333333 | 54.769231 | 30.166667 |
| **15000000.50** | NaN | 21.250000 | 35.333333 | 39.500000 |
| **50000000.00** | NaN | NaN | 37.000000 | NaN |

This table is confusing, so let's look at just the means of each of the columns.

In [26]:

```
df.pivot_table(index='amount_cleaned', values='non_disclosure_period(days)', 
```

Out[26]:

```
type
exchange         51.582288
purchase         40.964468
sale_full        38.550314
sale_partial     39.663331
dtype: float64
```

For some reason, or just sheer random chance, exchanges take on average almost 12 days longer to be disclosed than purchases, full sales, and partial sales. Interesting 🤔🤔

This concludes our EDA section.

# Assessment of Missingness

## Follwing are all the columns that have one or more null values:

- `asset_description` : 4
- `owner` : 5333
- `ticker` : 1141
- `transaction_date` : 5

Because `owner` has the most amount of missing values, we will use that column to analyze if the missingness of `owner` is depended on any columns. In order to find out, we have to run permutation testing. Permutation testing will allow us to analyze whether the missingness of `owner` is depended on any other columns.

## Let's begin permutation testing!

In [27]:
```
df_copy = df.copy()
# assigning `owner_missing` column to True if owner val is missing; else Fals
df_copy['owner_missing'] = df_copy['owner'].isna()
```

We will use ks_2samp from SciPy library as a test stat for our permutation testing. But ks is used on columns that are only numerical so let's focus on two numerical columns for now.

In [28]:
```
cols_to_choose = 'disclosure_year amount_cleaned'.split(" ")
```

In [29]:
```
new_dict = {}

for col in cols_to_choose:
    # when 'owner' is missing
    col_owner_mis = df_copy.loc[df_copy['owner_missing'], col]

    # when 'owner' is not missing
    col_owner_not_mis = df_copy.loc[~df_copy['owner_missing'], col]

    # ks_2samp will perform Kolmogorov-Smirnov test for goodness of fit
    val = stats.ks_2samp(col_owner_mis, col_owner_not_mis)

    new_dict[col] = val

new_dict
```

Out[29]:
```
{'disclosure_year': KstestResult(statistic=0.10445941940037393, pvalue=4.37301
9393958429e-32),
 'amount_cleaned': KstestResult(statistic=0.02360406573781591, pvalue=0.047921
07763601705)}
```

The p-val for `disclosure_year` and `amount_cleaned` is extremely low. This means that the distribution of `disclosure_year` , for instance, when `owner` is missing and the distribution of `disclosure_year` when `owner` is not missing are likely different, which means that the missingness of `owner` likely depends on `disclosure_year` . Same goes for `amount_cleaned` .

Let's find one more column where the depended-on column for missingness of `owner` is categorical. The test stat we need to use if the depended-on column is categorical is TVD.

In [30]:
```
# making a copy so we don't modify the original df
shuffled = df.copy()
# again assigning `owner_missing` column to True if owner val is missing; els
shuffled['owner_missing'] = shuffled['owner'].isna()
tvds = []
for _ in range(500):
    # shuffles the values in the district column and puts it back to the df
    shuffled['state'] = np.random.permutation(shuffled['state'])
    # resulting df will have 2 rows; one for where `owner` val is missing and
    # the columns are the district
    pivoted = (
        shuffled
        .pivot_table(index='owner_missing', columns='state', aggfunc='size')
        .apply(lambda x: x / x.sum(), axis=1)
    )
    tvd = pivoted.diff().iloc[:, -1].abs().sum() / 2
    tvds.append(tvd)
```

In [31]:
```
df_copy = df.copy()
dist = (
    df_copy
    .assign(owner_missing=df_copy['owner'].isna())
    .pivot_table(index='state', columns='owner_missing', aggfunc='size')
)
dist = dist / dist.sum()
obs_tvd = dist.diff(axis=1).iloc[:, -1].abs().sum() / 2
obs_tvd
```

Out[31]:
```
0.4196815040820534
```

In [32]:
```
pval = np.mean(tvds >= obs_tvd)
pval
```

Out[32]:
```
0.0
```

Here, we see that the p-val is 0.0, implying that the distribution of `state` when `owner` is missing and the distribution of `state` when `owner` is not missing are likely different, so the missingness of `owner` likely depends on `state`.

# Hypothesis Testing

The question we've decided to explore is the case of *why are Michigan reps so damn good at trading stocks?*

Here's a little bit of background again to refresh your memory.

