Introduction

The event in which a private company becomes a public company is called an Initial Public Offering (IPO). At the moment of becoming a public company the private company makes available all or partial shares of its company to the general public in a stock exchange of its choosing. The general public, an individual or entity is able to purchase company shares at particular price, which allows the private company to raise additional capital and/or for previous investors to recover back their investment, hopefully at a gain.

The process to become a public company is very rigorous, such that the private company must disclose vast amounts of relevant information about its business and hire external entities, such as investment banks where (an) underwriter(s), help(s) to evaluate and recommend the number of shares and the price at which such shares should be initially offered. Using the information disclosed by the company, together with other primary and secondary information gathered by the underwriter(s), the underwriting company prepares and files a document to the government listing the opportunities, risks and financial detials about the company to be made available to any interested party including investors and regulators.

Prior to these shares being offered to the general public, these shares will first be sold to institutional investors, typically at a discount to the recommended initial offer price, to compensate these investors for their risk prior the market actually 'agreeing' upon what these shares are worth. Thus, at the end of the first trading day, the price of these shares may go higher or lower than the initial offer price. If the price goes up by a significant amount, then the private company going public will have foregone the opportunity to earn more money from the institutional investors who took the risk to buy first, while in the opposite case, institutional investors will be disatsified and the underwriting company's reputation may be affected. Additionally, this could trigger litigation against the issuing company or the underwriting company in the case that specific information was not disclosed when it should have.

For this reason, it is of utmost importance, but also to a great financial advtange (by selling or using these predictions) to be able to make correct predictions about the direction and magnitude of the price of a stock at the end of its first day of trading with respect to the initial offer price. Not surprisingly, our data includes these two important variables, the "offer price" and the "close price". Additionally, we are provided with supplemental information compiled across multiple sources of information, in regards to each company's industry, past investors, and financials, but also about its underwriters and IPO Prospectus.

Initial Set up

Import libraries

We begin this project by importing useful libraries, loading the data set and storing the data sets' layout.

In [2]:
```
# Import useful libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statistics as sts
from scipy import stats
%matplotlib inline
pd.set_option('display.max_columns', 100)

# scikit-learn
from sklearn.dummy           import DummyClassifier
from sklearn.linear_model    import LinearRegression, LogisticRegression, Ridge,
from sklearn.svm             import SVC
from sklearn.neighbors       import KNeighborsClassifier
from sklearn.tree            import DecisionTreeClassifier
from sklearn.ensemble        import RandomForestClassifier
from sklearn.ensemble        import GradientBoostingClassifier
from sklearn.preprocessing import OneHotEncoder

# Supporting functions from scikit-learn
from sklearn.metrics         import confusion_matrix
from sklearn.metrics         import roc_curve
from sklearn.metrics         import roc_auc_score
from sklearn.metrics         import f1_score
from sklearn.metrics         import recall_score
from sklearn.metrics         import accuracy_score
from sklearn.metrics         import mean_squared_error
from sklearn.pipeline        import Pipeline
from sklearn.preprocessing   import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_val_score
from sklearn.tree            import export_graphviz
from sklearn.decomposition   import PCA

# Visualize missing values
import missingno as msno

# ignore some warnings
import warnings
warnings.filterwarnings('ignore')

# Tunning plots
import matplotlib.lines as mlines
from matplotlib.ticker import MaxNLocator

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# Special Plotting
import pyLDAvis
import pyLDAvis.gensim  # don't skip this

from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.decomposition import TruncatedSVD

# Read/Write xlsx files
from openpyxl import workbook
from openpyxl import load_workbook

import os
```

```
/home/irene/Anaconda3/lib/python3.7/site-packages/openpyxl/xml/functions.py:30:
DeprecationWarning: defusedxml.lxml is no longer supported and will be removed i
n a future release.
  from defusedxml.lxml import fromstring as _fromstring, tostring
```

Useful functions

In [7]:

```
# Splitting data into training and validation set
# Seed for replication
SEED = 1
def split_data(vectors, target):
    """Splits the vectors matrix into train and validate matrices.

    Arguments: vectors (matrix)
               target (numpy array )
    """
    # Splitting the matrices into train and validate matrices
    x_train, x_test, y_train, y_test = train_test_split(vectors, target, random_
    return x_train, x_test, y_train, y_test

def plot_roc(fpr, tpr, title='ROC Curve', note=''):
    """
    Function to plot an ROC curve in a consistent way.

    Args:
        fpr        False Positive Rate (list of multiple points)
        tpr        True Positive Rate (list of multiple points)
        title      Title above the plot
        note       Note to display in the bottom-right of the plot
    """
    plt.figure(1)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr, tpr, color = 'b')
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title(title)
    if note: plt.text(0.6, 0.2, note)

# Train and test a model
def train_and_test_model(model, X_train,y_train,X_test, y_test, opt_alpha):
    """ Function to return the final ridge model and to print the MSE scores usi

        Args:
        X_train pandas data frame containing the train data set
        y_train numpy array containing the train target variable
        X_test  pandas data frame containing the train data set
        y_test numpy array containing the test target variable
        opt_alpha float number for the optimum alpha to run the model
    """
    # Train the model with optimum parameter
```

```
    model_ = model
    model_.fit(X_train,y_train)

    # Make predictions
    y_train_pred = model_.predict_proba(X_train)
    # Compute MSE of the predictions
    AUC = roc_auc_score(y_train, y_train_pred[:,1])
    print("The AUC score for the model with parameter alpha {0:.3f} is: {1:.3f}

    # Make predictions using the test data set
    y_test_pred = model_.predict_proba(X_test)
    # Compute MSE of the predictions using the test data set
    AUC = roc_auc_score(y_test, y_test_pred[:,1])
    print("The AUC score for the model using the test data set with parameter al

    # Compute other metrics
    # F1
    F1 = f1_score(y_test, model_.predict(X_test))

    # Recall
    accuracy = accuracy_score(y_test, model_.predict(X_test))

    # Compute fpr and tpr
    fpr, tpr, _ = roc_curve(y_test, y_test_pred[:,1])

    # Plot ROC curve
    plot_roc(fpr, tpr)

    # Display AUC score
    label = mlines.Line2D([], [], color='b', marker='',
                          markersize=15, label='AUC score {}'.format(round(AUC,4
    plt.legend(handles=[label])
    return AUC, F1, accuracy

def plot_AUC_train_test(AUC_list_train, AUC_list_test, choose_index, value_param
    plt.plot(AUC_list_train, color = 'r')
    plt.plot(AUC_list_test, 'b')
    plt.axvline(x = choose_index, color = 'g')
    plt.title('Bias & Variance Trade-off')
    plt.xlabel('Index parameter {0}'.format(symbol_parameter))
    plt.ylabel('AUC Score')

    train = mlines.Line2D([], [], color='r', marker='',
                          markersize=15, label='train AUC')
    test = mlines.Line2D([], [], color='b', marker='',
                          markersize=15, label='test AUC')
    opt_value = mlines.Line2D([], [], color='g', marker='',
                          markersize=15, label='Optimum value: {}'.format(value_
    plt.legend(handles=[train, test, opt_value])
    plt.show()

def print_model_scores(model_scores, model_names):
    # Print number of missing values per feature and its corresponding percentag
    dash = '-' * 65
    print(dash)
    print('{:<15s}{:<15s}{:<15s}{:<15s}'.format('Model name','AUC score','F1 sco
    print(dash)
    for i in range(len(model_scores)):
        print('{:<15s}{:<15.3f}{:<15.3f}{:<15.3f}'.format(model_names[i], model_
    print(dash)
```

```
# Display the target variables' distribution for any prediction
def display_target_distribution(data):
    count = data.reset_index()[['target', 'index']].groupby('target').count().re
    count.columns = ['target', 'number']
    count['percentage'] = round(100*count['number']/data.shape[0],3)
    return count

def build_scores(y_true, prediction, threshold):
    y_test_pred_df = pd.DataFrame(prediction, columns = ['y_proba'])
    y_test_pred_df['y_binary'] = y_test_pred_df['y_proba'].apply(lambda x: 1.0 i
    cm = confusion_matrix(y_true, y_test_pred_df['y_binary'])
    tn, fp, fn, tp = cm.ravel()
    accuracy = (tn + tp)/len(prediction)
    return [threshold, accuracy]

def choose_thresholds(y_true, prediction, step_size):
    lista = []
    for i in np.arange(0, 1, step_size):
        lista.append(build_scores(y_true, prediction, i))
    thresholds_list = max(lista, key=lambda x: x[1])
    threshold = thresholds_list[0]
    accuracy = thresholds_list[1]
    print("The threshold that maximizes the accuracy index is: ", threshold,
         "and the accuracy score is: ", accuracy)
    return threshold, accuracy, lista

def plot_across_thresholds(thresholds_list, winner_threshold):
    # Plot accuracy across thresholds
    ax = sns.lineplot(*zip(*thresholds_list))

    # Tunning plot
    ax.set(xlabel = 'Probability thresholds', ylabel = 'Accuracy score', title =
    plt.axvline(winner_threshold, color = 'red')

def plot_score_test(score_list_test, choose_index, value_parameter, symbol_param
    plt.plot(score_list_test, 'b')
    plt.axvline(x = choose_index, color = 'g')
    plt.title('Optimum parameter {} selection using given score metric'.format(s
    plt.xlabel('Index parameter {0}'.format(symbol_parameter))
    plt.ylabel('Sum score metric - test')

    opt_value = mlines.Line2D([], [], color='g', marker='',
                              markersize=15, label='Optimum value: {0:.4f}'.format(v
    plt.legend(handles=[opt_value])
    plt.show()

def plot_mse_train_test(mse_list, choose_index, value_parameter, symbol_paramete
    plt.plot(mse_list, 'b')
    plt.axvline(x = choose_index, color = 'g')
    plt.title('Optimum parameter {} selection using MSE'.format(symbol_parameter
    plt.xlabel('Index parameter {0}'.format(symbol_parameter))
    plt.ylabel('MSE')

    opt_value = mlines.Line2D([], [], color='g', marker='',
                              markersize=15, label='Optimum value: {}'.format(va
    plt.legend(handles=[opt_value])
    plt.show()

def export_to_excel(pred_num, pred_array):
```

```
# this function exports our predictions into the data_to_predict excel file
    LOCAL_PATH = "data/"
    IPO_DATA_P = os.path.join(LOCAL_PATH, 'IPO_data_to_predict.xlsx')
    wb = load_workbook(IPO_DATA_P)
    sheets = wb.sheetnames
    S1 = wb[sheets[0]]
    xlrow = 2
    col_num = (46+pred_num)
    #add the predicted value into the column of the corresponding row of IPO com
    for num in pred_array:
        S1.cell(row = xlrow, column = col_num).value = pred_array[num] #eg. This
        xlrow += 1
    wb.save(IPO_DATA_P)

def print_model_scores_mse(model_scores, model_names):
    # Print number of missing values per feature and its corresponding percentag
    dash = '-' * 50
    print(dash)
    print('{:<15s}{:<15s}{:<15s}{:<15s}'.format('Model name','MSE', 'RMSE', 'R^2
    print(dash)
    for i in range(len(model_scores)):
        print('{:<15s}{:<15.3f}{:<15.3f}{:<15.3f}'.format(model_names[i], model_
    print(dash)

def MSE_train_test(X_train, X_test, y_train, y_test, model, opt_parameter):
    y_train_pred = model.predict(X_train)       # use validation set during hype
    MSE_train = mean_squared_error(y_train, y_train_pred)
    y_test_pred = model.predict(X_test)       # use validation set during hyper-
    MSE_test = mean_squared_error(y_test, y_test_pred)
    R_2 = model.score(X_test, y_test)
    RMSE_test = np.sqrt(MSE_test)
    print("The MSE for the model: {0:.4f}\n".format(MSE_train))
    print("The MSE for the model using the test data set: {0:.4f}\n".format(MSE_
    print("The square root of the MSE using the test data set: {0:.4f}\n".format
    print("The R^2 for the model using the test data set: {0:.4f}".format(R_2))
    return [MSE_test, RMSE_test, R_2]

def plot_score_metric_train_test(SM_list_train, SM_list_test, choose_index, valu
    plt.plot(SM_list_train, color = 'r')
    plt.plot(SM_list_test, 'b')
    plt.axvline(x = choose_index, color = 'g')
    plt.title('Bias & Variance Trade-off')
    plt.xlabel('Index parameter {0}'.format(symbol_parameter))
    plt.ylabel('Score Metric')

    train = mlines.Line2D([], [], color='r', marker='',
                          markersize=15, label='train Score Metric')
    test = mlines.Line2D([], [], color='b', marker='',
                          markersize=15, label='test Score MEtric')
    opt_value = mlines.Line2D([], [], color='g', marker='',
                          markersize=15, label='Optimum value: {}'.format(va
    plt.legend(handles=[train, test, opt_value])
    plt.show()

def print_cv_scores(mean_scores, std_scores, model_names):
    # Print number of missing values per feature and its corresponding percentag
    dash = '-' * 40
    print(dash)
    print('{:<15s}{:<15s}{:<15s}'.format('Model name','Mean AUC', 'Std AUC'))
    print(dash)
```

```
    for i in range(len(mean_scores)):
        print('{:<15s}{:<15.3f}{:<15.3f}'.format(model_names[i], mean_scores[i],
    print(dash)
```

Exploratory Analysis

Data sets' characteristics

The data set IPO_df contains 3,330 observations and 46 features. It has no duplicate rows.

In [8]:
```
# Define a function to print the number of observations and features
def print_shape(data):
    """ Function to print the the number of features and observations in a data

        Args:
        data pandas data frame containing the input variables
    """
    print("Observations: {} \nFeatures: {}".format(data.shape[0], data.shape[1])
print_shape(IPO_df)
```

```
Observations: 3330
Features: 46
```

In [9]:
```
# Check if data set have any duplicate rows
print("Duplicate rows: {}".format(np.sum(IPO_df.duplicated(subset=None, keep='fi
```

```
Duplicate rows: 0
```

The IPO_df data set contains mostly float data type features, it has in a lower proportion integer, bool and object data type features. A table with the number of features per data type can be found below.

Data types

In [10]:
```
def print_number_dtypes(data):
    """ Function to print the the number of features per data type in a data set

        Args:
        data pandas data frame containing the input variables
    """
    # Determine data types in a data frame
    data_types = pd.DataFrame(data.dtypes)\
                   .reset_index()

    # Count how many variables per data type
    data_types.columns = ['feature_name', 'data_type']
    data_types = data_types.sort_values(['feature_name'])
    data_types_count = data_types.groupby(['data_type'])\
                                 .count()\
                                 .reset_index()
    data_types_count.columns = ['data_type', 'number']

    # Print number of variables per data type
    dash = '-' * 35
```

```
    print(dash)
    print('{:<15s}{:<3s}'.format('Data type','Number of features'))
    print(dash)
    for i in range(data_types_count.shape[0]):
        print('{:<15s}{:<3f}'.format(str(data_types_count.data_type[i]), data_ty
    print(dash)
    return data_types

# Print number of features per data type in a data set using the function print_
data_types = print_number_dtypes(IPO_df)
```

```
-----------------------------------
Data type      Number of features
-----------------------------------
bool           5.000000
int64          5.000000
float64        28.000000
object         8.000000
-----------------------------------
```

For the following data exploration steps, we first subset the features' names and descriptions per data type. The previous will make the exploration phase easier as some techniques are only meant to be used with a certain type of data. For instance, to observe a features' distribution we can plot a histogram for float features. For object features we can instead plot a bar chart with the number of observations per category.

