Summary of Findings

Introduction

The following notebook analyzes publicly available information regarding stock trades made by US House of Representatives members to reach inferences. We first cleaned up the dataset and combined *The 116th U.S. House of Representatives* at https://www.kaggle.com/datasets/aavigan/house-of-representatives-congress-116 which contains information regarding political affiliation of congresspeople. Next, we evaluate the dataset's `owner` column's missingness association.

After that, we began processing the dataset for insights, and try to explore answers to the following questions:

- Does one party trade more often?
- Does one party make larger trades?
- What congresspeople have made the most trades?
- What companies are most traded by congresspeople?
- When are stocks bought and sold? Is there a day of the week that is most common? Or a month of the year?

Cleaning and EDA

After obtaining the complete dataset of stock trade disclosures from https://housestockwatcher.com/api, we discover that the data need to be cleaned up since they are fairly untidy. To clean it, we did what is shown below:

1. Change `disclosure_date` and `transaction_date` column to `datetime` type.
2. Replace '--' value in `ticker` column with `np.NaN` .
3. Replace '--' value in `owner` column with `np.NaN` .
4. Convert `amount` to a `pd.Categorical` series.

The political affiliation of congressmen is missing from the dataset after it has been cleaned up, therefore we choose to utilize one from **Kaggle** at https://www.kaggle.com/datasets/aavigan/house-of-representatives-congress-116. Due of the distinctions in the names between the two datasets, we combined them using the first and last names of each participant. Then we examine a few rare occurrences and manually resolve them. We were able to successfully integrate stock trade activity with representative political allegiance as an outcome.

Moving on, we process to EDA and find out that:

- `owner`, `ticker`, `transaction_date`, and `asset_description` are 4 columns that contain missing data, some of the missingness in `transaction_date` is because of the incorrect value. For example, there are a few cell with value `0009-06-09` which is clearly not a valid date.
- `transaction_date` range between `2012-06-19` and `2022-10-21`.
- Most of the congresspeople are either from `Democrat` or `Republican`, there is only one house member who is listed as `Independent` regrading their political affiliation.

What congresspeople have made the most trades?

- By plotting the value counts of `representative` column, we have discovered that the representative **Josh Gottheimer** has made the most trades.

What companies are most traded by congresspeople?

- By plotting the value counts of `ticker` column, we have discovered that the ticker **MSFT**, which is **Microsoft Corp.**, has the most trade transactions.

When are stocks bought and sold? Is there a day of the week that is most common? Or a month of the year?

- By grouping the dateset by weekday of `transaction_date`, such that most of the transactions happened during weekdays, while only a tiny amount of transactions are done in weekend. Among weekdays, **Thurday** seems to have a slightly higher transaction volume.
- By grouping the dataset by month of `transation_date`, we discover that **February** is the month has largest volume of transactions.

Assessment of Missingness

In this section we decided to evaluate the missingness of `owner` column as it has the most missing values across all columns. It has values like `self`, `joint`, `dependent`, and `np.NaN`. We think the missingness of `owner` column could be associated with `type` column. This concept arises from the fact that `type` describes the sort of transaction that is performed; if the type of transaction is not a stock exchange, it is less likely to fall into the `self`, `joint`, or `dependant` categories and end up as an empty value.

In order to validate this assumption, we have to perform a permutation test. To begin with, we determine the test statistic to be **Total Variation Distance (TVD)**, as `type` is a categorical data. Then, we calculate the observed statistic for the original dataset, which is `0.07390` . Afterwards, we shuffle the `owner` column and calculate the simulate statistics. By repeating this prcoess for 5,000 times, we then calculate the p-value for this permutation. As a result, we get a p-value of `0.0` which indicates that none of the simulate statistics has a more extreme result than the observed statistics. In conclusion, we conclude that the missingness of `owner` is **Missing at Random (MAR)**, and it's dependent on `type` column the most.

Hypothesis Test

Which party trade more often?

- **Null hypothesis**: the distribution of trading frequency among congresspeople from different party is the same. The difference between the two observed sample is due to chance.

- **Alternative hypothesis**: the distribution of trading frequency among congresspeople from different party are different.

For the test statistics, we calculate the average trading transactions per month of each party and take the absolute difference between them. The observed statistics is `58.5469` , and we shuffle the `party` column and run the permutation test for 5,000 times. At the end, we get a p-value of `0.8756` , which indicates that majority of the permutation test cases have more extreme result than the observed statistics. Therefore, we **fail to reject** the null hypothesis, the distribution of trading frequency among congresspeople from various parties is probrably the same.

Code

```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
```

Load transaction dataset

```
transactions = pd.read_csv('data/all_transactions.csv')
```

```
transactions.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	disclosure_year	disclosure_date	transaction_date	owner	ticker	a
0	2021	10/04/2021	2021-09-27	joint	BP	B
1	2021	10/04/2021	2021-09-13	joint	XOM	E C
2	2021	10/04/2021	2021-09-10	joint	ILPT	I L P C
3	2021	10/04/2021	2021-09-28	joint	PM	P I
4	2021	10/04/2021	2021-09-17	self	BLK	B

Cleaning and EDA

```
cleaned = transactions.copy()

# convert `disclosure_date`, `transaction_date` to datetime type
cleaned['disclosure_date'] = pd.to_datetime(cleaned['disclosure_date'])
cleaned['transaction_date'] = pd.to_datetime(cleaned['transaction_date'], e

# change `ticker` null values
cleaned['ticker'] = cleaned['ticker'].replace('--', np.NaN)
```

```
# cahnge `owner` null values
cleaned['owner'] = cleaned['owner'].replace('--', np.NaN)

# convert `amount` to categorical type
cleaned['amount'] = pd.Categorical(cleaned['amount'])

cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15674 entries, 0 to 15673
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   disclosure_year        15674 non-null  int64
 1   disclosure_date        15674 non-null  datetime64[ns]
 2   transaction_date       15667 non-null  datetime64[ns]
 3   owner                  8346 non-null   object
 4   ticker                 14378 non-null  object
 5   asset_description      15670 non-null  object
 6   type                   15674 non-null  object
 7   amount                 15674 non-null  category
 8   representative         15674 non-null  object
 9   district               15674 non-null  object
 10  ptr_link               15674 non-null  object
 11  cap_gains_over_200_usd  15674 non-null  bool
dtypes: bool(1), category(1), datetime64[ns](2), int64(1), object(7)
memory usage: 1.2+ MB
```

```
cleaned.isna().sum()
```

```
disclosure_year             0
disclosure_date             0
transaction_date            7
owner                    7328
ticker                   1296
asset_description           4
type                        0
amount                      0
representative              0
district                    0
ptr_link                    0
cap_gains_over_200_usd      0
dtype: int64
```

Combine with political affliation dataset

```
# remove unwanted name suffixs
suffixs = ['Hon\\.', 'Mr\\.', 'Mrs\\.', 'None', 'Aston', 'S\\.', 'W\\.']
cleaned['representative'] = (cleaned['representative']
                             .str.replace('|'.join(suffixs), '', regex=True
                             .str.strip())

cleaned['representative'].head()
```

```
0      Virginia Foxx
1      Virginia Foxx
2      Virginia Foxx
3      Virginia Foxx
4    Alan  Lowenthal
Name: representative, dtype: object
```

```
# split representative name into `first_name` and `last_name` for later mer
cleaned['first_name'] = cleaned['representative'].apply(lambda x: x.split()
cleaned['last_name'] = cleaned['representative'].apply(lambda x: x.split()[

# fix special cases
cleaned.loc[cleaned['representative'] == 'Neal Patrick Dunn MD, FACS', 'las

cleaned['first_name'].head()
```

```
0    virginia
1    virginia
2    virginia
3    virginia
4        alan
Name: first_name, dtype: object
```

