02_EDA

December 11, 2018

```
In [1]: import requests
        from IPython.core.display import HTML
        styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A/maste
        HTML(styles)

        from bs4 import BeautifulSoup
        import re
        import pandas as pd
        import time
        import json
        from pathlib import Path
        import numpy as np
        import os
        from os import listdir
        from os.path import isfile, join
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.preprocessing import StandardScaler
        from sklearn.linear_model import LogisticRegression,LogisticRegressionCV, LinearRegress
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.feature_selection import SelectFromModel
        from sklearn.metrics import accuracy_score
        import scipy.stats as stats
```

1 EDA

1.1 Datasummary information

```
In [2]: #Get the data extracted from different sources
        house_df = pd.read_csv('data/ready_to_use_dataset.csv')
        house_df = house_df.drop_duplicates(['year', 'state', 'district', 'name'])
        display(house_df.shape)
```

```
(9974, 20)
```

```
In [3]: #check for datatypes
        display(house_df.dtypes)
```

```
district                             object
is_incumbent                        float64
name                                 object
party                                object
percent                             float64
state                                object
votes                               float64
won                                   int64
year                                  int64
first_time_elected                  float64
count_victories                       int64
unemployement_rate                  float64
is_presidential_year                float64
president_can_be_re_elected         float64
president_party                      object
president_overall_avg_job_approval    float64
last_D_house_seats                  float64
last_R_house_seats                  float64
last_house_majority                  object
fundraising                         float64
dtype: object
```

In [4]: display(house_df.head())

```
     district  is_incumbent          name party  percent    state    votes  \
0  District 1           0.0  Ratliff Boon     D     42.1  Indiana   4281.0
1  District 1           1.0  Ratliff Boon     D     42.8  Indiana   5202.0
2  District 1           1.0  Ratliff Boon     D     52.2  Indiana   7272.0
3  District 1           0.0      John Law     D     49.1  Indiana  10868.0
4  District 1           1.0  Ratliff Boon     D     50.9  Indiana  11280.0

   won  year  first_time_elected  count_victories  unemployement_rate  \
0    1  1824              1824.0                0                 NaN
1    1  1826              1824.0                1                 NaN
2    1  1828              1824.0                2                 NaN
3    0  1830              1860.0                0                 NaN
4    1  1830              1824.0                3                 NaN

   is_presidential_year  president_can_be_re_elected president_party  \
0                   NaN                          NaN             NaN
1                   NaN                          NaN             NaN
2                   NaN                          NaN             NaN
3                   NaN                          NaN             NaN
4                   NaN                          NaN             NaN

   president_overall_avg_job_approval  last_D_house_seats  last_R_house_seats  \
0                                 NaN                 NaN                 NaN
```

```
1                                 NaN                 NaN                 NaN
2                                 NaN                 NaN                 NaN
3                                 NaN                 NaN                 NaN
4                                 NaN                 NaN                 NaN

  last_house_majority  fundraising
0                 NaN          NaN
1                 NaN          NaN
2                 NaN          NaN
3                 NaN          NaN
4                 NaN          NaN
```

In [5]:
```
#get columns with NaN data
house_df.isna().sum()
```

Out[5]:
```
district                                0
is_incumbent                          112
name                                    0
party                                   0
percent                                15
state                                   0
votes                                  67
won                                     0
year                                    0
first_time_elected                   4445
count_victories                         0
unemployement_rate                    979
is_presidential_year                  102
president_can_be_re_elected           102
president_party                       102
president_overall_avg_job_approval   1060
last_D_house_seats                    102
last_R_house_seats                    102
last_house_majority                   102
fundraising                          7172
dtype: int64
```

In [6]:
```
#data normalisation
def normalise_df(df, mins, maxs):
    df = (df - mins)/(maxs - mins)
    return df
```

In [7]:
```
def clean_nan_model(data):
    #model based imputation for columns fundraising and president_overall_avg_job_appr
    #data: dataframe which is cleaned of NaNs but not for the 2 mentioned variables

    #These are the 2 columns which will be imputed
    missing_cols = ['fundraising', 'president_overall_avg_job_approval']
```

```
data_origin = data.copy()
target_col = ['party'] #response variable

#category variables will be dropped
del_columns = ['district','president_party','last_house_majority', 'name', 'state']
data = data.drop(columns = del_columns)

#model can not deal with NaN values so we change them to the number 1 which didn't
#for those columns in missing_cols
data = data[missing_cols].fillna(1)

# dataset without any missing values; not normalised
clean_data = data[~((data[missing_cols[0]]==1) |
                  (data[missing_cols[1]]==1))]

# dataset with missing values that need to be imputed; not normalised
unclean_data = data[((data[missing_cols[0]]==1) |
                  (data[missing_cols[1]]==1))]

unclean_df = unclean_data.copy() # making fresh copy of unclean dataset
train_data = data.copy() #start with original dataset

# running for 20 iterations for robustness
for it in range(20):
    # finding missing values to be imputed using multiple linear regression model
    for col in missing_cols:
        sub_train = train_data
        sub_test = unclean_data[unclean_data[col] == 1] # subset of unclean data w

        #split the data
        sub_xtrain, sub_ytrain = sub_train[sub_train.columns.difference([col]+targe
        sub_xtest, sub_ytest = sub_test[sub_test.columns.difference([col]+target_co

        # normalising the train and test predictors
        sub_mins, sub_maxs = sub_xtrain.min(), sub_xtrain.max()
        sub_xtrain = normalise_df(sub_xtrain, sub_mins, sub_maxs)
        sub_xtest = normalise_df(sub_xtest, sub_mins, sub_maxs)

        #Imputation with linear regression
        linreg = LinearRegression(fit_intercept=True)
        linreg.fit(sub_xtrain, sub_ytrain)
        sub_ytest_hat = linreg.predict(sub_xtest)

        # impute values in the unclean dataframe
        unclean_df[col].replace([1]*len(sub_ytest_hat), sub_ytest_hat, inplace=True

        # re-construct the train dataset by combining clean data with newly impute
```

```
            train_data = unclean_df.append(clean_data)

        return  train_data[missing_cols]
```