In [11]:

```
# Subset of features' names and descriptions per data type

# Subset of features' names and descriptions  - float data type
float_features = list(data_types['feature_name'][data_types['data_type'] == 'flo
float_features_description = list(IPO_layout['feature_description'][IPO_layout['

# Subset of features' names and descriptions  - int data type
int_features = list(data_types['feature_name'][data_types['data_type'] == 'int']
int_features_description = list(IPO_layout['feature_description'][IPO_layout['fe

# Subset of features' names and descriptions  - object data type
object_features = list(data_types['feature_name'][data_types['data_type'] == 'ob
object_features_description = list(IPO_layout['feature_description'][IPO_layout[

# Subset of features' names and descriptions  - boolean data type
boolean_features = list(data_types['feature_name'][data_types['data_type'] == 'b
boolean_features_description = list(IPO_layout['feature_description'][IPO_layout
```

Missing values

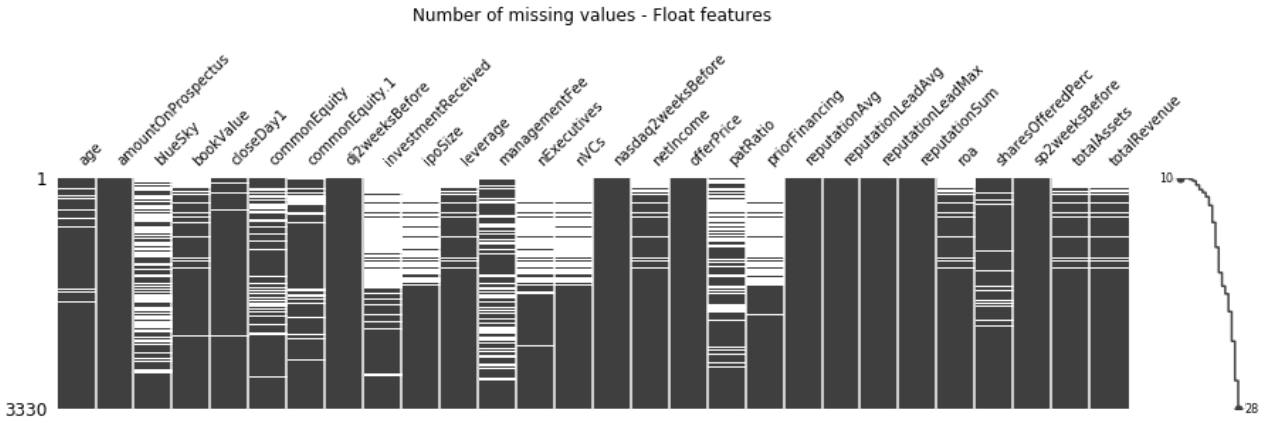
We begin the data exploration by computing the number of missing values in each feature per data type. We first plot the results, the missing values are displayed as `white` wholes. Then we give a detailed summary of the number of missing values per feature and data type.

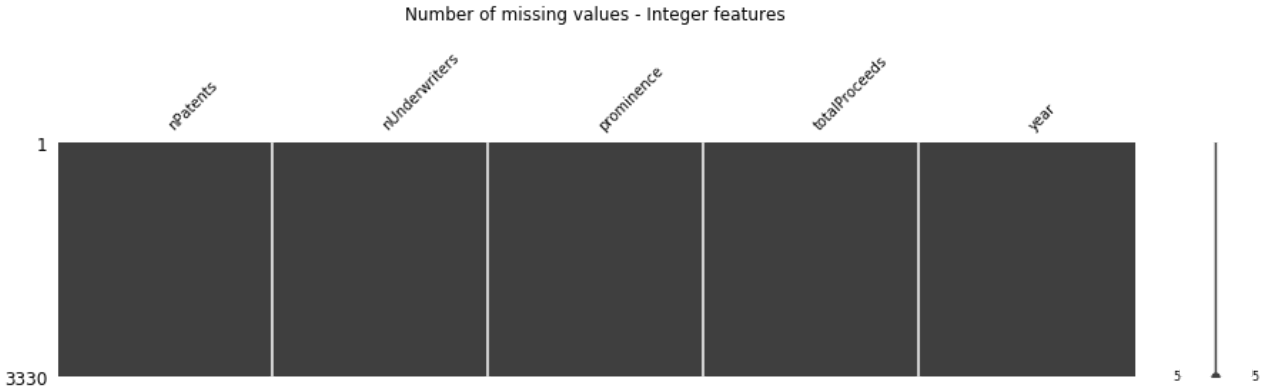
Taking a look at the following plots, we observe the following:

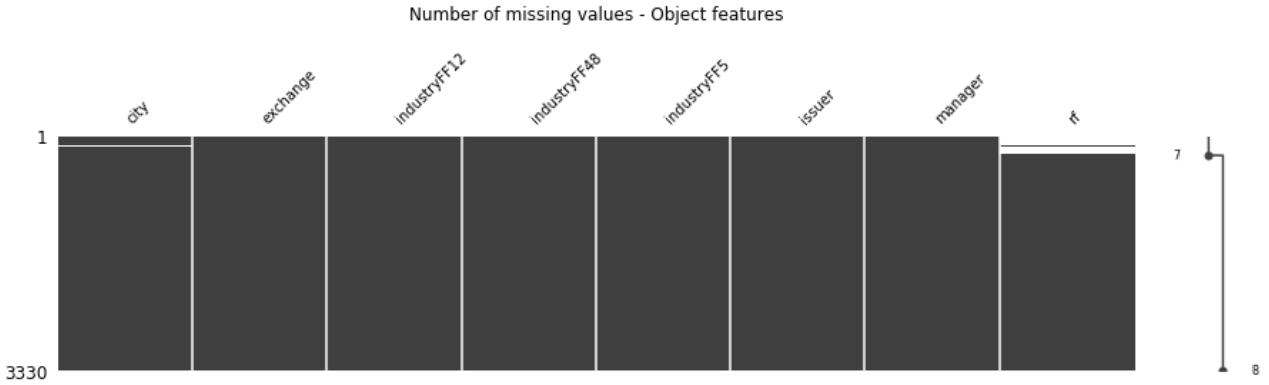
- **Float data type features**

  - The features `investmentReceived`, `ipoSize`, `nExecutives`, `nVCs`, `patRatio`, `priorFinancing` seem to have the most missing values for this type of data.
- **Object data type features**
  - Only the feature `rf` contain missing values
- **Integer and Boolean data type features**
  - No feature contain missing values

In [12]:
```
# Plot number of missing values per type of data
print_data_type = ['Float', 'Integer', 'Object', 'Boolean']
k = 0
for lista in [float_features, int_features, object_features, boolean_features]:
    sorted_data = msno.nullity_sort(IPO_df[lista], sort='ascending')
    ax0 = msno.matrix(sorted_data, figsize=(15,3), fontsize=10)
    ax0.set_title("Number of missing values - {} features\n".format(print_data_t
    k = k + 1
```







The previous plots were useful to have an idea of the missing values in each feature. In order to take decisions, we perform a detailed analysis of the features' missing values.

In [13]:

```
# Create a function to print a detailed summary about the features description,
def print_info_nan(data, data_type_features_names, data_type_features_descriptio
    """ Function to print the the number of features per data type in a data set

        Args:
        data pandas data frame containing the input variables
        data_type_features_names list containing the same data type features nam
        data_type_features_description list containing the same data type featur
    """
    # Compute number of missing values per feature and store it in a list
    n_missing_values = list(np.sum(data[data_type_features_names].isnull()))

    # Compute the percentage of missing values per feature and store it in a lis
    per_missing_values = list(np.sum(data[data_type_features_names].isnull())/da

    # Sort features' names, descriptions and percentage of missing values by the
    per_missing_values_ = [x for _,x,_,_ in sorted(zip(n_missing_values, per_mis
    data_type_features_names_ = [x for _,_,x,_ in sorted(zip(n_missing_values, p
    data_type_features_description_ = [x for _,_,_,x in sorted(zip(n_missing_val
    n_missing_values_ = sorted(n_missing_values, reverse = True)

    # Print number of missing values per feature and its corresponding percentag
    dash = '-' * 110
    print(dash)
    print('{:<20s}{:<65s}{:<15s}{:>5s}'.format('Feature','Description', 'N NaN',
    print(dash)
    for i in range(len(data_type_features_names)):
        print('{:<20s}{:<65s}{:<15f}{:<5f}%'.format(data_type_features_names_[i]
    print(dash)
```

The missing values summary for the Float data type features reveals that the features

`investmentReceived`, `nExecutives`, `priorFinancing`, `nVCs`, `blueSky`, `ipoSize` have more than 38% of missing values.

In [14]:
```
# Print NaN information about the float features in the IPO data set using the f
print_info_nan(IPO_df, float_features, float_features_description)
```

```
------------------------------------------------------------------------------
------------------------------
Feature             Description
N NaN          % NaN
------------------------------------------------------------------------------
------------------------------
investmentReceived  Total known amount invested in company ($000)
1500.000000    45.045045%
nExecutives         Count of executives
1429.000000    42.912913%
priorFinancing      Prior financing received
1417.000000    42.552553%
nVCs                Count of VC firms backing IPO firm
1329.000000    39.909910%
blueSky             Blue sky expenses
1328.000000    39.879880%
ipoSize             IPO size in USD
1316.000000    39.519520%
patRatio            Frequency of word patent in Risk Factors rel prospectus len
1275.000000    38.288288%
managementFee       Total management fee
1096.000000    32.912913%
commonEquity        Tangible Common Equity Ratio Before Offer
679.000000     20.390390%
commonEquity.1      Common Equity as % Cap
505.000000     15.165165%
bookValue           Book value
418.000000     12.552553%
roa                 Return on assets
392.000000     11.771772%
netIncome           Net income
392.000000     11.771772%
leverage            Leverage
376.000000     11.291291%
totalRevenue        Total revenue
375.000000     11.261261%
totalAssets         Total assets
357.000000     10.720721%
sharesOfferedPerc   Shares offered as % of shares outstanding after offer
262.000000     7.867868%
age                 Firm age
182.000000     5.465465%
closeDay1           Closing price at the end of the first trading day
115.000000     3.453453%
sp2weeksBefore      S&P 500 Average 2 Weeks Before Offer Date
0.000000       0.000000%
reputationSum       Sum of reputations of all underwriters
0.000000       0.000000%
reputationLeadMax   Lead underwriter reputation, max if more than one
0.000000       0.000000%
reputationLeadAvg   Lead underwriter reputation, avg if more than one
0.000000       0.000000%
reputationAvg       Avg of reputations of all underwriters
0.000000       0.000000%
offerPrice          Offer price
0.000000       0.000000%
nasdaq2weeksBefore  NASDAQ Average 2 Weeks Before Offer
```

```
0.000000       0.000000%
dj2weeksBefore      Dow Jones Indust Average 2 Weeks Before Offer
0.000000       0.000000%
amountOnProspectus  Total Amt on Prospectus (US Mil, Global)
0.000000       0.000000%
------------------------------------------------------------------------------
------------------------------
```

Both the Integer features and the Boolean features do not contain any missing values

In [15]:
```
# Print NaN information about the int features in the IPO data set using the fun
print_info_nan(IPO_df, int_features, int_features_description)
```

```
------------------------------------------------------------------------------
------------------------------
Feature             Description
N NaN          % NaN
------------------------------------------------------------------------------
------------------------------
year                Issue year
0.000000       0.000000%
totalProceeds       Total proceeds
0.000000       0.000000%
prominence          VC prominence
0.000000       0.000000%
nUnderwriters       Count of underwriters
0.000000       0.000000%
nPatents            Count of patents granted at the time of IPO
0.000000       0.000000%
------------------------------------------------------------------------------
------------------------------
```

In [16]:
```
# Print NaN information about the boolean features in the IPO data set using the
print_info_nan(IPO_df, boolean_features, boolean_features_description)
```

```
------------------------------------------------------------------------------
------------------------------
Feature             Description
N NaN          % NaN
------------------------------------------------------------------------------
------------------------------
vc                  Venture capital backing indicator
0.000000       0.000000%
pe                  Private equity backing indicator
0.000000       0.000000%
html                HTML text file indicator TXT file otherwise
0.000000       0.000000%
highTech            High tech firm indicator
0.000000       0.000000%
egc                 Emerging Growth Company indicator
0.000000       0.000000%
------------------------------------------------------------------------------
------------------------------
```

The only object variable in the data set that contains missing values is the feature `rf`, with 8.19%.

In [17]:
```
# Print NaN information about the object features in the IPO data set using the
print_info_nan(IPO_df, object_features, object_features_description)
```

```
------------------------------------------------------------------------------
```

```
------------------------------
Feature             Description
N NaN          % NaN
------------------------------------------------------------------------------
------------------------------
rf                  Risk Factors section of the IPO prospectus
273.000000     8.198198%
city                City of IPO firm address
1.000000       0.030030%
manager             Lead managers
0.000000       0.000000%
issuer              Issuer name
0.000000       0.000000%
industryFF5         Fama French (FF) 5-industries classification
0.000000       0.000000%
industryFF48        Fama French (FF) 48-industries classification
0.000000       0.000000%
industryFF12        Fama French (FF) 12-industries classification
0.000000       0.000000%
exchange            Exchange where shares will be listed on
0.000000       0.000000%
------------------------------------------------------------------------------
------------------------------
```

Basic statistics

Float features

We continue the exploratory phase by analyzing some basic statistics. We compute the mean, median, mode and standard deviation for each float data type variable. We also plot their respective histograms.

We can observe the following.