```
# import member table 1
members1 = pd.read_csv('data/us-house.csv')
members1 = members1[['party', 'first_name', 'last_name']]
members1['first_name'] = members1['first_name'].str.lower()
members1['last_name'] = members1['last_name'].str.lower()
members1['party'] = members1['party'].str.capitalize()

members1.head(10)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
```

```
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	party	first_name	last_name
0	Republican	don	young
1	Republican	jerry	carl
2	Republican	felix	moore
3	Republican	mike	rogers
4	Republican	robert	aderholt
5	Republican	mo	brooks
6	Republican	gary	palmer
7	Democrat	terri	sewell
8	Republican	rick	crawford
9	Republican	french	hill

```
# import member table 2
members2 = pd.read_csv('data/house_members_116.csv')
members2['first_name'] = members2['name'].apply(
    lambda x: x.split('-')[0].lower())
members2['last_name'] = members2['name'].apply(
    lambda x: x.split('-')[-1].lower())
members2 = members2.rename(columns={'current_party': 'party'})[
    ['first_name', 'last_name', 'party']]

# unify party values
members2.loc[members2['party'] == 'Democratic', 'party'] = 'Democrat'

members2.head(10)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

</style>

	first_name	last_name	party
0	ralph	abraham	Republican
3	pete	aguilar	Democrat
4	rick	allen	Republican
5	colin	allred	Democrat
6	justin	amash	Independent
7	mark	amodei	Republican
8	kelly	armstrong	Republican
9	jodey	arrington	Republican


```
# combine 2 member tables
members = (pd.concat([members1, members2])
           .sort_values(['first_name', 'last_name'])
           .drop_duplicates(subset=['first_name', 'last_name'])
           .reset_index(drop=True))

# fix mismatch names
members.loc[members['first_name'] == 'k', 'first_name'] = 'k.'
members.loc[members['first_name'] == 'raul', 'first_name'] = 'raúl'
members.loc[members['first_name'] == 'wm', 'first_name'] = 'wm.'
members.loc[members['first_name'] == 'ro', 'first_name'] = 'rohit'
members.loc[members['first_name'] == 'cynthia', 'first_name'] = 'cindy'
members.loc[members['last_name'] == 'allen', 'first_name'] = 'richard'
members.loc[members['last_name'] == 'steube', 'first_name'] = 'greg'
members.loc[members['last_name'] == 'banks', 'first_name'] = 'james'
members.loc[(members['first_name'] == 'j') & (
    members['last_name'] == 'hill'), 'first_name'] = 'james'
members.loc[(members['first_name'] == 'mike') & (
    members['last_name'] == 'garcia'), 'first_name'] = 'michael'
members.loc[members['last_name'] == 'crenshaw', 'first_name'] = 'daniel'
members.loc[members['last_name'] == 'taylor', 'first_name'] = 'nicholas'
members.loc[members['last_name'] == 'gallagher', 'first_name'] = 'michael'
members.loc[(members['first_name'] == 'gregory') & (
    members['last_name'] == 'murphy'), 'first_name'] = 'greg'
members.loc[members['first_name'] == 'ashley', 'last_name'] = 'arenholz'
members.loc[members['last_name'] == 'buck', 'first_name'] = 'kenneth'
```

```
members.loc[members['last_name'] == 'costa', 'first_name'] = 'james'
members.loc[members['last_name'] == 'hagedorn', 'first_name'] = 'james'

# drop duplicate rows
members = members.drop_duplicates(subset=['first_name', 'last_name'])

# output cleaned representative table
members.to_csv('data/cleaned_members.csv', index=False)

members.shape
```

```
(547, 3)
```

```
# transaction table with member info table
combined = cleaned.merge(members, how='left', on=['first_name', 'last_name'

combined.loc[combined['party'].isna(), 'representative'].unique()
```

```
array([], dtype=object)
```

```
combined.shape
```

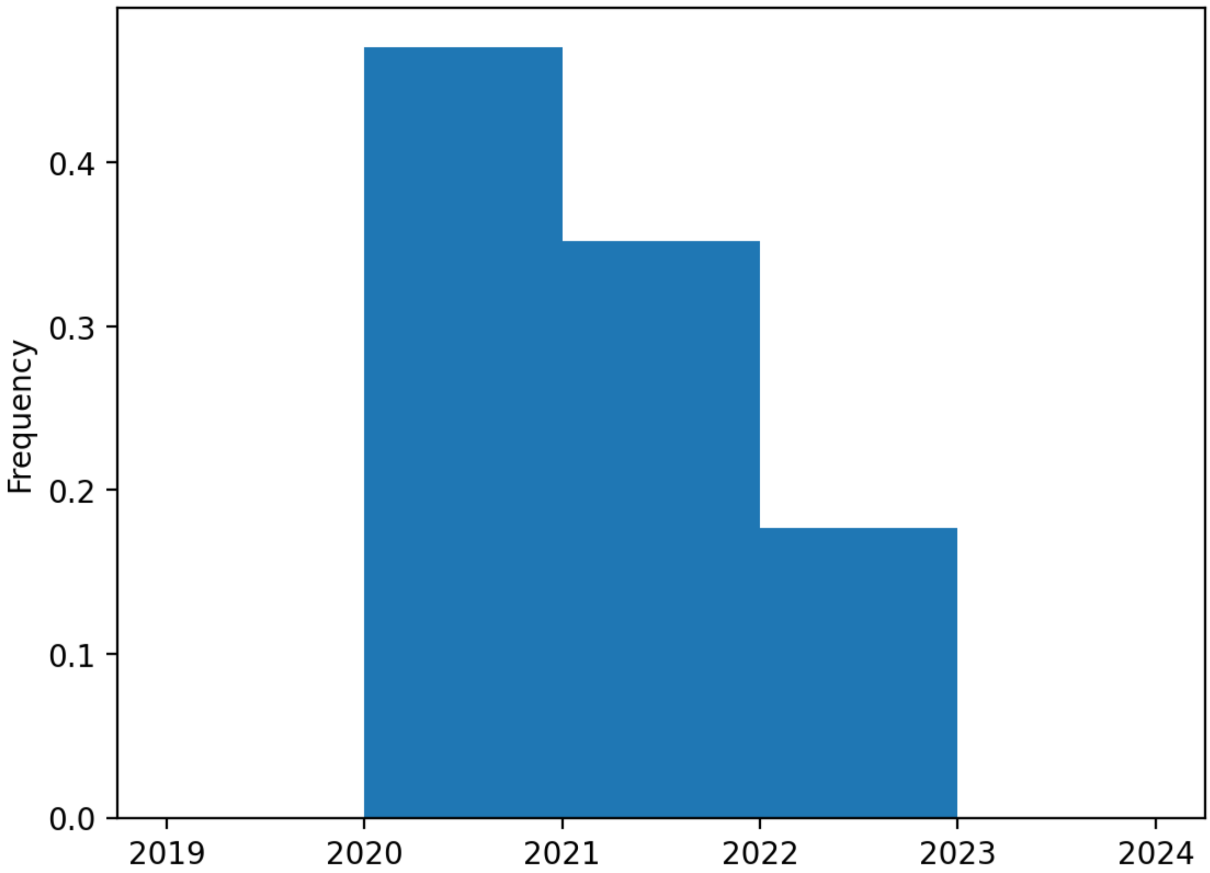
```
(15674, 15)
```