In [8]:
```
#get rid of NaNs
def clean_nan(data,i_type='mean'):
    #cleans NaNs
    #data: dataframe
    #i_type: if mean -> mean imputation only
    #            if model -> model imputation for undraising and president_overall_avg_

    #delete duplicates
    data = data.drop_duplicates(['year', 'state', 'district', 'name'])

    #needed just in case if not all NaNs are imputed with aggregated mean for fundrais
    mean_fund = data.fundraising.mean()

    #groups needed for imputation
    gr_dist = data.groupby(['state', 'district'])
    gr_year = data.groupby(['state', 'district','year'])

    #imputation of values
    if i_type == 'mean':
        data['president_overall_avg_job_approval'].fillna(gr_dist['president_overall_av
        data['fundraising'].fillna(gr_dist['fundraising'].transform('mean'), inplace=Tr
        data['fundraising'].fillna(mean_fund, inplace=True) #necessary if in first fun
    else:
        model_df = clean_nan_model(data)
        data['fundraising'] = model_df.fundraising
        data['president_overall_avg_job_approval']=model_df.president_overall_avg_job_a
    data['votes'].fillna(gr_dist['votes'].transform('mean'), inplace=True)
    data['last_D_house_seats'].fillna(gr_dist['last_D_house_seats'].transform('mean'),
    data['last_R_house_seats'].fillna(gr_dist['last_R_house_seats'].transform('mean'),
    data['percent'].fillna(100 - gr_year['percent'].transform('sum'), inplace=True)
    data['unemployement_rate'].fillna(gr_dist['unemployement_rate'].transform('mean'),
    data['is_presidential_year'].fillna(0, inplace=True)
    data['president_can_be_re_elected'].fillna(1, inplace=True)
    data['president_party'].fillna(0, inplace=True)
    s = gr_year['is_incumbent'].transform("sum")
    r=[]
    for index, item in enumerate(s):
        if s[item] > 0:
            r.append(0)
        else:
            r.append(1)
    r = pd.Series(r)
    data['is_incumbent'].fillna(r, inplace=True)
    data['last_house_majority'].fillna(gr_dist['last_house_majority'].transform(lambda
```

```
        data.loc[data['first_time_elected'].isna() & (data['won']==1),'first_time_elected']
        data.loc[data['first_time_elected'].isna() & (data['won']==0),'first_time_elected']

        return data
```

In [ ]:
```
#clean orignal dataset nan_df with mean only
house_df = pd.read_csv('data/ready_to_use_dataset.csv')
house_df_mean = clean_nan(house_df,i_type='mean')
```

In [10]:
```
#test if imputation worked well
house_df_mean.isna().sum()
```

Out[10]:
```
district                             0
is_incumbent                         0
name                                 0
party                                0
percent                              0
state                                0
votes                                0
won                                  0
year                                 0
first_time_elected                   0
count_victories                      0
unemployement_rate                   0
is_presidential_year                 0
president_can_be_re_elected          0
president_party                      0
president_overall_avg_job_approval   0
last_D_house_seats                   0
last_R_house_seats                   0
last_house_majority                  0
fundraising                          0
dtype: int64
```

In [ ]:
```
#model based imputation
house_df = pd.read_csv('data/ready_to_use_dataset.csv')
house_df_model = clean_nan(house_df, i_type='model')
```

In [12]:
```
#test if model based imputation worked
house_df_model.isna().sum()
```

Out[12]:
```
district                             0
is_incumbent                         0
name                                 0
party                                0
percent                              0
state                                0
votes                                0
won                                  0
```

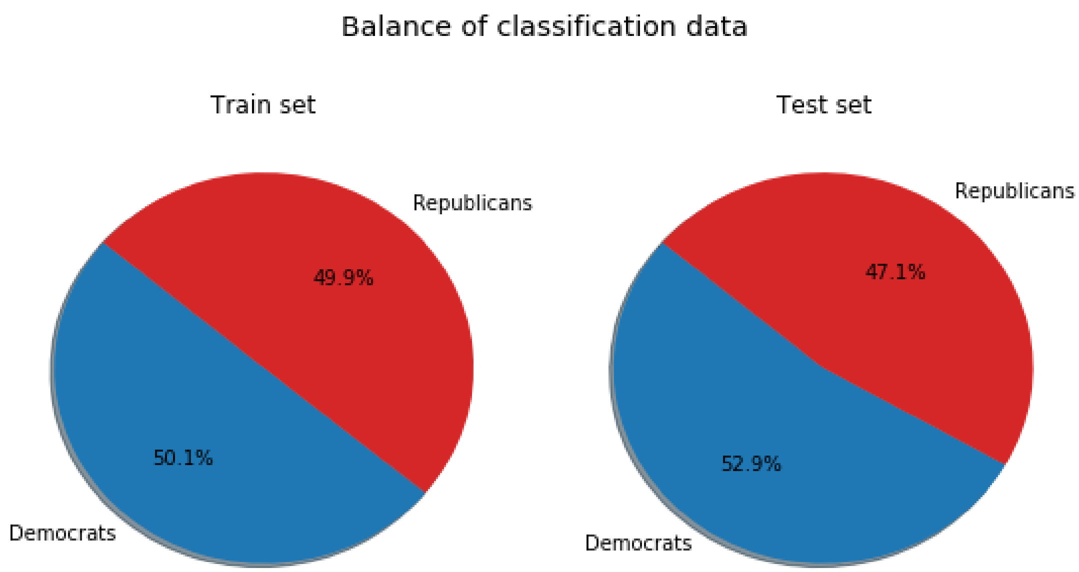
```
year                                  0
first_time_elected                    0
count_victories                       0
unemployement_rate                    0
is_presidential_year                  0
president_can_be_re_elected           0
president_party                       0
president_overall_avg_job_approval    0
last_D_house_seats                    0
last_R_house_seats                    0
last_house_majority                   0
fundraising                           0
dtype: int64
```

In [13]:
```
#save model and mean imputed data to csv
house_df_mean.to_csv('data/house_mean_imputation.csv', index=False)
house_df_model.to_csv('data/house_model_imputation.csv', index=False)
```