- All features take values in different ranges, so a scaling method could be helpful in the preprocessing step.
- All features have a high variance.
- Most of them are positively skewed:
    - investmentReceived
    - nExecutives
    - priorFinancing
    - nVCs
    - blueSky
    - ipoSize
    - managementFee
    - leverage
    - totalRevenue
    - age
    - closeDay1
    - reputationSum

  - dj2weeksBefore
  - amountOnProspectus

- Some variables are negatively skewed:
  - roa
  - reputationLeadMax
  - reputationLeadAvg

- Several features have visible outliers:
  - investmentReceived
  - nExecutives
  - priorFinancing
  - blueSky
  - commonEquity1
  - commonEquity2
  - roa
  - netIncome
  - totalRevenue
  - totalAssets
  - amountOnProspectus

In [18]:
```
def compute_basic_statistics_onevar(data, variable):
    """ Function that returns and prints the mean, median, mode and std of a fea

        Args:
        data   pandas DataFrame
        variable   String with the feature's name
    """
    # Compute the mean, median, mode and std
    mean = np.nanmean(data[variable])
    median = np.nanmedian(data[variable])
    mode = sts.mode(data[variable])
    std = np.std(data[variable])
    return mean, median, mode, std
```

In [19]:
```
# Plots for variables float
fig, ax = plt.subplots(7,4,figsize=(25,30))
k = 0
for i in range(7):
    for j in range(4):
        # Compute mean, median, mode and std using the function compute_basic_st
        mean, median, mode, std = compute_basic_statistics_onevar(IPO_df[np.isfi

        # Plot histograms of float variables
        ax[i,j].hist(IPO_df[float_features[k]][np.isfinite(IPO_df[float_features
        ax[i,j].set_title("Histogram - {}".format(float_features[k]))
        ax[i,j].set_yscale("log")

        # Tunning the plot
        ax[i,j].axvline(mean, color='r')
        ax[i,j].axvline(median, color='g')
```

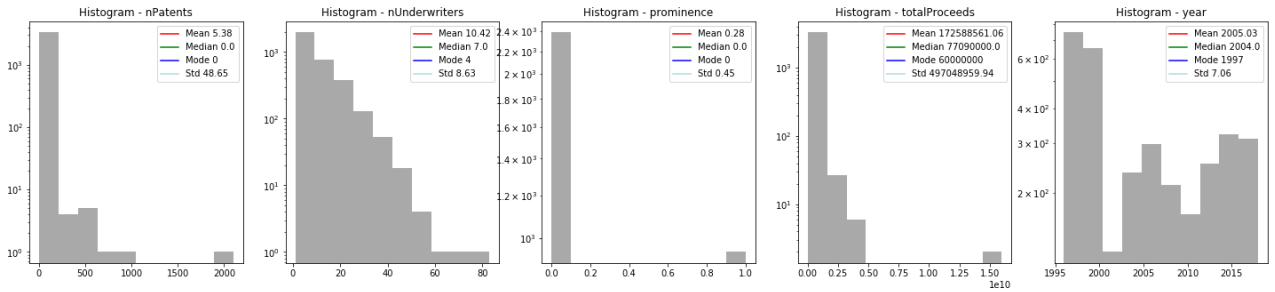
```
        ax[i,j].axvline(mode, color='b')

        # Write the mean, median and mode on the graph as labels
        label_mean = mlines.Line2D([], [], color='r', marker='',
                          markersize=15, label='Mean {}'.format(round(mean,2)))
        label_median = mlines.Line2D([], [], color='g', marker='',
                          markersize=15, label='Median {}'.format(round(median,2
        label_mode = mlines.Line2D([], [], color='b', marker='',
                          markersize=15, label='Mode {}'.format(round(mode,2)))
        label_std = mlines.Line2D([], [], color='lightblue', marker='',
                          markersize=15, label='Std {}'.format(round(std,2)))
        ax[i,j].legend(handles=[label_mean, label_median, label_mode, label_std]
        k = k + 1
```

Integer features

Now, we proceed to plot histograms for the integer variables.

ToDo: insights about integer variables

In [20]:
```
# Plots for variables int = []
fig, ax = plt.subplots(1,5,figsize=(25,5))
for k in range(5):
    # Compute mean, median, mode and skewness of the Sprint Speed feature
    mean, median, mode, std = compute_basic_statistics_onevar(IPO_df[np.isfinite

    # Make plot
    ax[k].hist(IPO_df[int_features[k]][np.isfinite(IPO_df[int_features[k]])].val
    ax[k].set_title("Histogram - {}".format(int_features[k]))
    ax[k].set_yscale("log")

    # Tunning plot
    # Write the mean, median and mode on the graph as labels
    label_mean = mlines.Line2D([], [], color='r', marker='',
                          markersize=15, label='Mean {}'.format(round(mean,2)))
    label_median = mlines.Line2D([], [], color='g', marker='',
                          markersize=15, label='Median {}'.format(round(median,2
    label_mode = mlines.Line2D([], [], color='b', marker='',
                          markersize=15, label='Mode {}'.format(round(mode,2)))
    label_std = mlines.Line2D([], [], color='lightblue', marker='',
                          markersize=15, label='Std {}'.format(round(std,2)))
    ax[k].legend(handles=[label_mean, label_median, label_mode, label_std])
```

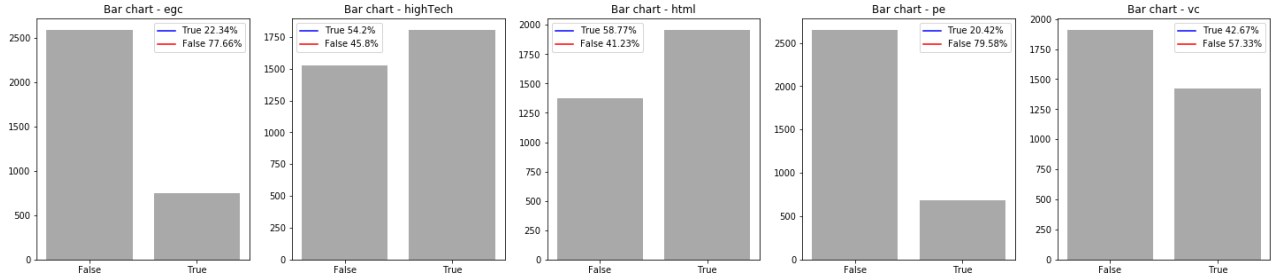


Boolean features

ToDo: insights about boolean variables

In [21]:
```
# Plots for variables Boolean = []
fig, ax = plt.subplots(1,5,figsize=(25,5))
for k in range(5):
    # Create count data frame
    count = IPO_df[[boolean_features[k], 'exchange']].groupby(boolean_features[k
    count.columns = ['feature', 'count']

    # Compute percentage true and false observations
    per_true = float(count['count'][count['feature'] == True]/IPO_df.shape[0])*1
    per_false = float(count['count'][count['feature'] == False]/IPO_df.shape[0])

    # Plot bar charts for boolean variables
    x_pos = [i for i, _ in enumerate(count['feature'])]
    ax[k].bar(x_pos, count['count'], color='darkgray')
```

```
    # Tunning the plot
    ax[k].set_title("Bar chart - {}".format(boolean_features[k]))
    ax[k].set_xticklabels([' ', 'False', 'True'], fontdict=None, minor=False)
    ax[k].xaxis.set_major_locator(MaxNLocator(integer=True))

    label_true = mlines.Line2D([], [], color='b', marker='',
                               markersize=15, label='True {}%'.format(round(per_true,
    label_false = mlines.Line2D([], [], color='r', marker='',
                               markersize=15, label='False {}%'.format(round(per_fals
    ax[k].legend(handles=[label_true, label_false])
```



In [22]:
```
# Check relationship between having a True 'pe' and/or 'vc'

pd.crosstab(IPO_df['vc'], IPO_df['pe'], margins=True, margins_name="total")
```

Out[22]:

pe	False	True	total
vc			
False	1246	663	1909
True	1404	17	1421
total	2650	680	3330

In [23]:
```
# Comparing this relationship to the number missing values under the columns 'nV
# we can see that there is a high correlation between not having both VC and PE
# in these columns

print("Out of 1167 companies without Venture Capital or Private Equity investors
print(IPO_df.loc[((IPO_df.pe == 0) & (IPO_df.vc == 0)),'nVCs'].isna().sum(),"NaN
print(IPO_df.loc[((IPO_df.pe == 0) & (IPO_df.vc == 0)),'nExecutives'].isna().sum
print(IPO_df.loc[((IPO_df.pe == 0) & (IPO_df.vc == 0)),'priorFinancing'].isna().
print(IPO_df.loc[((IPO_df.pe == 0) & (IPO_df.vc == 0)),'ipoSize'].isna().sum(),"

print("\nOut of 647 companies without Venture Capital and with Private Equity in
print(IPO_df.loc[((IPO_df.pe == 1) & (IPO_df.vc == 0)),'nVCs'].isna().sum(),"NaN
print(IPO_df.loc[((IPO_df.pe == 1) & (IPO_df.vc == 0)),'nExecutives'].isna().sum
print(IPO_df.loc[((IPO_df.pe == 1) & (IPO_df.vc == 0)),'priorFinancing'].isna().
print(IPO_df.loc[((IPO_df.pe == 1) & (IPO_df.vc == 0)),'ipoSize'].isna().sum(),"

print("\nOut of 1384 companies with Venture Capital and without Private Equity i
print(IPO_df.loc[((IPO_df.pe == 0) & (IPO_df.vc == 1)),'nVCs'].isna().sum(),"NaN
print(IPO_df.loc[((IPO_df.pe == 0) & (IPO_df.vc == 1)),'nExecutives'].isna().sum
print(IPO_df.loc[((IPO_df.pe == 0) & (IPO_df.vc == 1)),'priorFinancing'].isna().
print(IPO_df.loc[((IPO_df.pe == 0) & (IPO_df.vc == 1)),'ipoSize'].isna().sum(),"
```

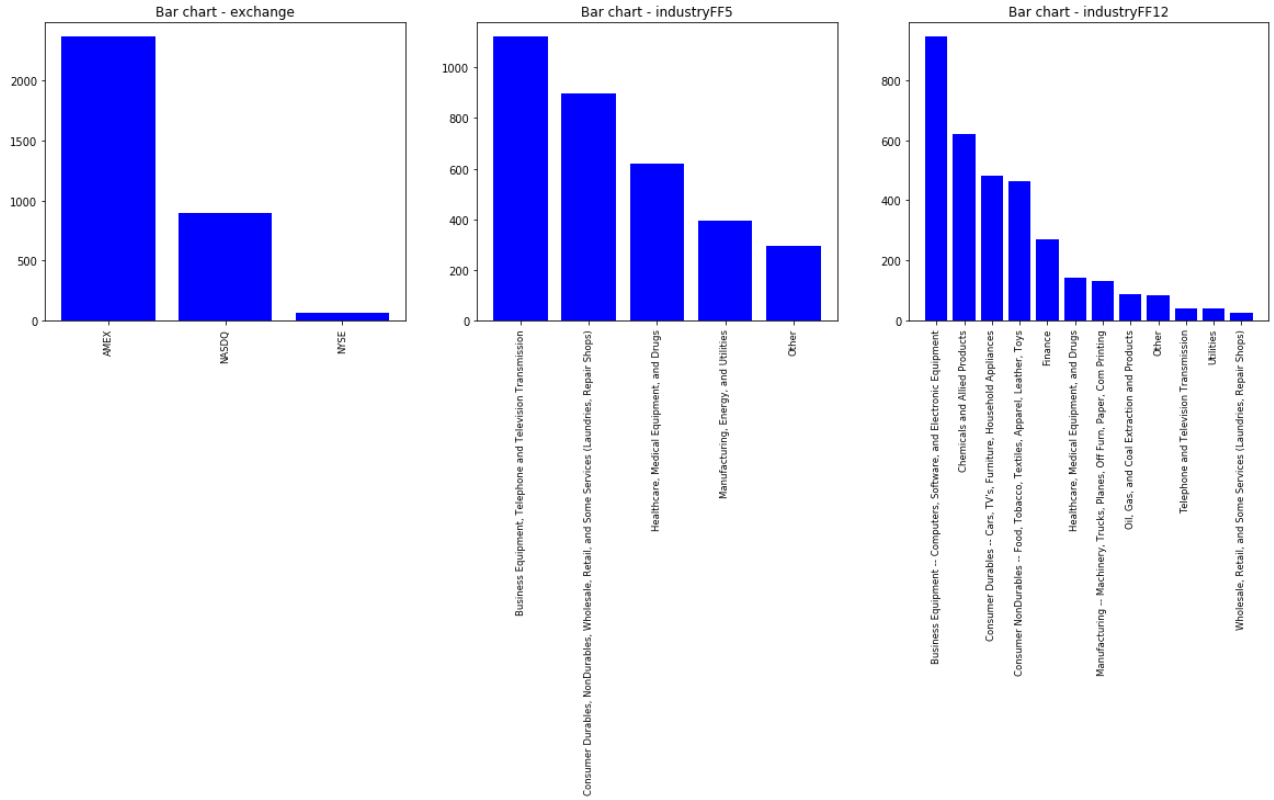
```
Out of 1167 companies without Venture Capital or Private Equity investors we hav
e:
1233 NaN rows in nVCs
1235 NaN rows in nExecutives
1236 NaN rows in priorFinancing
1233 NaN rows in ipoSize

Out of 647 companies without Venture Capital and with Private Equity investors w
e have:
88 NaN rows in nVCs
128 NaN rows in nExecutives
158 NaN rows in priorFinancing
80 NaN rows in ipoSize

Out of 1384 companies with Venture Capital and without Private Equity investors
we have:
8 NaN rows in nVCs
66 NaN rows in nExecutives
22 NaN rows in priorFinancing
3 NaN rows in ipoSize
```

Object features

Some object variables have more than 50 categories, we do not plot them using a bar chart. We only plot the veriables that have less than 50 categories.

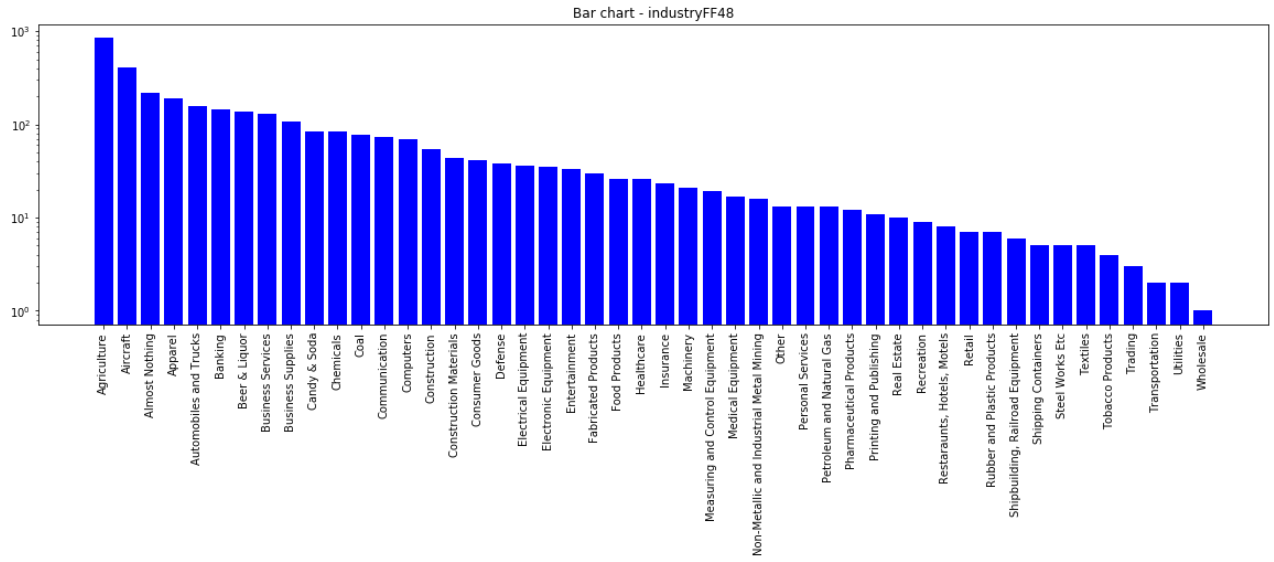
In [24]:
```
# Subset the variables that have less than 50 categories and store them in a lis
object_features_analysis = ['exchange','industryFF5', 'industryFF12', 'industryF
```

In [25]:
```
# Plots for variables ['exchange','industryFF5', 'industryFF12', 'industryFF48']
fig, ax = plt.subplots(1,3,figsize=(20,5))
for k in range(3):
    # Create count data frame
    count = IPO_df.reset_index()[[object_features_analysis[k], 'index']].groupby
    count.columns = ['feature', 'count']
    count = count.sort_values(['count'], ascending = False)

    # Plot bar charts for the object variables using the count data frame
    x_pos = [i for i, _ in enumerate(count['feature'])]
    ax[k].bar(x_pos, count['count'], color='b')

    # Tunning the plot
    ax[k].set_title("Bar chart - {}".format(object_features_analysis[k]))
    ax[k].set_xticks(count.index)
    ax[k].set_xticklabels(count.feature, rotation=90, fontsize=8.5)
```



In [26]:

```
# Plots for variables ['altchord','timesig','global_key','local_key']
fig, ax = plt.subplots(figsize=(20,5))
for k in range(1):
    count = IPO_df.reset_index()[[object_features_analysis[k+3], 'index']].group
    count.columns = ['feature', 'count']
    count = count.sort_values(['count'], ascending = False)
    x_pos = [i for i, _ in enumerate(count['feature'])]
    ax.bar(x_pos, count['count'], color='b')
    ax.set_title("Bar chart - {}".format(object_features_analysis[k+3]))
    ax.set_yscale("log")
    ax.set_xticks(count.index)
    ax.set_xticklabels(count.feature, rotation=90, fontsize=10)
```



In [27]:

```
# Define a function to count observations per category
```

```
def create_count_feature_df(data, features_list, byVar):
    """ Function that returns a data frame that contains the count of observatio

        Args:
        data   pandas DataFrame
        features_list List containing features' names to subset from data
        byVar   String with the feature's name to group by the count
    """
    count_df = data[features_list].groupby(byVar).count().reset_index()
    count_df.columns = ['Category', 'Number']
    count_df = count_df.sort_values(['Number'], ascending = False)
    count_df['Percentage'] = count_df['Number']/data.shape[0]
    return count_df
```

The object variable `City` has 948 different cities. We show the top 5 most frequent cities in the data set instead of plotting a bar chart.

