Stock Trades by Members of the US House of Representatives

- A few example prediction questions to pursue are listed below. However, don't limit yourself to them!
  - Can you predict the party affiliation of a representative from their stock trades?
  - Can you predict the geographic region that the representative comes from using their stock trades? E.g., west coast, east coast, south, etc.
  - Can you predict whether a particular trade is a BUY or SELL?

Be careful to justify what information you would know at the "time of prediction" and train your model using only those features.

Summary of Findings

Introduction

We are looking at the stock trades made by members of the US House of Representatives. This data is collected and maintained by Timothy Carambat as part of the House Stock Watcher project. All the data is in the disclosure year 2020 or 2021. Members of Congress must report periodic reports of their asset transactions. This data is purely for an informative purpose and aid in transparency.

We also pulled data from Kaggle on the 117th US Congress (https://www.kaggle.com/adamshl/complete-education-details-117th-us-congress (https://www.kaggle.com/adamshl/complete-education-details-117th-us-congress) (https://www.kaggle.com/adamshl/complete-education-details-117th-us-congress) (https://www.kaggle.com/adamshl/complete-education-details-117th-us-congress))) to match each senator to their political party. Doing so requires cleaning to match the names on both dataframes.

We are using this dataset to predict whether a particular trade is Buy or Sell. We will be using a classification model to solve this problem. We will be looking at the type of purchase in this dataset. Type has 4 different variables: purchase, sale_partial, sale_full, and exchange. Since exchange is only 0.91% of the dataset and it means both buy and sell, we will be dropping all columns to have a clear distinction between buy and sell. We consider purchase as buy and the rest as sell. We will be looking at the accuracy of the model to see whether the model is a good model, over the F1-score because Buy and Sell have the same level of importance.

Baseline Model

In this model we use 9 features out of which 1 of them are quantitative (disclosure_year) and the rest are nominal (amount, district, owner, representative, ticker, type, Political Party, cap_gains_over_200_usd) and we drop the rest of the columns. We drop disclosure_date and transaction_date because we cannot impute datetime objects in a pipeline. We drop asset description because we have tickers instead that are better representative of the stock and drop ptr_link becuase it is irrelevent to the the data

We then one hot encode amount, district, owner, representative, ticker, Political Party. We then use SimpleImputers to fill up the null values because there only exist null values in ticker and owner and we can't make sense of them by keeping them as null values.

We also use simpleimputer to fill any null values in the numerical columns (disclosure_year and cap_gains_over_200_usd) we decide not to standardize it as they aren't continous numerical values.

This model has an R-squared value of 0.6757 and and RMSE of 0.0773. We also get an accuracy of 0.6757. These scores are okay considering the fact that we did not feature engineer anything yet and a lot of data was based on one-hot encoded columns.

Final Model

For our first engineered feature, since we couldn't impute transaction date and disclosure date into our pipeline we create a new feature that is the number of days between the transaction date and disclosure date. We believe this will be a good measure that can be used to predict whether a trade is a sell or a buy. This feature is added to the dataframe under the column name 'transaction to disclosure days'

The next feature we engineer is take out the minimum values for the amount category that the stocks are sold at or bought at. We believe this will be a good measure as sales might be at a higher margin than prices and thus can be used as a feature to predict whether a trade is a sell or a buy. This feature is added to the dataframe under the column name 'amount lower limit'

We decided to go ahead with a DecisionTreeClassifier because of its ability to use different feature subsets and decision rules at different stages of classification. We then used GridSearchCV on the parameters to figure out the best parameters for the final model and we got max_depth = None, min_samples_leaf = 2, and min_samples_split = 2. We also used PCA to remove any correlated columns in the pipeline. We did not use any other hyperparameters because the computation time was too high.

We also looked into LogisticRegression, RandomForestClassifier and DummyClassifier, but in the end DecisionTreeClassifier gave us the best accuracy.

This model has an R-squared value of 0.6901 and and RMSE of 0.0458. We also get an accuracy of 0.6901. This is better than the baseline model and therefore this model is better suited for our predictions.