```
transactions.shape
```

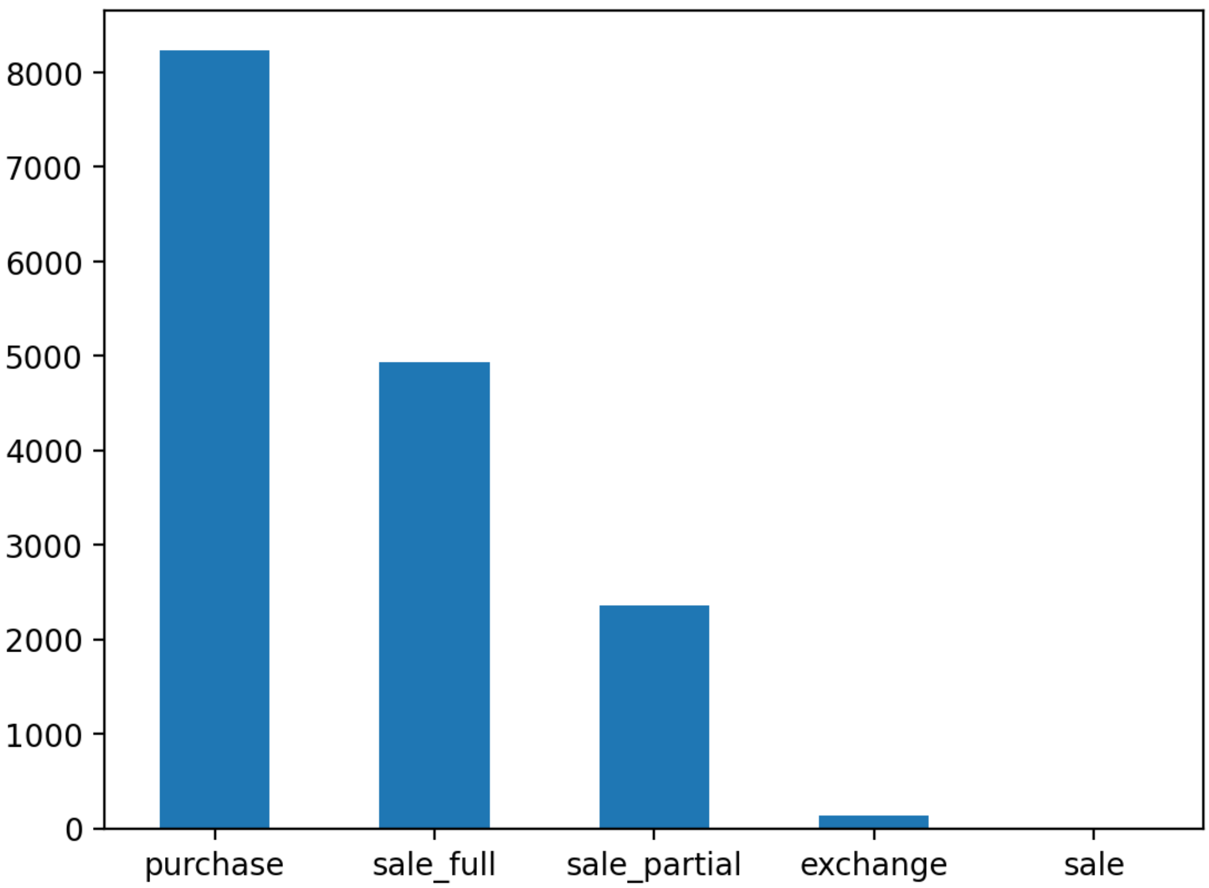
```
(15674, 12)
```

```
combined.to_csv('data/combined_transactions.csv', index=False)
```

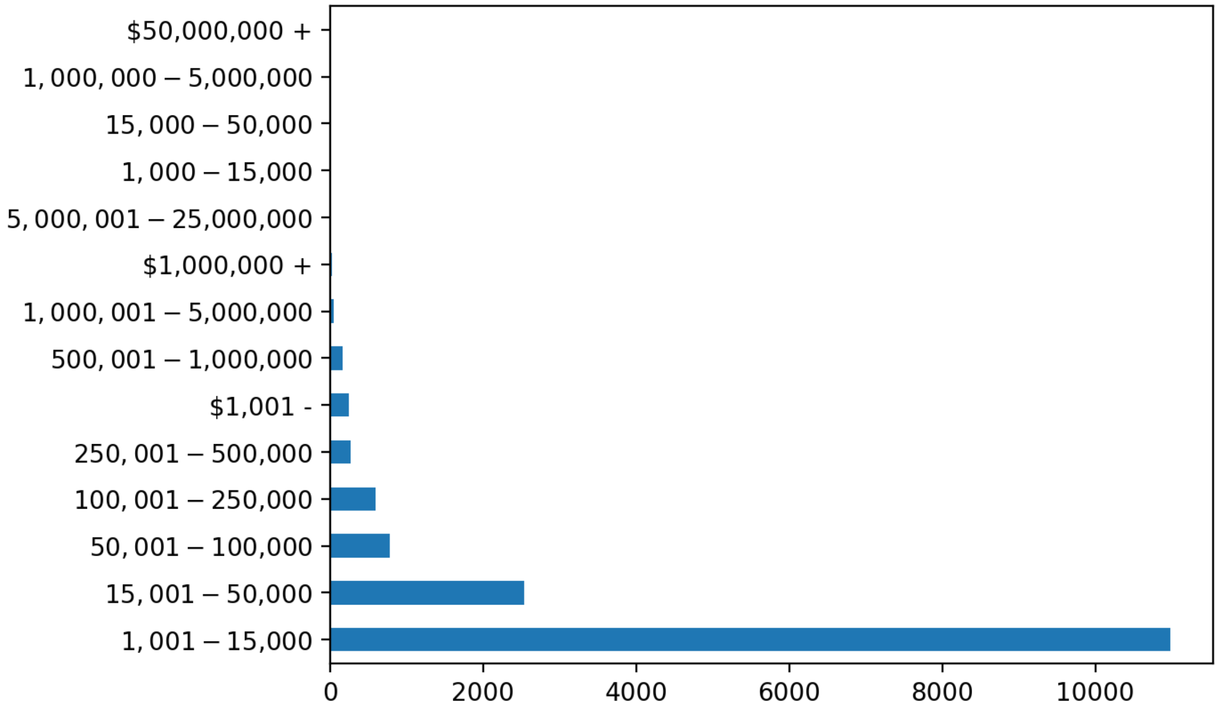
```
combined['disclosure_year'].plot(kind='hist', density=True, bins=range(2019
```



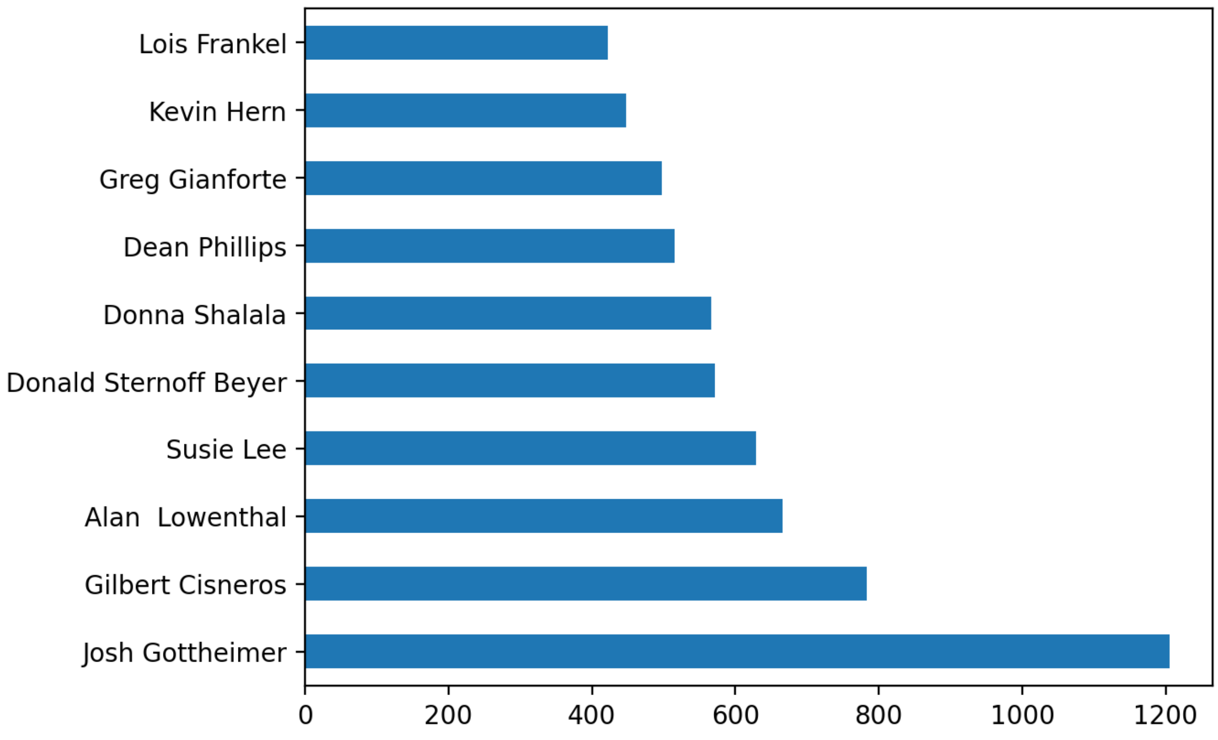