The most frequent city for IPOs is `New York`

In [28]:
```
# Print number of cities in variables City
print("Number of distinct cities in the IPO data frame: {}".format(np.size(IPO_d
```

```
Number of distinct cities in the IPO data frame: 948
```

In [29]:
```
# Display top 5 most frequent cities in the IPO training data set
create_count_feature_df(IPO_df.reset_index(), ['city', 'index'], 'city').head(5)
```

Out[29]:

	Category	Number	Percentage
573	NEW YORK	195	0.058559
381	HOUSTON	98	0.029429
727	SAN DIEGO	87	0.026126
122	CAMBRIDGE	78	0.023423
728	SAN FRANCISCO	71	0.021321

The top 5 most frequent managers in the IPO data set are:

In [30]:
```
# Display top 5 most frequent managers for IPOs
create_count_feature_df(IPO_df.reset_index(), ['manager', 'index'], 'manager').h
```

Out[30]:

	Category	Number	Percentage
521	Goldman Sachs & Co	126	0.037838
977	Merrill Lynch & Co Inc	94	0.028228
162	CS First Boston Corp	85	0.025526
1213	Morgan Stanley Dean Witter & Co	73	0.021922
473	Donaldson Lufkin & Jenrette Inc	68	0.020420

There are 3,215 distinct issuers in the data set.

In [31]:
```
# Print number of cities in variables City
print("Number of distinct issuers (companies) in the IPO data frame: {}".format(
```

```
Number of distinct issuers (companies) in the IPO data frame: 3330
```

Pandas Profiling

We execute a Pandas profiling report and export it as an html file for easier review and handling.

In [32]:
```
#create pandas profiling report
# import pandas_profiling as pdpfg
# pdpfg.ProfileReport(df_learn)
# profile = df_learn.profile_report(title='Pandas Profiling Report')
# profile.to_file(output_file="IPO Filings data profiling.html")
```

General Preprocessing

The `IPO` data frame contains real-world data, meaning it is very *dirty*. In this part of the notebook, we describe the different methods for cleaning the complete data set.

All the nine predictions we need to do involve the veriable `closeDay1` , that is the reason why we drop the observations that contain a NaN value in it. We saw before that this feature only contains 115 missing values, so the new data set has 3,215 observations.

In [33]:
```
# Drop observations with missing value in closeDay1
IPO_df = IPO_df.dropna(subset=['closeDay1'])
print_shape(IPO_df)
```

```
Observations: 3215
Features: 46
```

In [34]:
```
#convert true and false into integer
IPO_df[['egc','html','highTech','vc','pe']]= IPO_df[['egc','html','highTech','vc
```

Most of the predictions have as target variable how the stock price variates after the first day. We proceed to create a feature called `Price_var` that contains the stock's price variation between the offering price and the closing price.

In [35]:
```
# Create variable that reflects the stock's price variation
IPO_df['Price_var'] = (IPO_df['closeDay1'] / IPO_df['offerPrice'])-1

# Display IPO_df to see the variable was created
IPO_df[['closeDay1', 'offerPrice','Price_var']].head()
```

Out[35]:

	closeDay1	offerPrice	Price_var
974	35.5625	14.0	1.540179
2585	20.0000	18.0	0.111111
1936	26.1500	21.0	0.245238

	closeDay1	offerPrice	Price_var
842	13.8125	12.0	0.151042
2026	18.1500	12.5	0.452000

Some values in the variable `patRatio` have `inf` values, the reason could the that the variable represents a ratio of the number of times the word `Patent` appears in the Prospectus document. The variable was computed by dividing the number of times the word `Patent` by the length of the Prospectus Risk content. If the numerator was the number `0`, then python outputs the value `inf`

In [36]:
```
IPO_df['patRatio'][IPO_df['patRatio']==np.inf]
```

Out[36]:
```
449     inf
149     inf
2394    inf
471     inf
2508    inf
107     inf
3654    inf
3184    inf
1884    inf
3529    inf
2264    inf
2247    inf
Name: patRatio, dtype: float64
```

After knowing the previous, we procceed to replace all infinite values in the variable by `numpy.NaN`

In [37]:
```
IPO_df['patRatio'] = IPO_df['patRatio'].replace([np.inf, -np.inf], np.nan)
```

We saw that the variable `city` has more than 948 occurences and that the variable `manager` has also many categories. The We decided to drop those columns for the model. We also drop the variable `issuer` as it contains 3,215 different categories.

In [38]:
```
IPO_df = IPO_df.drop(columns = ['city','manager', 'issuer', 'industryFF48'])
```

As we saw in the previous parts, there are many missing values in the table of data. For the columns with few missing values, we decided to replace the missing values by the median of the sample (and not the mean not be influenced by outliers). Then, we tried to drop the column when the percentage of missing values was superior to 40%. We have also tried for bigger and lower levels but the AUC was decreasing, reason why we finally chose 40%.

In [39]:
```
IPO_df = IPO_df.drop(columns = ['investmentReceived', 'nExecutives', 'priorFinan
```

Predictions

Modeling Strategy

In this part of the project, we train different tasks to predict. A summary of them can be found below.

Prediction Task	Target type	Description
Prediction 1	Binary	Whether the closing price at the end of the first day of trading will go up from the original offer price
Prediction 2	Binary	Whether the closing price at the end of the first day of trading will go up from the original offer price
Prediction 3	Binary	Whether the closing price at the end of the first day of trading will go up from the original offer price
Prediction 4	Binary	Whether the closing price at the end of the first day of trading will go up more than 20% from the original offer price
Prediction 5	Binary	Whether the closing price at the end of the first day of trading will go down by more than 20% from the original offer price
Prediction 6	Continuous	Predict share price at the end of the first day
Prediction 7	Binary	Whether the closing price at the end of the first day of trading will go up by more than 5% from the original offer price using a score metris
Prediction 8	Binary	Whether the closing price at the end of the first day of trading will go up by more than 50% from the original offer price using a score metris
Prediction 9	Binary	Whether the closing price at the end of the first day of trading will go down by more than 10% from the original offer price using a score metris

Eight of these predictions are classification problems as we have a binary target. Only one prediction is a regression problem, as it has a continuous target variable.

The strategy we implemented for modeling the above problems is the following:

Splitting the data We split the data using the `sklearn` function. We asign 70% of the data set for training and the rest 30% for testing. We decided not to have a third validate data set, as the IPO data frame has around 3,000 observations.

Training Depending on the prediction task, we choose different types of models.

For classification problems we choose among the following models.

- Logit model with penalize parameter `C`
- k-NN model with number of neighbors `n`
- Decision trees
- Random Forest with number of trees `n`

For regression problems, we choose a Ridge regression with parameter `alpha`, which is a linear regression model with a penalize term. Such term is the L2 norm.

For binary classification predictions 1, 2, 3, 4 and 5 we trained one of the models described above and computed the `AUC score` for both the train and the test data set for a set of different parameters. We chose the optimum parameter that satisfied the following:

- Parameter that maximizes the `AUC score` in the test data set
- Parameter that minimizes the difference of `AUC score` between train and test data set, to avoid overfitting.

For binary classification predictions 7, 8 and 9: We compute the requiered score metric and choose the parameter that minimizes it.

For regression problem 6, we used RidgeCV, a `sklearn` function that performs CV on a set of parameters `alpha` to choose the optimum one.

Testing

For classification problems: We test all predictions using the test data set. We did not only compute the `accuracy` score but the `AUC score`, `F1 score`, as `accuracy score` could be misleading for model selection, specially if the target variable is unbalanced.

For regression problems: We test all predictions using the test data set. We compute the `MSE`, `RMSE` and the R^2.

Model selection

For classification problems: For selecting the optimum model, we first display a summary table with the scores of the different models trained. Then we perform a 5-fold cross validation on each model. We select the model that has on average a higher `AUC` score, and also a small standard deviation. This way, we select the most robust model that has high `AUC` score but that will be consistent when predicting new data sets.

For regression problems: For selection the optimum model, we first display a summary table with the scores of the different models trained. We do not perform K-fold cross validation as the `sklearn` function RidgeCV did it for us. We select the model that has the minimum `RMSE`.

Making prediction

For classification problems:

For predictions 1, 2, 3, 4 and 5: After choosing the best model for each binary classification prediction, we choose the optimum probability threshold to predict the label `0` or `1`. We make a prediction on the `IPO_to_predict` and we asign the label.

For binary classification predictions 7, 8 and 9: After choosing the best model for each prediction, we compute the $P(Y == 1)$ using the data set `IPO_to_predict`

No that we have described our main strategy, we procceed to solve each of the 9 predictions.

Prediction 1

Prediction 1 is about predicting if the closing price at the end of the first day will go up.

In [40]:
```
# Drop variable rf for the prediction 1, as we are not allowed to use it
Pred1_df = IPO_df.drop(columns = ['rf', 'closeDay1'])

# Compute the target variable for the second prediction
Pred1_df['target'] = Pred1_df['Price_var'].apply(lambda x: 1 if x>0 else 0)

# Store target in a numpy array to use it later
Pred1_target = np.array(Pred1_df['target'])

#Display target distribution using function display_target_distribution
display_target_distribution(Pred1_df)
```

Out[40]:

	target	number	percentage
0	0	842	26.19
1	1	2373	73.81

In [41]:
```
#Drop target and Price_var from the training set
Pred1_df = Pred1_df.drop(columns = ['Price_var', 'target'])
```

In [42]:
```
#One hot encoded exchange for variables 'exchange', 'industryFF5', 'industryFF12
Pred1_df = pd.get_dummies(data = Pred1_df, columns = ['exchange', 'industryFF5',
```

In [43]:
```
# Replace missing values with median of each feature
Pred1_df = Pred1_df.fillna(Pred1_df.median())
```

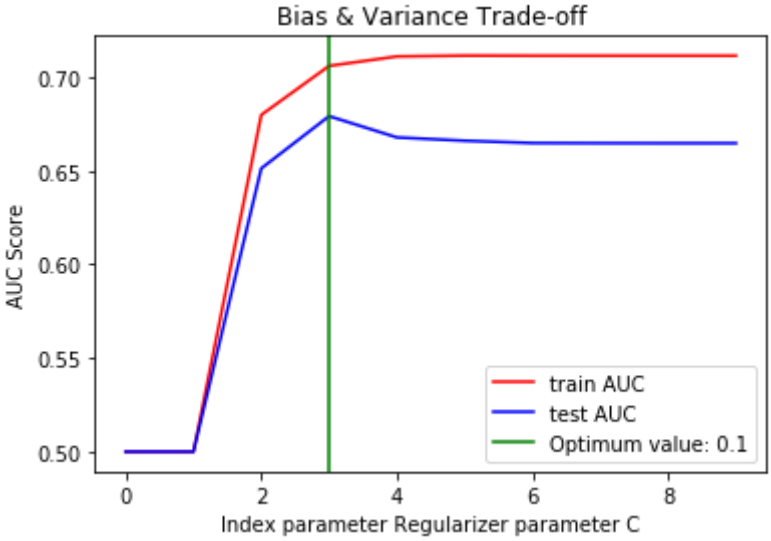
In [44]:
```
#Split the data and the target into a training and a testing set
Pred1_X_train, Pred1_X_test, Pred1_y_train, Pred1_y_test = split_data(Pred1_df,
```

Pred 1 Model 1

First, let's use a logit model.