Fairness Evaluation

To evaluate the fairness of our model we look at the days between transaction date and disclosure date, specifically the feature engineered column 'transaction to disclosure days'. Using this column we create is_long by binning the values into above or below 35 days. We decided to go with 35 days as it was the median. The metric that we decide to use is the accuracy parity. We decide to us this parity because the impacts of false positives and false negatives are similar in the final model. If our permutation test gives a p-value less than 0.05 then we can say that there is bias and our model is not treating is_long column fairly.

Our null hypothesis is that our model is fair and there is no difference in accuracy for the two subsets. Our alternative hypothesis is that our model isn't fair and there is a significant difference in our metric. We use the significance level of 0.05.

After running the permutation test we get a p-value of 0.35. Thus, we accept the null hypothesis and cannot conclude that there is a significant difference between the accuracy of both subsets. This has us believe that our model is treating all values in the is_long column fairly.

Code

In [1]:
```
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
```

In [2]:
```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
import sklearn.preprocessing as pp

from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import TruncatedSVD
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestRegressor
from sklearn.decomposition import PCA
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.dummy import DummyClassifier
```

Cleaning Data

In [660]:
```
#actual dataset
url = "https://house-stock-watcher-data.s3-us-west-2.amazonaws.com/data/all
fp = pd.read_json(url)
fp.head()
```

Out[660]:

	amount	asset_description	cap_gains_over_200_usd	disclosure_date	disclosure_year	district
0	$1,001 - $15,000	BP plc	False	10/04/2021	2021	NC05
1	$1,001 - $15,000	Exxon Mobil Corporation	False	10/04/2021	2021	NC05
2	$15,001 - $50,000	Industrial Logistics Properties Trust - Common...	False	10/04/2021	2021	NC05
3	$15,001 - $50,000	Phillip Morris International Inc	False	10/04/2021	2021	NC05
4	$1,001 - $15,000	BlackRock Inc	False	10/04/2021	2021	CA47

In [661]:
```
df = fp.copy()
df['representative'] = df['representative'].str.lstrip('Hon. ')
```

In [662]:
```
#dataset containing political party
new_fp = os.path.join('archive (1)', 'Updated Education.xlsx')
fs = pd.read_excel(new_fp)
fs.head()
```

Out[662]:

	Name	Twitter	Instagram	Political Party	House or Senate	DOB	Date Values	Birth Year	Generation	E
0	Richard Shelby	93.1K	4104	Republican	Senate	5/6/1934	1934-06-05 00:00:00	1934	NaN	
1	Tommy Tuberville	74.3K	9984	Republican	Senate	09/18/1954	#VALUE!	#VALUE!	NaN	
2	Dan Sullivan	56.4K	5810	Republican	Senate	11/13/64	#VALUE!	#VALUE!	NaN	
3	Lisa Murkowski	317.3K	15.8K	Republican	Senate	5/22/57	#VALUE!	#VALUE!	NaN	
4	Mark Kelly	495.K	185K	Democratic	Senate	21/02/1964	1964-02-21 00:00:00	1964	NaN	

5 rows × 26 columns

In [663]:
```
def name_fix(x):
    val = x.split()
    name = val[0] + " " + val[-1]
    return name

df['fixed_names'] = df['representative'].apply(name_fix)
```