```
combined['type'].value_counts().plot(kind='bar')
plt.xticks(rotation=0);
```



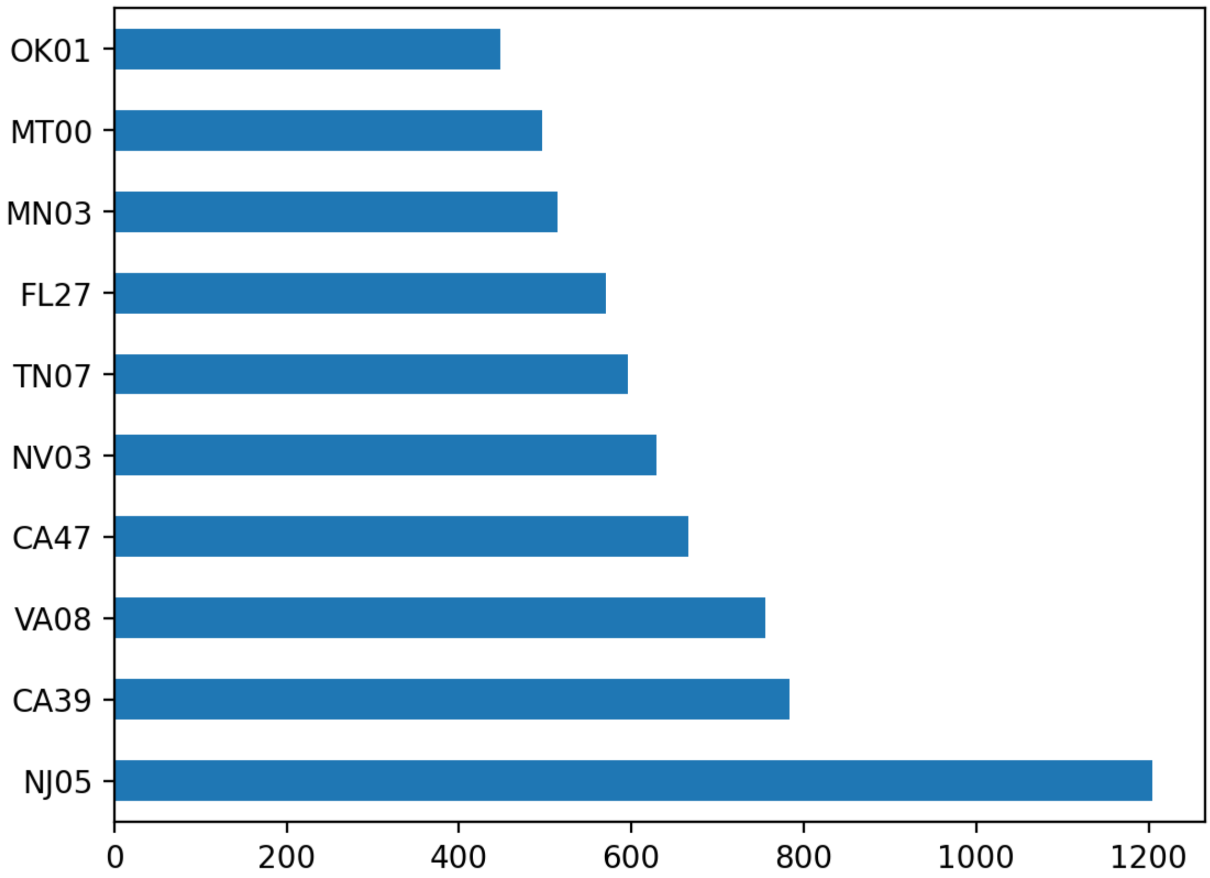
```
combined['amount'].value_counts().plot(kind='barh');
```



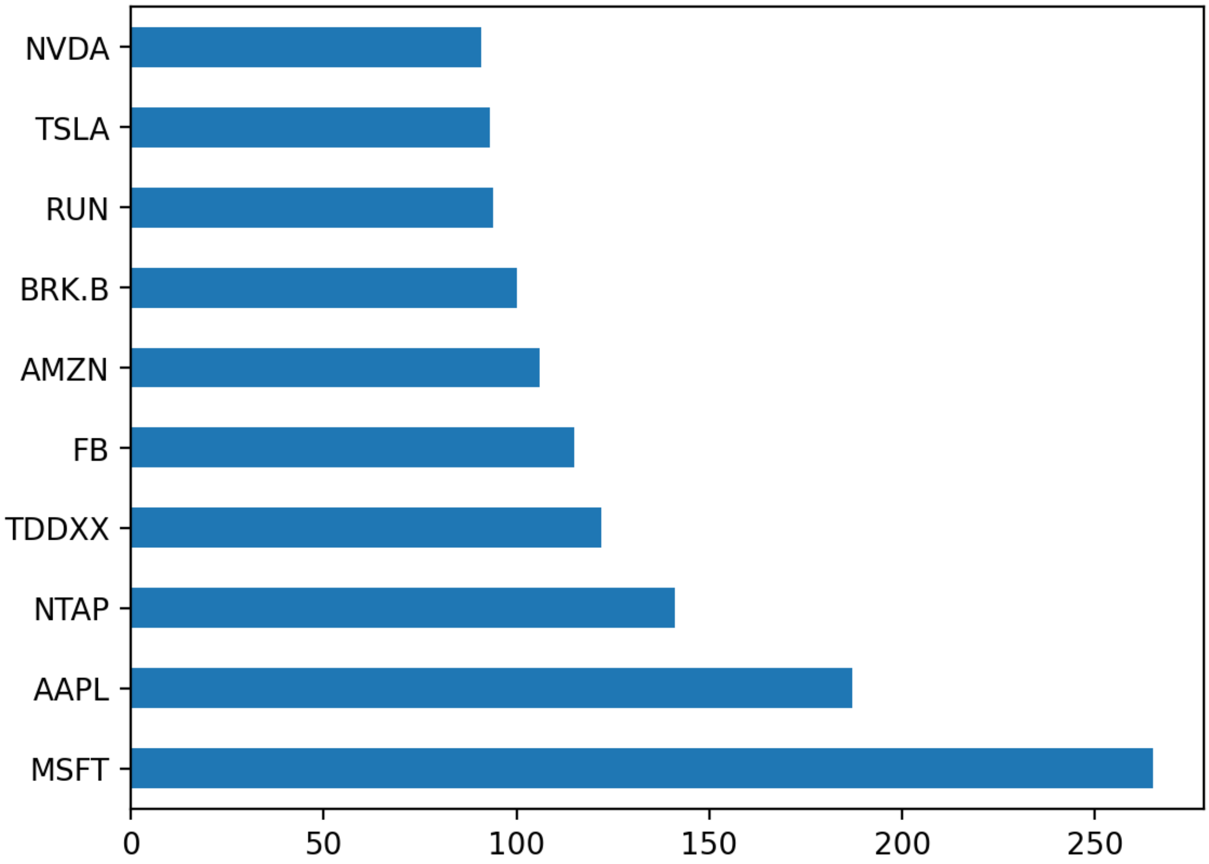

```
combined['representative'].value_counts().head(10).plot(kind='barh');
```




```
combined['district'].value_counts().head(10).plot(kind='barh');
```




```
combined['ticker'].value_counts().head(10).plot(kind='barh');
```



Assessment of Missingness