In our interesting aggregates / pivot tables section, we wanted to answer: **Which state has the highest average number of trades that result in capital gains over $200?**

In [33]:
```
(
    df.pivot_table(index = 'state', values = 'cap_gains_over_200_usd')
    .sort_values(by = 'cap_gains_over_200_usd', ascending=False)
    .head()
)
```

Out[33]:

| | cap_gains_over_200_usd |
|---|---|
| **state** | |
| **MI** | 0.772871 |
| **SC** | 0.379310 |
| **NY** | 0.280702 |
| **ID** | 0.222222 |
| **LA** | 0.222222 |

**Formalized...**

- **Question:**

The reps from the state of Michigan have 77% of trades that result in the `cap_gains_over_200_usd` column having a True value.

Is this just by chance, or are the representatives from some states, like *The Great Lakes State*, just that damn much better at trading then representatives from other states?

- **Null hypothesis:** Reps from Michigan *are not* more likely to have an average trade proportion that result in capital gains over $200 higher than the rest of the population.

- **Alternative Hypothesis:** Reps from Michigan *are* more likely to have an average trade proportion that result in capital gains over $200 higher than the rest of the population.

- **Test statistic:** Difference of means between average value of `cap_gains_over_200_usd` column for Michigan reps vs average value of `cap_gains_over_200_usd` column for non-Michigan reps.
    - Difference of means is appropriate here because we have numerical/quantitiative data, and we cn have just one aggregation statistic, for each Michigan only group, and non-Michigan only group.
- **Significance level:** 1%
    - Why 1% vs 5%? No particular reason, we just want to have results that are as statistically significant as possible.

There's an easy solution to this problem - **Permutation Testing!**

We're choosing permutation testing over hypothesis testing because we are given *2* observed samples (versus the only *one* that hypothesis testing works on). One group is all trades from reps from Michigan, and the other group are all trades that are not from reps from Michigan. We need to see if they are they fundamentally different, or could they have been generated by the same process?

Let's begin the permutation testing.

First, let's find the observed test statistic: difference in means.

In [34]:
```
# let's only choose the columns we need
ptest_df = df[['state', 'cap_gains_over_200_usd']]

# seperate into 2 groups
ptest_df_MI = ptest_df.loc[ptest_df['state'] == 'MI']
ptest_df_NOTMI = ptest_df.loc[ptest_df['state'] != 'MI']

# calculate difference of means
observed_diff_means = ptest_df_MI['cap_gains_over_200_usd'].mean() - \
                        ptest_df_NOTMI['cap_gains_over_200_usd'].mean()
observed_diff_means
```

Out[34]:
```
0.7210881633608198
```

So the reps from Michigan have an average value of `cap_gains_over_200_usd` substantially higher than non-Michigan reps, by about 72%.

Now let's run the permutation test 500 times.

In [35]:

```
diff_means = []
N = 500
for _ in range(N):

    # shuffle the cap_gains_over_200_usd column
    shuffled_gains = (
        ptest_df['cap_gains_over_200_usd']
        .sample(frac=1)
        # we need to reset_index w/ drop=True otherwise the old index values
        .reset_index(drop=True)
    )

    # assign a new dataframe with the shuffled column
    ptest_df_shuffled = (
        ptest_df.assign(**{'shuffled_cap_gains_over_200_usd': shuffled_gains}
    )

    # Michigan only values
    MI_only = (
        ptest_df_shuffled.loc[ptest_df_shuffled['state'] == 'MI']
    )

    # NON-Michigan only values
    not_MI = (
        ptest_df_shuffled.loc[ptest_df_shuffled['state'] != 'MI']
    )

    # compute difference in means by subtracting MI-only group's mean with
    # NON-MI-only group's mean.
    test_stat = (
        MI_only['shuffled_cap_gains_over_200_usd'].mean() - \
            not_MI['shuffled_cap_gains_over_200_usd'].mean()
    )
    diff_means.append(test_stat)
```

Now let's calculate the p-value!

The p-value represents what proportion of our `observed_diff_means` list had more extreme values than our observed test-statistic, `diff_means` .

In [36]:

```
p_val = np.mean(diff_means >= observed_diff_means)
p_val
```

Out[36]:

```
0.0
```

**0!**

So this means that our result is **highly significant**, and we can reject the null hypothesis since our p-value (0.0) is lower than our significance level (0.01).

I.E., Reps from Michigan *did indeed* have an average trade proportion that resulted in capital gains over 200 usd *higher* than the rest of the population, and this was **not due by chance**.

If it was only due by chance, our p-value would not be so low, and we would not be rejecting the null hypothesis.

A few implications from our results:

- Does this mean MI representatives are doing more insider trading?
    - We can't say this, because insider trading would require us to look at timing of some specific stock trades, outside information (investigative journalism, etc), and would also require us to rule out some other scenarios.
- Are MI represenatatives 'better' traders?
    - We can't say this either, because proportion of trades that result in capital gains over 200 USD isn't a fool proof indicator of how good one is at trading. Consider this: MI representatives each only made 1 trade that resulted in capital gains over 200 USD, but CA representatives each made 100 trades each resulting in 199 USD in capital gains. Which state can say thay have the better traders in this case?