In [20]:
```
#palettes for parties or other
Parties_palette=[(0.12156862745098039, 0.4666666666666667, 0.7058823529411765),
             (0.8392156862745098, 0.15294117647058825, 0.1568627450980392),
             (0.17254901960784313, 0.6274509803921569, 0.17254901960784313),
             (1.0, 0.4980392156862745, 0.054901960784313725),
             (0.5803921568627451, 0.403921568627451, 0.7411764705882353),
             (0.5490196078431373, 0.33725490196078434, 0.29411764705882354),
             (0.8901960784313725, 0.4666666666666667, 0.7607843137254902),
             (0.4980392156862745, 0.4980392156862745, 0.4980392156862745),
             (0.7372549019607844, 0.7411764705882353, 0.13333333333333333),
             (0.09019607843137255, 0.7450980392156863, 0.8117647058823529)]
WinLosePalette=[(0.17254901960784313, 0.6274509803921569, 0.17254901960784313),
             (1.0, 0.4980392156862745, 0.054901960784313725),
             (0.5803921568627451, 0.403921568627451, 0.7411764705882353),
             (0.5490196078431373, 0.33725490196078434, 0.29411764705882354),
             (0.8901960784313725, 0.4666666666666667, 0.7607843137254902),
             (0.4980392156862745, 0.4980392156862745, 0.4980392156862745),
             (0.7372549019607844, 0.7411764705882353, 0.13333333333333333),
             (0.09019607843137255, 0.7450980392156863, 0.8117647058823529)]
DEM_blue=Parties_palette[0]
REP_red=Parties_palette[1]
#sns.palplot(Parties_palette)
```

In [21]:
```
#check balance of classification data
test_D = len(house_df_mean[(house_df_mean['won']==1) & (house_df_mean['party']=='D')&
test_R = len(house_df_mean[(house_df_mean['won']==1) & (house_df_mean['party']=='R')&
train_D = len(house_df_mean[(house_df_mean['won']==1) & (house_df_mean['party']=='D')&
train_R = len(house_df_mean[(house_df_mean['won']==1) & (house_df_mean['party']=='R')&

labels = ['Democrats', 'Republicans']
```

```
sets = ['Train set', 'Test set']
sizes = [[train_D,train_R],[test_D,test_R]]
colors = [DEM_blue,REP_red]

fig = plt.figure(figsize=(8,4))
fig.suptitle('Balance of classification data', y=1.1,fontsize=14)
for i in range(len(sizes)):
    plt.subplot(1,2,i+1)
    plt.pie(sizes[i], labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)
    plt.title(sets[i])
fig
plt.tight_layout()
```




```
In [22]: #Get correlations between variables  when Democrats or Republicans win
         def corr_heat(dataframe,party='D'):
             f, ax = plt.subplots(figsize=(8, 6))
             corr = dataframe[dataframe['party'] == party].drop('party', axis=1).corr()
             sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_pal
                         square=True, ax=ax)
             ax.set_title('Correlations when party {} won'.format(party))

In [23]: #Display correlations
         corr_heat(house_df_mean[house_df_mean['year']!=2018], party='D')
         corr_heat(house_df_mean[house_df_mean['year']!=2018], party='R')
```


Correlations when party D won
is_incumbent
percent
votes
won
year
first_time_elected
count_victories
unemployement_rate
is_presidential_year
president_can_be_re_elected
president_overall_avg_job_approval
last_D_house_seats
last_R_house_seats
fundraising
0.9
0.6
0.3
0.0
−0.3
−0.6

2 Variable Selection

```
In [24]: #these are the categorical variables
         cat_cols=['president_party','state','district','last_house_majority','name']
```

```
In [25]: #one-hot-coding is necessary because of category variables
         #because it is no ordinary data we cannot do label encoding. So we do one-hot-encodin
         def one_hot_coding(data,cat_cols,y_year=2018):
             #returns x and y variables for test and train
             #data: dataframe
             #cat_cols: enter array with category columns
             #y_year: enter year for test data

             #create dummy features
             data = pd.get_dummies(data, columns=cat_cols)
```

```
    #create data sets for test and training
    sel_train, sel_test=data[data['year']!=y_year], data[data['year']==y_year]

    #split for x and y
    x_train, y_train=sel_train.drop('party', axis=1), sel_train['party']
    x_test, y_test=sel_test.drop('party', axis=1), sel_test['party']

    return x_train, y_train, x_test, y_test
```

2.0.1 Variable Selection - categorical variables

```
In [26]: #Chi Square Test
         def chi2_test(x_col,y_col):
             x = x_col.astype(str)
             y= y_col.astype(str)

             obs_val = pd.crosstab(y,x)
             chi2, p, dof, expected = stats.chi2_contingency(obs_val.values)

             return chi2, p, dof
```

```
In [27]: def print_chi2_result(data,y_col='party',cat_cols=cat_cols,alpha=0.05):
             for i in range(len(cat_cols)):
                 chi2, p, dof = chi2_test(data[cat_cols[i]],data[y_col])
                 if p>alpha:
                     print('Important for the prediction model: {} (p-value: {:+.3f}, chi2: {:+
                 else:
                     print('\033[1mNOT\033[0m important for the prediction model: \033[1m{}\033
```

```
In [28]: #Print the result of Chi Square Test
         print_chi2_result(house_df)
```

```
Important for the prediction model: president_party (p-value: +0.210, chi2: +3.1)
Important for the prediction model: state (p-value: +1.000, chi2: +19.7)
Important for the prediction model: district (p-value: +1.000, chi2: +14.7)
Important for the prediction model: last_house_majority (p-value: +0.314, chi2: +2.3)
NOT important for the prediction model: name (p-value: +0.000, chi2: +9955.0)
```

Interpretation:

- Column "name" is not useful in the models
- But because of feature engineering during the modeling it will be needed

2.1 Variable Selection - Random Forest with one-hot-coding

```
In [29]: #copy of original dataset
         forest_df = house_df_mean.copy()
```

```python
#Exclude column name because of low p-value - see chapter "Variable Selection - categ
forest_df = forest_df.drop(columns = 'name')

#categorical columns for Random Forest model
forest_cat=['president_party','state','district','last_house_majority']
```