```python
combined.isna().sum()
```

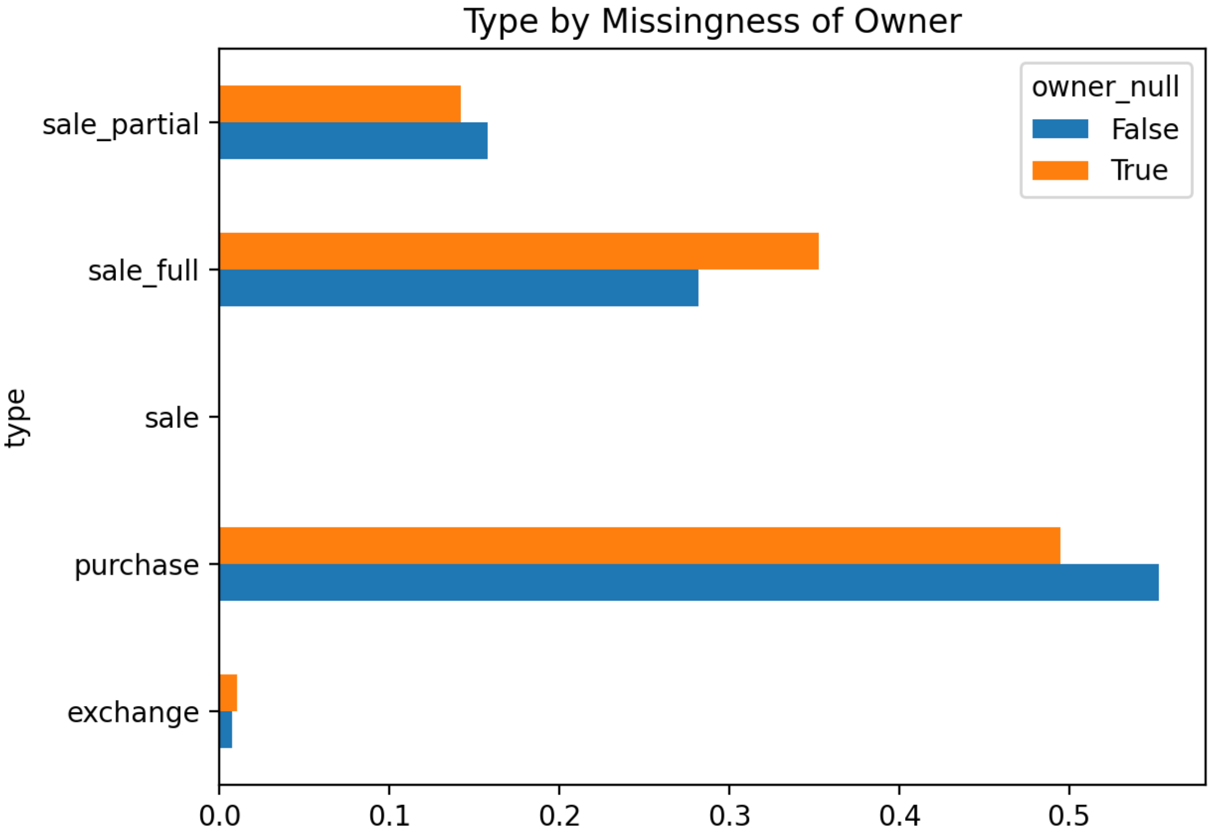
```
disclosure_year             0
disclosure_date             0
transaction_date            7
owner                    7328
ticker                   1296
asset_description           4
type                        0
amount                      0
representative              0
district                    0
ptr_link                    0
cap_gains_over_200_usd      0
first_name                  0
last_name                   0
party                       0
dtype: int64
```

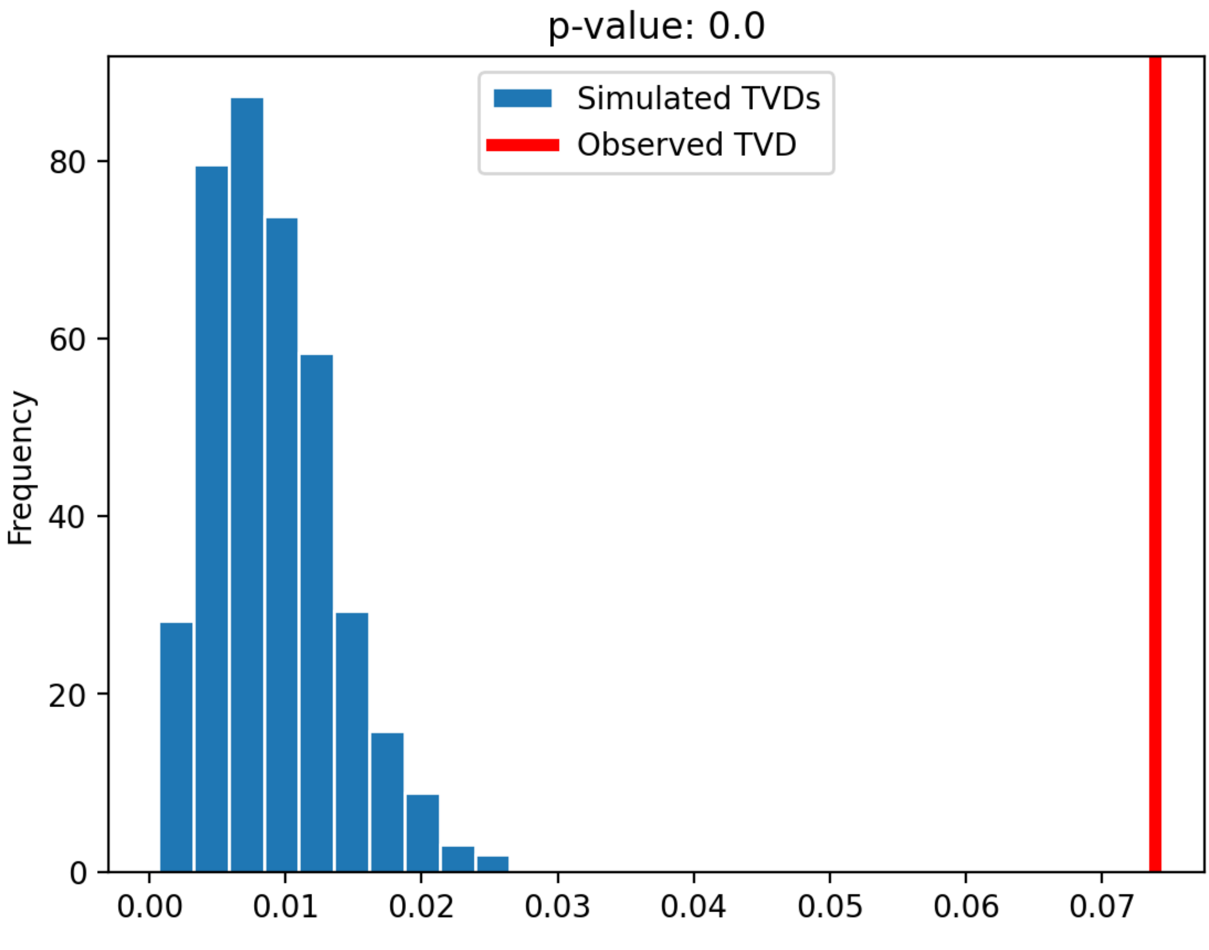
```python
combined['owner'].unique()
```

```
array(['joint', 'self', nan, 'dependent'], dtype=object)
```

```python
def make_dist(df, missing_col, col):
    dist = (
        df
        .assign(**{f'{missing_col}_null': df[missing_col].isna()})
        .pivot_table(index=col, columns=f'{missing_col}_null', aggfunc='siz
    )
    dist = dist / dist.sum()

    dist.plot(kind='barh', legend=True, title=f"{col.capitalize()} by Missi
    plt.show()

    return dist


def calc_tvd(df, missing_col, col):
    dist = (
        df
        .assign(**{f'{missing_col}_null': df[missing_col].isna()})
        .pivot_table(index=col, columns=f'{missing_col}_null', aggfunc='siz
    )
    dist = dist / dist.sum()
    return dist.diff(axis=1).iloc[:, -1].abs().sum() / 2
```

```python
def missingness_perm_test(df, missing_col, col):
    shuffled = df.copy()
    shuffled[f'{missing_col}_null'] = shuffled[missing_col].isna()

    make_dist(df, missing_col, col)
    obs_tvd = calc_tvd(df, missing_col, col)

    n_repetitions = 1000
    tvds = []
    for _ in range(n_repetitions):

        # Shuffling genders and assigning back to the DataFrame
        shuffled[col] = np.random.permutation(shuffled[col])

        # Computing and storing TVD
        tvd = calc_tvd(shuffled, missing_col, col)
        tvds.append(tvd)

    tvds = np.array(tvds)
    pval = np.mean(tvds >= obs_tvd)