In [664]:
```
merge_df = fs[['Name' , 'Political Party']]
merge_df['Name'] = merge_df['Name'].str.rstrip()
```

```
/Users/milon/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

In [665]:
```
present_names = dict({
"Mr. TJ John (Tj) Cox" : "TJ Cox" ,
"Robert J. Wittman" : "Rob Wittman" ,
"Michael T. McCaul" : "Mike McCaul" ,
"William R. Keating" : "Bill Keating",
"Gerald E. Connolly" : "Gerry Connolly",
"Sean Patrick Maloney" : "Sean P. Maloney" ,
"Michael K. Simpson" : "Mike Simpson" ,
"Donald Sternoff Honorable Beyer" : "Don Beyer" ,
"Mark Dr Green" : "Mark E. Green" ,
'Robert C. "Bobby" Scott' : "Bobby Scott" ,
"Earl Leroy Carter" : "Buddy Carter" ,
"akeem S. Jeffries" : "Hakeem Jeffries" ,
"Rohit Khanna" : "Ro Khanna" ,
"Richard W. Allen" : "Rick W. Allen" ,
"James E Hon Banks" : "Jim Banks" ,
"arold Dallas Rogers" : "Hal Rogers" ,
"Peter J. Visclosky" : "Pete Visclosky" ,
"David P. Joyce" : "Dave Joyce" ,
"James French Hill" : "French Hill" ,
"Gilbert Cisneros" : "Gil Cisneros" ,
"K. Michael Conaway" : "Mike Conaway" ,
'Charles J. "Chuck" Fleischmann' : "Chuck Fleischmann" , "Mark Green" : "Ma
"James E. Banks" : "Jim Banks" ,
"Donald Sternoff Beyer" : "Don Beyer" ,
"Michael Garcia" : "Mike Garcia" ,
"Bradley S. Schneider" : "Brad Schneider" ,
"Nicholas Van Taylor" : "Van Taylor" ,
"Debbie Wasserman Schultz" : "Debbie Wasserman Schultz", "James Hagedorn" :
"S. Raja Krishnamoorthi" : "Raja Krishnamoorthi" , "Mario Diaz-Balart" : "M
"Linda T. Sanchez" : "Linda Sánchez" ,
"Joseph P. Kennedy" : "Joe Kennedy III" ,
"arley E. Rouda" : "Harley Rouda" ,
"Daniel Meuser" : "Dan Meuser" ,
"David P. Roe" : "Phil Roe" ,
"Robert E. Latta" : "Bob Latta" ,
"Daniel Crenshaw" : "Dan Crenshaw" ,
"Kenneth R. Buck" : "Ken Buck" ,
"Michael John Gallagher" : "Mike Gallagher" ,
"W. Greg Steube" : "Greg Steube" ,
"Neal Patrick Dunn MD, FACS" : "Neal Dunn" ,
"Wm. Lacy Clay" : "Lacy Clay" ,
"John B. Larson" : "John B. Larson"
})
```

In [666]:
```
non_present_names = dict({
"None Kathy Manning": ["Kathy Manning", "Democratic"],
"Marie Newman": ["Marie Newman" ,"Democratic"] ,
"Christopher L. Jacobs" : ["Christopher L. Jacobs", "Republican"] ,
"Mr. Peter Meijer" : ["Mr. Peter Meijer" , "Republican"] ,
"Blake Moore" : ["Blake Moore" , "Republican"],
"Sara Jacobs" : ["Sara Jacobs" , "Democratic"],
"Marjorie Taylor Greene" : ["Marjorie Taylor Greene" , "Republican"], "Augu
"Scott Franklin" : ["Scott Franklin" , "Republican"],
"Victoria Spartz" : ["Victoria Spartz", "Republican"],
"Patrick Fallon" : ["Patrick Fallon" , "Republican"],
"Andrew Garbarino" : ["Andrew Garbarino" , "Republican"],
"Stephanie Bice" : ["Stephanie Bice" , "Republican"],
"Felix Barry Moore" : ["Felix Barry Moore" , "Republican"],
"Mr. Scott Franklin" : ["Scott Franklin" , "Republican"],
"None Victoria Spartz" : ["Victoria Spartz", "Republican"],
"None Marie Newman" : ["Marie Newman" ,"Democratic"] ,
"Mr. Scott Franklin" : ["Scott Franklin" , "Republican"],
"Mrs. Marjorie Taylor Greene" : ["Marjorie Taylor Greene" , "Republican"],
"None Sara Jacobs" : ["Sara Jacobs" , "Democratic"] ,