In [30]:
```python
def var_sel_RF(forest_df,forest_cat=forest_cat,y_year=2018, threshold=0.003):
    #returns 1) sorted list of most important features
    #        2) Accuracy of model with all features and with selected features
    #thresold: minimum feature importance

    x_train, y_train, x_test, y_test = one_hot_coding(forest_df,forest_cat,y_year)

    # Create a random forest classifier. number of trees set to 100
    clf = RandomForestClassifier(n_estimators=100, random_state=0, n_jobs=-1)

    # Train the classifier
    clf.fit(x_train, y_train)
    feat_labels = x_train.columns
    feat_imp = []

    # name and gini importance of each feature
    for feature in zip(clf.feature_importances_,feat_labels):
        feat_imp.append(feature)
    feat_imp.sort(reverse=True)

    #sorted list with most important features
    feat_imp = list(filter(lambda x: x[0] > threshold, feat_imp))

    # Create a selector object that will use the random forest classifier to identify
    # features that have an importance of more than 0.003
    sfm = SelectFromModel(clf, threshold=threshold)

    # Train the selector
    sfm.fit(x_train, y_train)

    # Transform the data to create a new dataset containing only the most important f
    # Note: We have to apply the transform to both the training X and test X data.
    X_important_train = sfm.transform(x_train)
    X_important_test = sfm.transform(x_test)

    # Create a new random forest classifier for the most important features
    clf_important = RandomForestClassifier(n_estimators=100, random_state=0, n_jobs=-

    # Train the new classifier on the new dataset containing the most important featu
    clf_important.fit(X_important_train, y_train)

    # Accuracy of model with all features
```

```
    y_pred = clf.predict(x_test)
    print('Accuracy of model with all features: {:+.3f}'.format(accuracy_score(y_test

    # Accuracy of model with most important features
    y_important_pred = clf_important.predict(X_important_test)
    print('Accuracy of model with most important features: {:+.3f}'.format(accuracy_s

    return feat_imp
```

In [31]:
```
var_sel_RF(forest_df,forest_cat=forest_cat,y_year=2018, threshold=0.005)
```

```
Accuracy of model with all features: +0.754
Accuracy of model with most important features: +0.683
```

Out[31]:
```
[(0.12887509285513132, 'percent'),
 (0.12211427246664142, 'votes'),
 (0.0915045815689258, 'fundraising'),
 (0.05292988093578521, 'unemployement_rate'),
 (0.05007418053233519, 'first_time_elected'),
 (0.03975649490654364, 'year'),
 (0.031002666065544072, 'last_D_house_seats'),
 (0.030495240178302283, 'last_R_house_seats'),
 (0.023551925361171365, 'count_victories'),
 (0.022220248454241392, 'president_overall_avg_job_approval'),
 (0.015216390095867081, 'state_California'),
 (0.014624051606127605, 'is_incumbent'),
 (0.01279188801836484, 'won'),
 (0.01090425880680488, 'district_District 1'),
 (0.010734454772533122, 'is_presidential_year'),
 (0.010715193184569984, 'district_District 2'),
 (0.008940875050554105, 'district_District 4'),
 (0.008605364653506718, 'state_Texas'),
 (0.00857605478759626, 'district_District 3'),
 (0.00752941926693872, 'state_New York'),
 (0.007336515171323253, 'district_District 6'),
 (0.007324742482375194, 'president_can_be_re_elected'),
 (0.007320299601452976, 'district_District 5'),
 (0.007004935550100393, 'president_party_D'),
 (0.006794508349814401, 'president_party_R'),
 (0.006771100403183179, 'state_Massachusetts'),
 (0.006451974070560702, 'state_Maryland'),
 (0.0061887415553304615, 'state_Connecticut'),
 (0.0061183167065578595, 'state_Florida'),
 (0.00603270974876755, 'district_District 8'),
 (0.00599316499908822, 'district_District 7'),
 (0.005988859960770931, 'district_District 9'),
 (0.005803693934539453, 'state_Colorado'),
```

```
 (0.005298202173353087, 'state_Oregon'),
 (0.0050957713777973835, 'last_house_majority_D')]
```

3 Baseline Model

```
In [32]: #further modeling for baseline with mean imputed data
         house_df = house_df_mean.copy()
```

```
In [33]: print(len(house_df))
```

```
9974
```

```
In [34]: #check that we always have one (and only one) winner per district
         house_df_grouped=house_df.groupby(['year', 'state', 'district'])['won'].sum().reset_i
         house_df_grouped[house_df_grouped['won']!=1]
```

```
Out[34]: Empty DataFrame
         Columns: [year, state, district, won]
         Index: []
```

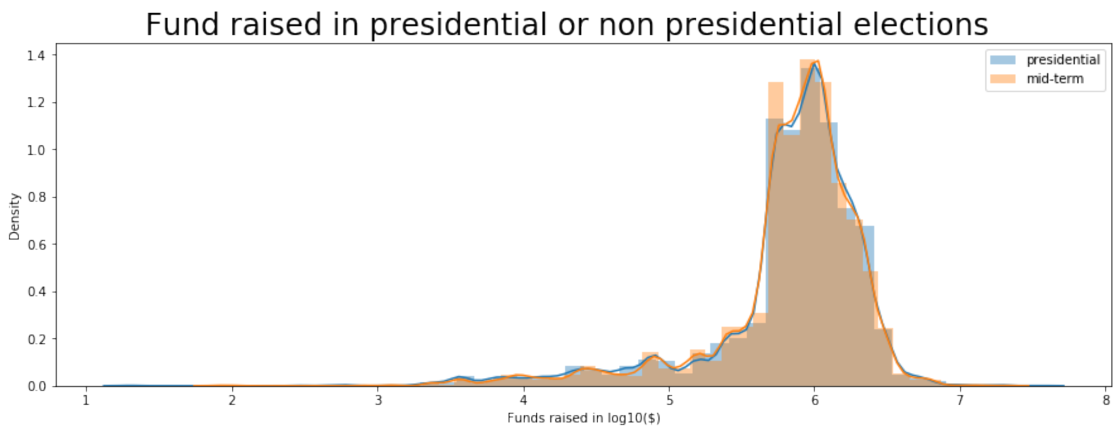
```
In [35]: #show that we have to remove first_time_elected if it's in the future, compared to cu
         house_df[(house_df['year']-house_df['first_time_elected']<=0)&(house_df['name']=='John
```

```
Out[35]:       district  is_incumbent      name party  percent    state    votes  won  \
         3   District 1           0.0  John Law     D     49.1  Indiana  10868.0    0
         21  District 1           0.0  John Law     D     55.7  Indiana  13476.0    1

             year  first_time_elected  count_victories  unemployement_rate  \
         3   1830              1860.0                0            5.790196
         21  1860              1860.0                0            5.790196

             is_presidential_year  president_can_be_re_elected president_party  \
         3                    0.0                          1.0               0
         21                   1.0                          1.0               R

             president_overall_avg_job_approval  last_D_house_seats  \
         3                             0.525667          200.179856
         21                            0.525667           98.000000

             last_R_house_seats last_house_majority  fundraising
         3           182.503597                   D  552917.8375
         21          116.000000                   R  552917.8375
```

```
In [36]: #fundraising
         def fundraisingVsPresidentialYear(df):
             df_plt=df.dropna(subset=['fundraising', 'is_presidential_year']).copy()
             #df_plt.loc[df_plt['fundraising']<=0, 'fundraising']=1 #remove zero values
```

```
    df_plt=df_plt[df_plt['fundraising']>0]
    df_plt['fundraising']=np.log10(df_plt['fundraising']) #take the log10
    fig, ax = plt.subplots(1, 1, figsize=(15, 5))
    fig.suptitle('Fund raised in presidential or non presidential elections', fontsiz
    #print(i, year)
    sns.distplot(df_plt[df_plt['is_presidential_year']==1]['fundraising'], ax=ax, labe
    sns.distplot(df_plt[df_plt['is_presidential_year']==0]['fundraising'], ax=ax, labe
    #set x label
    ax.set_xlabel('Funds raised in log10($)')
    #set y label
    ax.set_ylabel('Density')
    #set title
    #ax[i].set_title('year {}'.format(year))
    #set legend
    ax.legend()
fundraisingVsPresidentialYear(house_df)
```

```
C:\Users\IBM_ADMIN\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



In [37]:
```
house_df_district_count=house_df.loc[house_df['year']==2017]
house_df_district_count.groupby(['state', 'district'])['name'].first()