In [45]:
```
# Build pipeline
estimators = []
estimators.append(('standardize', StandardScaler()))          # tell it to standa
estimators.append(('logit_model_l1', LogisticRegression()))  # tell it to use a
pipeline = Pipeline(estimators)
pipeline.set_params(logit_model_l1__penalty='l1')             # tell it to regula
pipeline.set_params(logit_model_l1__random_state=SEED)

results_C = []
AUC_list_train = []
AUC_list_test = []

#Chose the good c
for c in np.logspace(-4, 5, 10):
    pipeline.set_params(logit_model_l1__C=c)
```

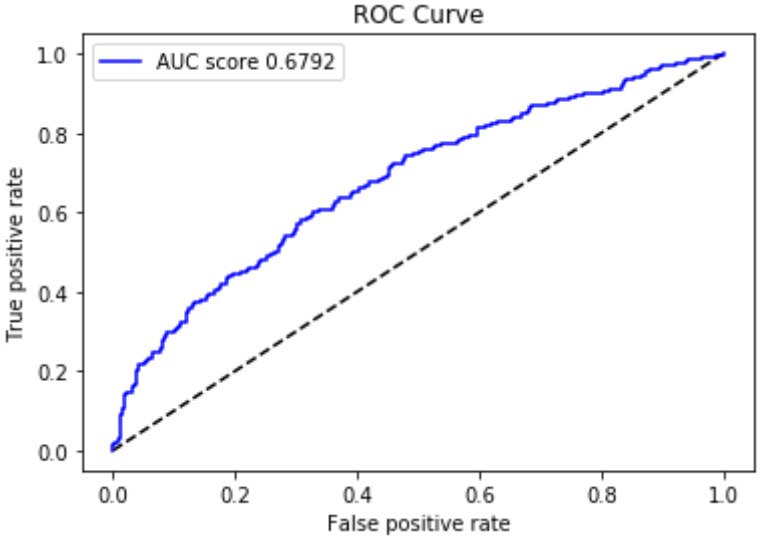
```
    pipeline.fit(Pred1_X_train, Pred1_y_train)
    Pred1_y_train_pred = pipeline.predict_proba(Pred1_X_train)       # use valid
    auc_train = roc_auc_score(Pred1_y_train, Pred1_y_train_pred[:,1])
    AUC_list_train.append(auc_train)
    Pred1_y_test_pred = pipeline.predict_proba(Pred1_X_test)       # use validat
    auc_test = roc_auc_score(Pred1_y_test, Pred1_y_test_pred[:,1])
    AUC_list_test.append(auc_test)
    results_C.append(c)

choose_index_list = list(np.array(AUC_list_test) - np.abs((np.array(AUC_list_tra
choose_index = choose_index_list.index(np.max(choose_index_list))
pipeline.set_params(logit_model_l1__C = results_C[choose_index])
logit_model_l1 = pipeline.named_steps['logit_model_l1']       # capture model so

#Let's plot the AUC score depending on C for the training and testing data sets
plot_AUC_train_test(AUC_list_train, AUC_list_test, choose_index, results_C[choos
```



In [46]:

```
# Test the model using the function train_and_test_model
AUC, F1, accuracy=train_and_test_model(pipeline, Pred1_X_train,Pred1_y_train,Pre
```

```
The AUC score for the model with parameter alpha 0.100 is: 0.706

The AUC score for the model using the test data set with parameter alpha 0.100 i
s: 0.679
```



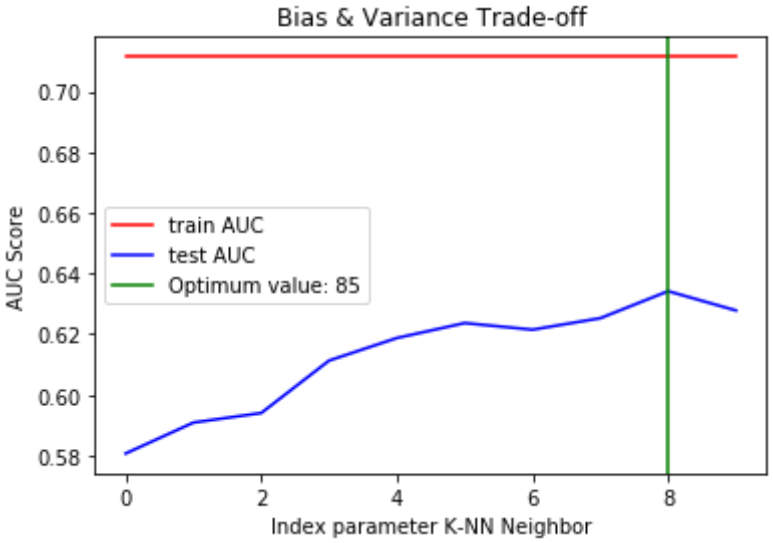
In [47]:
```
#Save the results for later
Pred1_model_1_scores = [AUC, F1, accuracy]
```

In [48]:
```
# Save the model for model comparison later
Pred1_model_1 = pipeline
```

Pred 1 Model 2

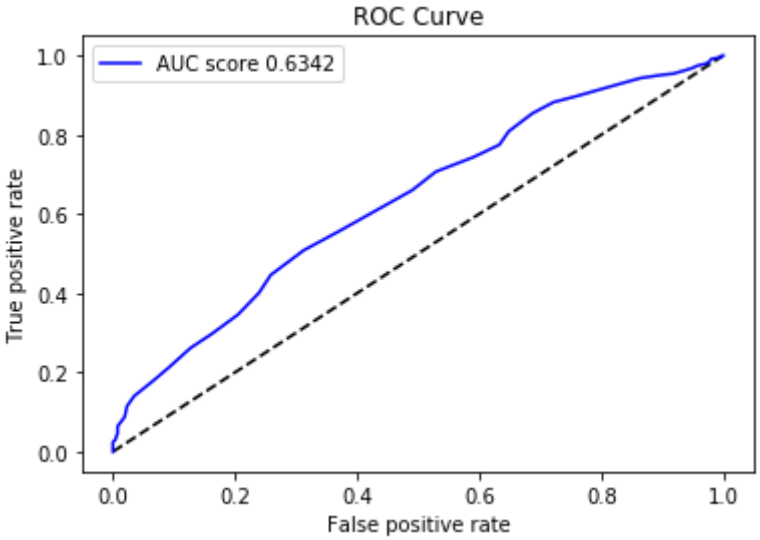
let's now use a KNN model

In [49]:
```
# Build pipeline
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('knn_model', KNeighborsClassifier()))
pipeline = Pipeline(estimators)


results_C = []
AUC_list_train = []
AUC_list_test = []

for k in range(5, 100, 10):
    pipeline.set_params(knn_model__n_neighbors=k)
    pipeline.fit(Pred1_X_train, Pred1_y_train)
    y_train_pred = pipeline.predict_proba(Pred1_X_train)       # use validation
    auc_train = roc_auc_score(Pred1_y_train, Pred1_y_train_pred[:,1])
    AUC_list_train.append(auc_train)
    Pred1_y_test_pred = pipeline.predict_proba(Pred1_X_test)       # use validat
    auc_test = roc_auc_score(Pred1_y_test, Pred1_y_test_pred[:,1])
    AUC_list_test.append(auc_test)
    results_C.append(k)

choose_index_list = list(np.array(AUC_list_test) - np.abs((np.array(AUC_list_tra
choose_index = choose_index_list.index(np.max(choose_index_list))
pipeline.set_params(knn_model__n_neighbors = results_C[choose_index])
knn_model = pipeline.named_steps['knn_model']       # capture model so we can us

#Let's plot the AUC score depending on C for the training and testing data sets
plot_AUC_train_test(AUC_list_train, AUC_list_test, choose_index, results_C[choos
```



In [50]:
```
# Test the model using the function train_and_test_model
AUC, F1, accuracy=train_and_test_model(pipeline, Pred1_X_train, Pred1_y_train, P
```

```
The AUC score for the model with parameter alpha 85.000 is: 0.664

The AUC score for the model using the test data set with parameter alpha 85.000
is: 0.634
```



In [51]:
```
# Save the parameters for model comparison later
Pred1_model_2_scores = [AUC, F1, accuracy]
```

In [52]:
```
# Save the model for model comparison later
Pred1_model_2 = pipeline
```

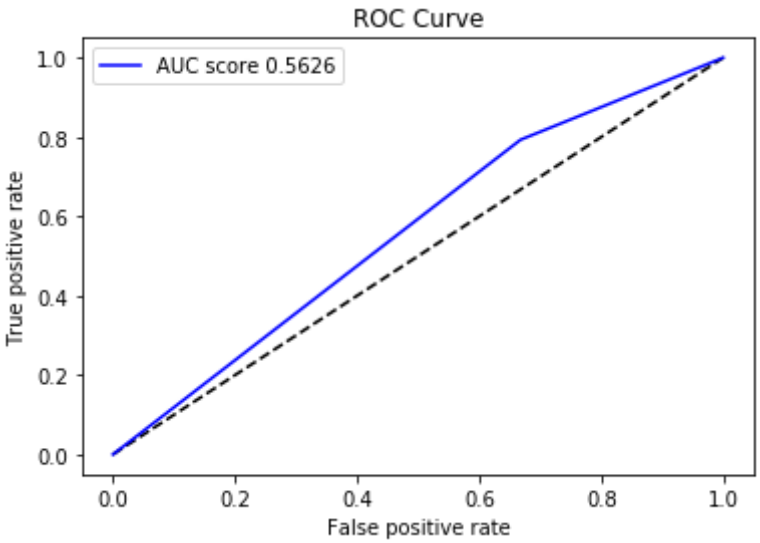
Pred 1 Model 3

Let's try to modelise with decision trees.

In [53]:
```
# Fit a one-tree Model
tree_model = DecisionTreeClassifier(random_state = SEED)  # just one tree, so no
```

```
tree_model.fit(Pred1_X_train, Pred1_y_train)
pass  # skip showing the model internals

# Test model
AUC, F1, accuracy=train_and_test_model(tree_model, Pred1_X_train, Pred1_y_train,
```

```
The AUC score for the model with parameter alpha 1.000 is: 1.000

The AUC score for the model using the test data set with parameter alpha 1.000 i
s: 0.563
```



In [54]:
```
Pred1_model_3_scores=[AUC, F1, accuracy]
```

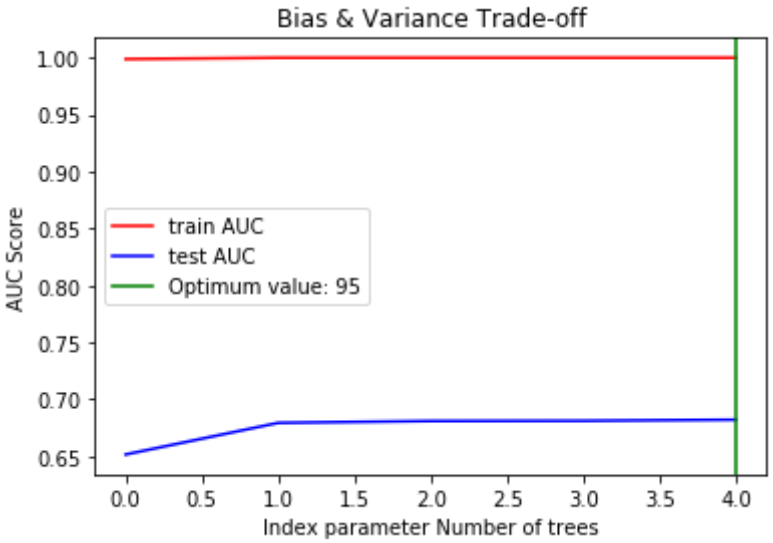
In [55]:
```
# Save the model for model comparison later
Pred1_model_3 = tree_model
```

Pred 1 Model 4

Let's try a random forest.

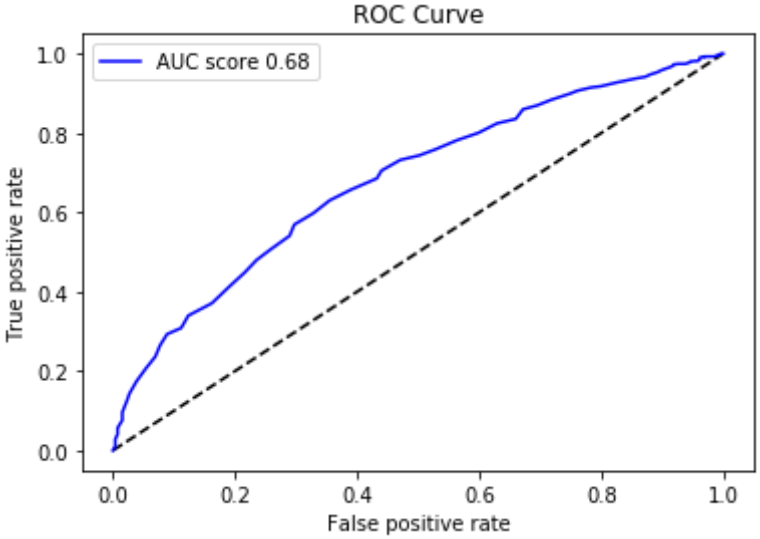
In [56]:
```
# Build pipeline
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('forest_model', RandomForestClassifier()))
pipeline = Pipeline(estimators)
pipeline.set_params(forest_model__random_state=SEED)

# Tune N
results_C = []
AUC_list_train = []
AUC_list_test = []

for n in [10, 50, 150, 200, 250]:
    pipeline.set_params(forest_model__n_estimators = n)
    pipeline.fit(Pred1_X_train, Pred1_y_train)
    Pred1_y_train_pred = pipeline.predict_proba(Pred1_X_train)       # use valid
    auc_train = roc_auc_score(Pred1_y_train, Pred1_y_train_pred[:,1])
    AUC_list_train.append(auc_train)
    Pred1_y_test_pred = pipeline.predict_proba(Pred1_X_test)       # use validat
```

```
    auc_test = roc_auc_score(Pred1_y_test, Pred1_y_test_pred[:,1])
    AUC_list_test.append(auc_test)
    results_C.append(k)

choose_index_list = list(np.array(AUC_list_test) - np.abs((np.array(AUC_list_tra
choose_index = choose_index_list.index(np.max(choose_index_list))
pipeline.set_params(forest_model__n_estimators = results_C[choose_index])
forest_model = pipeline.named_steps['forest_model']       # capture model so we

plot_AUC_train_test(AUC_list_train, AUC_list_test, choose_index, results_C[choos
```



In [57]:
```
# Test the model using the function train_and_test_model
AUC, F1, accuracy=train_and_test_model(pipeline, Pred1_X_train,Pred1_y_train, Pr
```

```
The AUC score for the model with parameter alpha 95.000 is: 1.000

The AUC score for the model using the test data set with parameter alpha 95.000
is: 0.680
```



In [58]:
```
# Save the parameters for model comparison later
Pred1_model_4_scores=[AUC, F1, accuracy]
```

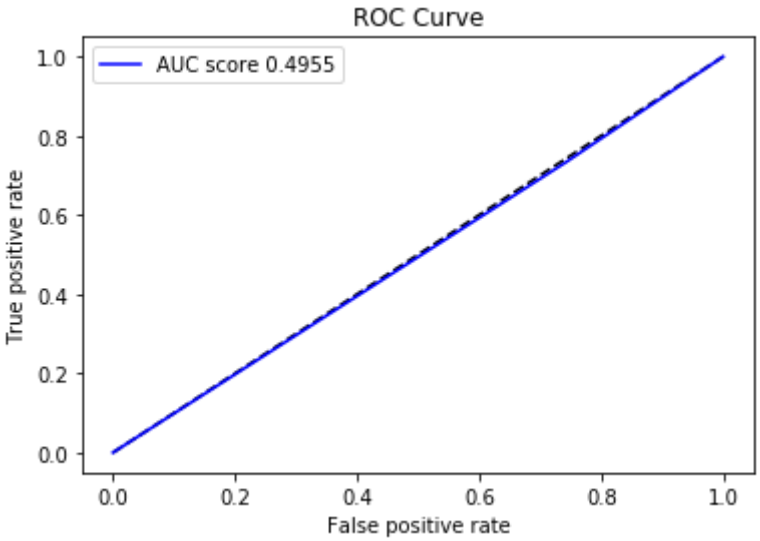
In [59]:
```
# Save the model for model comparison later
Pred1_model_4 = pipeline
```

Baseline model

Now let's implement a dummy model that we can compare with the other models implemented.

In [60]:
```
from sklearn.dummy import DummyClassifier
Pred1_dummy = DummyClassifier(strategy="stratified", random_state = SEED)# stati
AUC, F1, accuracy = train_and_test_model(Pred1_dummy, Pred1_X_train, Pred1_y_tra
Pred1_model_dummy_scores = [AUC, F1, accuracy]
```

```
The AUC score for the model with parameter alpha 0.000 is: 0.483

The AUC score for the model using the test data set with parameter alpha 0.000 i
s: 0.495
```



Choice of the model

Let's compute AUC sore, F1 score and accuracy score to choose which model is the best.

In [61]:
```
#Let's do a list with the different models and with their names
Pred1_model_scores = [Pred1_model_dummy_scores, Pred1_model_1_scores, Pred1_mode
Pred1_model_names = ['Dummy_model','Model 1', 'Model 2', 'Model 3','Model 4']
```

In [62]:
```
#Let's pint their parameters into a table in order to compare tthe models
print_model_scores(Pred1_model_scores, Pred1_model_names)
```

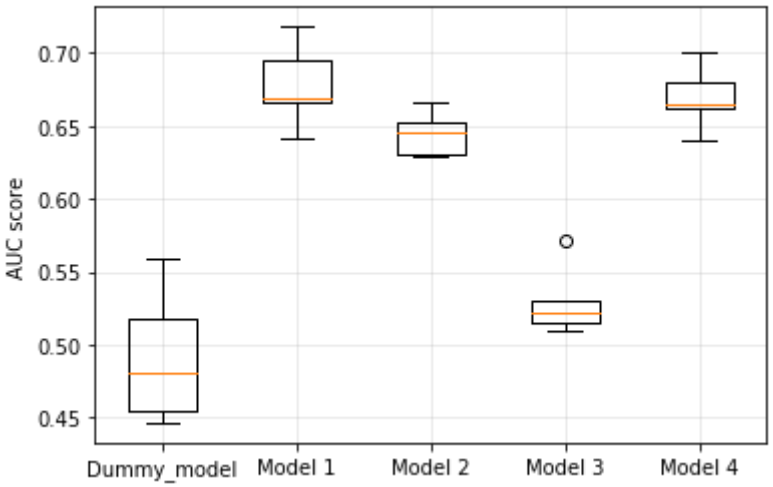
```
-----------------------------------------------------------------
Model name      AUC score       F1 score        Accuracy score
-----------------------------------------------------------------
Dummy_model     0.495           0.731           0.605
Model 1         0.679           0.842           0.733
Model 2         0.634           0.843           0.730
Model 3         0.563           0.778           0.669
Model 4         0.680           0.835           0.725
-----------------------------------------------------------------
```

In [63]:

```
#Let's implement data for the cross validation
Cross_val_df = Pred1_X_test
Cross_val_target = Pred1_y_test
```

In [64]:

```
#Let's display plotboxes to compare the different models
scores_list = []
from sklearn.model_selection import cross_val_score
for model in [Pred1_dummy, Pred1_model_1, Pred1_model_2, Pred1_model_3, Pred1_mo
    scores = cross_val_score(model, Cross_val_df, Cross_val_target, cv=5, scorin
    scores_list.append(scores)

fig, ax = plt.subplots()
pos = np.array(range(len(scores_list))) + 1
ax.boxplot(scores_list, positions=pos)
ax.set_xticklabels(Pred1_model_names)
ax.set_ylabel('AUC score')
ax.grid(alpha = 0.3)
plt.show()
```



In [65]:

```
# Save CV results
Pred1_mean_cv = list(np.mean(scores_list, axis = 1))
Pred1_std_cv = list(np.std(scores_list, axis = 1))

# Print CV results using the function print_cv_scores
print_cv_scores(Pred1_mean_cv, Pred1_std_cv, Pred1_model_names)
```

```
----------------------------------------
Model name      Mean AUC        Std AUC
----------------------------------------
Dummy_model     0.492           0.042
Model 1         0.678           0.026
Model 2         0.645           0.014
Model 3         0.530           0.022
Model 4         0.669           0.020
----------------------------------------
```

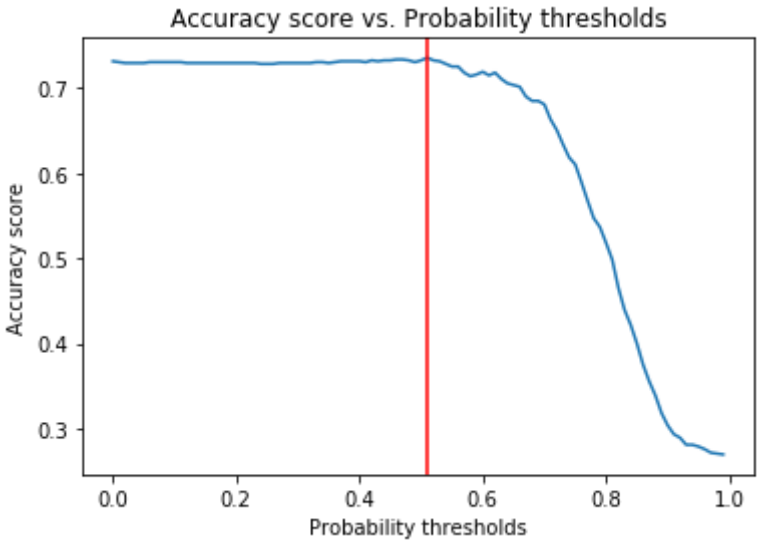
We choose model 1 because it has the best AUC and not much variance.

In [66]:

```
# Choose best model
Pred1_winner_model = Pred1_model_1
```

In [67]:
```
# Pick probability threshold that maximizes accuracy
Pred1_winner_probability = Pred1_winner_model.predict_proba(Pred1_X_test)[:,1]

Pred1_threshold, Pred1_accuracy, Pred1_thresholds_list = choose_thresholds(Pred1
plot_across_thresholds(Pred1_thresholds_list, Pred1_threshold)

Pred1_winner_label = pd.DataFrame(Pred1_winner_probability, columns = ['y_proba'
Pred1_winner_label['target'] = Pred1_winner_label['y_proba'].apply(lambda x: 1.0
```

```
The threshold that maximizes the accuracy index is:  0.51 and the accuracy score
is:  0.7357512953367875
```



Testing prediction in test data set.

In [68]:
```
# Display accuracy of predicted target
accuracy_score(Pred1_y_test, Pred1_winner_label['target'])
```

Out[68]:
```
0.7357512953367875
```

Prediction 2

Here we perform a text analysis for predicting if the closing price at the end of the first day will go up using only the variables `rf` , `year` and `industryFF12`

In [70]:
```
# Subset the variables rf, year and industryFF12 from IPO_df and store it in a d
Pred2_features = ['Price_var','rf', 'year', 'industryFF12']

# Drop rows that contain NaN in teh variable rf
Pred2_df = IPO_df[Pred2_features].dropna(subset=['rf'])

# Compute the target variable for the second prediction
Pred2_df['target'] = Pred2_df['Price_var'].apply(lambda x: 1 if x>0 else 0)

# Store target in a numpy array to use it later
Pred2_target = np.array(Pred2_df['target'])
```

Text analytics preprocessing

Text cleaning

Here we define NLP techniques to clean the variable `rf`

In [72]:
```
# Get English Stopwords from NLTK
# Import stopwords
from nltk.corpus import stopwords
import re

# Store English stopwords in a list
stop_words = list(set(stopwords.words('english')))

# Print the list's length
print("The stop words list has a length of {} words".format(len(stop_words)))
```

```
The stop words list has a length of 179 words
/home/irene/Anaconda3/lib/python3.7/site-packages/nltk/decorators.py:68: Deprecation
Warning: `formatargspec` is deprecated since Python 3.5. Use `signature` and
the `Signature` object directly
  regargs, varargs, varkwargs, defaults, formatvalue=lambda value: ""
/home/irene/Anaconda3/lib/python3.7/site-packages/nltk/lm/counter.py:15: Deprecation
Warning: Using or importing the ABCs from 'collections' instead of from 'col
lections.abc' is deprecated, and in 3.8 it will stop working
  from collections import Sequence, defaultdict
```

In [73]:
```
# Define a custom function to clean some given text
def clean_re(txt):
    """ Function to clean puntuaction with regular expressions

        Args:
        txt pandas series string to clean
    """
    # Tranform each word into lower case
    txt = pd.Series(txt).str.lower()

    # Remove and replace any non-whitespace character like \t\n\r\f\v
    txt = pd.Series(txt).apply(lambda x: re.sub(r'[^\w\s]',' ',x))

    # Remove very short words as they do not provide useful information
    txt = txt.apply(lambda x: ' '.join([w for w in x.split() if len(w)>3]))

    # Remove numbers
    txt = pd.Series(txt).apply(lambda x: re.sub(r'\b\d+(?:\.\d+)?\s+', ' ', x))

    # Remove words 'risk factors' as they appear in every row
    txt = pd.Series(txt).str.replace("risk factors", ' ')
    return txt

# Define custom function to remove stopwords
def remove_stopwords(txt):
    """ Function to remove stop words

        Args:
        txt pandas series string to clean
    """
    # Remove stop words
    txt = pd.Series(txt).apply(lambda x: ' '.join([item for item in x.split() if
    return txt
```

```
# Write a lemmatization function based on nltk.stem.WordNetLemmatizer()
from nltk.stem import WordNetLemmatizer
def apply_lemmatization(txt):
    """ Function to apply the lemmatization function to a sentece

        Args:
        txt pandas series string to clean
    """
    lemmatizer = WordNetLemmatizer()
    txt = [word for word in txt.split()]
    txt = ' '.join([lemmatizer.lemmatize(w) for w in txt])
    return txt

prospectus_text = ['factors including','risk factors', 'you should', 'the occurr
# Define custom function to remove stopwords
def remove_prospectus_sentences(txt):
    """ Function to remove prospectus sentences

        Args:
        txt pandas series string to clean
    """
    # Remove stop words
    txt = pd.Series(txt).apply(lambda x: ' '.join([item for item in x.split('.')
    return txt

# Write a stemming function based on nltk.stem.WordNetLemmatizer()
from nltk.stem import PorterStemmer
def apply_stemmer(txt):
    """ Function to apply the lemmatization function to a sentece

        Args:
        txt pandas series string to clean
    """
    stemmer = PorterStemmer()
    txt = [word for word in txt.split()]
    txt = ' '.join([stemmer.stem(w) for w in txt])
    return txt

# WordCloud is a little word cloud generator in Python
from wordcloud import WordCloud, ImageColorGenerator, STOPWORDS

# Plotting wordcloud method
def plot_wordcloud(data, feature):
    """ Function to plot a wordcloud made from the most frequent words in a text

        Args:
        data pandas data frame containing the text
        feature string containing the name of the feature to analyze
    """
    # Determines the wordcloud results based on the dataframe provided
    all_words = ' '.join([text for text in data[feature]])
    wordcloud = WordCloud(width=800, height=500,
                          random_state=21,
                          max_font_size=110,
                          background_color="white",
                          max_words = 25,
                          stopwords = set(STOPWORDS)).generate(all_words)
    return wordcloud
```

We start cleaning the variable `rf`

In [74]:
```
# Clean the rf variable using the variables defined above
Pred2_df['rf_clean'] = Pred2_df['rf'].apply(remove_prospectus_sentences)
Pred2_df['rf_clean'] = Pred2_df['rf_clean'].apply(clean_re)
Pred2_df['rf_clean'] = Pred2_df['rf_clean'].apply(remove_stopwords)
```

In the following part, we plot word clouds to see what are the most frequent words in the `IPO prospectus` for each industrya and each target value

In summary, for some industries, we got the following insights:

- For the Business Equipment -- Computers, Software, and Electronic Equipment industry:
  - When the stock price went up: `intellectual property` , `products service`
  - When the stock prince went down: `service` , `revenue`
- For the Healthcare, Medical Equipment, and Drugs industry:
  - When the stock price went up: `intellectual property` , `clinical trial`
  - When the stock prince went down: `clinical trial` `regulatory approval`
- For the Utilities industry:
  - When the stock price went up: `adquiscition`
  - When the stock prince went down: `cash flow`

In [75]:
```
# Store industryFF12 categoriies
industry_list = list(Pred2_df['industryFF12'].unique())
```

In [76]:
```
# Plot a word cloud for each of the categories in industryFF12 and for each targ
for i in range(np.size(industry_list)):
    # Create data sets with each industryFF12 category and target value
    Pred2_industry_df_target1 = Pred2_df[(Pred2_df['industryFF12'] == industry_l
    Pred2_industry_df_target0 = Pred2_df[(Pred2_df['industryFF12'] == industry_l
    print("Word cloud for the feature: {}\n ".format(industry_list[i]))

    # Plot the word clouds
    fig,ax = plt.subplots(1,2, figsize=(15,10))

    ax[0].imshow(plot_wordcloud(Pred2_industry_df_target1, 'rf_clean'), interpol
    ax[0].set_title('Word Cloud - Stock price going up')
    ax[0].axis('off')
    ax[1].imshow(plot_wordcloud(Pred2_industry_df_target0, 'rf_clean'), interpol
    ax[1].set_title('Word Cloud - Stock price going down')
    ax[1].axis('off')
    plt.show()
```

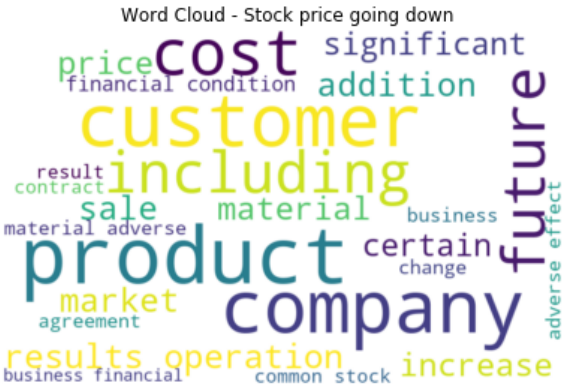
```
Word cloud for the feature: Business Equipment -- Computers, Software, and Elect
ronic Equipment
```

Word cloud for the feature: Finance

Word cloud for the feature: Other

Word cloud for the feature: Healthcare, Medical Equipment, and Drugs

Word cloud for the feature: Manufacturing -- Machinery, Trucks, Planes, Off Furn, Paper, Com Printing



Word cloud for the feature: Telephone and Television Transmission

Word cloud for the feature: Wholesale, Retail, and Some Services (Laundries, Rep air Shops)

Word cloud for the feature: Utilities

Word cloud for the feature: Consumer NonDurables -- Food, Tobacco, Textiles, App arel, Leather, Toys

Word cloud for the feature: Oil, Gas, and Coal Extraction and Products

Word cloud for the feature: Consumer Durables -- Cars, TV's, Furniture, Household Appliances

Word cloud for the feature: Chemicals and Allied Products

In [77]:
```
# Keep cleaining the rf variable using the variables defined above
Pred2_df['rf_clean'] = Pred2_df['rf_clean'].apply(apply_lemmatization)
```

```
Pred2_df['rf_clean'] = Pred2_df['rf_clean'].apply(apply_stemmer)
```

Word embeddings

Here we compute word embeddings, which are a sparse numerical matrix representation of text. This is helpful, as all machine learning models use numeric input.

In [78]:
```
# Splitting the data into X_train, X_test, y_train, y_test using the function sp
Pred2_X_train, Pred2_X_test, Pred2_y_train, Pred2_y_test = split_data(Pred2_df,
```

In [79]:
```
# Transforming text into vectors method
def text_transformer(X_train, X_test, varToVector, vectorizer):
    """ Function to apply the lemmatization function to a sentece

        Args:
        X_train pandas dataframe
        X_test  pandas dataframe
        varToVector string containing the name of teh feature to vectorize
        vectorizer object to transform form word to vector
    """
    # Transforms the pandas dataframe into a numerical matrix
    vectorizer_ = vectorizer
    vectorizer_.fit(X_train[varToVector])
    X_train = vectorizer_.transform(X_train[varToVector])
    X_test = vectorizer_.transform(X_test[varToVector])
    return X_train, X_test
```

We first split the data, as stated in the `Strategy` subsection in `Predictions` section

In [80]:
```
# Create word embeddings to train a model using the function text_transformer Tf
Pred2_X_train_emb, Pred2_X_test_emb = text_transformer(Pred2_X_train, Pred2_X_te
                                                                        lowerc

# Convert to pandas data frame the sparse matrix that contains the word embbedin
Pred2_X_train_emb = pd.DataFrame(Pred2_X_train_emb.toarray())
Pred2_X_test_emb = pd.DataFrame(Pred2_X_test_emb.toarray())

# Merge the word embbedings data frame with the variables 'year' and 'industryFF
Pred2_X_train = Pred2_X_train_emb.reset_index().merge(Pred2_X_train[['year', 'in
Pred2_X_test = Pred2_X_test_emb.reset_index().merge(Pred2_X_test[['year', 'indus
```

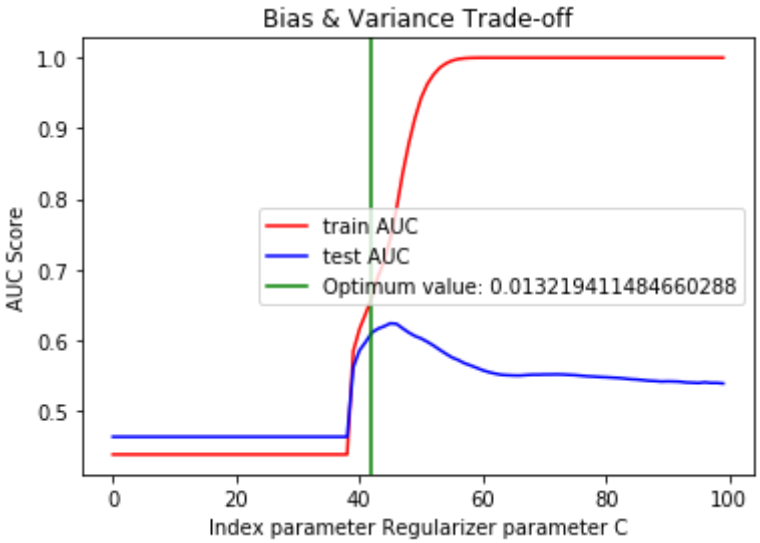
In [81]:
```
# One hot encode variable industryFF12
Pred2_X_train = pd.get_dummies(data = Pred2_X_train, columns = ['industryFF12'])
Pred2_X_test = pd.get_dummies(data = Pred2_X_test, columns = ['industryFF12'])
```

Pred 2 Model 1

This model trains a a logit model with `l1` norm. The optimum parameter is chosen using the `Strategy` subsection in `Predictions` section

In [82]:
```
# Build pipeline
estimators = []
```

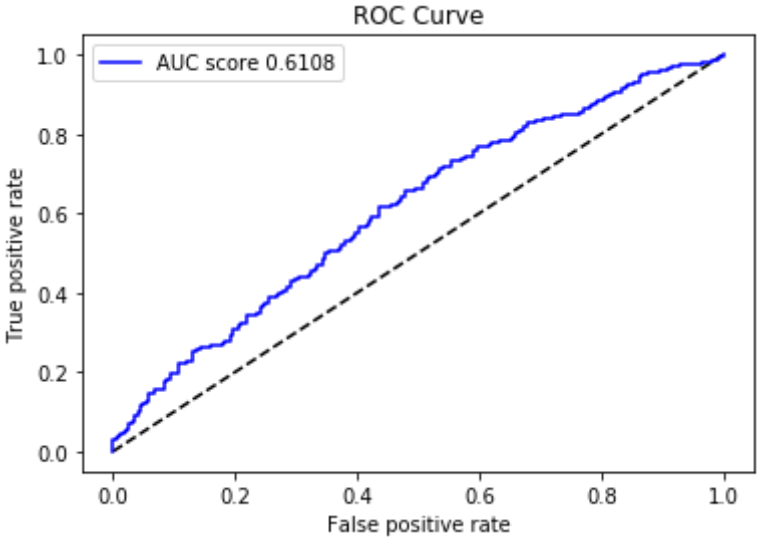
```
estimators.append(('standardize', StandardScaler(with_mean=False)))         # te
estimators.append(('log_reg_l1', LogisticRegression()))  # tell it to use a logi
pipeline = Pipeline(estimators)
pipeline.set_params(log_reg_l1__penalty='l1')         # tell it to regularize wi
pipeline.set_params(log_reg_l1__random_state=SEED)

# Tune C
# The tunning will be done by iterating the penalty value in the range [1.e-04,
results_C = []
AUC_list_train = []
AUC_list_test = []
for c in np.logspace(-4, 1, 100):
    pipeline.set_params(log_reg_l1__C = c)
    pipeline.fit(Pred2_X_train, Pred2_y_train)
    y_train_pred = pipeline.predict_proba(Pred2_X_train)       # use train set d
    auc_train = roc_auc_score(Pred2_y_train, y_train_pred[:,1])
    AUC_list_train.append(auc_train)
    y_test_pred = pipeline.predict_proba(Pred2_X_test)       # use test set duri
    auc_test = roc_auc_score(Pred2_y_test, y_test_pred[:,1])
    AUC_list_test.append(auc_test)
    results_C.append(c) # append parameter C

# Choose the optimum C, such that it maximizes AUC in test and minimizes AUC dif
choose_index_list = list(np.array(AUC_list_test) - np.abs((np.array(AUC_list_tra
choose_index = choose_index_list.index(np.max(choose_index_list))
pipeline.set_params(log_reg_l1__C = results_C[choose_index])
log_reg_l1 = pipeline.named_steps['log_reg_l1']      # capture model so we can u

# Plot AUC score for train and test data set over the different values for param
plot_AUC_train_test(AUC_list_train, AUC_list_test, choose_index, results_C[choos
```



After choosing the optimum parameter, we test it.

In [83]:
```
# Test the model using the function train_and_test_model
AUC, F1, accuracy = train_and_test_model(pipeline, Pred2_X_train, Pred2_y_train,
```

```
The AUC score for the model with parameter alpha 0.013 is: 0.660

The AUC score for the model using the test data set with parameter alpha 0.013 i
s: 0.611
```



In [84]:
```
# Save the scores for model comparison later
Pred2_model_1_scores = [AUC, F1, accuracy]
```

In [85]:
```
# Save the model for model comparison later
Pred2_model_1 = pipeline
```

Pred 2 Model 2

This model trains a logit model with `l2` norm. The model's pipeline includes a dimensionality reduction method called `SVD`. The optimum parameters are chosen using the `Strategy` subsection in `Predictions` section

In [86]:
```
from sklearn.decomposition import TruncatedSVD
# Build pipeline
estimators = []
estimators.append(('standardize', StandardScaler(with_mean=False)))         # te
estimators.append(('svd', TruncatedSVD()))
estimators.append(('log_reg_l2', LogisticRegression()))  # tell it to use a logi
pipeline = Pipeline(estimators)
pipeline.set_params(log_reg_l2__random_state=SEED)
pipeline.set_params(svd__random_state=SEED)

# Find the number n for SVD that maximizes the AUC score
AUC_list_train = []
AUC_list_test = []
n_comp = []
for i in range(40):
    pipeline.set_params(svd__n_components = i+1)
    pipeline.set_params(log_reg_l2__penalty= 'l2')
    pipeline.set_params(log_reg_l2__solver='lbfgs')  # tell it to use a l2 norm
    pipeline.fit(Pred2_X_train, Pred2_y_train)
    y_train_pred = pipeline.predict_proba(Pred2_X_train)       # use train set d
    auc_train = roc_auc_score(Pred2_y_train, y_train_pred[:,1])
    AUC_list_train.append(auc_train)
    y_test_pred = pipeline.predict_proba(Pred2_X_test)       # use test set duri
    auc_test = roc_auc_score(Pred2_y_test, y_test_pred[:,1])
    AUC_list_test.append(auc_test)
```

```
    n_comp.append(i+1)
    print("For {0} number of components in SVD, we get {1:.2f} of AUC score in t
```

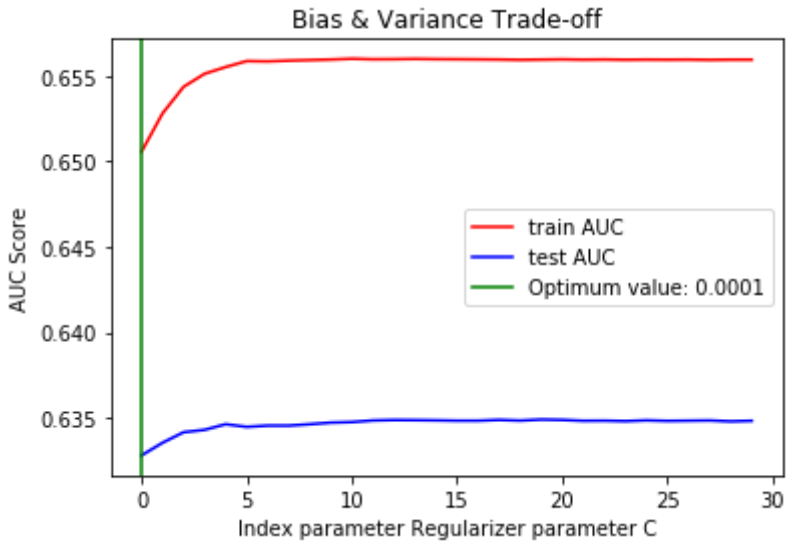
```
For 1 number of components in SVD, we get 0.52 of AUC score in train, and 0.55 i
n test
For 2 number of components in SVD, we get 0.60 of AUC score in train, and 0.59 i
n test
For 3 number of components in SVD, we get 0.60 of AUC score in train, and 0.62 i
n test
For 4 number of components in SVD, we get 0.61 of AUC score in train, and 0.62 i
n test
For 5 number of components in SVD, we get 0.61 of AUC score in train, and 0.62 i
n test
For 6 number of components in SVD, we get 0.61 of AUC score in train, and 0.62 i
n test
For 7 number of components in SVD, we get 0.61 of AUC score in train, and 0.62 i
n test
For 8 number of components in SVD, we get 0.61 of AUC score in train, and 0.62 i
n test
For 9 number of components in SVD, we get 0.62 of AUC score in train, and 0.61 i
n test
For 10 number of components in SVD, we get 0.63 of AUC score in train, and 0.62
in test
For 11 number of components in SVD, we get 0.63 of AUC score in train, and 0.62
in test
For 12 number of components in SVD, we get 0.63 of AUC score in train, and 0.62
in test
For 13 number of components in SVD, we get 0.63 of AUC score in train, and 0.62
in test
For 14 number of components in SVD, we get 0.63 of AUC score in train, and 0.62
in test
For 15 number of components in SVD, we get 0.63 of AUC score in train, and 0.62
in test
For 16 number of components in SVD, we get 0.63 of AUC score in train, and 0.62
in test
For 17 number of components in SVD, we get 0.63 of AUC score in train, and 0.62
in test
For 18 number of components in SVD, we get 0.63 of AUC score in train, and 0.62
in test
For 19 number of components in SVD, we get 0.63 of AUC score in train, and 0.62
in test
For 20 number of components in SVD, we get 0.64 of AUC score in train, and 0.62
in test
For 21 number of components in SVD, we get 0.64 of AUC score in train, and 0.62
in test
For 22 number of components in SVD, we get 0.65 of AUC score in train, and 0.63
in test
For 23 number of components in SVD, we get 0.65 of AUC score in train, and 0.62
in test
For 24 number of components in SVD, we get 0.65 of AUC score in train, and 0.63
in test
For 25 number of components in SVD, we get 0.65 of AUC score in train, and 0.63
in test
For 26 number of components in SVD, we get 0.65 of AUC score in train, and 0.63
in test
For 27 number of components in SVD, we get 0.65 of AUC score in train, and 0.63
in test
For 28 number of components in SVD, we get 0.65 of AUC score in train, and 0.62
in test
For 29 number of components in SVD, we get 0.65 of AUC score in train, and 0.62
in test
For 30 number of components in SVD, we get 0.65 of AUC score in train, and 0.63
in test
For 31 number of components in SVD, we get 0.65 of AUC score in train, and 0.64
```

```
in test
For 32 number of components in SVD, we get 0.65 of AUC score in train, and 0.63
in test
For 33 number of components in SVD, we get 0.65 of AUC score in train, and 0.64
in test
For 34 number of components in SVD, we get 0.66 of AUC score in train, and 0.63
in test
For 35 number of components in SVD, we get 0.66 of AUC score in train, and 0.63
in test
For 36 number of components in SVD, we get 0.65 of AUC score in train, and 0.63
in test
For 37 number of components in SVD, we get 0.65 of AUC score in train, and 0.63
in test
For 38 number of components in SVD, we get 0.66 of AUC score in train, and 0.63
in test
For 39 number of components in SVD, we get 0.66 of AUC score in train, and 0.63
in test
For 40 number of components in SVD, we get 0.66 of AUC score in train, and 0.63
in test
```

After choosing `40` number of components, as they maximize the test `AUC` and minimize `AUC` difference between train and test. We choose the optimum parameter for the penalizer norm for logit model.

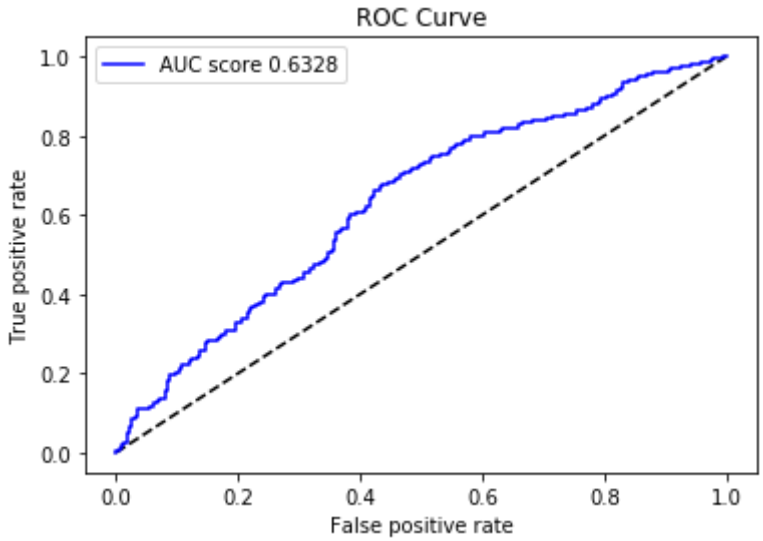
In [87]:

```
# Build pipeline
estimators = []
estimators.append(('standardize', StandardScaler(with_mean=False)))         # te
estimators.append(('svd', TruncatedSVD()))
estimators.append(('log_reg_l2', LogisticRegression()))  # tell it to use a logi
pipeline = Pipeline(estimators)
pipeline.set_params(log_reg_l2__penalty='l2')  # tell it to regularize with L2 n
pipeline.set_params(log_reg_l2__solver='lbfgs')
pipeline.set_params(svd__n_components = 40)
pipeline.set_params(log_reg_l2__random_state=SEED)
pipeline.set_params(svd__random_state=SEED)

# Tune C
# The tunning will be done by iterating the penalty value in the range [1.e-04,
results_C = []
AUC_list_train = []
AUC_list_test = []
for c in np.logspace(-4, 1, 30):
    pipeline.set_params(log_reg_l2__C = c)
    pipeline.fit(Pred2_X_train, Pred2_y_train)
    y_train_pred = pipeline.predict_proba(Pred2_X_train)       # use train set d
    auc_train = roc_auc_score(Pred2_y_train, y_train_pred[:,1])
    AUC_list_train.append(auc_train)
    y_test_pred = pipeline.predict_proba(Pred2_X_test)       # use test set duri
    auc_test = roc_auc_score(Pred2_y_test, y_test_pred[:,1])
    AUC_list_test.append(auc_test)
    results_C.append(c)

# Choose the optimum C, such that it maximizes AUC in test and minimizes AUC dif
choose_index_list = list(np.array(AUC_list_test) - np.abs((np.array(AUC_list_tra
choose_index = choose_index_list.index(np.max(choose_index_list))
pipeline.set_params(log_reg_l2__C = results_C[choose_index])
model = pipeline.named_steps['log_reg_l2']      # capture model so we can use it

# Plot AUC score for train and test data set over the different values for param
plot_AUC_train_test(AUC_list_train, AUC_list_test, choose_index, results_C[choos
```



In [88]:
```
# Test the model using the function train_and_test_model
AUC, F1, accuracy = train_and_test_model(pipeline, Pred2_X_train, Pred2_y_train,
```

```
The AUC score for the model with parameter alpha 0.000 is: 0.651

The AUC score for the model using the test data set with parameter alpha 0.000 i
s: 0.633
```



In [89]:
```
# Save the scores for model comparison later
Pred2_model_2_scores = [AUC, F1, accuracy]
```

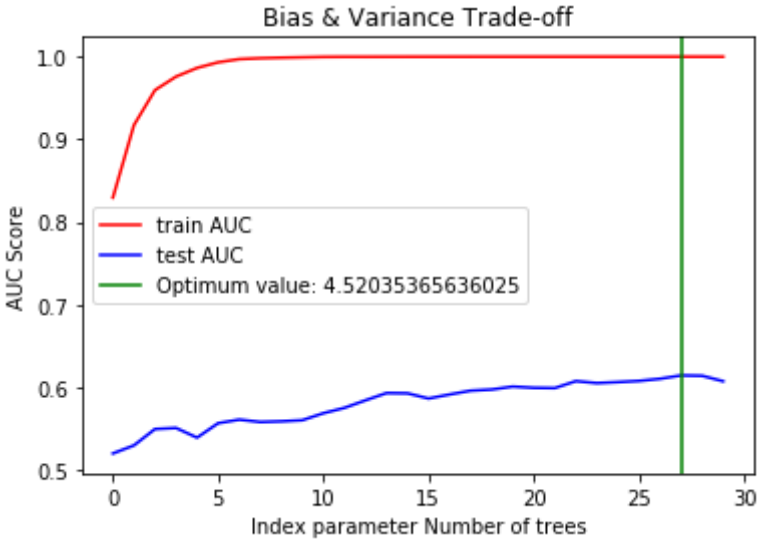
In [90]:
```
# Save the model for model comparison later
Pred2_model_2 = pipeline
```

Pred 2 Model 3

This model trains a random forest. The optimum parameter is chosen using the `Strategy` subsection in `Predictions` section

In [91]:

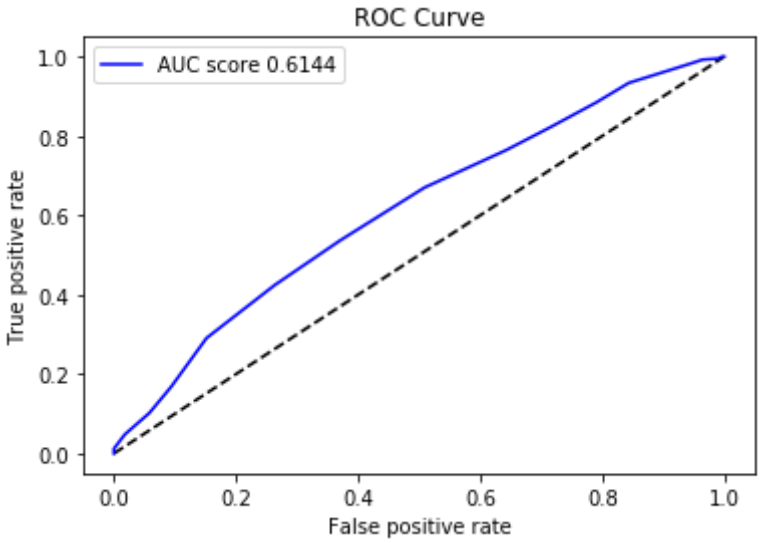
```
from sklearn.preprocessing   import StandardScaler
from sklearn.ensemble        import RandomForestClassifier
# Build pipeline, standardization, random forest
estimators = []
estimators.append(('standardize', StandardScaler(with_mean = False)))        # t
estimators.append(('random_forest',RandomForestClassifier()))  # tell it to choo
pipeline = Pipeline(estimators)
pipeline.set_params(random_forest__random_state=SEED)

# Tune N
AUC_list_train = []
AUC_list_test = []
N_array = []
for N in range(30):
    pipeline.set_params(random_forest__n_estimators=N+1)
    pipeline.fit(Pred2_X_train, Pred2_y_train)
    y_train_pred = pipeline.predict_proba(Pred2_X_train)       # use train set d
    auc_train = roc_auc_score(Pred2_y_train, y_train_pred[:,1])
    AUC_list_train.append(auc_train)
    y_test_pred = pipeline.predict_proba(Pred2_X_test)       # use test set duri
    auc_test = roc_auc_score(Pred2_y_test, y_test_pred[:,1])
    AUC_list_test.append(auc_test)
    N_array.append(N+1)
random_forest = pipeline.named_steps['random_forest']

# Choose the optimum C, such that it maximizes AUC in test and minimizes AUC dif
choose_index_list = list(np.array(AUC_list_test) - np.abs((np.array(AUC_list_tra
choose_index = choose_index_list.index(np.max(choose_index_list))
pipeline.set_params(random_forest__n_estimators = N_array[choose_index])
model = pipeline.named_steps['random_forest']      # capture model so we can use

# Plot AUC score for train and test data set over the different values for param
plot_AUC_train_test(AUC_list_train, AUC_list_test, choose_index, results_C[choos
```



In [92]:

```
# Test the model using the function train_and_test_model
AUC, F1, accuracy = train_and_test_model(pipeline, Pred2_X_train, Pred2_y_train,
```

```
The AUC score for the model with parameter alpha 28.000 is: 1.000

The AUC score for the model using the test data set with parameter alpha 28.000
is: 0.614
```



In [93]:
```
# Save the scores for model comparison later
Pred2_model_3_scores = [AUC, F1, accuracy]
```

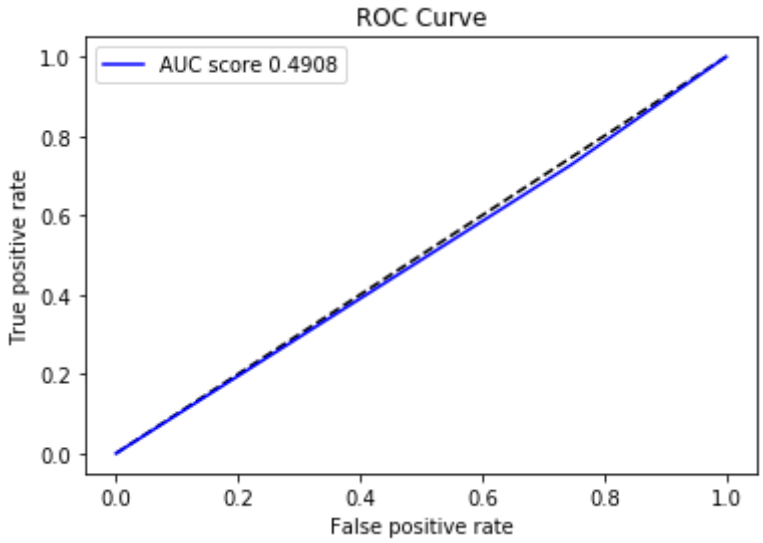
In [94]:
```
# Save the model for model comparison later
Pred2_model_3 = pipeline
```

Pred 2 Baseline model

We define a baseline model. We use a Dummy classifer with `stratified` strategy

In [95]:
```
# Define a dummy classifier as baseline model for model comparison
from sklearn.dummy import DummyClassifier
Pred2_dummy = DummyClassifier(strategy="stratified", random_state = SEED)

# Test the model using the function train_and_test_model
AUC, F1, accuracy = train_and_test_model(Pred2_dummy, Pred2_X_train, Pred2_y_tra

# Save the scores for model comparison later
Pred2_model_dummy_scores = [AUC, F1, accuracy]
```

```
The AUC score for the model with parameter alpha 0.000 is: 0.495

The AUC score for the model using the test data set with parameter alpha 0.000 i
s: 0.491
```



Pred 2 Model Selection

We procceed to select the model in terms of best `AUC scores`. First we display a summary of the different models' scores. The `model 2` seems to have the best score. Let's see if after 5-fold CV, we have the same results.

In [96]:
```
# Save models' scores in a list for model comparison
Pred2_model_scores = [Pred2_model_dummy_scores, Pred2_model_1_scores, Pred2_mode

# Save models' names in a list for model comparison
Pred2_model_names = ['Dummy model', 'Model 1', 'Model 2', 'Model 3']
```

In [97]:
```
# Print summary table for the prediction, it contains all models' scores usingth
print_model_scores(Pred2_model_scores, Pred2_model_names)
```

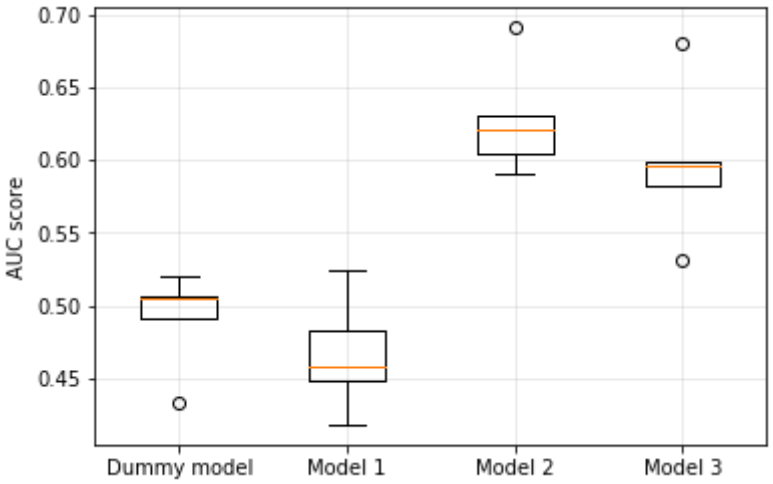
```
-----------------------------------------------------------------
Model name      AUC score       F1 score        Accuracy score
-----------------------------------------------------------------
Dummy model     0.491           0.735           0.608
Model 1         0.611           0.856           0.749
Model 2         0.633           0.857           0.750
Model 3         0.614           0.850           0.745
-----------------------------------------------------------------
```

In [98]:
```
# Create a cross validation data set to perfom K-fold CV
Pred2_Cross_val_df = Pred2_X_test
Pred2_Cross_val_target = Pred2_y_test
```

Looking at the box plot, we can see that the `model 2` is the best one, as it has the highest `AUC` score and it is not overfitting.

In [99]:
```
# Perform 5-fold CV
scores_list = []
from sklearn.model_selection import cross_val_score
for model in [Pred2_dummy, Pred2_model_1, Pred2_model_2, Pred2_model_3]:
    scores = cross_val_score(model, Pred2_Cross_val_df, Pred2_Cross_val_target,
    scores_list.append(scores)
```

```
# Plot CV results using  boxplot
fig, ax = plt.subplots()
pos = np.array(range(len(scores_list))) + 1
ax.boxplot(scores_list, positions=pos)
ax.set_xticklabels(Pred2_model_names)
ax.set_ylabel('AUC score')
ax.grid(alpha = 0.3)
plt.show()
```



In [100…

```
# Save CV results
Pred2_mean_cv = list(np.mean(scores_list, axis = 1))
Pred2_std_cv = list(np.std(scores_list, axis = 1))

# Print CV results using the function print_cv_scores
print_cv_scores(Pred2_mean_cv, Pred2_std_cv, Pred2_model_names)
```

```
----------------------------------------
Model name     Mean AUC       Std AUC
----------------------------------------
Dummy model    0.491          0.031
Model 1        0.466          0.036
Model 2        0.628          0.035
Model 3        0.598          0.048
----------------------------------------
```

We decide to select the `model 2` for this prediction. We select the threshold that optimized the `accuracy` score.

In [101…

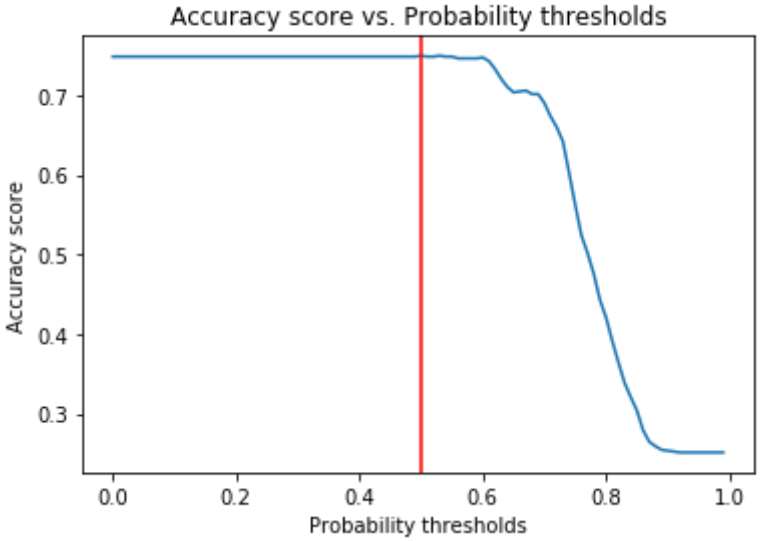
```
# Save winner model
Pred2_winner_model = Pred2_model_2
```

In [102…

```
# Determine optimum probability threshold
Pred2_winner_probability = Pred2_winner_model.predict_proba(Pred2_X_test)[:,1]
Pred2_threshold, Pred2_accuracy, Pred2_thresholds_list = choose_thresholds(Pred2

# Plot optimum probability threshold using the function choose_thresholds
plot_across_thresholds(Pred2_thresholds_list, Pred2_threshold)

# Predict label for test data set
```

```
Pred2_winner_label = pd.DataFrame(Pred2_winner_probability, columns = ['y_proba'
Pred2_winner_label['target'] = Pred2_winner_label['y_proba'].apply(lambda x: 1.0
```

```
The threshold that maximizes the accuracy index is:  0.5 and the accuracy score
is:  0.7497194163860831
```



In [103…

```
# Display accuracy of predicted target
accuracy_score(Pred2_y_test, Pred2_winner_label['target'])
```

Out[103… 0.7497194163860831