"None Deborah K. Ross" : ["Deborah K. Ross" , "Democratic"] ,
"Kathy Manning" : ["Kathy Manning" , "Democratic"] ,
"Deborah K. Ross" : ["Deborah K. Ross" , "Democratic"] ,
"David Madison Cawthorn" : ["David Madison Cawthorn" , "Republican"] , "Non
"Cindy Axne" : ["Cindy Axne" , "Democratic"] ,
"Pete Sessions" : ["Pete Sessions" , "Republican"],
"Aston McEachin" : ["Aston McEachin", "Democratic"],
"Michael Burgess": ["Michael Burgess", "Republican"]
})
```

In [667]:
```
#fix names to merge
for x in present_names.keys():
    vals = df[df['representative'] == x]['fixed_names']
    new_val = [present_names[x] for y in range(len(vals))]
    new_val = pd.Series(new_val)
    new_val.index = vals.index
    df.loc[new_val.index, 'fixed_names'] = new_val
```

In [668]:
```
merged_dfs = df.merge(merge_df, how = 'left', left_on = 'fixed_names', rig
```

In [669]:
```
#impute non existent names
for x in non_present_names.keys():
    vals = merged_dfs[merged_dfs['representative'] == x]['fixed_names']
    new_val = [non_present_names[x][0] for y in range(len(vals))]
    new_party = [non_present_names[x][1] for y in range(len(vals))]
    new_val = pd.Series(new_val)
    new_party = pd.Series(new_party)
    new_val.index = vals.index
    new_party.index = vals.index
    merged_dfs.loc[new_val.index, 'fixed_names'] = new_val
    merged_dfs.loc[new_party.index, 'Political Party'] = new_party
```

In [670]:
```
merged_dfs = merged_dfs.drop(['representative', 'Name'], axis = 1)
```

In [671]:
```
merged_dfs = merged_dfs.rename(columns = {'fixed_names' : 'representative'}
```

In [672]:
```
#clean df
merged_dfs.replace('--', np.NaN, inplace = True)
```

In [673]:
```
 # change the type of disclosure date to datetime
merged_dfs['disclosure_date'] = pd.to_datetime(merged_dfs['disclosure_date'
```

In [674]:
```
merged_dfs['transaction_date'].replace('0009-06-09', '2021-06-09', inplace
merged_dfs['transaction_date'].replace('0021-08-02', '2021-08-02', inplace
merged_dfs['transaction_date'].replace('0021-06-22', '2021-06-22', inplace
merged_dfs['transaction_date'].replace('0201-06-22', '2021-06-22', inplace
```

In [675]:
```
 #change the type of transaction date to datetime
merged_dfs['transaction_date'] = pd.to_datetime(merged_dfs['transaction_dat
```

In [676]:
```
#impute data
merged_dfs.loc[751, 'asset_description'] = "Ball Corporation"
merged_dfs.loc[3302, 'asset_description'] = 'Celo'
merged_dfs.loc[10808, 'asset_description'] = 'Urban-gro Inc'
merged_dfs.loc[10809, 'asset_description'] = 'Urban-gro Inc'
```

In [677]:
```
#impute data
merged_dfs.loc[5, 'Political Party'] = "Democratic"
merged_dfs.loc[700, 'Political Party'] = "Democratic"
merged_dfs.loc[5073, 'Political Party'] = "Democratic"
merged_dfs.loc[5074, 'Political Party'] = "Democratic"
merged_dfs.loc[5075, 'Political Party'] = "Democratic"
merged_dfs.loc[3469, 'Political Party'] = "Republican"
merged_dfs.loc[3470, 'Political Party'] = "Republican"
```

Feature Engineering

In [156]:
```
# create new column with amount lower limit
def fix_amount(val):
    val = val.strip('$').split()[0].replace(',', '')
    return int(val)

merged_dfs['amount lower limit'] = merged_dfs['amount'].apply(fix_amount)
```

In [162]:
```
#create new column with days between transaction and disclosure date
merged_dfs['transaction to disclosure days'] = (merged_dfs['disclosure_date
                                                merged_dfs['transaction_dat
```

In [192]:
```
def type_num(val):
    if val == 'purchase':
        return 1
    else:
        return 2
```

In [193]:
```
#Drop Exchange values
merged_df = merged_dfs[merged_dfs['type'] != 'exchange']
```

In [254]:
```
#Change type values to numerical
merged_df['type'] = merged_df['type'].apply(type_num)
```

```
/Users/milon/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  """Entry point for launching an IPython kernel.
```

In [339]:
```
#fill all None and nan values
merged_df = merged_df.fillna(value = np.nan)
```

Baseline Model

In [697]:
```
cols = ['disclosure_date', 'transaction_date', 'ptr_link', 'type', 'asset_d
       'transaction to disclosure days']
```

In [698]:
```
X = merged_df.drop(cols, axis = 1)
y = merged_df['type']
```

In [699]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y)
```

In [700]:
```
types = X.dtypes
catcols = types.loc[types == np.object].index #all categorical columns
numcols = types.loc[types != np.object].index #all numerical columns
```

In [701]:
```
cats = Pipeline([
    ('imp', SimpleImputer(strategy='constant', fill_value='NULL')),
    ('ohe', pp.OneHotEncoder(handle_unknown='ignore', sparse=False)),
])

ct = ColumnTransformer([
    ('catcols', cats, catcols),
    ('numcols', SimpleImputer(strategy='constant', fill_value=0), numcols)
])

baseline_pl = Pipeline([('feats', ct), ('reg', DecisionTreeClassifier())])
```

In [702]:
```
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.25)
```

In [703]:
```
baseline_pl.fit(X_tr, y_tr)
baseline_pl.score(X_ts, y_ts) #R^2
```

Out[703]:
```
0.6757874642061724
```

In [704]:
```
preds = baseline_pl.predict(X_ts)
np.sqrt(np.mean(preds - y_ts)**2) #RMSE
```

Out[704]:
```
0.07731466751511296
```

In [705]:
```
tn, fp, fn, tp= metrics.confusion_matrix(y_ts, preds).ravel() / len(preds)
```

In [706]:
```
accuracy = tn + tp
accuracy
```

Out[706]:
```
0.6757874642061725
```

Hypertuning Parameters

In [628]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

In [629]:
```
baseline_pl.get_params().keys()
```

Out[629]:
```
dict_keys(['memory', 'steps', 'feats', 'reg', 'feats__n_jobs', 'feats__re
mainder', 'feats__sparse_threshold', 'feats__transformer_weights', 'feats
__transformers', 'feats__catcols', 'feats__numcols', 'feats__catcols__mem
ory', 'feats__catcols__steps', 'feats__catcols__imp', 'feats__catcols__oh
e', 'feats__catcols__imp__copy', 'feats__catcols__imp__fill_value', 'feat
s__catcols__imp__missing_values', 'feats__catcols__imp__strategy', 'feats
__catcols__imp__verbose', 'feats__catcols__ohe__categorical_features', 'f
eats__catcols__ohe__categories', 'feats__catcols__ohe__dtype', 'feats__ca
tcols__ohe__handle_unknown', 'feats__catcols__ohe__n_values', 'feats__cat
cols__ohe__sparse', 'feats__numcols__copy', 'feats__numcols__fill_value',
'feats__numcols__missing_values', 'feats__numcols__strategy', 'feats__num
cols__verbose', 'reg__class_weight', 'reg__criterion', 'reg__max_depth',
'reg__max_features', 'reg__max_leaf_nodes', 'reg__min_impurity_decrease',
'reg__min_impurity_split', 'reg__min_samples_leaf', 'reg__min_samples_spl
it', 'reg__min_weight_fraction_leaf', 'reg__presort', 'reg__random_stat
e', 'reg__splitter'])
```

In [517]:
```
parameters = {
    'reg__max_depth': [2,3,4,5,7,10,13,15,18,None],
    'reg__min_samples_split':[2,3,5,7,10,15,20],
    'reg__min_samples_leaf':[2,3,5,7,10,15,20]
}
```

In [518]:
```
clf = GridSearchCV(baseline_pl, parameters, cv = 5)
```

In [519]:
```
clf.fit(X_train, y_train)
```

Out[519]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('feats', ColumnTransformer(n_jobs=None, remainder='drop', sp
arse_threshold=0.3,
         transformer_weights=None,
         transformers=[('catcols', Pipeline(memory=None,
     steps=[('imp', SimpleImputer(copy=True, fill_value='NULL', missing_v
alues=nan,
       strategy='constant', verbose=...      min_weight_fraction_leaf=0.
0, presort=False, random_state=None,
            splitter='best'))]),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'reg__max_depth': [2, 3, 4, 5, 7, 10, 13, 15, 18, Non
e], 'reg__min_samples_split': [2, 3, 5, 7, 10, 15, 20], 'reg__min_samples
_leaf': [2, 3, 5, 7, 10, 15, 20]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [540]:
```
clf.best_params_ # best parameters
```

Out[540]:
```
{'reg__max_depth': None,
 'reg__min_samples_leaf': 2,
 'reg__min_samples_split': 2}
```

Final Model

In [707]:
```
cols = ['disclosure_date', 'transaction_date', 'ptr_link', 'type', 'asset_d
```

In [708]:
```
X = merged_df.drop(cols, axis = 1)
y = merged_df['type']
```

In [709]:
```
types = X.dtypes
catcols = types.loc[types == np.object].index
numcols = types.loc[types != np.object].index
```

In [710]:
```
cats = Pipeline([
    ('imp', SimpleImputer(strategy='constant', fill_value='NULL')),
    ('ohe', pp.OneHotEncoder(handle_unknown='ignore', sparse=False)),
    ('pca', PCA(svd_solver='full', n_components=0.85))
])

ct = ColumnTransformer([
    ('catcols', cats, catcols),
    ('numcols', SimpleImputer(strategy='constant', fill_value=0), numcols)
])

pl = Pipeline([('feats', ct), ('reg', DecisionTreeClassifier(max_depth = No
 min_samples_leaf = 2,
 min_samples_split = 2))])
```

In [711]:
```
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.25)
```

In [712]:
```
pl.fit(X_tr, y_tr)
pl.score(X_ts, y_ts) #R^2
```

Out[712]:
```
0.6901049952274897
```

In [713]:
```
preds = pl.predict(X_ts)
np.sqrt(np.mean(preds - y_ts)**2) #RMSE
```

Out[713]:
```
0.04581609926821508
```

In [714]:
```
tn, fp, fn, tp = metrics.confusion_matrix(y_ts, preds).ravel() /len(preds)
```

In [715]:
```
accuracy = (tn + tp)
accuracy
```

Out[715]:
```
0.6901049952274897
```

In [716]:
```
metrics.confusion_matrix(y_ts, preds) / len(preds)
```

Out[716]:
```
array([[0.38625517, 0.13203945],
       [0.17785555, 0.30384983]])
```

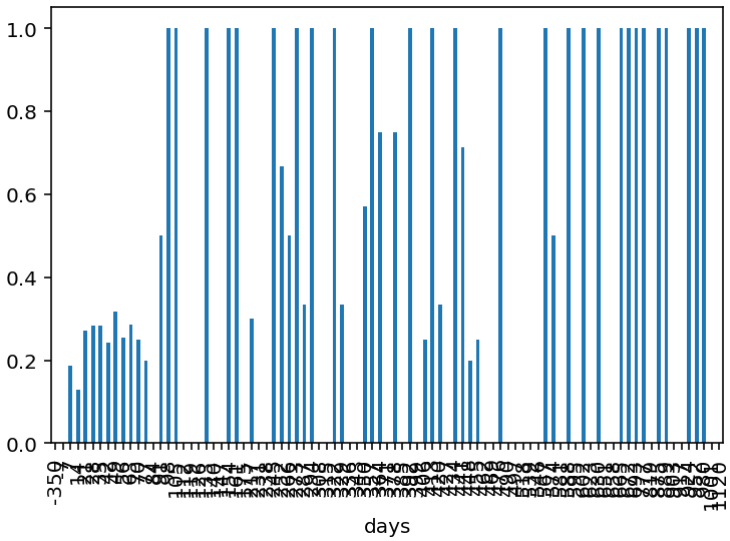
Fairness Evaluation

In [717]:
```
results = X_ts
results['days'] = results['transaction to disclosure days'].apply(lambda x:
results['prediction'] = preds
results['type'] = y_ts

(
    results
    .groupby('days')
    .apply(lambda x:1 - metrics.recall_score(x.type, x.prediction))
    .plot(kind='bar')
)
```

```
/Users/milon/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)

/Users/milon/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  This is separate from the ipykernel package so we can avoid doing impor
ts until
/Users/milon/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  after removing the cwd from sys.path.
/Users/milon/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classi
fication.py:1145: UndefinedMetricWarning: Recall is ill-defined and being
set to 0.0 due to no true samples.
  'recall', 'true', average, warn_for)
/Users/milon/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classi
fication.py:1145: UndefinedMetricWarning: Recall is ill-defined and being
set to 0.0 due to no true samples.
  'recall', 'true', average, warn_for)
/Users/milon/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classi
fication.py:1145: UndefinedMetricWarning: Recall is ill-defined and being
set to 0.0 due to no true samples.
  'recall', 'true', average, warn_for)
/Users/milon/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classi
fication.py:1145: UndefinedMetricWarning: Recall is ill-defined and being
set to 0.0 due to no true samples.
  'recall', 'true', average, warn_for)
/Users/milon/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classi
```

```
fication.py:1145: UndefinedMetricWarning: Recall is ill-defined and being
set to 0.0 due to no true samples.
  'recall', 'true', average, warn_for)
```

Out[717]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd79987f710>



In [718]:
```
results['is_long'] = (results['transaction to disclosure days'] <= 35).repl
```

```
/Users/milon/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:
1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  """Entry point for launching an IPython kernel.
```

In [720]:
```
#Accuracy Parity
(
    results
    .groupby('is_long')
    .apply(lambda x: metrics.accuracy_score(x.type, x.prediction))
    .rename('accuracy')
    .to_frame()
)
```

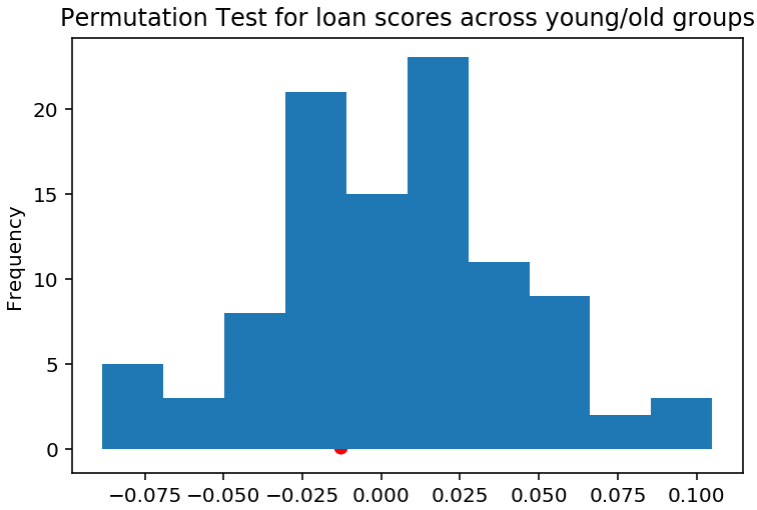
Out[720]:

is_long	accuracy
long	0.698980
not_long	0.686084

In [721]:
```
#permutation test
obs = results.groupby('is_long').apply(lambda x: metrics.accuracy_score(x.t

metrs = []
for _ in range(100):
    s = (
        results[['is_long', 'prediction', 'type']]
        .assign(is_long=results.is_long.sample(frac=1.0, replace=False).res
        .groupby('is_long')
        .apply(lambda x: metrics.accuracy_score(x.type, x.prediction))
        .diff()
        .iloc[-1]
    )

    metrs.append(s)
```

In [722]:
```
print(pd.Series(metrs <= obs).mean()) #p-value
pd.Series(metrs).plot(kind='hist', title='Permutation Test for loan scores
plt.scatter(obs, 0.1, c='r');
```

```
0.35
```



In [ ]:
```
```