house_df[(house_df['state']=='California')&(house_df['district']=='District 34')&(hous
```

Out[37]:
```
           district  is_incumbent            name party  percent       state  \
9122  District 34           0.0  Robert Lee Ahn     D     40.8  California
9126  District 34           0.0     Jimmy Gomez     D     59.2  California

        votes  won  year  first_time_elected  count_victories  \
9122  17610.0    0  2017                 0.0                0
9126  25569.0    1  2017              2017.0                0
```

```
      unemployement_rate  is_presidential_year  president_can_be_re_elected  \
9122                 6.8                   0.0                          1.0
9126                 6.8                   0.0                          1.0

     president_party  president_overall_avg_job_approval  last_D_house_seats  \
9122               0                            0.515259          241.017241
9126               0                            0.515259          241.017241

      last_R_house_seats last_house_majority  fundraising
9122          193.724138                   D   1658443.92
9126          193.724138                   D   1379556.75
```

In [38]:
```
#count how many observations we have for each district.
house_df_grouped=house_df[house_df['year']!=2018].groupby(['state', 'district'])['par
house_df_grouped.reset_index(drop=False).head()
```

Out[38]:
```
     state    district  party
0  Alabama  District 1     12
1  Alabama  District 2     16
2  Alabama  District 3     16
3  Alabama  District 4     12
4  Alabama  District 5     14
```

In [39]:
```
house_df2=house_df.copy()
house_df2['R_vs_D_Seats']=house_df2['last_R_house_seats']/(house_df2['last_R_house_sea
house_df2['WinLoseParty']=house_df2['party'].astype(str)+house_df2['won'].replace([0,
house_df2['won']=house_df2['won'].replace([0, 1], ['Loser', 'Winner'])
house_df2['LogFundraising']=house_df2['fundraising'].copy()
house_df2.loc[house_df2['LogFundraising']<=0, 'LogFundraising']=np.NaN
house_df2['LogFundraising']=np.log10(house_df2['LogFundraising']) #take the log10
#df['Year'].astype(str) + df['quarter']
house_df2.head()
```

Out[39]:
```
     district  is_incumbent          name party  percent    state    votes  \
0  District 1           0.0  Ratliff Boon     D     42.1  Indiana   4281.0
1  District 1           1.0  Ratliff Boon     D     42.8  Indiana   5202.0
2  District 1           1.0  Ratliff Boon     D     52.2  Indiana   7272.0
3  District 1           0.0      John Law     D     49.1  Indiana  10868.0
4  District 1           1.0  Ratliff Boon     D     50.9  Indiana  11280.0

      won  year  first_time_elected      ...        \
0  Winner  1824              1824.0      ...
1  Winner  1826              1824.0      ...
2  Winner  1828              1824.0      ...
3   Loser  1830              1860.0      ...
4  Winner  1830              1824.0      ...

   president_can_be_re_elected  president_party  \
```

```
0                             1.0                 0
1                             1.0                 0
2                             1.0                 0
3                             1.0                 0
4                             1.0                 0

   president_overall_avg_job_approval  last_D_house_seats last_R_house_seats  \
0                            0.525667          200.179856         182.503597
1                            0.525667          200.179856         182.503597
2                            0.525667          200.179856         182.503597
3                            0.525667          200.179856         182.503597
4                            0.525667          200.179856         182.503597

   last_house_majority  fundraising  R_vs_D_Seats WinLoseParty  LogFundraising
0                    D  552917.8375      0.476905      DWinner        5.742661
1                    D  552917.8375      0.476905      DWinner        5.742661
2                    D  552917.8375      0.476905      DWinner        5.742661
3                    D  552917.8375      0.476905       DLoser        5.742661
4                    D  552917.8375      0.476905      DWinner        5.742661