    # Draw the p-value graph
    pd.Series(tvds).plot(kind='hist', density=True, ec='w', bins=10, title=
    plt.axvline(x=obs_tvd, color='red', linewidth=4, label='Observed TVD')
    plt.legend()
    plt.show()

    return obs_tvd, pval

obs_tvd, pval = missingness_perm_test(combined, 'owner', 'type')
```



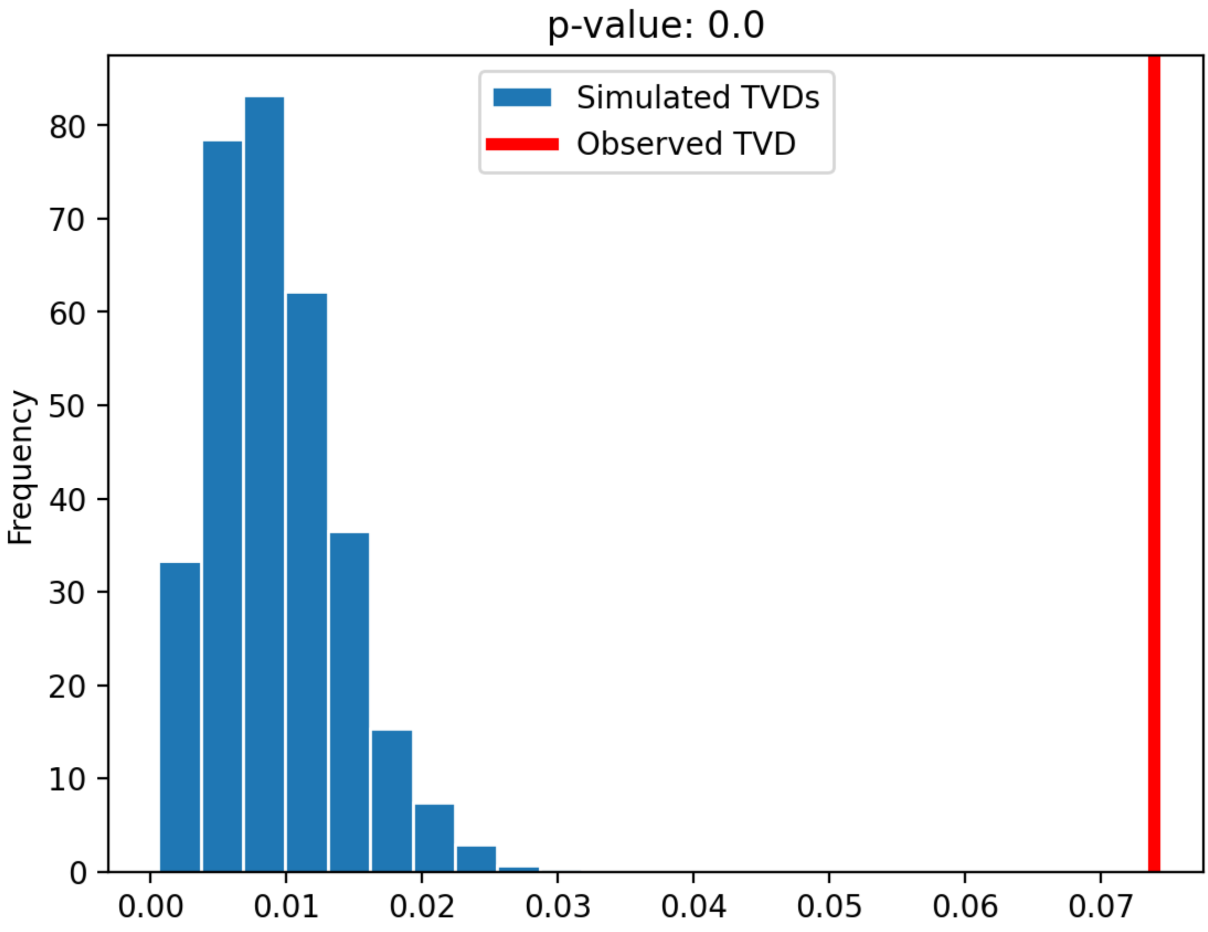



```
shuffled = combined.copy()
shuffled['owner_null'] = shuffled['owner'].isna()
```

```
n_repetitions = 1000
tvds = []
for _ in range(n_repetitions):

    # Shuffling genders and assigning back to the DataFrame
    shuffled['type'] = np.random.permutation(shuffled['type'])

    # Computing and storing TVD
    pivoted = (
        shuffled
        .pivot_table(index='type', columns='owner_null', aggfunc='size')
        .apply(lambda x: x / x.sum(), axis=0)
    )

    tvd = pivoted.diff(axis=1).iloc[:, -1].abs().sum() / 2
    tvds.append(tvd)

tvds = np.array(tvds)
tvds[:10]
```

```
array([0.00394066, 0.01337585, 0.01542276, 0.00559226, 0.01175366,
       0.00862066, 0.00919086, 0.00751793, 0.00661405, 0.01520279])
```

```
pval = np.mean(tvds >= obs_tvd)

pd.Series(tvds).plot(kind='hist', density=True, ec='w', bins=10, title=f'p-
plt.axvline(x=obs_tvd, color='red', linewidth=4, label='Observed TVD')
plt.legend();
```



So we conclude that the missingness of `owner` is **MAR**, and it's dependent on `type` column the most.

Hypothesis Test / Permutation Test

Which party trade more often?

- **Null hypothesis**: the distribution of trading frequency among congresspeople from different party is the same. The difference between the two observed sample is due to chance.
- **Alternative hypothesis**: the distribution of trading frequency among congresspeople from different party are different.

```
combined.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

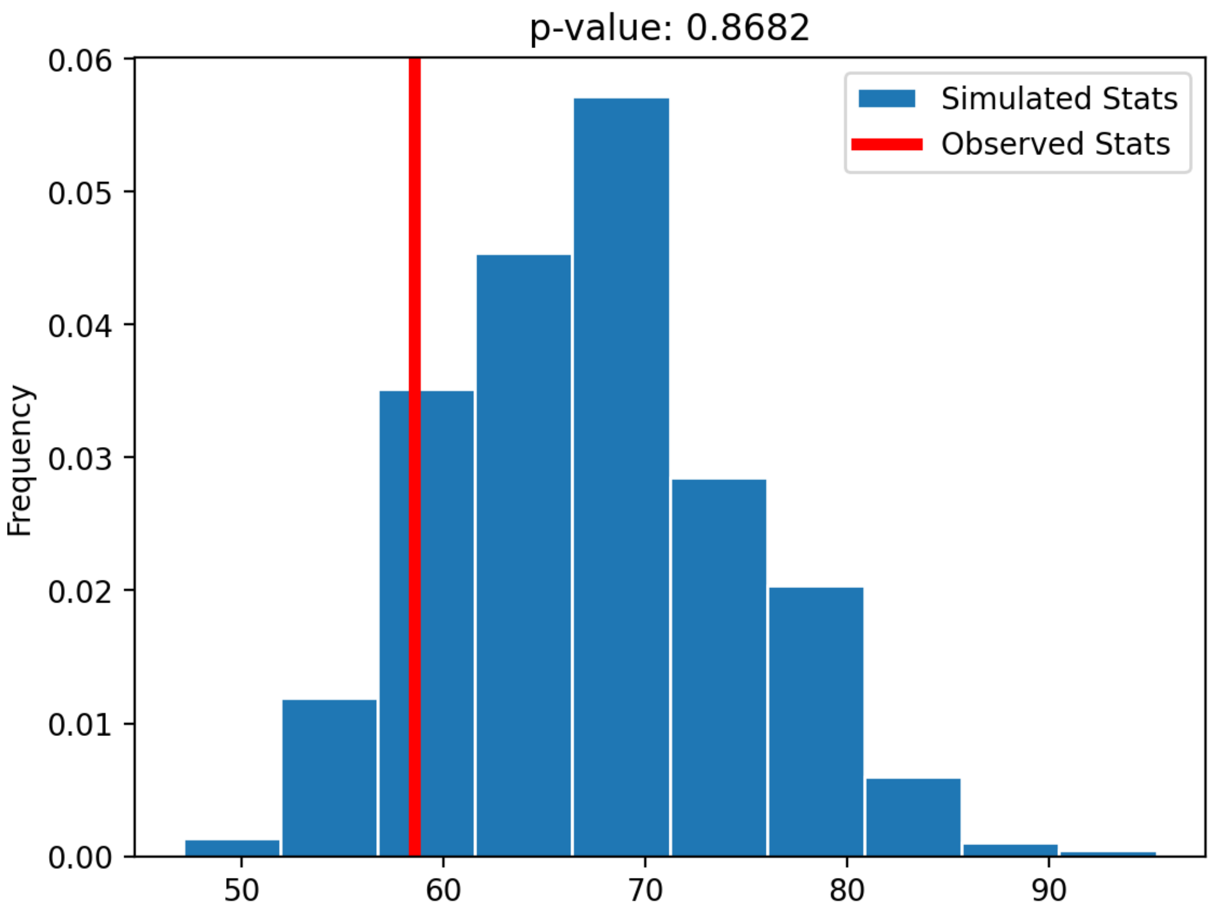
```
    text-align: right;
}
```

</style>

	disclosure_year	disclosure_date	transaction_date	owner	ticker	a
0	2021	2021-10-04	2021-09-27	joint	BP	B
1	2021	2021-10-04	2021-09-13	joint	XOM	E C
2	2021	2021-10-04	2021-09-10	joint	ILPT	Ir L P C
3	2021	2021-10-04	2021-09-28	joint	PM	P Ir
4	2021	2021-10-04	2021-09-17	self	BLK	B

```
df = combined.assign(transaction_year=combined['transaction_date'].dt.year,
                     transaction_month=combined['transaction_date'].dt.mont
df = (
    df
    .groupby(['transaction_year', 'transaction_month', 'party'])[['represen
    .count()
    .reset_index()
)
democrat_stats = df.loc[df['party'] == 'Democrat', 'representative'].sum()
republican_stats =  df.loc[df['party'] == 'Republican', 'representative'].s

obs_stats = abs(democrat_stats - republican_stats)

shuffled = combined.assign(transaction_year=combined['transaction_date'].dt
                           transaction_month=combined['transaction_date'].d

n_repetitions = 5000
stats = []
for _ in range(n_repetitions):

    # Shuffling genders and assigning back to the DataFrame
```

```
    shuffled['party'] = np.random.permutation(shuffled['party'])

    # Computing and storing TVD
    pivoted = (
        shuffled
        .groupby(['transaction_year', 'transaction_month', 'party'])[['repr
        .count()
        .reset_index()
    )

    democrat_stats = pivoted.loc[pivoted['party'] == 'Democrat', 'represent
    republican_stats =  pivoted.loc[pivoted['party'] == 'Republican', 'repr
    stats.append(abs(democrat_stats - republican_stats))

stats = np.array(stats)
pval = np.mean(stats >= obs_stats)

pd.Series(stats).plot(kind='hist', density=True, ec='w', bins=10, title=f'p
plt.axvline(x=obs_stats, color='red', linewidth=4, label='Observed Stats')
plt.legend();
```



Conclusion

The p-value of the permutation test is `0.8722`, which is way larger the the `0.05`. Thus, we **fail to reject** the null hypothesis, which means that distribution of trading frequency among congresspeople from different party may be the same.

```
df = combined.assign(transaction_year=combined['transaction_date'].dt.year,
                     transaction_month=combined['transaction_date'].dt.mont
df = (
    df
    .groupby(['transaction_year', 'transaction_month', 'party'])[['represen
    .count()
    .reset_index()
)
democrat_stats = df.loc[df['party'] == 'Democrat', 'representative'].sum()
republican_stats =  df.loc[df['party'] == 'Republican', 'representative'].s

democrat_stats, republican_stats
```

```
(178.2, 119.65306122448979)
```

```
obs_stats = abs(democrat_stats - republican_stats)
obs_stats
```

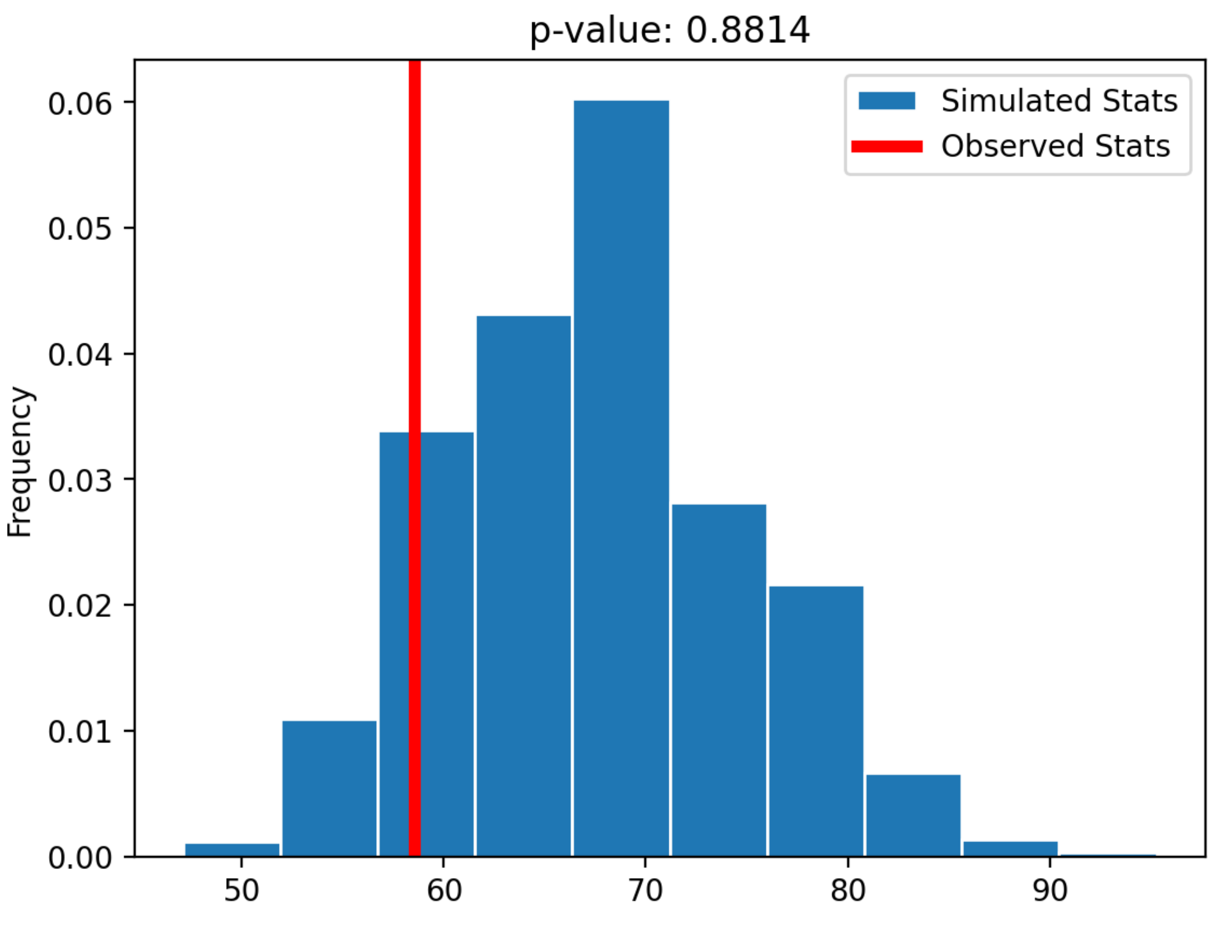
```
58.5469387755102
```

```
shuffled = combined.assign(transaction_year=combined['transaction_date'].dt
                           transaction_month=combined['transaction_date'].d

n_repetitions = 5000
stats = []
for _ in range(n_repetitions):

    # Shuffling genders and assigning back to the DataFrame
    shuffled['party'] = np.random.permutation(shuffled['party'])

    # Computing and storing TVD
    pivoted = (
        shuffled
        .groupby(['transaction_year', 'transaction_month', 'party'])[['repr
        .count()
        .reset_index()
    )

    democrat_stats = pivoted.loc[pivoted['party'] == 'Democrat', 'represent
    republican_stats =  pivoted.loc[pivoted['party'] == 'Republican', 'repr
    stats.append(abs(democrat_stats - republican_stats))
```

```
stats = np.array(stats)
stats[:10]
```

```
array([59.98214286, 72.09833091, 70.87735849, 65.37517483, 74.30188679,
       57.72      , 55.32653061, 68.71225071, 60.90181818, 65.33776224])
```

```
pval = np.mean(stats >= obs_stats)

pd.Series(stats).plot(kind='hist', density=True, ec='w', bins=10, title=f'p
plt.axvline(x=obs_stats, color='red', linewidth=4, label='Observed Stats')
plt.legend();
```




```
ser1 = (
    combined
    .groupby('party')['transaction_date'].agg(['min', 'max'])
    .diff(axis=1)
    .iloc[:, -1]
    .apply(lambda x: x.days)
)
ser2 = combined.groupby('party')['representative'].count()
```

```
ser2 / ser1
```

```
party
Democrat       2.596398
Independent         inf
Republican     3.725079
dtype: float64
```

```
df2 = combined.groupby('party')['representative'].count()
```

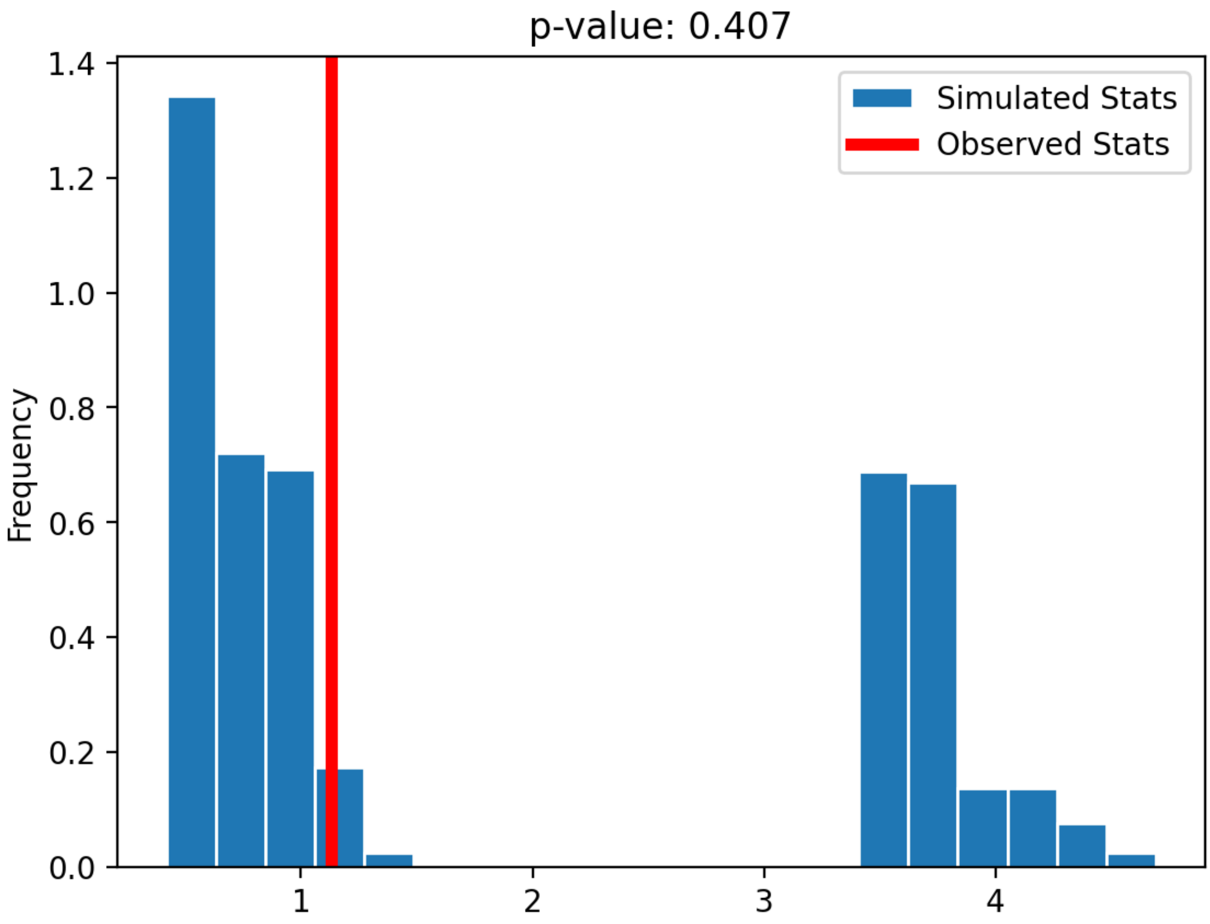
```
def calc_test_statistics(df):
    counts = df.groupby('party')['representative'].count()
    ranges = (
        df
        .groupby('party')['transaction_date'].agg(['min', 'max'])
        .diff(axis=1)
        .iloc[:, -1]
        .apply(lambda x: x.days)
    )
    result = counts / ranges
    return abs(result['Democrat'] - result['Republican'])
```

```
calc_test_statistics(combined)
```

```
1.1286810599946193
```

```
def permutation_test(df, n_repetitions=1000):

    shuffled = df.copy()
    obs_stats = calc_test_statistics(shuffled)

    sim_stats = []
    for _ in range(n_repetitions):

        # Shuffling genders and assigning back to the DataFrame
        shuffled['party'] = np.random.permutation(shuffled['party'])

        # Computing and storing TVD
        stats = calc_test_statistics(shuffled)
        sim_stats.append(stats)

    sim_stats = np.array(sim_stats)
    pval = np.mean(sim_stats >= obs_stats)
```

```
    pd.Series(sim_stats).plot(kind='hist', density=True, ec='w', bins=20, t
    plt.axvline(x=obs_stats, color='red', linewidth=4, label='Observed Stat
    plt.legend()
    plt.show()

    return pval, sim_stats

pval, sim_stats = permutation_test(combined)
sim_stats[:10]
```




```
array([3.49204486, 4.5007299 , 0.74758808, 1.03989987, 4.19301549,
       0.85884316, 3.71209501, 0.81663253, 0.58560186, 0.96367451])
```

```
sim_stats.min(), sim_stats.max()
```

```
(0.4204057417960114, 4.691782988465429)
```

```
combined.sort_values('transaction_date').head(10)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	disclosure_year	disclosure_date	transaction_date	owner	ti
10586	2021	2021-08-26	2012-06-19	NaN	B
11456	2022	2022-03-03	2017-09-05	NaN	S
11437	2022	2022-03-03	2017-12-06	NaN	C
11436	2022	2022-03-03	2018-04-17	NaN	B
11442	2022	2022-03-03	2018-04-30	NaN	C
11453	2022	2022-03-03	2018-05-08	NaN	G
11443	2022	2022-03-03	2018-06-27	NaN	C
11455	2022	2022-03-03	2018-06-27	NaN	I
11457	2022	2022-03-03	2018-06-27	NaN	V

	disclosure_year	disclosure_date	transaction_date	owner	ti
4285	2021	2021-09-28	2018-09-08	dependent	N

```
def to_weekday(x):
    day = x.weekday()
    if day == 0:
        return 'Monday'
    elif day == 1:
        return 'Tuesday'
    elif day == 2:
        return 'Wednesday'
    elif day == 3:
        return 'Thursday'
    elif day == 4:
        return 'Friday'
    elif day == 5:
        return 'Saturday'
    else:
        return 'Sunday'

df = combined.assign(weekday=combined['transaction_date'].apply(to_weekday)
df.groupby('weekday').count().rename(columns={'transaction_date': 'count'})
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	count
weekday	
Friday	3079
Monday	3059
Saturday	48

	count
weekday	
Sunday	27
Thursday	3358
Tuesday	3021
Wednesday	3075

```
df = combined.assign(month=combined['transaction_date'].dt.month)
df.groupby('month').count().rename(columns={'transaction_date': 'count'})[[
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	count
month	
1.0	1561
2.0	2110
3.0	2081
4.0	1438
5.0	1006
6.0	1485
7.0	971
8.0	972
9.0	1056
10.0	856
11.0	1131

	count
month	
12.0	1000