[5 rows x 23 columns]
```

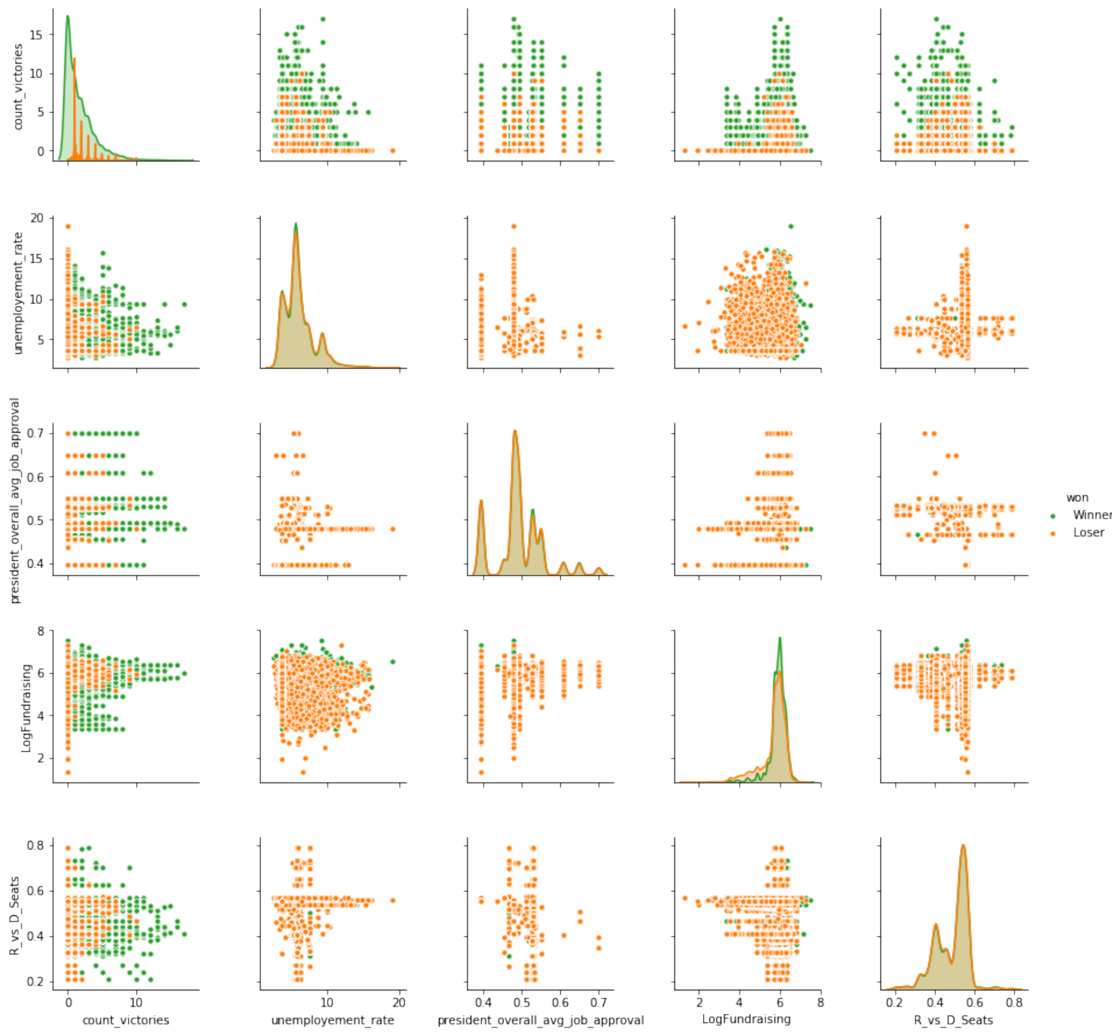
In [40]:
```
#palettes for parties or other
Parties_palette=[(0.12156862745098039, 0.4666666666666667, 0.7058823529411765),
             (0.8392156862745098, 0.15294117647058825, 0.1568627450980392),
             (0.17254901960784313, 0.6274509803921569, 0.17254901960784313),
             (1.0, 0.4980392156862745, 0.054901960784313725),
             (0.5803921568627451, 0.403921568627451, 0.7411764705882353),
             (0.5490196078431373, 0.33725490196078434, 0.29411764705882354),
             (0.8901960784313725, 0.4666666666666667, 0.7607843137254902),
             (0.4980392156862745, 0.4980392156862745, 0.4980392156862745),
             (0.7372549019607844, 0.7411764705882353, 0.13333333333333333),
             (0.09019607843137255, 0.7450980392156863, 0.8117647058823529)]
WinLosePalette=[(0.17254901960784313, 0.6274509803921569, 0.17254901960784313),
             (1.0, 0.4980392156862745, 0.054901960784313725),
             (0.5803921568627451, 0.403921568627451, 0.7411764705882353),
             (0.5490196078431373, 0.33725490196078434, 0.29411764705882354),
             (0.8901960784313725, 0.4666666666666667, 0.7607843137254902),
             (0.4980392156862745, 0.4980392156862745, 0.4980392156862745),
             (0.7372549019607844, 0.7411764705882353, 0.13333333333333333),
             (0.09019607843137255, 0.7450980392156863, 0.8117647058823529)]
```

In [41]:
```
sns.pairplot(house_df2[[
 'party',
 'count_victories',
 'unemployement_rate',
 'president_party',
 'president_overall_avg_job_approval',
```

```
        'last_house_majority',
        'LogFundraising',
        #'WinLoseParty',
        #'wonParty',
        'R_vs_D_Seats',
        'won']], hue="won",  palette=WinLosePalette, plot_kws=dict(s=25))
```

```
C:\Users\IBM_ADMIN\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
C:\Users\IBM_ADMIN\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:448: RuntimeWar
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
C:\Users\IBM_ADMIN\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:448: RuntimeWar
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
```

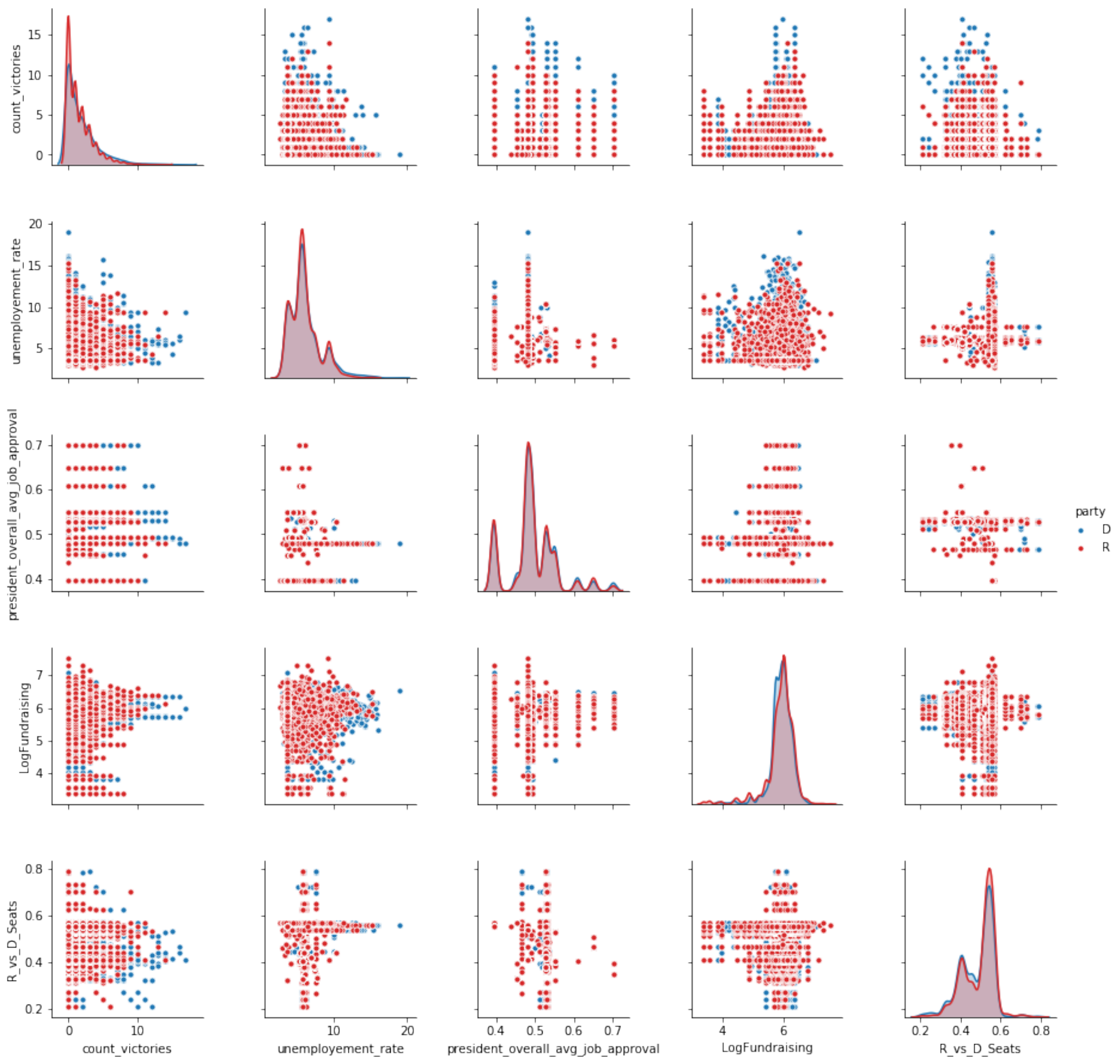
```
Out[41]: <seaborn.axisgrid.PairGrid at 0xcd2af98>
```



In [42]:
```
sns.pairplot(house_df2[house_df2['won']=='Winner'][[
        'party',
        'count_victories',
        'unemployement_rate',
        'president_party',
        'president_overall_avg_job_approval',
        'last_house_majority',
        'LogFundraising',
        #'WinLoseParty',
        #'wonParty',
        'R_vs_D_Seats',
        'won']], hue="party",  palette=Parties_palette, plot_kws=dict(s=25))
```

```
C:\Users\IBM_ADMIN\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
C:\Users\IBM_ADMIN\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:448: RuntimeWa
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
C:\Users\IBM_ADMIN\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:448: RuntimeWa
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
```

Out[42]:
```
<seaborn.axisgrid.PairGrid at 0xf1dcbe0>
```




```
In [ ]: house_df2=house_df.dropna().copy()
        house_df2_districts=house_df2[['state','district']]
        house_df2=house_df2.drop('state', axis=1).drop('district', axis=1).drop('name', axis=1)
        house_df2['party']=house_df2['party'].replace(['D', 'R'], [0, 1])
        house_df2['president_party']=house_df2['president_party'].replace(['D', 'R'], [0, 1])
        house_df2['last_house_majority']=house_df2['last_house_majority'].replace(['D', 'R'],

        data_train, data_test=house_df2[house_df2['year']!=2018], house_df2[house_df2['year']=

        x_train, y_train=data_train.drop('won', axis=1), data_train['won']

        x_test, y_test=data_test.drop('won', axis=1), data_test['won']
        baselineLogRegr=LogisticRegressionCV(cv=5, penalty='l2').fit(x_train, y_train)
```

```
In [44]: #Accuracy is defined as (TP+TN)/n
```

```
def printAccuracy(y_train, y_pred_train, y_test, y_pred_test):
    print('Training Set Accuracy: \t{:.2%}'.format(np.sum(y_train == y_pred_train) / l
    print('Test Set Accuracy: \t{:.2%}'.format(np.sum(y_test == y_pred_test) / len(y_t

y_pred_train=baselineLogRegr.predict(x_train)
y_pred_test=baselineLogRegr.predict(x_test)
printAccuracy(y_train, y_pred_train, y_test, y_pred_test)
print('Amount of districts in the predictions: {:.1%} of the total'.format(len(x_test
```

```
Training Set Accuracy:          97.38%
Test Set Accuracy:          97.02%
Amount of districts in the predictions: 100.0% of the total
```

In [45]:
```
#Baseline model
def winnerFilter(df):
    return df[df['won']==1][['state', 'district','party']]

def baselineTrain(df):
    df_grouped=df[df['won']==1 ].groupby(['state', 'district', 'party'])['won'].count
    df_grouped=df_grouped.groupby(['state', 'district']).agg({'won':'max',
                                                    'party': 'first'})
    return df_grouped.drop('won', axis=1).reset_index(drop=False)
```

In [46]:
```
y_pred=baselineTrain(house_df[house_df['year']!=2018]) #train simple average model, r
y=winnerFilter(house_df[house_df['year']==2018]) #extract winner party for each distr

results=[]
for state in y['state'].unique():
    for district in y[y['state']==state]['district']:
        actual=y.loc[(y['state']==state)&(y['district']==district), 'party']
        pred=y_pred.loc[(y_pred['state']==state)&(y_pred['district']==district), 'par
        #print('pred:{}, \nactual:{}, \npred.all():{}, \nactual.all():{}\n result:{}\
        results.append(actual.all()==pred.all())
print('Test Set Accuracy: \t{:.2%}'.format(sum(results)/len(results)))
```

```
Test Set Accuracy:          77.93%
```

In [47]:
```
def deductPartisanship(trainData, HistYears=50):
    #compute the prevalence of one party win against the other
    house_df_all_districts=trainData[(trainData['won']==1) & (trainData['year']>=(2018
    house_df_all_districts['R_occurence']=house_df_all_districts['party'].str.count('R

    avgHistData=house_df_all_districts['party'].str.len().mean() #Average amount of h
    histDataThreshold=avgHistData/2

    print('In average, in the last {} years, we have data from the last {:.1f} electi
```

```
    #3=traditionally Republican district
    #2=traditionally Democratic district
    #1=swing district
    #0=Recent district (Not enough historical data)
    house_df_all_districts['partisanship']=(house_df_all_districts['party'].str.len()>
                          (house_df_all_districts['R_occurence']>(2/3))*3
                        + (house_df_all_districts['R_occurence']<=(1/3))*2
                        + ((house_df_all_districts['R_occurence']>(1/3))
                          &(house_df_all_districts['R_occurence']<=(2/3)))*1
                        )
    return house_df_all_districts[['state', 'district', 'partisanship']]

def assignPartisanship(train_df, test_df):
    return test_df.join(deductPartisanship(train_df).set_index(['state', 'district'])
```

In [48]:
```
def preprocess(train_df, df):
    out_df=assignPartisanship(train_df, df).copy()
    out_df['first_time_elected']=out_df['year']-out_df['first_time_elected']
    out_df.loc[out_df['first_time_elected']<0, 'first_time_elected']=np.NaN
    out_df['Log10fundraising']=out_df['fundraising']
    out_df.loc[out_df['Log10fundraising']<=0, 'Log10fundraising']=np.NaN
    out_df['Log10fundraising']=np.log10(out_df['fundraising']) #take the log10
    return out_df[['is_incumbent',
                  'party',
                  'first_time_elected',
                  'count_victories',
                  'unemployement_rate',
                  'is_presidential_year',
                  'president_can_be_re_elected',
                  'president_party',
                  'president_overall_avg_job_approval',
                  'last_D_house_seats',
                  'last_R_house_seats',
                  'last_house_majority',
                  'fundraising',
                  'won'
                 ]]

msk=house_df['year']!=2018
data_train=preprocess(house_df[msk], house_df[msk])
data_test=preprocess(house_df[msk], house_df[~msk])
```

```
In average, in the last 50 years, we have data from the last 9.0 elections in each district.
Some districts are "new" as they exist only after a redistribution for a new congress.
We evaluate the partisanships of districts which exist at least since the last 4.5 elections
In average, in the last 50 years, we have data from the last 9.0 elections in each district.
Some districts are "new" as they exist only after a redistribution for a new congress.
We evaluate the partisanships of districts which exist at least since the last 4.5 elections
```

```
C:\Users\IBM_ADMIN\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: divide
  import sys
C:\Users\IBM_ADMIN\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: invalic
  import sys
```

In [49]:
```
# table with all correlations for Republicans win
drop = ['won','votes', 'percent', 'year', 'first_time_elected', 'is_presidential_year
corr_df = house_df2.copy()
corr_df = corr_df.drop(drop, axis=1)
corr_df[corr_df['party'] == 1].drop('party', axis=1).corr().style.format("{:.2}").bacl
```

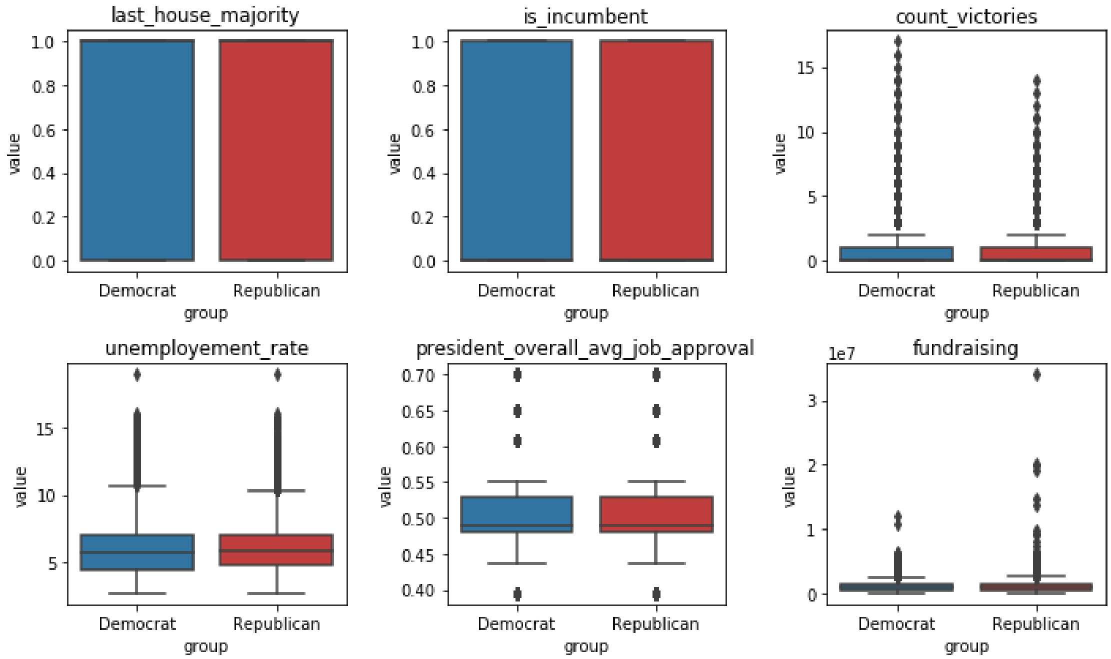
Out[49]:
```
<pandas.io.formats.style.Styler at 0x19a75940>
```

In [50]:
```
# table with all correlations for Democrats win
corr_df[corr_df['party'] == 0].drop('party',axis=1).corr().style.format("{:.2}").backg
```

Out[50]:
```
<pandas.io.formats.style.Styler at 0x19e58d68>
```

In [55]:
```
var_all = ['last_house_majority','is_incumbent', 'count_victories','unemployement_rate

# comparison of variables with boxplots
def expl_boxplots(dataframe,variables):
    house_df2_D = dataframe[dataframe['party'] == 0]
    house_df2_R = dataframe[dataframe['party'] == 1]
    fig = plt.figure(figsize=(10,6))
    for i in range(len(var_all)):
        plt.subplot(2,3,i+1)
        a = pd.DataFrame({ 'group' : np.repeat('Democrat',house_df2_D.shape[0]), 'valu
        b = pd.DataFrame({ 'group' : np.repeat('Republican',house_df2_R.shape[0]), 'va
        plt.title(var_all[i])
        df=a.append(b)
        # boxplot with colors
        my_pal = {DEM_blue,REP_red}
        sns.boxplot(x='group', y='value', data=df,palette=my_pal)
    fig

    plt.tight_layout()
```

In [56]:
```
expl_boxplots(house_df2,var_all)
```


last_house_majority
is_incumbent
count_victories
unemployement_rate
president_overall_avg_job_approval
fundraising
value
group
Democrat
Republican
1e7

In [ ]: