Data Science Project 3

- Student Name: Natalya Doris
- Student Pace: Flex / 40 weeks
- Scheduled Project Review Date / Time: Thurs, Sept 16 / 12pm
- Instructor Name: Abhineet Kulkarni

Setup, EDA, Preprocessing

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from datetime import datetime

from sklearn.model_selection import train_test_split, GridSearchCV, \
cross_val_score
from sklearn.impute import SimpleImputer
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, \
BaggingClassifier, ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, recall_score, \
precision_score, confusion_matrix, classification_report, roc_curve, auc, \
average_precision_score
from sklearn.preprocessing import StandardScaler, LabelEncoder, \
MinMaxScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE

import warnings
warnings.filterwarnings('ignore')
```

Load in Data

In [2]:
```
df = pd.read_csv('data/investments_VC.csv', encoding = "ISO-8859-1")
```

Basic Cleaning

In [3]:
```
# getting rid of extra spaces in market and funding_total_usd
df = df.rename(columns={' market ': 'market',
                        ' funding_total_usd ': 'funding_total_usd'})
```

Dropping irrelevant columns:

In [4]:
```
df = df.drop(columns=['permalink', 'homepage_url', 'category_list',
                           'founded_quarter', 'post_ipo_equity',
                           'post_ipo_debt', 'secondary_market'],
                  axis=1)
```

In [5]:
```
# converting to float
df['funding_total_usd'][~df['funding_total_usd'].isnull()] = \
    [float(num.replace(" ", "0").replace(",", "0").replace("-", "0")) \
     for num in df['funding_total_usd'][~df['funding_total_usd'].isnull()]]
```

Dropping duplicates, if any:

In [6]:
```
df = df.drop_duplicates()
```

In [7]:
```
df['status'].value_counts()
```

Out[7]:
```
operating    41829
acquired      3692
closed        2603
Name: status, dtype: int64
```

Exploratory Analysis - Full Dataset

In [8]:
```
data = df
```

In [9]:
```
print(data.info())
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 49439 entries, 0 to 49438
Data columns (total 32 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   name                  49437 non-null  object 
 1   market                45470 non-null  object 
 2   funding_total_usd     49438 non-null  object 
 3   status                48124 non-null  object 
 4   country_code          44165 non-null  object 
 5   state_code            30161 non-null  object 
 6   region                44165 non-null  object 
 7   city                  43322 non-null  object 
 8   funding_rounds        49438 non-null  float64
 9   founded_at            38554 non-null  object 
 10  founded_month         38482 non-null  object 
 11  founded_year          38482 non-null  float64
 12  first_funding_at      49438 non-null  object 
 13  last_funding_at       49438 non-null  object 
 14  seed                  49438 non-null  float64
 15  venture               49438 non-null  float64
 16  equity_crowdfunding   49438 non-null  float64
 17  undisclosed           49438 non-null  float64
 18  convertible_note      49438 non-null  float64
 19  debt_financing        49438 non-null  float64
 20  angel                 49438 non-null  float64
 21  grant                 49438 non-null  float64
 22  private_equity        49438 non-null  float64
 23  product_crowdfunding  49438 non-null  float64
 24  round_A               49438 non-null  float64
 25  round_B               49438 non-null  float64
 26  round_C               49438 non-null  float64
 27  round_D               49438 non-null  float64
 28  round_E               49438 non-null  float64
 29  round_F               49438 non-null  float64
 30  round_G               49438 non-null  float64
 31  round_H               49438 non-null  float64
dtypes: float64(20), object(12)
memory usage: 12.4+ MB
None
```

Out[9]:

	name	market	funding_total_usd	status	country_code	state_code	region
0	#waywire	News	1.705e+09	acquired	USA	NY	New York City
1	&TV Communications	Games	4e+09	operating	USA	CA	Los Angeles
2	'Rock' Your Paper	Publishing	4e+06	operating	EST	NaN	Tallinn
3	(In)Touch Network	Electronics	1.5e+09	operating	GBR	NaN	London
4	-R- Ranch and Mine	Tourism	6e+06	operating	USA	TX	Dallas

5 rows × 32 columns

In [10]:
```
df.describe()
```

Out[10]:

	funding_rounds	founded_year	seed	venture	equity_crowdfunding	u
count	49438.000000	38482.000000	4.943800e+04	4.943800e+04	4.943800e+04	4.94
mean	1.696205	2007.359129	2.173215e+05	7.501051e+06	6.163322e+03	1.3
std	1.294213	7.579203	1.056985e+06	2.847112e+07	1.999048e+05	2.9
min	1.000000	1902.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.0
25%	1.000000	2006.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.0
50%	1.000000	2010.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.0
75%	2.000000	2012.000000	2.500000e+04	5.000000e+06	0.000000e+00	0.0
max	18.000000	2014.000000	1.300000e+08	2.351000e+09	2.500000e+07	2.9

In [11]:
```
data.isnull().sum()
```

Out[11]:
```
name                         2
market                    3969
funding_total_usd            1
status                    1315
country_code              5274
state_code               19278
region                    5274
city                      6117
funding_rounds               1
founded_at               10885
founded_month            10957
founded_year             10957
first_funding_at             1
last_funding_at              1
seed                         1
venture                      1
equity_crowdfunding          1
undisclosed                  1
convertible_note             1
debt_financing               1
angel                        1
grant                        1
private_equity               1
product_crowdfunding         1
round_A                      1
round_B                      1
round_C                      1
round_D                      1
round_E                      1
round_F                      1
round_G                      1
round_H                      1
dtype: int64
```

In [12]:
```
data['founded_at'].head()
```

Out[12]:
```
0    2012-06-01
1           NaN
2    2012-10-26
3    2011-04-01
4    2014-01-01
Name: founded_at, dtype: object
```

In [13]:
```
data['region'].value_counts()
```

Out[13]:
```
SF Bay Area      6804
New York City    2577
Boston           1837
London           1588
Los Angeles      1389
                 ...
Stirling            1
Igualada            1
MUS - Other         1
Cheadle Hulme       1
Reigate             1
Name: region, Length: 1089, dtype: int64
```

In [14]:
```
data['city'].value_counts()
```

Out[14]:
```
San Francisco    2615
New York         2334
London           1257
Palo Alto         597
Austin            583
                 ...
Tel Mond            1
Ra'ananna           1
Southwark           1
Benson              1
Alvorada            1
Name: city, Length: 4188, dtype: int64
```

In [15]:
```
data['state_code'].value_counts()
```

Out[15]:
```
CA    9917
NY    2914
MA    1969
TX    1466
WA     974
      ...
MB      13
AK      12
NB       8
SK       4
PE       2
Name: state_code, Length: 61, dtype: int64
```

In [16]:
```
data['country_code'].value_counts()
```

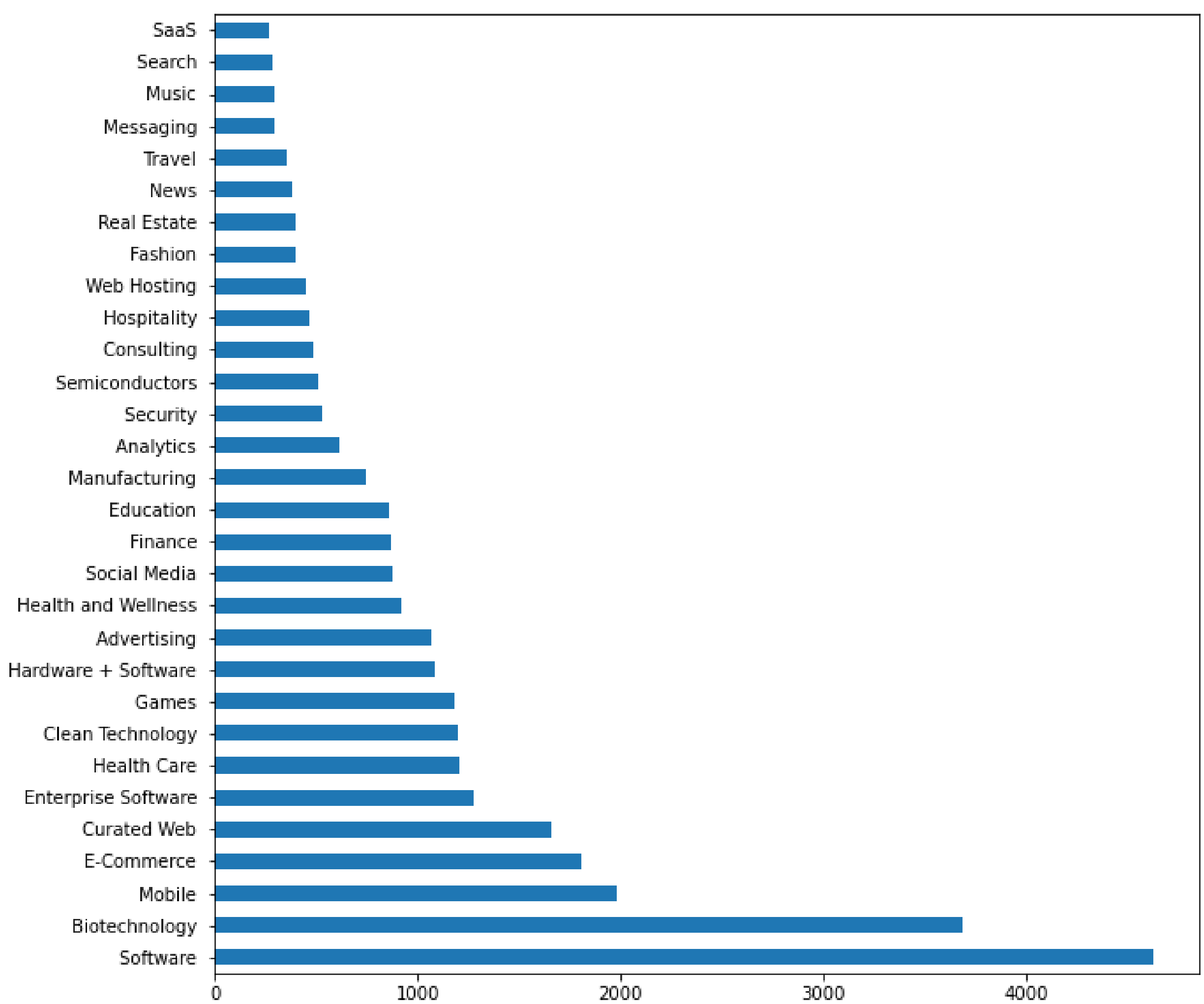
Out[16]:
```
USA    28793
GBR     2642
CAN     1405
CHN     1239
DEU      968
        ...
SOM        1
OMN        1
CIV        1
TTO        1
MOZ        1
Name: country_code, Length: 115, dtype: int64
```

In [17]:
```
# inspecting market feature - lots of catgories
print("Number of unique markets: ", len(set(data['market'])))
print("Markets with more than 200 companies: ",
      str(sum(data['market'].value_counts() >= 200)))
```

```
Number of unique markets:  754
Markets with more than 200 companies:  39
```

In [18]:
```
fig, ax = plt.subplots(figsize = (10,10))
data['market'].value_counts()[:30].plot(kind='barh')
```

Out[18]: <AxesSubplot:>



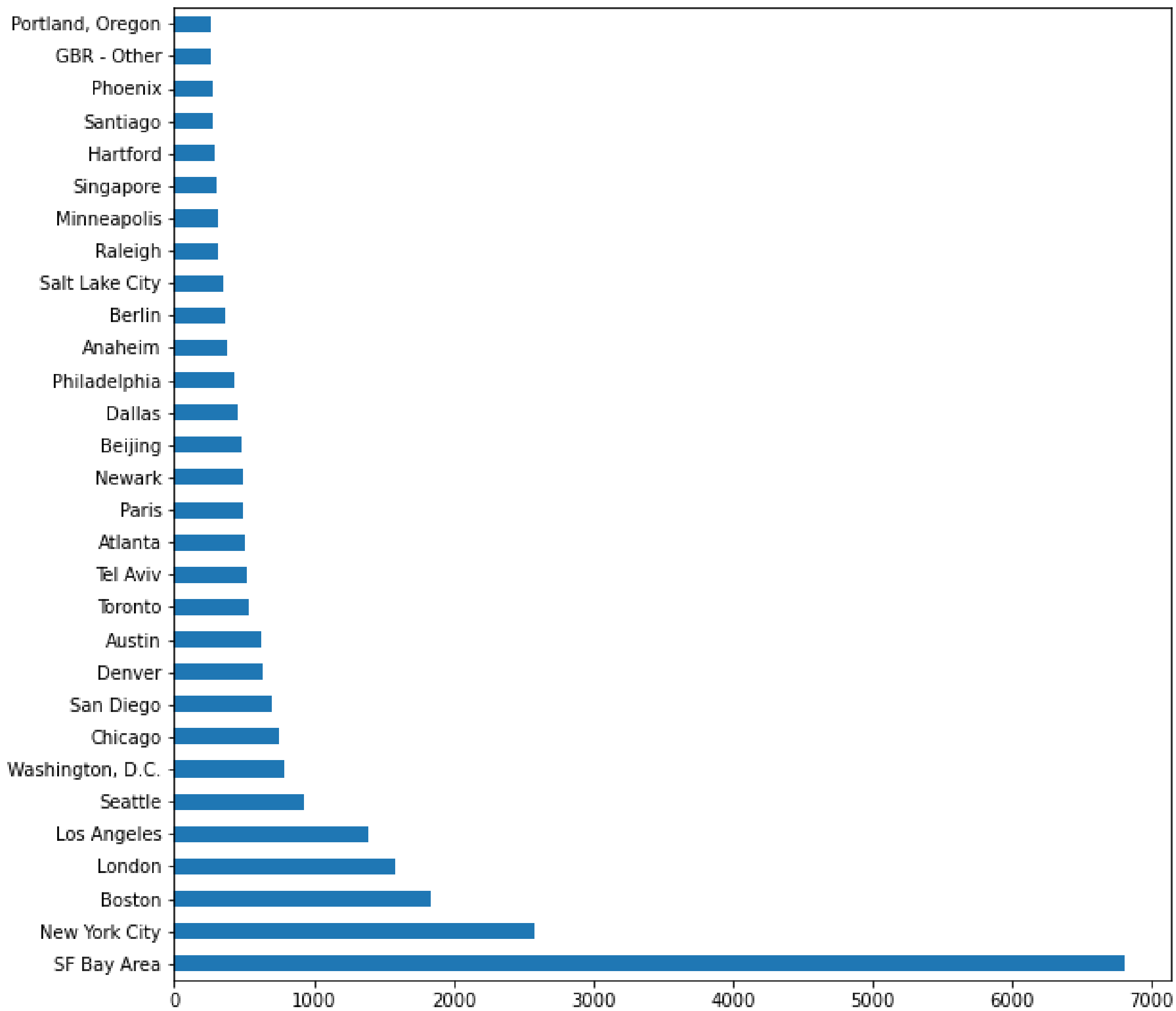
In [19]:
```
# same deal with region
print("Number of unique regions: ", len(set(data['region'])))
```

```
print("Regions with more than 200 companies: ",
      str(sum(data['region'].value_counts() >= 200)))
```

```
Number of unique regions:  1090
Regions with more than 200 companies:  44
```

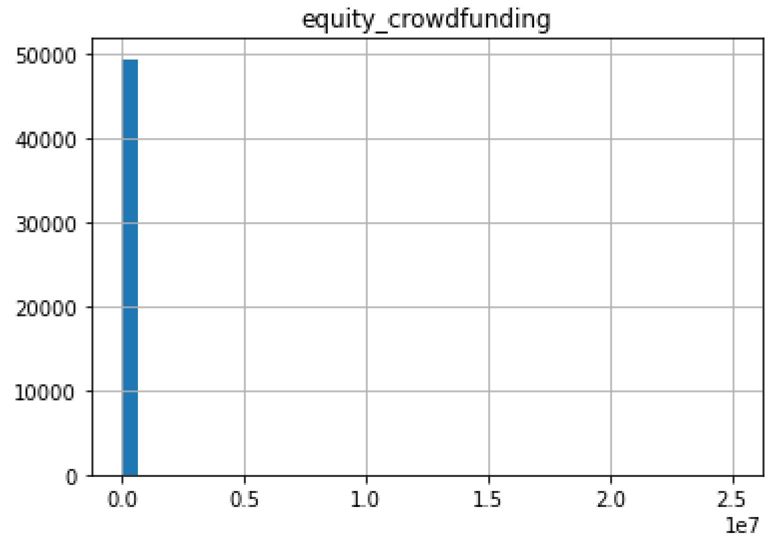
In [20]:
```
fig, ax = plt.subplots(figsize = (10,10))
data['region'].value_counts()[:30].plot(kind='barh')
```

Out[20]: <AxesSubplot:>



In [21]:
```
data.hist('equity_crowdfunding', bins=40)
# there are some very skewed columns in the funding area
```

Out[21]:
```
array([[<AxesSubplot:title={'center':'equity_crowdfunding'}>]],
      dtype=object)
```



Preprocessing & Feature Engineering

In [22]:
```
data.columns
```

Out[22]:
```
Index(['name', 'market', 'funding_total_usd', 'status', 'country_code',
       'state_code', 'region', 'city', 'funding_rounds', 'founded_at',
       'founded_month', 'founded_year', 'first_funding_at', 'last_funding_at',
       'seed', 'venture', 'equity_crowdfunding', 'undisclosed',
       'convertible_note', 'debt_financing', 'angel', 'grant',
       'private_equity', 'product_crowdfunding', 'round_A', 'round_B',
       'round_C', 'round_D', 'round_E', 'round_F', 'round_G', 'round_H'],
      dtype='object')
```

Missing Values

- I am removing rows with missing status or company name.
- I am also dropping rows with a missing founded_year because this feature has high importance in many of the models & thus would prefer not to impute at risk of skewing the model. I tried both dropping and keeping the missing founded_years (imputed). The proportion of companies in each status category & model results are fairly similar with & without imputing the median of year so I am comfortable dropping these rows.

In [23]:
```
data = data.dropna(subset=['status', 'name', 'founded_year'])
data.isnull().sum()
```

Out[23]:
```
name                         0
market                    1801
funding_total_usd            0
status                       0
country_code              2936
state_code               13332
region                    2936
city                      3357
funding_rounds               0
founded_at                   0
founded_month                0
founded_year                 0
first_funding_at             0
last_funding_at              0
seed                         0
venture                      0
equity_crowdfunding          0
undisclosed                  0
convertible_note             0
debt_financing               0
angel                        0
grant                        0
private_equity               0
product_crowdfunding         0
round_A                      0
round_B                      0
round_C                      0
round_D                      0
round_E                      0
round_F                      0
round_G                      0
round_H                      0
dtype: int64
```

In [24]:
```
len(data)
```

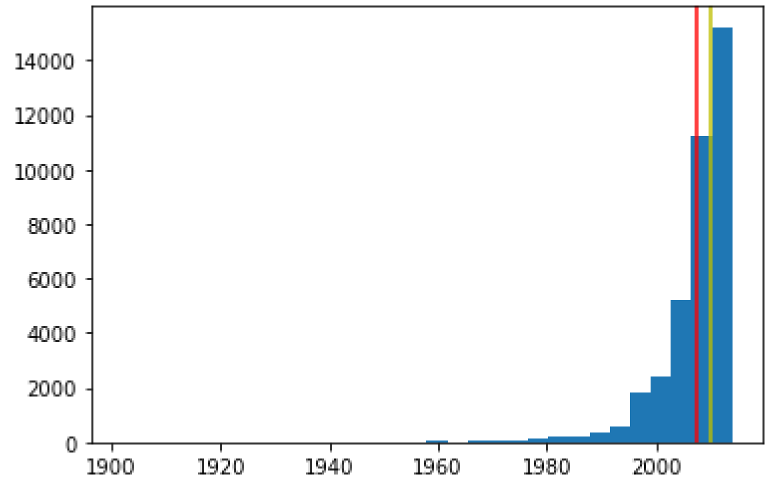
Out[24]:
```
37563
```

In [25]:
```
# filling categoricals
data = data.fillna(value={'market': 'other', 'country_code': 'other',
                          'region': 'other', 'city': 'other',
                          'state_code': 'other'})
```

In [26]:
```
# distribution of non NA years
plt.hist(data['founded_year'][~data['founded_year'].isnull()], bins=30)
plt.axvline(x=np.nanmean(data['founded_year']), color='r')
plt.axvline(x=np.nanmedian(data['founded_year']), color='y')

p10 = np.percentile(data['founded_year'][~data['founded_year'].isnull()], 10)
print("10th percentile of founded_year: ", p10)

#10th percentile year is 2000, we will subset for startups founded on or after
print("Num Rows before 2000: ", len(data[data['founded_year'] < 2000]))
data = data[data['founded_year'] >= 2000].reset_index(drop=True)
len(data)
```

```
10th percentile of founded_year:  2000.0
Num Rows before 2000:  3518
```

Out[26]:
```
34045
```



In [27]:
```
len(df[(df['founded_year'] < 2000) & (df['status'] != 'operating')])
```

Out[27]: 838

Date data - convert to datetime:

In [28]:
```
data['founded_at'] = [datetime.strptime(day, '%Y-%m-%d') for day \
                      in data['founded_at'][~data['founded_at'].isnull()]]
data['first_funding_at'] = [datetime.strptime(day, '%Y-%m-%d').date() for day \
                            in data['first_funding_at']]
data['last_funding_at'] = [datetime.strptime(day, '%Y-%m-%d').date() for day \
                           in data['last_funding_at']]
data['founded_month'] = [datetime.strptime(mth, "%Y-%m").month for mth in data[
```

Fill year & founded_at with simple imputer (only if not removing NaN rows). Since the data is skewed (see plot above), we will use median rather than mean.

In [29]:
```
# uncomment the below to impute founded_year with mean, only if we are not
# dropping nans for this feature

#imp_median = SimpleImputer(missing_values=np.nan, strategy='median')

#data['founded_year'] = imp_median.fit_transform(data[['founded_year']])
#data['founded_at'] = imp_median.fit_transform(data[['founded_at']])
```

In [30]:
```
data['founded_at'] = pd.to_datetime(data['founded_at'])
data['founded_year'] = [day.year for day in data['founded_at']]
```

In [31]:
```
data['founded_at'].value_counts()
```

Out[31]:
```
2012-01-01    2100
2011-01-01    2096
2010-01-01    1810
2009-01-01    1561
2013-01-01    1535
              ...
2004-04-25       1
2009-08-12       1
2014-08-04       1
2002-08-02       1
2010-11-06       1
Name: founded_at, Length: 2935, dtype: int64
```

In [32]:
```
data.isna().sum()
```

Out[32]:
```
name                    0
market                  0
funding_total_usd       0
status                  0
country_code            0
state_code              0
region                  0
city                    0
funding_rounds          0
founded_at              0
founded_month           0
founded_year            0
first_funding_at        0
last_funding_at         0
seed                    0
venture                 0
equity_crowdfunding     0
undisclosed             0
convertible_note        0
debt_financing          0
angel                   0
grant                   0
private_equity          0
product_crowdfunding    0
round_A                 0
round_B                 0
round_C                 0
round_D                 0
round_E                 0
round_F                 0
round_G                 0
round_H                 0
dtype: int64
```

Basic data cleaning:

In [33]:
```
# getting rid of extra spaces in market, city, state code, region
data['market'] = [x.strip() for x in data['market']]
data['country_code'] = [x.strip() for x in data['country_code']]
data['state_code'] = [x.strip() for x in data['state_code']]
data['region'] = [x.strip() for x in data['region']]
data['city'] = [x.strip() for x in data['city']]
```

In [34]:
```
data.dtypes
```

Out[34]:
```
name                              object
market                            object
funding_total_usd                 object
status                            object
country_code                      object
state_code                        object
region                            object
city                              object
funding_rounds                   float64
founded_at                datetime64[ns]
founded_month                      int64
founded_year                       int64
first_funding_at                  object
last_funding_at                   object
seed                             float64
venture                          float64
equity_crowdfunding              float64
undisclosed                      float64
convertible_note                 float64
debt_financing                   float64
angel                            float64
grant                            float64
private_equity                   float64
product_crowdfunding             float64
round_A                          float64
round_B                          float64
round_C                          float64
round_D                          float64
round_E                          float64
round_F                          float64
round_G                          float64
round_H                          float64
dtype: object
```

In [35]:
```
# need to convert this data type to integer
data['funding_total_usd'].value_counts()
```

Out[35]:
```
0.000000e+00    5632
1.000000e+09     627
1.000000e+08     582
5.000000e+08     573
4.000000e+06     466
                ...
3.012099e+11       1
1.045077e+08       1
1.000108e+12       1
1.609205e+09       1
2.068042e+11       1
Name: funding_total_usd, Length: 10560, dtype: int64
```

In [36]:
```
# data['funding_total_usd'] = [float(num.replace(" ", "0").replace(",", "0").\
#                                     replace("-", "0"))
#                               for num in data['funding_total_usd']]
# data['funding_total_usd'].dtypes
```

Feature Engineering

In [37]:
```
# creating column that labels country as domestic or international
data['international'] = [0 if country=='USA' else 1 for country in data['countr
```

In [38]:
```
# creating temporary columns to aid in calculation of time to first funding

data['founded_at_temp'] = [day.date() for day in data['founded_at']]
data['founded_at_temp'] = pd.to_datetime(data['founded_at_temp'],
                                         format = '%Y-%m-%d')

data['first_funding_at_temp'] = pd.to_datetime(data['first_funding_at'],
                                               format = '%Y-%m-%d',
                                               errors='coerce')
data['last_funding_at_temp'] = pd.to_datetime(data['last_funding_at'],
                                              format = '%Y-%m-%d',
                                              errors='coerce')

data['time_to_first_funding'] = (data['first_funding_at_temp'] - \
                                 data['founded_at_temp']) / pd.Timedelta(days=3

data['time_first_to_last_funding'] = (data['last_funding_at_temp'] - \
                                      data['first_funding_at_temp']) / pd.Timec
```

In [39]:
```
# checking for nulls
print(data['first_funding_at'][data['time_to_first_funding'].isnull()])
print(data['last_funding_at'][data['time_first_to_last_funding'].isnull()])
```

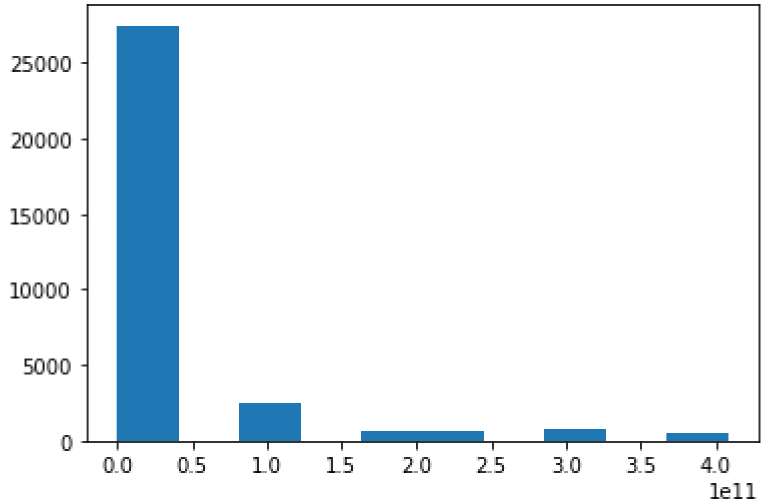
```
1030     0020-06-14
4514     0019-11-20
9863     0201-01-01
20287    0007-05-13
21784    0001-05-14
Name: first_funding_at, dtype: object
1030     2013-06-01
4514     2013-04-01
9863     0201-01-01
20287    2014-09-25
21784    0001-05-14
Name: last_funding_at, dtype: object
```

In [40]:
```
# dropping these
data = data.dropna(subset=['time_to_first_funding'])
```

OPTIONAL - Outliers

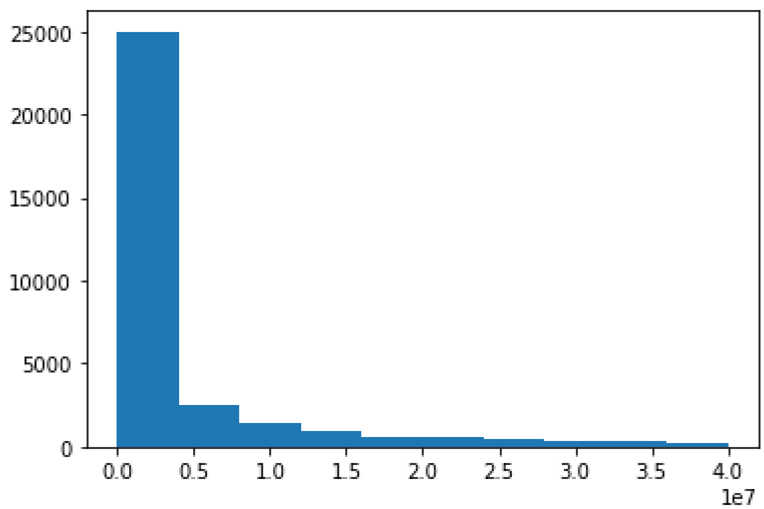
In [41]:
```
# Funding total USD
print(np.percentile(data['funding_total_usd'], 95))

plt.hist(data['funding_total_usd'][data['funding_total_usd'] < 408206869399.0])
print(len(data[data['funding_total_usd'] < 408206869399.0]))
print(len(data))
```

```
408206869399.0
32338
34040
```



In [42]:
```
np.percentile(data['venture'], 95)

plt.hist(data['venture'][data['venture'] < 40000000.0])
print(len(data[data['venture'] < 40000000.0]))
print(len(data[(data['venture'] < 40000000.0) &
               (data['funding_total_usd'] < 408206869399.0)]))
print(len(data))
```

```
32301
31990
34040
```



In [43]:
```
### THIS DOESN'T HELP THE MODELS, SO I COMMENTED THIS OUT
```

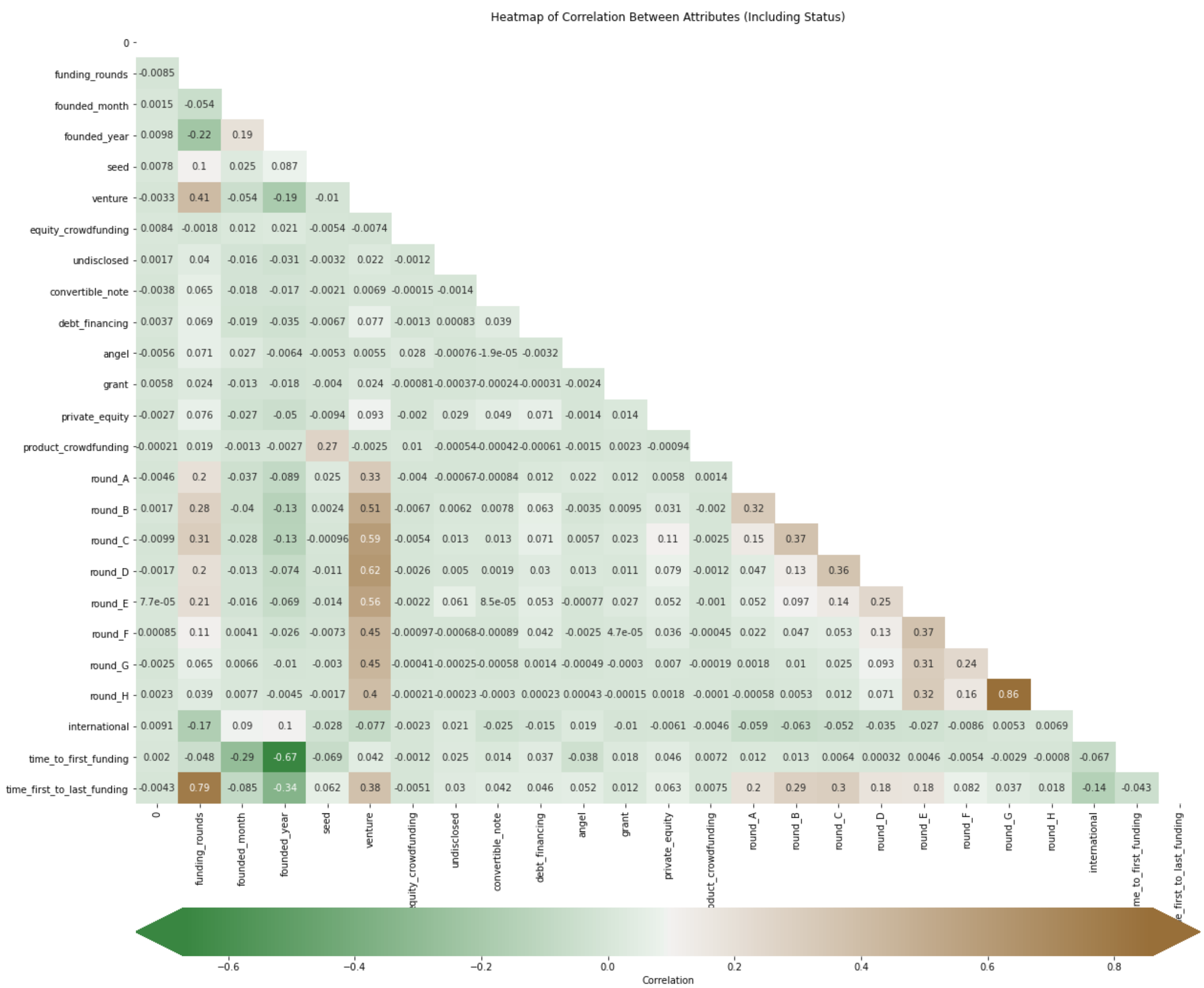
In [44]:
```
# uncomment to remove outliers

#data = data[(data['venture'] < 40000000.0) &
#             (data['funding_total_usd'] < 408206869399.0)].reset_index(drop=Tru
#len(data)
```

Correlations

Based on the below, the strongest correlations occur between debt_financing and funding_total_usd, round_H and round_G, followed by venture and all of the rounds of funding

In [45]:
```
# with status
encoder = LabelEncoder()
heatmap_data = pd.concat([pd.Series(encoder.fit_transform(data['status'])),
                          data.drop('status', axis=1)], axis=1)

# without status
#heatmap_data = data.drop('status', axis=1)

h_corr = heatmap_data.corr()
fig, ax = plt.subplots(figsize=(20, 20))
mask = np.triu(np.ones_like(h_corr, dtype=bool))
cmap = sns.diverging_palette(130, 50, as_cmap=True)
cbar_kws = {'label': 'Correlation', 'orientation': 'horizontal',
            'pad': .1, 'extend': 'both'}
sns.heatmap(data=h_corr, mask=mask, ax=ax, annot=True, cbar_kws=cbar_kws, cmap=
ax.set_title('Heatmap of Correlation Between Attributes (Including Status)')
```

Out[45]:
```
Text(0.5, 1.0, 'Heatmap of Correlation Between Attributes (Including Status)')
```



In [46]:
```
#data_corr=data.drop(columns=['status'], axis=1).corr()
corr = h_corr.abs().stack().reset_index().sort_values(0, ascending=False)
corr['pairs'] = list(zip(corr.level_0, corr.level_1))
corr.set_index(['pairs'], inplace = True)
corr.drop(columns=['level_1', 'level_0'], inplace = True)
```

```
# cc for correlation coefficient
corr.columns = ['cc']
corr.drop_duplicates(inplace=True)

corr[(corr['cc'] > 0.7) & (corr['cc'] < 1)]
```

Out[46]:

pairs	cc
(round_H, round_G)	0.859849
(funding_rounds, time_first_to_last_funding)	0.793787

In [47]:

```
# dropping temp columns
data = data.drop(columns=['founded_at', 'first_funding_at', 'last_funding_at',
                          'first_funding_at_temp', 'last_funding_at_temp',
                          'founded_at_temp'], axis=1).reset_index(drop=True)

# creating csv file to work from
data.to_csv('data/final_working_data.csv')
```

In [48]:

```
# dropping columns with correlation coefficient greater than 0.7
data_uncorr = data.drop(columns=['round_H', 'time_first_to_last_funding'],
                        axis=1).reset_index(drop=True)

#renaming full dataset
data_full = data
```

OPTIONAL - Binary Representation of Funding Rounds

In [49]:

```
# data['had_round_A'] = [0 if x==0 else 1 for x in data['round_A']]
# data['had_round_B'] = [0 if x==0 else 1 for x in data['round_B']]
# data['had_round_C'] = [0 if x==0 else 1 for x in data['round_C']]
# data['had_round_D'] = [0 if x==0 else 1 for x in data['round_D']]
# data['had_round_E'] = [0 if x==0 else 1 for x in data['round_E']]
# data['had_round_F'] = [0 if x==0 else 1 for x in data['round_F']]
# data['had_round_G'] = [0 if x==0 else 1 for x in data['round_G']]
# data['had_venture'] = [0 if x==0 else 1 for x in data['venture']]
# data['had_seed'] = [0 if x==0 else 1 for x in data['seed']]
# data['had_eq_crowdfunding'] = [0 if x==0 else 1 for x in data['equity_crowdfu
# data['had_pd_crowdfunding'] = [0 if x==0 else 1 for x in data['product_crowdi
# data['had_angel'] = [0 if x==0 else 1 for x in data['angel']]
# data['had_grant'] = [0 if x==0 else 1 for x in data['grant']]
# data['had_pe'] = [0 if x==0 else 1 for x in data['private_equity']]
# data['had_convert'] = [0 if x==0 else 1 for x in data['convertible_note']]
```

Functions

train_test_preprocess

In [50]:

```
'''
Function to perform train_test_split and necessary preprocessing / scaling
'''

def train_test_preprocess(X, y):
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

    # check that there are the same number of rows in X as values in y
    assert X_train.shape[0] == y_train.shape[0]

    # Categorizing features in preparation for scaling / encoding
    X_train_cat_eng = X_train.select_dtypes(include=['int64']).reset_index(drop
    X_test_cat_eng = X_test.select_dtypes(include=['int64']).reset_index(drop=T

    X_train_cont = X_train.select_dtypes(exclude=['object','int64']).reset_inde
    X_test_cont = X_test.select_dtypes(exclude=['object','int64']).reset_index(

    cat_columns = ['market', 'region']
    cat_train = X_train[cat_columns].reset_index(drop=True)
    cat_test = X_test[cat_columns].reset_index(drop=True)

    # Scale continuous variables using Min Max Scaler:
    scaler = MinMaxScaler() # instantiate MinMaxScaler

    ## TRAIN
    # Fit and transform X_train
    X_train_cont_scaled = scaler.fit_transform(X_train_cont)
    X_train_cont_scaled = pd.DataFrame(X_train_cont_scaled, columns=X_train_con

    # One hot encode categoricals
    ohe = OneHotEncoder(handle_unknown = 'ignore')
    encoded_train = ohe.fit_transform(cat_train).toarray()
    X_train_cat = pd.DataFrame(encoded_train, columns=ohe.get_feature_names(cat

    # Putting it all together:
    X_train_processed = pd.concat([X_train_cat, X_train_cont, X_train_cat_eng],
    X_train_scaled = pd.concat([X_train_cat, X_train_cont_scaled, X_train_cat_e

    ## TEST
    # Scale continuous features
    X_test_cont_scaled = scaler.transform(X_test_cont)
    X_test_cont_scaled = pd.DataFrame(X_test_cont_scaled, columns=X_test_cont.c

    # One hot encoding categoricals
    encoded_test = ohe.transform(cat_test).toarray()
    X_test_cat = pd.DataFrame(encoded_test, columns=ohe.get_feature_names(cat_t

    # Putting it all together
    X_test_scaled = pd.concat([X_test_cat, X_test_cont_scaled, X_test_cat_eng],
    X_test_processed = pd.concat([X_test_cat, X_test_cont, X_test_cat_eng], axi

    return X_train_processed, X_train_scaled, X_test_processed, X_test_scaled,
```

print_scores

In [51]:

```
'''
Function to print relevant scoring metrics
'''

def print_scores(y_train, y_hat_train, y_test, y_hat_test, binary=True):
    if binary:
        print('Training Recall: ',
              recall_score(y_train, y_hat_train))
        print('Testing Recall: ',
```

```
            recall_score(y_test, y_hat_test))
        print('\n')
        print('Training F1: ',
            f1_score(y_train, y_hat_train))
        print('Testing F1: ',
            f1_score(y_test, y_hat_test))
        print('\n')
        false_positive_rate, true_positive_rate, thresholds = \
        roc_curve(y_test, y_hat_test)
        roc_auc = auc(false_positive_rate, true_positive_rate)
        print('ROC AUC: ', roc_auc)
        print('PR AUC: ', average_precision_score(y_test, y_hat_test))
        print('\n')

    else:
        print('Training Recall (weighted avg): ',
            recall_score(y_train, y_hat_train, average='weighted'))
        print('Testing Recall (weighted avg): ',
            recall_score(y_test, y_hat_test, average='weighted'))
        print('\n')
        print('Training Recall (macro avg): ',
            recall_score(y_train, y_hat_train, average='macro'))
        print('Testing Recall (macro avg): ',
            recall_score(y_test, y_hat_test, average='macro'))
        print('\n')
        print('Training F1-Score (weighted avg): ',
            f1_score(y_train, y_hat_train, average='weighted'))
        print('Testing F1-Score (weighted avg): ',
            f1_score(y_test, y_hat_test, average='weighted'))
        print('\n')
        print('Training F1-Score (macro avg): ',
            f1_score(y_train, y_hat_train, average='macro'))
        print('Testing F1-Score (macro avg): ',
            f1_score(y_test, y_hat_test, average='macro'))
        print('\n')
        print('Testing Recall (failure class): ',
            recall_score(y_test, y_hat_test, average=None, labels=[1]))
        print('\n')

    print('Training Accuracy: ', accuracy_score(y_train, y_hat_train))
    print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test))
```

return_scores

In [52]:

```
'''
Function that stores relevant scoring metrics
'''

def return_scores(y_train, y_hat_train, y_test, y_hat_test):
    r_train = recall_score(y_train, y_hat_train)
    r_test = recall_score(y_test, y_hat_test)

    f1_train = f1_score(y_train, y_hat_train)
    f1_test = f1_score(y_test, y_hat_test)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_h
    roc_auc = auc(false_positive_rate, true_positive_rate)

    pr_auc = average_precision_score(y_test, y_hat_test)
```

```
    ac_train = accuracy_score(y_train, y_hat_train)
    ac_test = accuracy_score(y_test, y_hat_test)

    return r_train, r_test, f1_train, f1_test, ac_train, ac_test, roc_auc, pr_a
```

train_test_check

In [53]:
```
'''
Function that checks new train & test splits for proper shape
'''

def train_test_check(X_train_processed, X_train_scaled, X_test_processed,
                     X_test_scaled, y_train, y_test):

    assert X_train_processed.shape[0] == y_train.shape[0]
    assert X_train_scaled.shape[0] == y_train.shape[0]

    assert X_test_processed.shape[0] == y_test.shape[0]
    assert X_test_scaled.shape[0] == y_test.shape[0]

    print("There are {} features in train set".format(len(X_train_processed.col
    print("There are {} features in test set".format(len(X_test_processed.colum
    print('\n')

    print("There are {} features in train set (scaled)".format(len(X_train_scal
    print("There are {} features in test set (scaled)".format(len(X_test_scalec
    print('\n')

    print(f"y_train is a Series with {y_train.shape[0]} values")
    print('\n')
    print("target breakdown: ", y_train.value_counts(normalize=True))

    display(X_train_processed.head())
    display(X_train_scaled.head())
```

correlation_check

In [54]:
```
'''
Function that checks for excessive correlations across features
'''
def correlation_check(X_train_processed):

    df_corr=X_train_processed.corr()

    df = df_corr.abs().stack().reset_index().sort_values(0, ascending=False)
    df['pairs'] = list(zip(df.level_0, df.level_1))
    df.set_index(['pairs'], inplace = True)
    df.drop(columns=['level_1', 'level_0'], inplace = True)

    # cc for correlation coefficient
    df.columns = ['cc']
    df.drop_duplicates(inplace=True)

    display(df[(df.cc>.5) & (df.cc<1)])
```

Acquired or Closed Subset

In [55]:
```
data_ac = df[df['status'] != 'operating']
```

EDA / Preprocessing

In [56]:
```
data_ac.isnull().sum()
```

Out[56]:
```
name                      2
market                  611
funding_total_usd         1
status                 1315
country_code            835
state_code             2582
region                  835
city                    951
funding_rounds            1
founded_at             1720
founded_month          1725
founded_year           1725
first_funding_at          1
last_funding_at           1
seed                      1
venture                   1
equity_crowdfunding       1
undisclosed               1
convertible_note          1
debt_financing            1
angel                     1
grant                     1
private_equity            1
product_crowdfunding      1
round_A                   1
round_B                   1
round_C                   1
round_D                   1
round_E                   1
round_F                   1
round_G                   1
round_H                   1
dtype: int64
```

In [57]:
```
len(data_ac)
```

Out[57]:
```
7610
```

In [58]:
```
data_ac = data_ac.dropna(subset=['status', 'name'])
len(data_ac)
```

Out[58]:
```
6294
```

In [59]:
```
data_ac.dtypes
```

Out[59]:
```
name                      object
market                    object
funding_total_usd         object
status                    object
country_code              object
state_code                object
region                    object
city                      object
funding_rounds           float64
founded_at                object
founded_month             object
founded_year             float64
first_funding_at          object
last_funding_at           object
seed                     float64
venture                  float64
equity_crowdfunding      float64
undisclosed              float64
convertible_note         float64
debt_financing           float64
angel                    float64
grant                    float64
private_equity           float64
product_crowdfunding     float64
round_A                  float64
round_B                  float64
round_C                  float64
round_D                  float64
round_E                  float64
round_F                  float64
round_G                  float64
round_H                  float64
dtype: object
```

In [60]:
```
# converting to float
data_ac['funding_total_usd'] = [float(num) for num in data_ac['funding_total_us
```

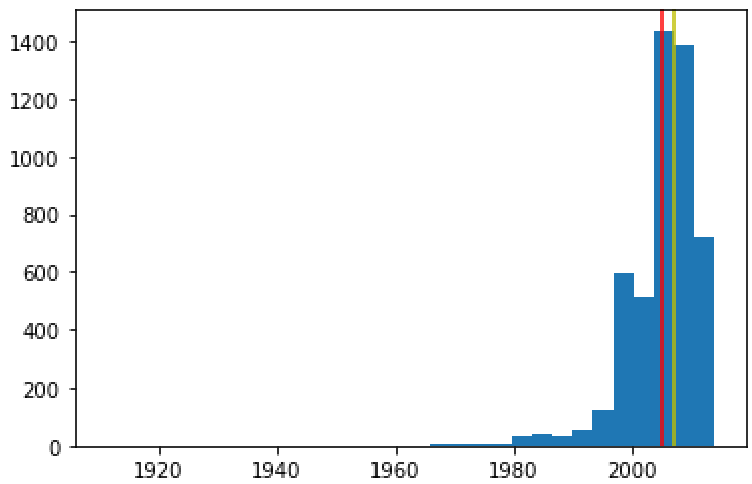
In [61]:
```
data_ac['status'].value_counts(normalize=True)
```

Out[61]:
```
acquired    0.58659
closed      0.41341
Name: status, dtype: float64
```

In [62]:
```
# filling categoricals
data_ac = data_ac.fillna(value={'market': 'other', 'country_code': 'other',
                                'region': 'other', 'city': 'other',
                                'state_code': 'other'})
```

In [63]:
```
# distribution of non NA years
plt.hist(data_ac['founded_year'][~data_ac['founded_year'].isnull()], bins=30)
plt.axvline(x=np.nanmean(data_ac['founded_year']), color='r')
plt.axvline(x=np.nanmedian(data_ac['founded_year']), color='y')

p1 = np.percentile(data_ac['founded_year'][~data_ac['founded_year'].isnull()],
print("1st percentile of founded_year: ", p1)

#1980 looks like a good cutoff point
print("Num Rows before 1980: ", len(data_ac[data_ac['founded_year'] < 1980]))
data_ac = data_ac[data_ac['founded_year'] >= 1980].reset_index(drop=True)
```

```
1st percentile of founded_year:  1982.0
Num Rows before 1980:  35
```



In [64]:
```
data_ac['founded_at'] = [datetime.strptime(day, '%Y-%m-%d') for day \
                         in data_ac['founded_at'][~data_ac['founded_at'].isnul
data_ac['first_funding_at'] = [datetime.strptime(day, '%Y-%m-%d').date() for da
                               in data_ac['first_funding_at']]
data_ac['last_funding_at'] = [datetime.strptime(day, '%Y-%m-%d').date() for day
                              in data_ac['last_funding_at']]
data_ac['founded_month'] = [datetime.strptime(mth, "%Y-%m").month for mth in da
```

In [65]:
```
# imputing median for day, then pulling founded year and month from that
imp_median = SimpleImputer(missing_values=np.nan, strategy='median')

data_ac['founded_at'] = imp_median.fit_transform(data_ac[['founded_at']])

data_ac['founded_at'] = pd.to_datetime(data_ac['founded_at'])
data_ac['founded_year'] = [day.year for day in data_ac['founded_at']]
data_ac['founded_month'] = [day.month for day in data_ac['founded_at']]
```

In [66]:
```
# creating column that labels country as domestic or international
data_ac['international'] = [0 if country=='USA' else 1 for country in data_ac['
```

In [67]:
```
# creating temporary columns to aid in calcuation of time to first funding

data_ac['founded_at_temp'] = [day.date() for day in data_ac['founded_at']]
data_ac['founded_at_temp'] = pd.to_datetime(data_ac['founded_at_temp'],
                                            format = '%Y-%m-%d')

data_ac['first_funding_at_temp'] = pd.to_datetime(data_ac['first_funding_at'],
                                                  format = '%Y-%m-%d',
                                                  errors='coerce')
data_ac['last_funding_at_temp'] = pd.to_datetime(data_ac['last_funding_at'],
                                                 format = '%Y-%m-%d',
                                                 errors='coerce')

data_ac['time_to_first_funding'] = (data_ac['first_funding_at_temp'] - \
                                    data_ac['founded_at_temp']) / pd.Timedelta(

data_ac['time_first_to_last_funding'] = (data_ac['last_funding_at_temp'] - \
                                         data_ac['first_funding_at_temp']) / pd
```

In [68]:
```
# checking for nulls
```

```
print(data_ac['first_funding_at'][data_ac['time_to_first_funding'].isnull()])
print(data_ac['last_funding_at'][data_ac['time_first_to_last_funding'].isnull()
```

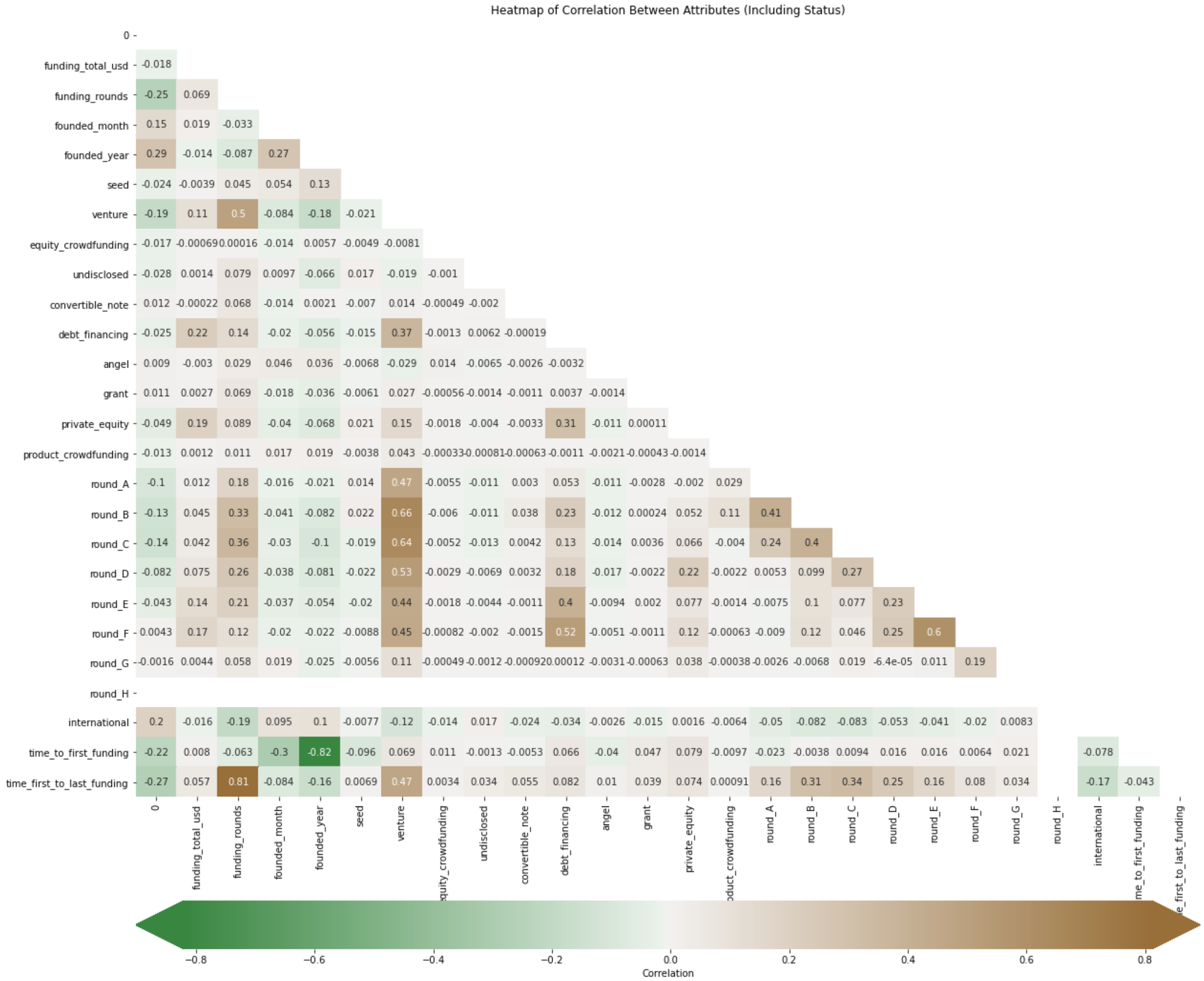
```
3697    0011-11-14
Name: first_funding_at, dtype: object
3697    2012-07-24
Name: last_funding_at, dtype: object
```

In [69]:
```
data_ac = data_ac.dropna(subset=['time_to_first_funding']).reset_index(drop=Tru
```

In [70]:
```
# getting rid of extra spaces

data_ac['market'] = [x.strip() for x in data_ac['market']]
data_ac['country_code'] = [x.strip() for x in data_ac['country_code']]
data_ac['state_code'] = [x.strip() for x in data_ac['state_code']]
data_ac['region'] = [x.strip() for x in data_ac['region']]
data_ac['city'] = [x.strip() for x in data_ac['city']]
```

In [71]:
```
# correlation check
# with status
encoder = LabelEncoder()
heatmap_data = pd.concat([pd.Series(encoder.fit_transform(data_ac['status'])),
                          data_ac.drop('status', axis=1)], axis=1)

# without status
#heatmap_data = data.drop('status', axis=1)

h_corr = heatmap_data.corr()
fig, ax = plt.subplots(figsize=(20, 20))
mask = np.triu(np.ones_like(h_corr, dtype=bool))
cmap = sns.diverging_palette(130, 50, as_cmap=True)
cbar_kws = {'label': 'Correlation', 'orientation': 'horizontal',
            'pad': .1, 'extend': 'both'}
sns.heatmap(data=h_corr, mask=mask, ax=ax, annot=True, cbar_kws=cbar_kws, cmap=
ax.set_title('Heatmap of Correlation Between Attributes (Including Status)')
```

Out[71]:
```
Text(0.5, 1.0, 'Heatmap of Correlation Between Attributes (Including Status)')
```



In [72]:
```python
corr = h_corr.abs().stack().reset_index().sort_values(0, ascending=False)
corr['pairs'] = list(zip(corr.level_0, corr.level_1))
corr.set_index(['pairs'], inplace = True)
corr.drop(columns=['level_1', 'level_0'], inplace = True)

# cc for correlation coefficient
corr.columns = ['cc']
corr.drop_duplicates(inplace=True)

corr[(corr['cc'] > 0.7) & (corr['cc'] < 1)]
```

Out[72]:

	cc
pairs	
(time_to_first_funding, founded_year)	0.819908
(funding_rounds, time_first_to_last_funding)	0.812019

In [73]:
```python
# reducing number of categories in market & region

print(sum(data_ac['market'].value_counts() >= 15)) # top ~50 markets
data_ac['market'][data_ac['market'].map(data_ac['market'].value_counts()) < 15]
print(data_ac['market'].value_counts())

print(sum(data_ac['region'].value_counts() >= 60)) # top 10-15 regions
```

```
data_ac['region'][data_ac['region'].map(data_ac['region'].value_counts()) < 60]
print(data_ac['region'].value_counts())
```

```
48
other                      1162
Software                    569
Curated Web                 347
Mobile                      281
Enterprise Software         200
Biotechnology               189
Advertising                 173
Games                       172
E-Commerce                  164
Social Media                136
Hardware + Software         100
Semiconductors               96
Security                     92
Web Hosting                  84
Clean Technology             78
Health Care                  75
Finance                      71
Analytics                    59
Messaging                    55
Search                       54
News                         47
Music                        45
Education                    43
Public Relations             42
Video                        41
Travel                       40
Networking                   38
Photography                  36
Social Network Media         34
Consulting                   33
Health and Wellness          32
SaaS                         29
Sports                       25
Sales and Marketing          23
Web Development              23
Internet                     23
Manufacturing                21
Cloud Computing              20
Android                      20
iPhone                       19
Fashion                      19
Apps                         18
Marketplaces                 18
Facebook Applications        18
Automotive                   17
Hospitality                  17
Shopping                     17
Real Estate                  15
Name: market, dtype: int64
13
other                 2128
SF Bay Area           1250
New York City          338
Boston                 260
Los Angeles            186
London                 146
Seattle                143
Washington, D.C.        85
Chicago                 85
```

```
Austin                84
San Diego             77
Denver                75
Tel Aviv              73
Name: region, dtype: int64
```

In [74]:
```
#data_ac['region'].value_counts()[data_ac['region'].value_counts() > 75]
```

In [75]:
```
# dropping correlated columns for logistic regression, which can be
# sensitive to correlated features
data_final = data_ac.drop(columns=['founded_at', 'first_funding_at', 'last_func
                                   'first_funding_at_temp', 'last_funding_at_te
                                   'founded_at_temp', 'round_H', 'founded_year'
                                   'time_first_to_last_funding'],
                        axis=1)
# all inclusive
data_final2 = data_ac.drop(columns=['founded_at', 'first_funding_at', 'last_fur
                                    'first_funding_at_temp', 'last_funding_at_t
                                    'founded_at_temp', 'round_H'],
                        axis=1)
```

In [76]:
```
data_final.columns
```

Out[76]:
```
Index(['name', 'market', 'funding_total_usd', 'status', 'country_code',
       'state_code', 'region', 'city', 'funding_rounds', 'founded_month',
       'seed', 'venture', 'equity_crowdfunding', 'undisclosed',
       'convertible_note', 'debt_financing', 'angel', 'grant',
       'private_equity', 'product_crowdfunding', 'round_A', 'round_B',
       'round_C', 'round_D', 'round_E', 'round_F', 'round_G', 'international',
       'time_to_first_funding'],
      dtype='object')
```

In [77]:
```
data_final2.columns
```

Out[77]:
```
Index(['name', 'market', 'funding_total_usd', 'status', 'country_code',
       'state_code', 'region', 'city', 'funding_rounds', 'founded_month',
       'founded_year', 'seed', 'venture', 'equity_crowdfunding', 'undisclose
d',
       'convertible_note', 'debt_financing', 'angel', 'grant',
       'private_equity', 'product_crowdfunding', 'round_A', 'round_B',
       'round_C', 'round_D', 'round_E', 'round_F', 'round_G', 'international',
       'time_to_first_funding', 'time_first_to_last_funding'],
      dtype='object')
```

X/Y Split

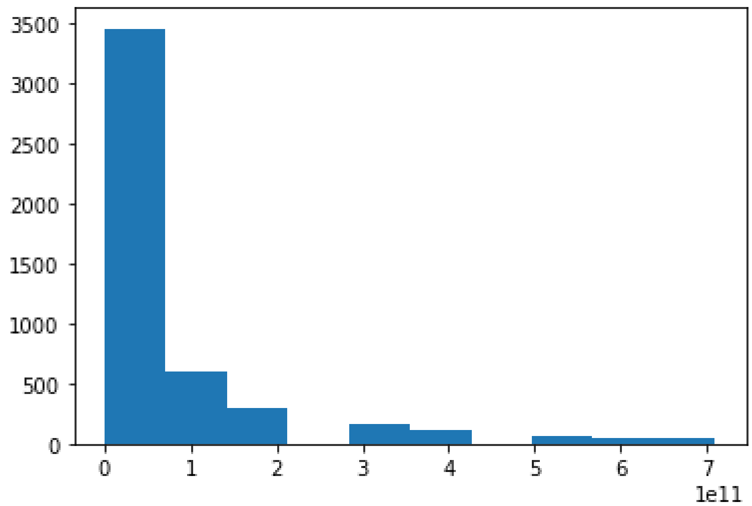
In [78]:
```
X_ac = data_final2.drop(columns=['status', 'name', 'country_code',
                                 'state_code', 'city'], axis=1)

encoder = LabelEncoder()
y_ac = pd.Series(encoder.fit_transform(data_final2['status']))
# acquired is 0, closed is 1

y_ac.value_counts(normalize=True)
```

Out[78]:
```
0    0.59716
1    0.40284
dtype: float64
```

For the logistic regression model, we'll get rid of some extre outliers and exclude highly correlated features

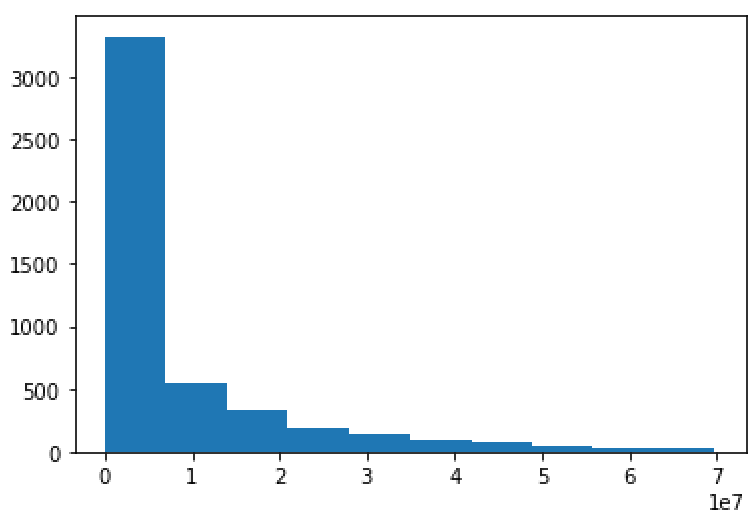
In [79]:
```
# Funding total USD
print(np.percentile(data_final['funding_total_usd'], 97.5))

plt.hist(data_final['funding_total_usd'][data_final['funding_total_usd'] < 8000
print(len(data_final[data_final['funding_total_usd'] >= 800000000000.0]))
print(len(data_final))
```

```
807015451469.2499
147
4930
```



In [80]:
```
print(np.percentile(data_final['venture'], 97.5))

plt.hist(data_final['venture'][data_final['venture'] < 70000000.0])
print(len(data_final[data_final['venture'] > 70000000.0]))
print(len(data_final[(data_final['venture'] > 70000000.0) &
               (data_final['funding_total_usd'] > 800000000000.0)]))
print(len(data_final))
```

```
69561028.64999995
119
96
4930
```



In [81]:
```
data_lr = data_final[(data_final['venture'] < 70000000.0) &
                     (data_final['funding_total_usd'] < 800000000000.0)].reset_
```

In [82]:
```
correlation_check(data_lr)
```

pairs	cc
(venture, funding_total_usd)	0.920258
(round_B, venture)	0.612658
(venture, round_C)	0.601337
(funding_total_usd, round_C)	0.562161
(funding_total_usd, round_B)	0.557115
(venture, funding_rounds)	0.537058
(funding_total_usd, funding_rounds)	0.509066

In [83]:
```
X_ac_lr = data_lr.drop(columns=['status', 'name', 'country_code',
                                'state_code', 'city', 'funding_total_usd'],
                       axis=1)

encoder = LabelEncoder()
y_ac_lr = pd.Series(encoder.fit_transform(data_lr['status']))
# acquired is 0, closed is 1

y_ac_lr.value_counts(normalize=True)
```

Out[83]:
```
0    0.588742
1    0.411258
dtype: float64
```

Baseline Model

In [84]:
```
X_train_processed, X_train_scaled, X_test_processed, \
X_test_scaled, y_train, y_test = train_test_preprocess(X_ac_lr, y_ac_lr)
```

In [85]:
```
train_test_check(X_train_processed, X_train_scaled, X_test_processed,
                 X_test_scaled, y_train, y_test)
```

```
There are 82 features in train set
There are 82 features in test set


There are 82 features in train set (scaled)
There are 82 features in test set (scaled)


y_train is a Series with 3570 values


target breakdown:  0    0.581793
1    0.418207
dtype: float64
```

	market_Advertising	market_Analytics	market_Android	market_Apps	market_Automotive	m
0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	0.0	

5 rows × 82 columns

	market_Advertising	market_Analytics	market_Android	market_Apps	market_Automotive	m
0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	0.0	

5 rows × 82 columns

In [86]:
```
baseline_model = LogisticRegression(random_state=42)
baseline_model.fit(X_train_scaled, y_train)
```

Out[86]:
```
LogisticRegression(random_state=42)
```

In [87]:
```
y_hat_train = baseline_model.predict(X_train_scaled)
y_hat_test = baseline_model.predict(X_test_scaled)

print_scores(y_train, y_hat_train, y_test, y_hat_test)
```

```
Training Recall:  0.6262558606831882
Testing Recall:  0.6


Training F1:  0.6572934973637962
Testing F1:  0.6058631921824104


ROC AUC:  0.6780991735537191
PR AUC:  0.5232765477926554


Training Accuracy:  0.726890756302521
Testing Accuracy:  0.6952141057934509
```

Baseline Analysis

- 60% recall is not a great start, but would like to see if hypertuning can improve this
- 70% accuracy also leaves something to be desired

Grid Search on Baseline**

In [88]:
```
param_grid = {'penalty':['l2', None],
              'solver':['lbfgs', 'sag'],
              'C': [1.0, 1e12],
              'class_weight': [None, 'balanced']
             }
```

In [89]:
```
grid_logreg = GridSearchCV(baseline_model, param_grid, cv = 5,
                           scoring='recall') # macro or weighted
grid_logreg.fit(X_train_scaled, y_train)
grid_logreg.best_params_
```

Out[89]:
```
{'C': 1000000000000.0,
 'class_weight': 'balanced',
 'penalty': 'l2',
 'solver': 'lbfgs'}
```

In [90]:
```
y_preds_grid_lr_train = grid_logreg.predict(X_train_scaled)
y_preds_grid_lr = grid_logreg.predict(X_test_scaled)

print_scores(y_train, y_preds_grid_lr_train, y_test, y_preds_grid_lr)
```

```
Training Recall:  0.769591426657736
Testing Recall:  0.7569892473118279


Training F1:  0.6993304930006086
Testing F1:  0.6597938144329897


ROC AUC:  0.706318315115969
PR AUC:  0.5375031955182031


Training Accuracy:  0.7232492997198879
Testing Accuracy:  0.6952141057934509
```

In [91]:
```
## Top coefficients
lr = LogisticRegression(random_state=42, class_weight='balanced',
                        solver='lbfgs', C=1e12, penalty='l2')
lr.fit(X_train_scaled, y_train)

coef_df = pd.DataFrame(lr.coef_, columns=X_train_scaled.columns).transpose()
coef_df.to_csv('coef_logreg.csv')

coef_df.columns=['coef']
coef_df['coef_abs'] = abs(coef_df['coef'])

coef_df.sort_values(by='coef_abs', ascending=False)[:20]
```

Out[91]:

	coef	coef_abs
time_to_first_funding	-13.318211	13.318211
private_equity	-4.029339	4.029339
funding_rounds	-3.113184	3.113184
undisclosed	-3.080005	3.080005
seed	-2.994236	2.994236
convertible_note	2.754146	2.754146
venture	-2.022008	2.022008
market_Clean Technology	2.021825	2.021825
round_A	-1.502466	1.502466
market_Consulting	1.492824	1.492824
market_Analytics	-1.264251	1.264251
debt_financing	-1.262151	1.262151
market_Sports	1.223371	1.223371
market_Biotechnology	1.194763	1.194763
region_other	1.120454	1.120454
region_Denver	0.919126	0.919126
market_Hardware + Software	0.860133	0.860133
market_Apps	-0.849601	0.849601
round_F	0.849171	0.849171
grant	-0.826037	0.826037

In [92]:
```
X_train_scaled['private_equity'].value_counts()
# vast majority of observations are 0, so coefficient probably isn't
# as important as its ranking implies
```

Out[92]:
```
0.000000    3529
0.026667       2
0.932000       1
0.006667       1
0.800000       1
0.133333       1
0.126667       1
0.933333       1
0.011106       1
0.840000       1
0.046667       1
0.033333       1
0.123335       1
0.022267       1
0.160000       1
0.533333       1
0.960000       1
0.000173       1
1.000000       1
0.408116       1
0.148482       1
0.529627       1
0.042667       1
0.248854       1
0.186667       1
0.252775       1
0.019813       1
0.053333       1
0.074248       1
0.060000       1
0.154667       1
0.240000       1
0.087976       1
0.066667       1
0.005051       1
0.081411       1
0.244853       1
0.624849       1
0.724281       1
0.006000       1
0.120000       1
Name: private_equity, dtype: int64
```

Custom Pipeline

In [93]:
```
scores = pd.DataFrame(columns = ['recall_train', 'recall_test', 'f1_train',
                                 'f1_test', 'accuracy_train', 'accuracy_test',
                                 'roc_auc', 'pr_auc', 'params'])
```

In [94]:
```
# creates a data frame with various scores for each model
def customPipe(model, model_name, X, y):
    X_train_processed, X_train_scaled, X_test_processed, X_test_scaled, \
    y_train, y_test = train_test_preprocess(X, y)

    if ('lr' in model_name) | ('knn' in model_name):
        model.fit(X_train_scaled, y_train)
        y_hat_train = model.predict(X_train_scaled)
        y_hat_test = model.predict(X_test_scaled)
```

```
    else:
        model.fit(X_train_processed, y_train)
        y_hat_train = model.predict(X_train_processed)
        y_hat_test = model.predict(X_test_processed)

    r_train, r_test, f1_train, f1_test, ac_train, ac_test, roc_auc, pr_auc = \
    return_scores(y_train, y_hat_train, y_test, y_hat_test)

    score_list = []
    score_list.extend((r_train, r_test, f1_train, f1_test,
                       ac_train, ac_test, roc_auc, pr_auc, str(model)))

    scores.loc[model_name] = score_list
    return scores
```

In [95]:

```
# model inputs
lr = LogisticRegression(random_state=42, class_weight='balanced',
                        solver='lbfgs', C=1e12, penalty='l2')
rf = RandomForestClassifier(random_state=42)
dtc = DecisionTreeClassifier(random_state=42)
ext = ExtraTreesClassifier(random_state=42)
xgb = XGBClassifier(random_state=42)
knn = KNeighborsClassifier()
```

In [96]:

```
# running the function
customPipe(lr, 'lr', X_ac_lr, y_ac_lr)
customPipe(rf, 'rf', X_ac, y_ac)
customPipe(dtc, 'dtc', X_ac, y_ac)
customPipe(ext, 'ext', X_ac, y_ac)
customPipe(xgb, 'xgb', X_ac, y_ac)
customPipe(knn, 'knn', X_ac, y_ac)
```

Out[96]:

	recall_train	recall_test	f1_train	f1_test	accuracy_train	accuracy_test	roc_auc	
lr	0.769591	0.756989	0.699330	0.659794	0.723249	0.695214	0.706318	0.5
rf	1.000000	0.589178	1.000000	0.643326	1.000000	0.735604	0.712164	0.5
dtc	1.000000	0.577154	1.000000	0.588957	1.000000	0.673966	0.658468	0.
ext	1.000000	0.571142	1.000000	0.617551	1.000000	0.713706	0.690884	0.5
xgb	0.933423	0.607214	0.927188	0.641949	0.941033	0.725872	0.706877	0.
knn	0.709482	0.577154	0.714528	0.576577	0.771977	0.656934	0.644163	0.5

Looking at recall and accuracy, the Random Forest and XGBoost models seem to yield the best scores. Will hyper tune each of these models.

In [97]:

```
# storing dataframe for easy access
scores_base = scores # models pre hyper parameter tuning
```

Other Models - Hypertuning

In [98]:
```
X_train_processed, X_train_scaled, X_test_processed, \
X_test_scaled, y_train, y_test = train_test_preprocess(X_ac, y_ac)
```

In [99]:
```
train_test_check(X_train_processed, X_train_scaled, X_test_processed,
                 X_test_scaled, y_train, y_test)
```

```
There are 85 features in train set
There are 85 features in test set


There are 85 features in train set (scaled)
There are 85 features in test set (scaled)


y_train is a Series with 3697 values


target breakdown:  0    0.597782
1    0.402218
dtype: float64
```

	market_Advertising	market_Analytics	market_Android	market_Apps	market_Automotive	n
0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	0.0	

5 rows × 85 columns

	market_Advertising	market_Analytics	market_Android	market_Apps	market_Automotive	n
0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	0.0	

5 rows × 85 columns

Random Forest

In [100…
```
rf = RandomForestClassifier(random_state=42)

# Initial search
rf_param_grid = {
    'criterion':['gini','entropy','log_loss'],
    'max_depth':[8,12,20],
    'min_samples_leaf': [5,10],
    'class_weight': [None, 'balanced']
```

```
}

## results from initial search
# {'class_weight': 'balanced',
#  'criterion': 'gini',
#  'max_depth': 20,
#  'min_samples_leaf': 5}

# Fine tuning
rf_param_grid2 = {
    'criterion':['gini'],
    'max_depth':[12,20,25],
    'min_samples_leaf': [10,15,20],
    'class_weight': ['balanced'],
    'n_estimators': [100, 200]
}

grid_rfc = GridSearchCV(rf, rf_param_grid2, cv = 5, scoring='recall')
grid_rfc.fit(X_train_processed, y_train)

grid_rfc.best_params_
```

Out[100]:

```
{'class_weight': 'balanced',
 'criterion': 'gini',
 'max_depth': 12,
 'min_samples_leaf': 20,
 'n_estimators': 100}
```

In [101…

```
y_preds_grid_rfc_train = grid_rfc.predict(X_train_processed)
y_preds_grid_rfc = grid_rfc.predict(X_test_processed)

print_scores(y_train, y_preds_grid_rfc_train, y_test, y_preds_grid_rfc)
```

```
Training Recall:  0.7222595830531271
Testing Recall:  0.6973947895791583


Training F1:  0.7049556941253693
Testing F1:  0.6850393700787402


ROC AUC:  0.7335747789857645
PR AUC:  0.5918918112508587


Training Accuracy:  0.7568298620503111
Testing Accuracy:  0.740470397404704
```

In [102…

```
rf_best = RandomForestClassifier(random_state = 42, class_weight='balanced',
                                 criterion='gini', max_depth=12,
                                 min_samples_leaf=20, n_estimators=100)
customPipe(rf_best, 'rf_best', X_ac, y_ac)
```

Out[102]:

	recall_train	recall_test	f1_train	f1_test	accuracy_train	accuracy_test	roc_auc
lr	0.769591	0.756989	0.699330	0.659794	0.723249	0.695214	0.706318
rf	1.000000	0.589178	1.000000	0.643326	1.000000	0.735604	0.712164
dtc	1.000000	0.577154	1.000000	0.588957	1.000000	0.673966	0.658468
ext	1.000000	0.571142	1.000000	0.617551	1.000000	0.713706	0.690884
xgb	0.933423	0.607214	0.927188	0.641949	0.941033	0.725872	0.706877
knn	0.709482	0.577154	0.714528	0.576577	0.771977	0.656934	0.644163
rf_best	0.722260	0.697395	0.704956	0.685039	0.756830	0.740470	0.733575

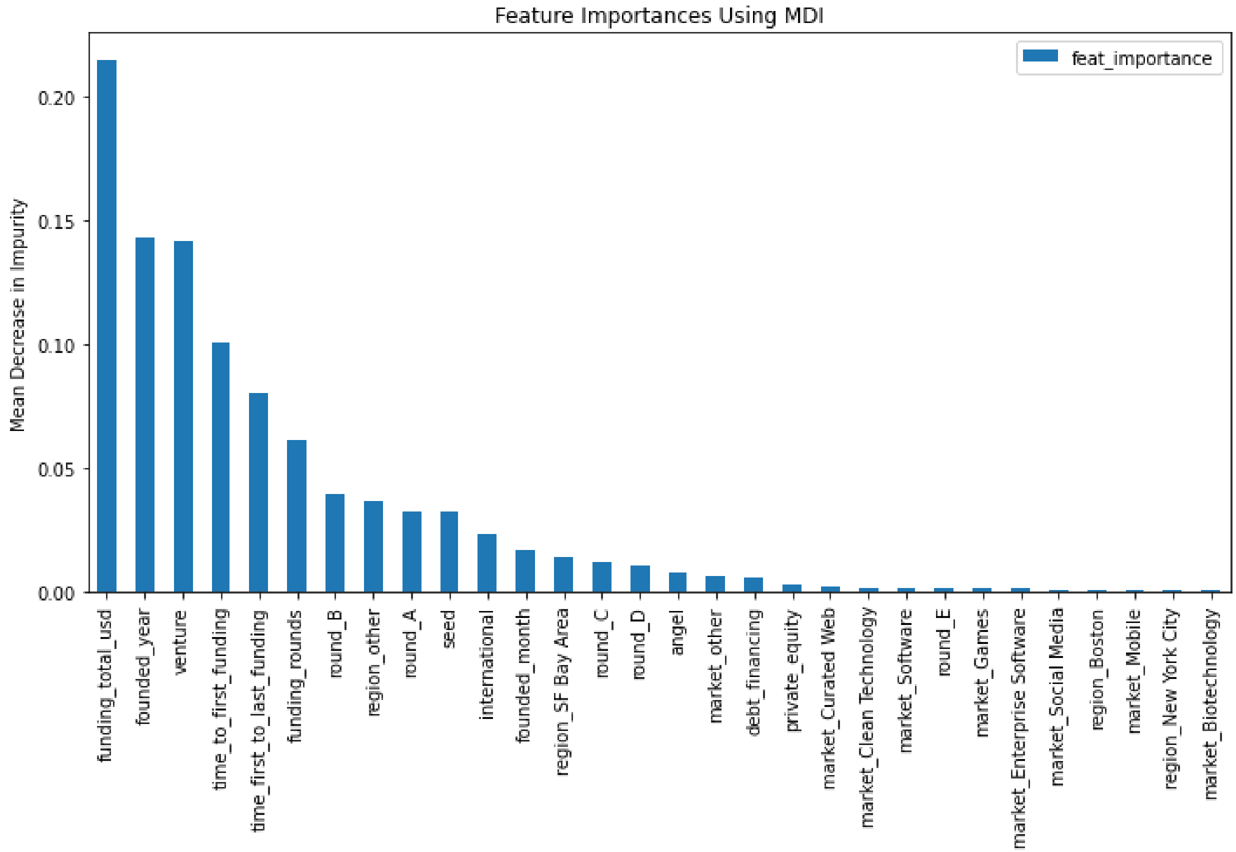
Feature Importance

In [103…

```
feats = {} # a dict to hold feature_name: feature_importance
for feature, importance in zip(X_train_processed.columns, rf_best.feature_impor
    feats[feature] = importance #add the name/value pair

feats_rf = pd.DataFrame(feats.items())

feats_rf.columns = ['col', 'feat_importance']
feats_rf = feats_rf.sort_values(by=['feat_importance'], ascending=False)
feats_rf_20 = feats_rf[:20]
feats_rf_30 = feats_rf[:30]
```

In [104…

```
fig, ax = plt.subplots(figsize = (10,7))
feats_rf_30.plot.bar(ax=ax) # yerr=std,
ax.set_title("Feature Importances Using MDI")
ax.set_ylabel("Mean Decrease in Impurity")
ax.set_xticklabels(feats_rf_30['col'])
fig.tight_layout()
```



In [105…

```
list(feats_rf['col'][:10])
```

Out[105]:

```
['funding_total_usd',
 'founded_year',
 'venture',
 'time_to_first_funding',
 'time_first_to_last_funding',
 'funding_rounds',
 'round_B',
 'region_other',
 'round_A',
 'seed']
```

XG Boost*

In [106…

```
# 45+ runtime
xgb = XGBClassifier(random_state=42)

# initial grid search
xgb_param_grid = {
    'learning_rate': [0.1, 0.2],
    'max_depth': [4,6,8],
    'min_child_weight': [3,5,7],
    'subsample': [0.5, 0.7],
    'scale_pos_weight':[1.5, 2]
}

## result from initial grid search:
# {'learning_rate': 0.1,
#  'max_depth': 4,
#  'min_child_weight': 7,
```

```
#   'scale_pos_weight': 2,
#   'subsample': 0.7}

# fine tuning
xgb_param_grid2 = {
    'learning_rate': [0.1,0.2],
    'max_depth': [1,2,4],
    'min_child_weight': [3,7,10],
    'subsample': [0.7],
    'scale_pos_weight':[2]
}

grid_xgb = GridSearchCV(xgb, xgb_param_grid2, cv = 5, scoring='recall')
grid_xgb.fit(X_train_processed, y_train)

grid_xgb.best_params_
```

Out[106]:

```
{'learning_rate': 0.2,
 'max_depth': 1,
 'min_child_weight': 7,
 'scale_pos_weight': 2,
 'subsample': 0.7}
```

In [107…

```
xgb_best = XGBClassifier(random_state=42, max_depth=1, min_child_weight=7,
                         subsample=0.7, learning_rate=0.2, scale_pos_weight=2)

customPipe(xgb_best, 'xgb_best', X_ac, y_ac)
```

Out[107]:

	recall_train	recall_test	f1_train	f1_test	accuracy_train	accuracy_test	roc_au
lr	0.769591	0.756989	0.699330	0.659794	0.723249	0.695214	0.7063
rf	1.000000	0.589178	1.000000	0.643326	1.000000	0.735604	0.7121
dtc	1.000000	0.577154	1.000000	0.588957	1.000000	0.673966	0.6584
ext	1.000000	0.571142	1.000000	0.617551	1.000000	0.713706	0.6908
xgb	0.933423	0.607214	0.927188	0.641949	0.941033	0.725872	0.7068
knn	0.709482	0.577154	0.714528	0.576577	0.771977	0.656934	0.6441
rf_best	0.722260	0.697395	0.704956	0.685039	0.756830	0.740470	0.7335
xgb_best	0.803631	0.799599	0.706682	0.696943	0.731674	0.718573	0.7315

Feature Importance

In [108…

```
xgb_best.fit(X_train_processed, y_train)
```

Out[108]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.2, max_delta_step=0, max_depth=1,
              min_child_weight=7, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=4
2,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=2, subsample=0.7,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [109…

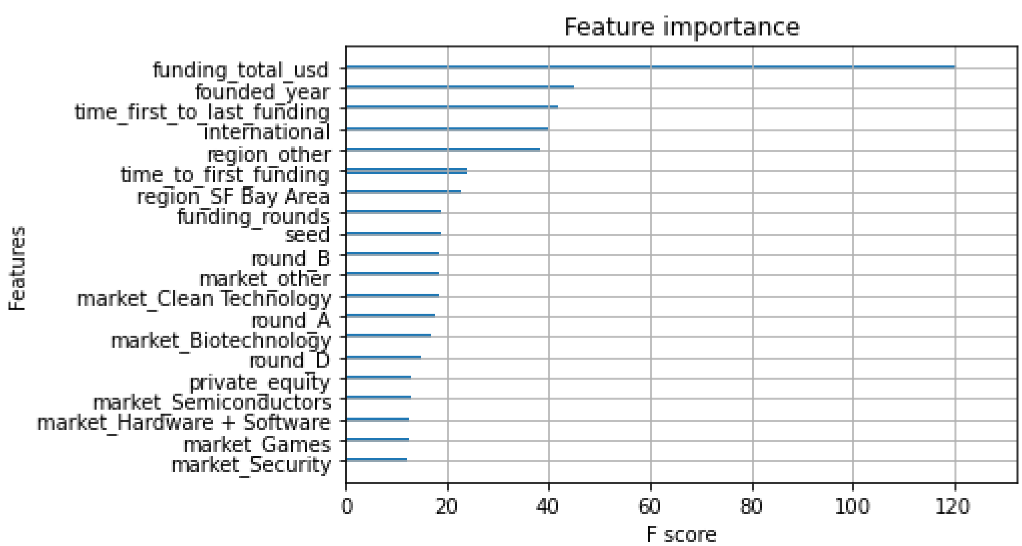
```
f = 'gain' # importance type
feat_imp = xgb_best.get_booster().get_score(importance_type= f)
feats_xg = pd.DataFrame(sorted(feat_imp.items(), key=lambda item: item[1],
                               reverse=True))
feats_xg.columns = ['col', 'feat_importance']
feats_xg[:15]
```

Out[109]:

	col	feat_importance
0	funding_total_usd	120.335433
1	founded_year	45.184254
2	time_first_to_last_funding	41.682578
3	international	39.813609
4	region_other	38.373383
5	time_to_first_funding	24.078237
6	region_SF Bay Area	22.668856
7	funding_rounds	18.984874
8	seed	18.831061
9	round_B	18.659117
10	market_other	18.592387
11	market_Clean Technology	18.477624
12	round_A	17.832843
13	market_Biotechnology	16.774651
14	round_D	15.123461

In [110…

```
from xgboost import plot_importance
plot_importance(xgb_best, max_num_features=20, importance_type=f,
                show_values=False)
plt.show()
```



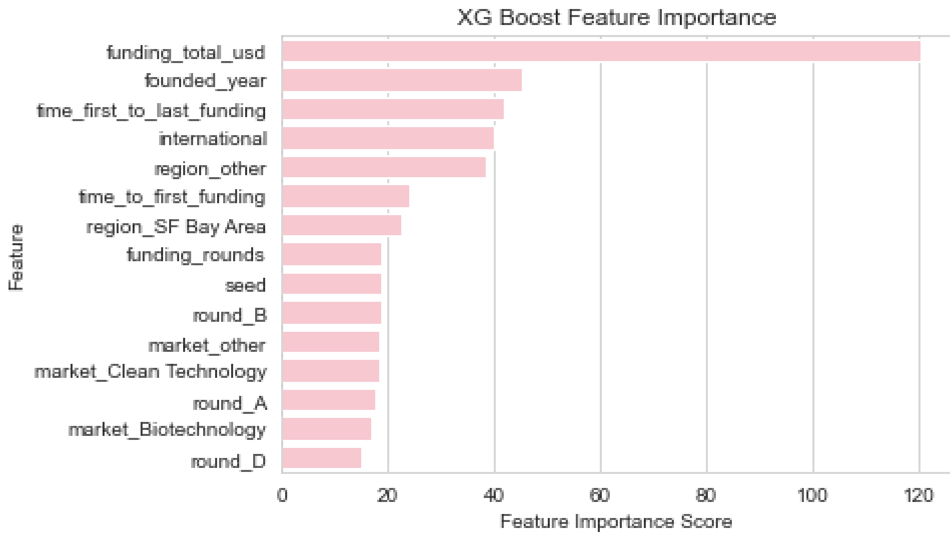
In [111…

```
feats_xg[:15]
```

```
sns.set_style("whitegrid")
sns.set_palette('Blues')

g = sns.barplot(x='feat_importance', y='col', data=feats_xg[:15], color='pink')
# g.set_xticklabels(["U.S.", "International"])
g.set(xlabel='Feature Importance Score',
      ylabel='Feature',
      title="XG Boost Feature Importance")
```

Out[111]:

```
[Text(0.5, 0, 'Feature Importance Score'),
 Text(0, 0.5, 'Feature'),
 Text(0.5, 1.0, 'XG Boost Feature Importance')]
```



ExtraTrees

In [112…

```
ext_param_grid = {'criterion':['entropy','gini'],
                 'max_depth':[15,20,25],
                 'min_samples_leaf': [1,5],
                 'class_weight': ['balanced'],
                 'max_features': ['auto']
                }

ext = ExtraTreesClassifier(random_state=42)

grid_ext = GridSearchCV(ext, ext_param_grid, cv = 5, scoring='recall')
grid_ext.fit(X_train_processed, y_train)

grid_ext.best_params_
```

Out[112]:

```
{'class_weight': 'balanced',
 'criterion': 'gini',
 'max_depth': 15,
 'max_features': 'auto',
 'min_samples_leaf': 5}
```

In [113…

```
ext_grid = ExtraTreesClassifier(random_state=42, class_weight='balanced',
                                criterion='gini', max_depth= 15,
                                max_features='auto', min_samples_leaf= 5)
customPipe(ext_grid, 'ext_best', X_ac, y_ac)
```

Out[113]:

	recall_train	recall_test	f1_train	f1_test	accuracy_train	accuracy_test	roc_a
lr	0.769591	0.756989	0.699330	0.659794	0.723249	0.695214	0.7063
rf	1.000000	0.589178	1.000000	0.643326	1.000000	0.735604	0.7121
dtc	1.000000	0.577154	1.000000	0.588957	1.000000	0.673966	0.6584
ext	1.000000	0.571142	1.000000	0.617551	1.000000	0.713706	0.6908
xgb	0.933423	0.607214	0.927188	0.641949	0.941033	0.725872	0.7068
knn	0.709482	0.577154	0.714528	0.576577	0.771977	0.656934	0.6441
rf_best	0.722260	0.697395	0.704956	0.685039	0.756830	0.740470	0.7335
xgb_best	0.803631	0.799599	0.706682	0.696943	0.731674	0.718573	0.7315
ext_best	0.759247	0.705411	0.711854	0.663525	0.752773	0.710462	0.7096

Decision Tree***

In [114…

```
dtc_param_grid = {'criterion':['gini','entropy'],
                  'max_depth':[5,10,15],
                  'min_samples_leaf': [15,20,25],
                  'class_weight': [None, 'balanced']
             }

dtc = DecisionTreeClassifier(random_state=42)

grid_dtc = GridSearchCV(dtc, dtc_param_grid, cv = 5, scoring='recall')
grid_dtc.fit(X_train_processed, y_train)

grid_dtc.best_params_
```

Out[114]:

```
{'class_weight': 'balanced',
 'criterion': 'gini',
 'max_depth': 5,
 'min_samples_leaf': 15}
```

In [115…

```
dtc_best = DecisionTreeClassifier(random_state=42, class_weight='balanced',
                                  criterion='gini', max_depth= 5,
                                  min_samples_leaf= 15)

dtc_best.fit(X_train_processed, y_train)
customPipe(dtc_best, 'dtc_best', X_ac, y_ac)
```

Out[115]:

	recall_train	recall_test	f1_train	f1_test	accuracy_train	accuracy_test	roc_au
lr	0.769591	0.756989	0.699330	0.659794	0.723249	0.695214	0.7063
rf	1.000000	0.589178	1.000000	0.643326	1.000000	0.735604	0.7121
dtc	1.000000	0.577154	1.000000	0.588957	1.000000	0.673966	0.6584
ext	1.000000	0.571142	1.000000	0.617551	1.000000	0.713706	0.6908
xgb	0.933423	0.607214	0.927188	0.641949	0.941033	0.725872	0.7068
knn	0.709482	0.577154	0.714528	0.576577	0.771977	0.656934	0.6441
rf_best	0.722260	0.697395	0.704956	0.685039	0.756830	0.740470	0.7335
xgb_best	0.803631	0.799599	0.706682	0.696943	0.731674	0.718573	0.7315
ext_best	0.759247	0.705411	0.711854	0.663525	0.752773	0.710462	0.7096
dtc_best	0.806994	0.787575	0.710900	0.687063	0.736002	0.709651	0.7221

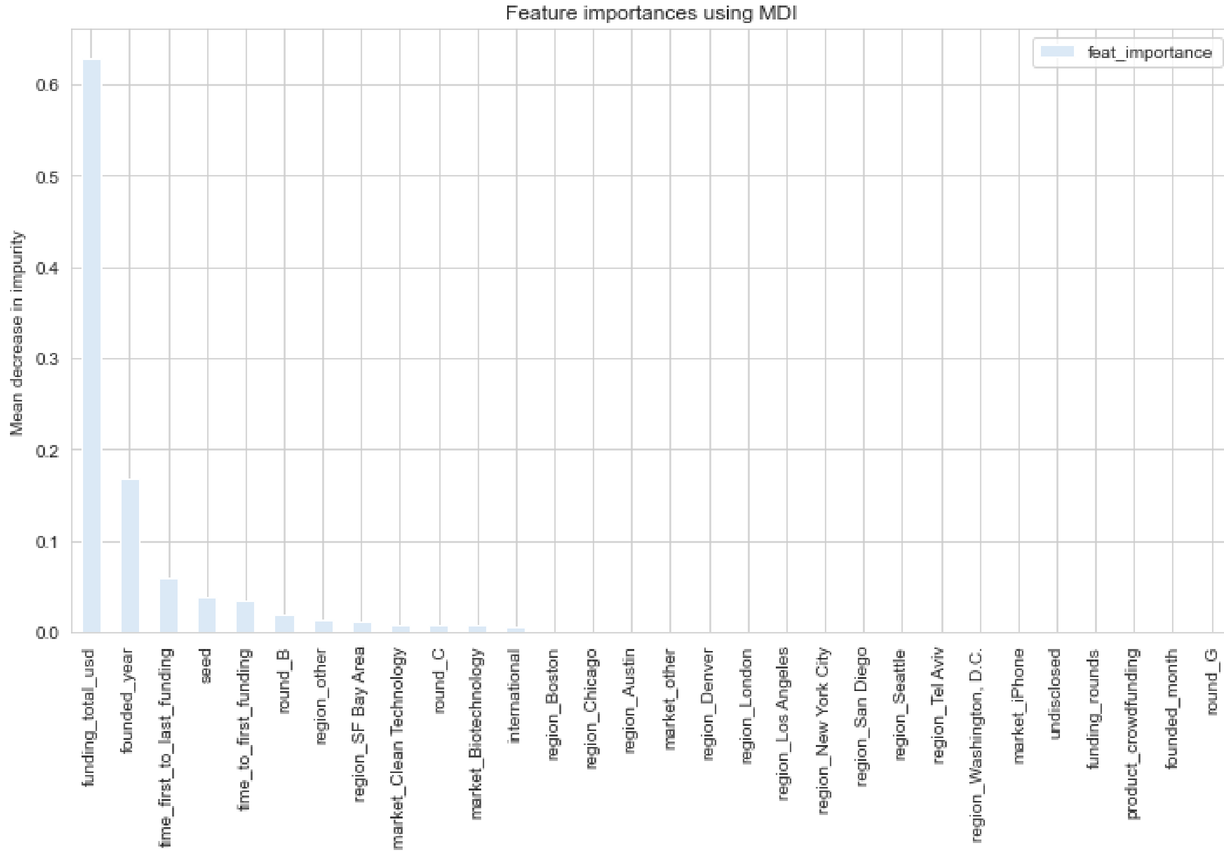
Feature Importance

In [116…

```
feats = {} # a dict to hold feature_name: feature_importance
for feature, importance in zip(X_train_processed.columns, dtc_best.feature_impo
    feats[feature] = importance #add the name/value pair

feats_dtc = pd.DataFrame(feats.items())

feats_dtc.columns = ['col', 'feat_importance']
feats_dtc = feats_dtc.sort_values(by=['feat_importance'], ascending=False)
feats_dtc_30 = feats_dtc[:30]
```

In [117…

```
fig, ax = plt.subplots(figsize = (10,7))
feats_dtc_30.plot.bar(ax=ax) # yerr=std,
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
ax.set_xticklabels(feats_dtc_30['col'])
fig.tight_layout()
```



Final Model

- XG Boost is the best model, with ~80% recall, 72% accuracy and 73% AUC
- Now that we have narrowed down the best models, we will run them with reduced features & SMOTE to see if that generates any improvement

In [118…

```
X_train_processed, X_train_scaled, X_test_processed, \
X_test_scaled, y_train, y_test = train_test_preprocess(X_ac, y_ac)
```

Feature Reduction

In [119…

```
# top 20 features
X_train_xg = X_train_processed[list(feats_xg['col'][:20])]
X_train_dtc = X_train_processed[list(feats_dtc['col'][:20])]

X_test_xg = X_test_processed[list(feats_xg['col'][:20])]
X_test_dtc = X_test_processed[list(feats_dtc['col'][:20])]
```

In [120…

```
# XG Boost
final_model = XGBClassifier(random_state=42, max_depth=1, min_child_weight=7,
                            subsample=0.7, learning_rate=0.2, scale_pos_weight=

final_model.fit(X_train_xg, y_train)

y_hat_train = final_model.predict(X_train_xg)
y_hat_test = final_model.predict(X_test_xg)
```

```
print_scores(y_train, y_hat_train, y_test, y_hat_test)
```

```
Training Recall:  0.8049764626765299
Testing Recall:  0.7875751503006012


Training F1:  0.7105966162065894
Testing F1:  0.6906854130052724


ROC AUC:  0.7262126432701916
PR AUC:  0.570346385974146


Training Accuracy:  0.7362726535028401
Testing Accuracy:  0.7145174371451744
```

In [121…

```
# Decision Trees - 2nd place
final_model = dtc_best
final_model.fit(X_train_dtc, y_train)

y_hat_train = final_model.predict(X_train_dtc)
y_hat_test = final_model.predict(X_test_dtc)

print_scores(y_train, y_hat_train, y_test, y_hat_test)
```

```
Training Recall:  0.8069939475453934
Testing Recall:  0.7875751503006012


Training F1:  0.7109004739336493
Testing F1:  0.6870629370629371


ROC AUC:  0.7221254498097012
PR AUC:  0.565840551507965


Training Accuracy:  0.7360021639166892
Testing Accuracy:  0.7096512570965126
```

Feature reduction doesn't improve the models.

SMOTE

In [122…

```
#Using SMOTE to further reduce class imbalance
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_sample(X_train_processed, y_tr
```

In [123…

```
# addressing class imbalance within the hyperparameters results in higher recal
final_model_smote = XGBClassifier(random_state=42, max_depth=1, min_child_weigh
                                  subsample=0.7, learning_rate=0.2)

final_model_smote.fit(X_train_resampled, y_train_resampled)

y_hat_train = final_model_smote.predict(X_train_resampled)
y_hat_test = final_model_smote.predict(X_test_processed)
```

```
print_scores(y_train_resampled, y_hat_train, y_test, y_hat_test)
```

```
Training Recall:  0.7773755656108597
Testing Recall:  0.7394789579158316


Training F1:  0.7712682379349045
Testing F1:  0.7001897533206831


ROC AUC:  0.7430364816827113
PR AUC:  0.5970874784794594


Training Accuracy:  0.7694570135746607
Testing Accuracy:  0.7437145174371452
```

Model Selection

In [124…

```
final_model = XGBClassifier(random_state=42, max_depth=1, min_child_weight=7,
                            subsample=0.7, learning_rate=0.2, scale_pos_weight=

customPipe(final_model, 'final_model', X_ac, y_ac)
```

Out[124]:

	recall_train	recall_test	f1_train	f1_test	accuracy_train	accuracy_test	roc_
lr	0.769591	0.756989	0.699330	0.659794	0.723249	0.695214	0.70
rf	1.000000	0.589178	1.000000	0.643326	1.000000	0.735604	0.71
dtc	1.000000	0.577154	1.000000	0.588957	1.000000	0.673966	0.65
ext	1.000000	0.571142	1.000000	0.617551	1.000000	0.713706	0.69
xgb	0.933423	0.607214	0.927188	0.641949	0.941033	0.725872	0.70
knn	0.709482	0.577154	0.714528	0.576577	0.771977	0.656934	0.64
rf_best	0.722260	0.697395	0.704956	0.685039	0.756830	0.740470	0.73
xgb_best	0.803631	0.799599	0.706682	0.696943	0.731674	0.718573	0.73
ext_best	0.759247	0.705411	0.711854	0.663525	0.752773	0.710462	0.70
dtc_best	0.806994	0.787575	0.710900	0.687063	0.736002	0.709651	0.72
final_model	0.803631	0.799599	0.706682	0.696943	0.731674	0.718573	0.73

In [125…

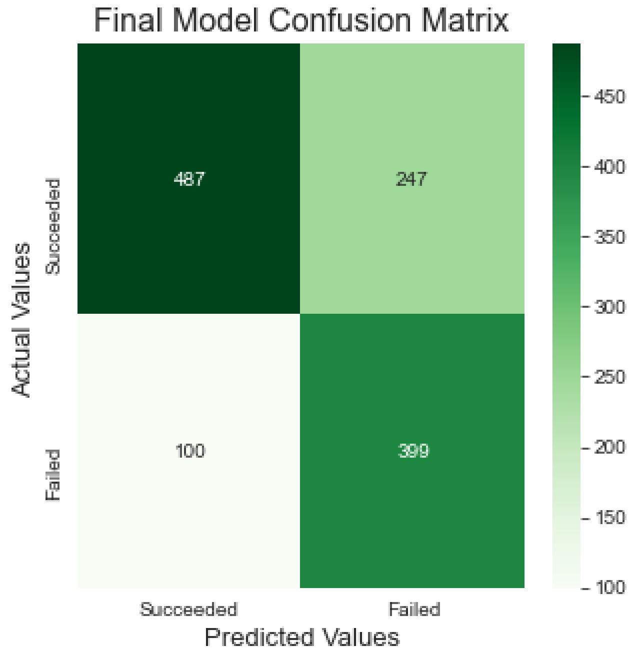
```
# storing dataframe for easy access
scores_final = scores
scores_final.to_csv('final_scores.csv')
scores_final.sort_values(by=['recall_test'], ascending=False)[1:]
```

Out[125]:

	recall_train	recall_test	f1_train	f1_test	accuracy_train	accuracy_test	roc_
final_model	0.803631	0.799599	0.706682	0.696943	0.731674	0.718573	0.73
dtc_best	0.806994	0.787575	0.710900	0.687063	0.736002	0.709651	0.72
lr	0.769591	0.756989	0.699330	0.659794	0.723249	0.695214	0.70
ext_best	0.759247	0.705411	0.711854	0.663525	0.752773	0.710462	0.70
rf_best	0.722260	0.697395	0.704956	0.685039	0.756830	0.740470	0.73
xgb	0.933423	0.607214	0.927188	0.641949	0.941033	0.725872	0.70
rf	1.000000	0.589178	1.000000	0.643326	1.000000	0.735604	0.71
dtc	1.000000	0.577154	1.000000	0.588957	1.000000	0.673966	0.65
knn	0.709482	0.577154	0.714528	0.576577	0.771977	0.656934	0.64
ext	1.000000	0.571142	1.000000	0.617551	1.000000	0.713706	0.69

In [126…

```
# visualizing predictions

y_hat_test = final_model.predict(X_test_processed)

conf_matrix = confusion_matrix(y_test, y_hat_test)

fig, ax = plt.subplots(figsize=(5,5))

ax = sns.heatmap(conf_matrix, annot=True, cmap='Greens', fmt='d')

ax.set_title('Final Model Confusion Matrix', fontsize=16);
ax.set_xlabel('Predicted Values',fontsize=13)
ax.set_ylabel('Actual Values ', fontsize=13);

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['Succeeded','Failed'])
ax.yaxis.set_ticklabels(['Succeeded','Failed'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



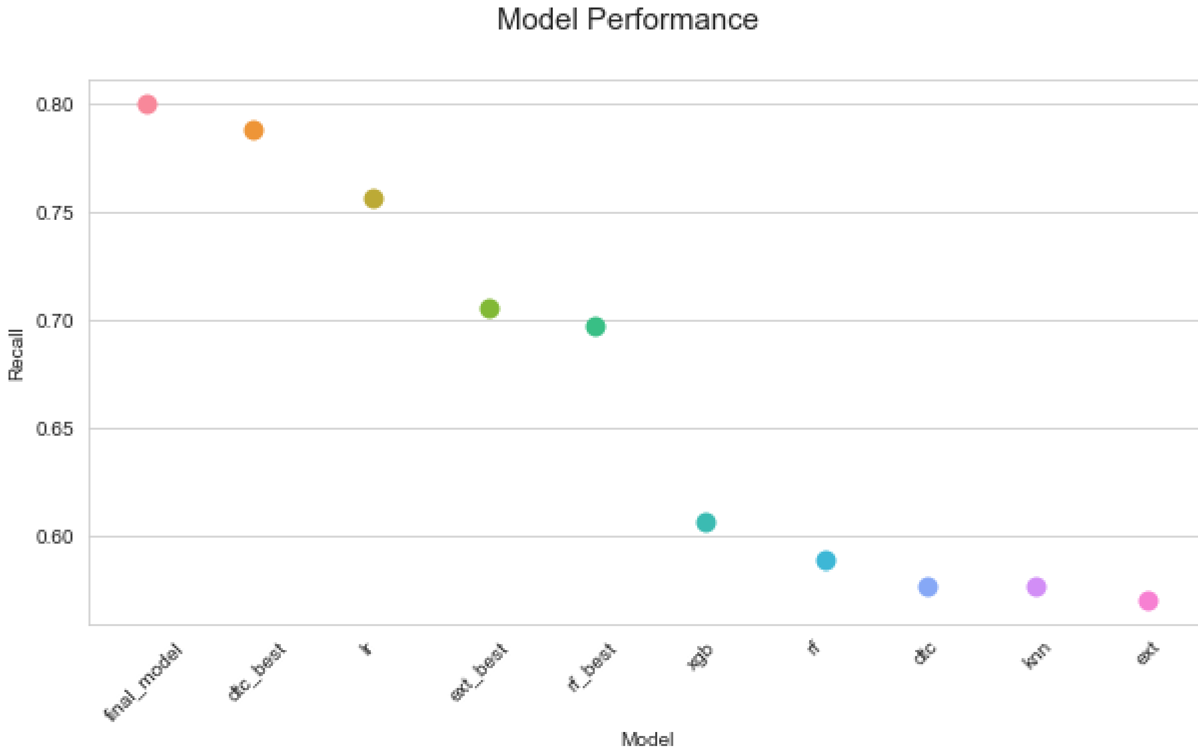
Final Analysis & Visualizations

In [127…

```
## Model Comparison
scores_viz = scores_final.sort_values(by=['recall_test'], ascending=False)[1:].

fig, ax = plt.subplots(figsize=(10, 5))
sns.stripplot(x="index", y="recall_test", data=scores_viz, size=10)
plt.xticks(rotation = 45)
ax.set_xlabel("Model", fontsize=10)
ax.set_ylabel("Recall", fontsize=10)
fig.suptitle("Model Performance", fontsize=15)
```

Out[127]:

```
Text(0.5, 0.98, 'Model Performance')
```



In [128…

```
## Visualizing Important Features
```

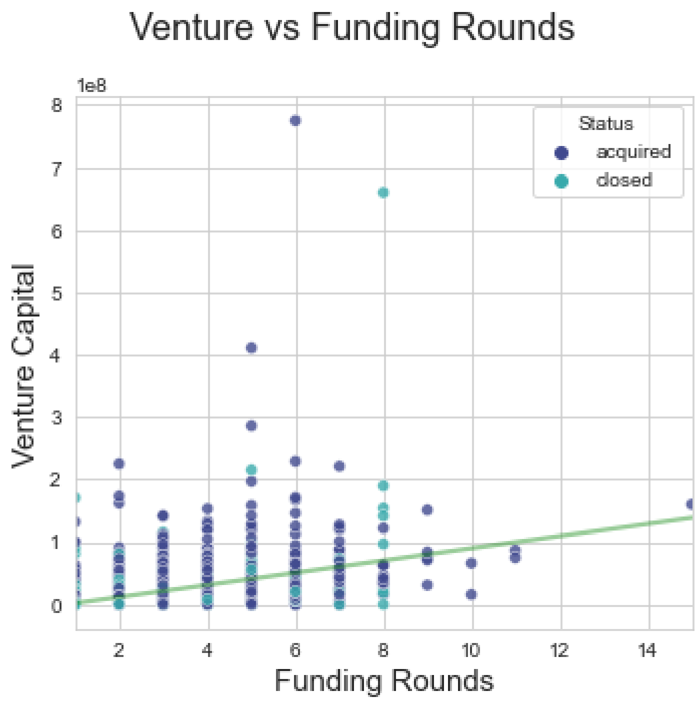
In [129…

```
df = data_final2
df['int_category'] = ['international' if x==1 else 'U.S.' for x in df['internat
df['one_funding_round'] = ['one' if x==1 else 'multiple' for x in df['funding_r

df_box = df[(df['venture'] < 40000000.0)]

cmap = sns.diverging_palette(130, 50, as_cmap=True)
```

In [130…

```
# setting up the figure with a single subplot
fig = plt.figure(figsize=(5, 5))
fig, ax = plt.subplots(figsize=(5, 5))
# scatterplot using seaborn
plot = sns.scatterplot(x='funding_rounds', y='venture', data=df,
hue=df.status, legend='full', alpha = 0.8, palette='mako')
# adding regression line using seaborn regplot
sns.regplot(data=df, x='funding_rounds', y='venture', scatter=False,
            ax=ax, ci=False, color='g', line_kws={'alpha':0.4})

# updating figure title, adding labels for x- and y-axis
fig.suptitle("Venture vs Funding Rounds", fontsize=18)
ax.set_xlabel("Funding Rounds", fontsize=15)
ax.set_ylabel("Venture Capital", fontsize=15)
# setting legend title
ax.get_legend().set_title("Status")
# getting everything to fit nicely on the plot
plt.tight_layout()
```

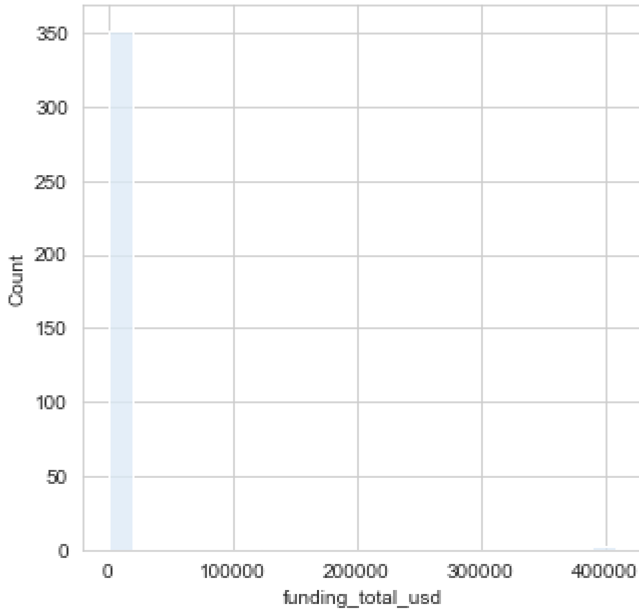
```
<Figure size 360x360 with 0 Axes>
```



In [131…

```
fig = plt.figure(figsize=(5, 5))
fig, ax = plt.subplots(figsize=(5, 5))

df_fund = df[(df['funding_total_usd'] < 1000000) &
             (df['time_to_first_funding'] > -10) &
             (df['funding_total_usd'] > 0)]

# plot = sns.scatterplot(x='time_to_first_funding', y='funding_total_usd',
#                        data=df_fund, hue=df.status, legend='full',
#                        alpha = 0.8, palette='mako')


# histograms - funding usd for successes & failures
df_success = df[df['status']=='acquired']
df_failure = df[df['status']=='closed']

plot = sns.histplot(x='funding_total_usd',
                    data=df_success[df_success['funding_total_usd'] < 750000],
                    bins=20)
```

```
<Figure size 360x360 with 0 Axes>
```

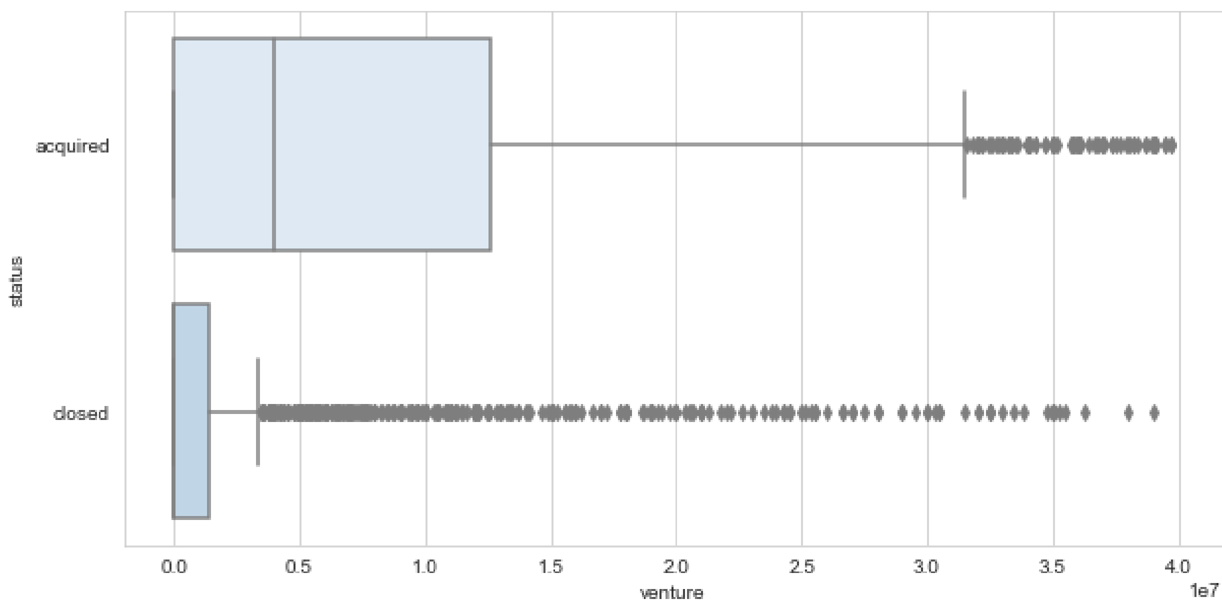


In [132…

```
df_box = df[(df['venture'] < 40000000.0)]
#df_box = df[(df['funding_total_usd'] < 1000000.0)]

plt.figure(figsize=(10,5))
sns.boxplot(x=df_box['venture'], y=df_box['status'])
```

Out[132]:

```
<AxesSubplot:xlabel='venture', ylabel='status'>
```

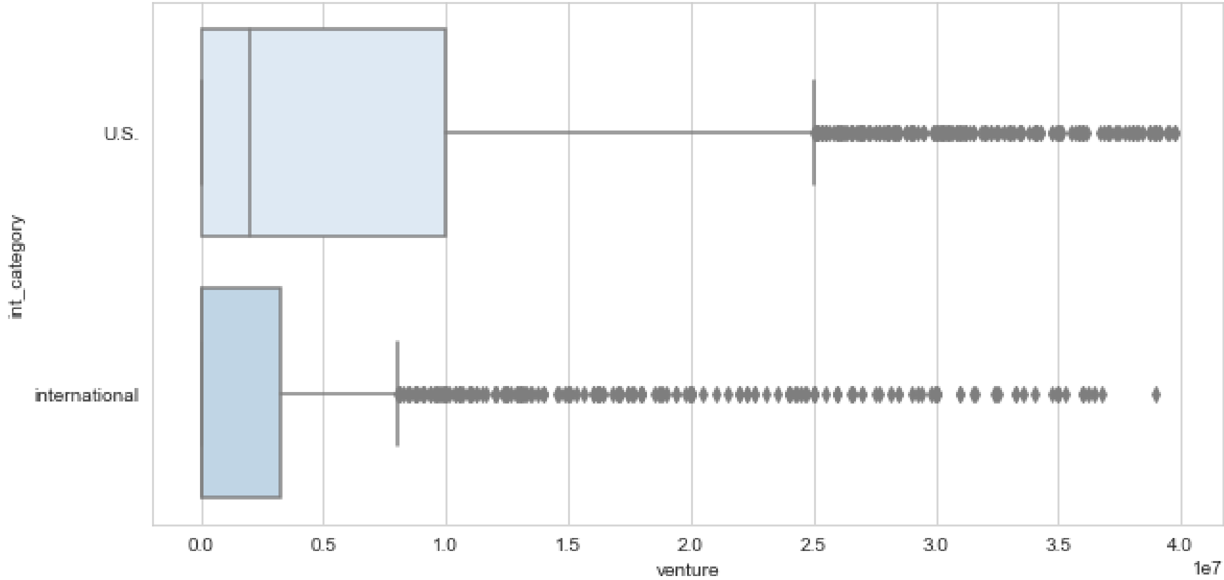


In [133…

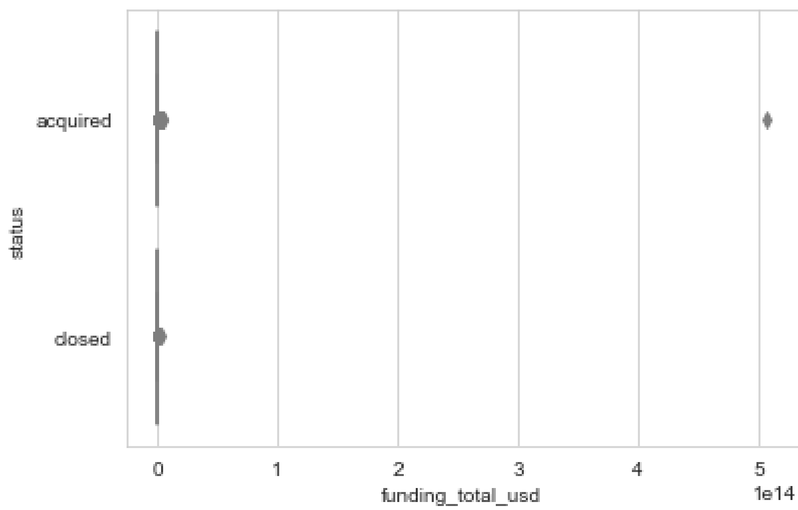
```
plt.figure(figsize=(10,5))

sns.boxplot(x=df_box['venture'], y=df_box['int_category'])

df_box['int_category'].value_counts()
```

Out[133]:

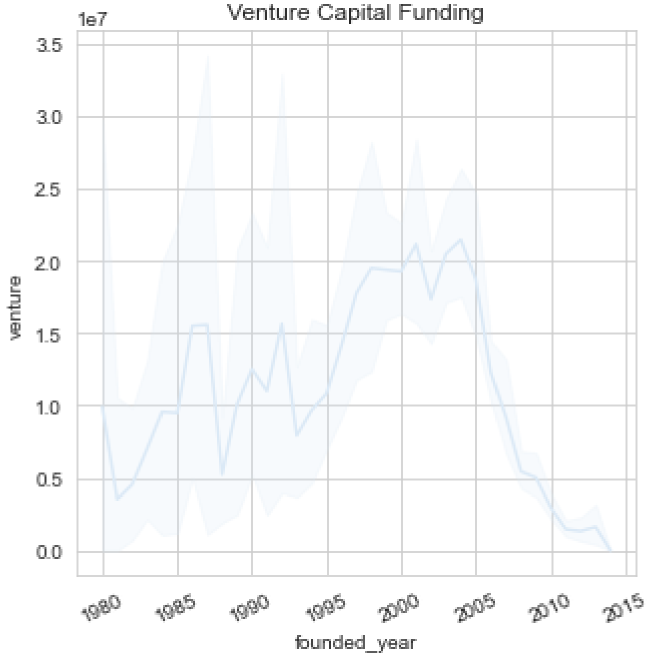
```
U.S.             3143
international    1443
Name: int_category, dtype: int64
```

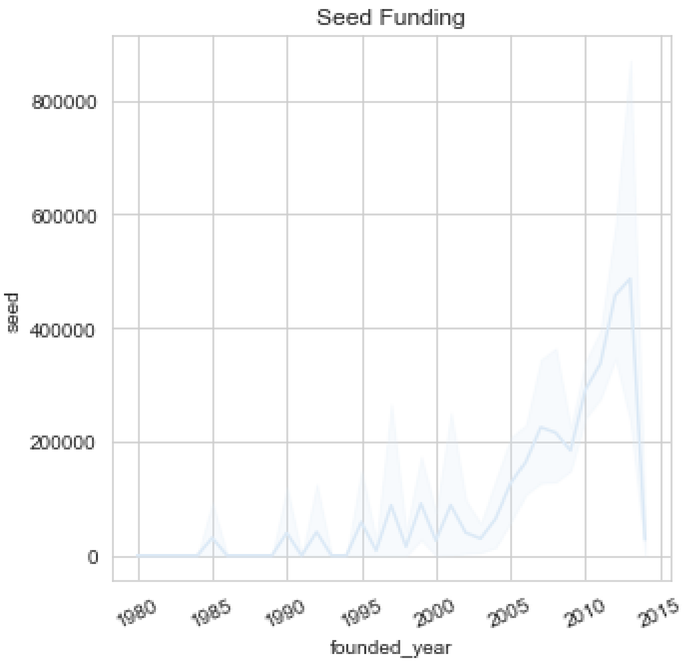


In [134…
```
sns.boxplot(x=df_box['funding_total_usd'], y=df_box['status'])
```
Out[134]:
```
<AxesSubplot:xlabel='funding_total_usd', ylabel='status'>
```



In [135…
```
plt.figure(figsize=(5,5))
plt.title('Venture Capital Funding')
sns.lineplot(x = "founded_year", y = "venture", data = df)
plt.xticks(rotation = 25)

plt.figure(figsize=(5,5))
sns.lineplot(x = "founded_year", y = "seed", data = df)
plt.xticks(rotation = 25)
plt.title('Seed Funding')
```
Out[135]:
```
Text(0.5, 1.0, 'Seed Funding')
```

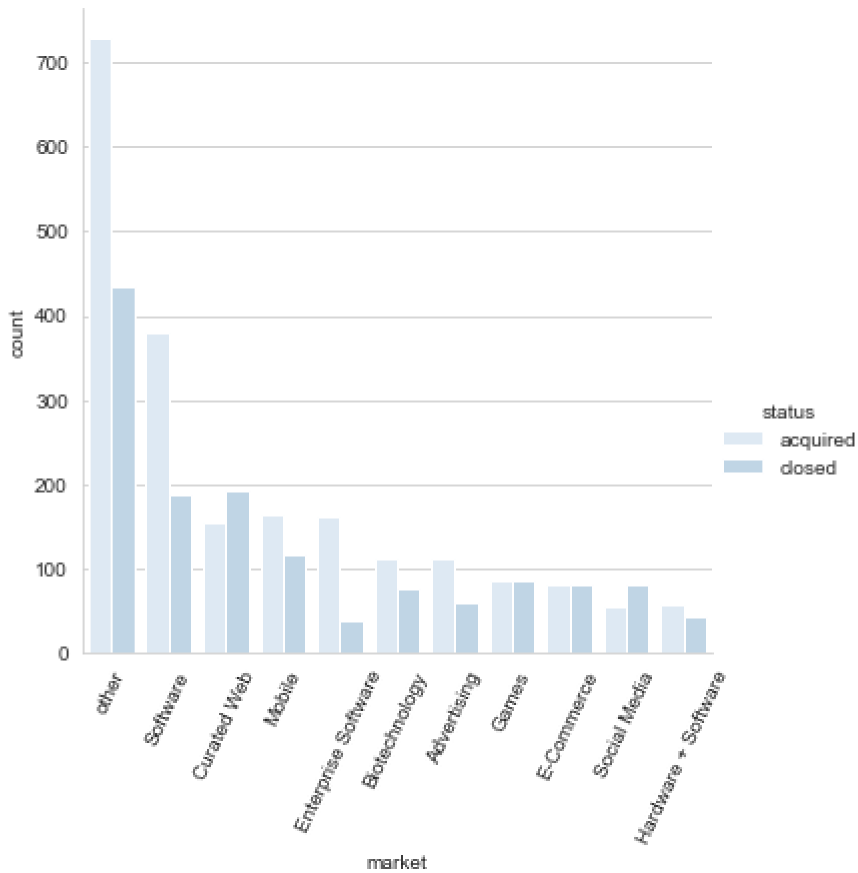




In [136…

```
# top 10 markets

#print(set(df['market'][df['market'].map(df['market'].value_counts()) >= 100]))
df['market'].value_counts()[1:11]
top10_mkt = list(set(df['market'][df['market'].map(df['market'].value_counts())
top10_mkt

plt.figure(figsize=(10,10))
sns.catplot(x='market',hue='status',data=df[df['market'].isin(top10_mkt)],
            kind="count",
            order = df['market'][df['market'].isin(top10_mkt)].value_counts().i
plt.xticks(rotation = 65)
```

Out[136]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 [Text(0, 0, 'other'),
  Text(1, 0, 'Software'),
  Text(2, 0, 'Curated Web'),
  Text(3, 0, 'Mobile'),
  Text(4, 0, 'Enterprise Software'),
  Text(5, 0, 'Biotechnology'),
  Text(6, 0, 'Advertising'),
  Text(7, 0, 'Games'),
  Text(8, 0, 'E-Commerce'),
  Text(9, 0, 'Social Media'),
  Text(10, 0, 'Hardware + Software')])
<Figure size 720x720 with 0 Axes>
```

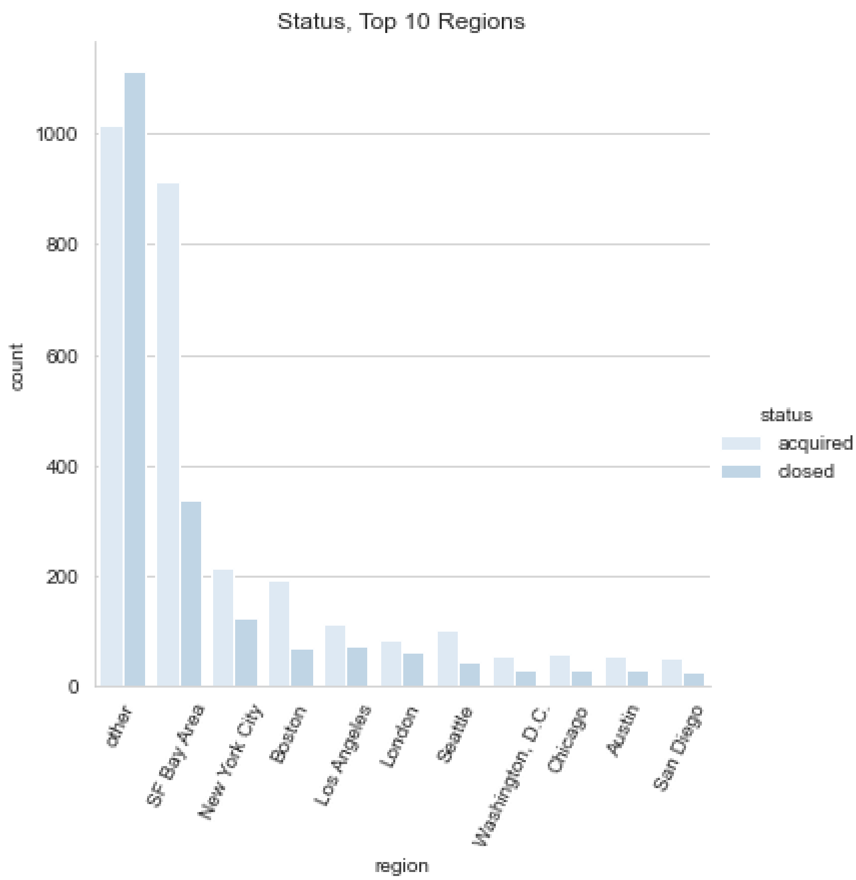


In [137…
```
print(df['region'].value_counts()[1:11])
top10_reg = list(set(df['region'][df['region'].map(df['region'].value_counts())
top10_reg

# fig= plt.figure(figsize=(10,5))
# fig, ax = plt.subplots(figsize=(10, 5))
# ax.set_xlabel("Region", fontsize=10)
# ax.set_ylabel("Count", fontsize=10)
# fig.suptitle("Status, Top 10 Regions", fontsize=15)

sns.catplot(x='region',hue='status',data=df[df['region'].isin(top10_reg)],
            kind="count",
            order = df['region'][df['region'].isin(top10_reg)].value_counts().i
plt.title('Status, Top 10 Regions')
plt.xticks(rotation = 65)
```

```
SF Bay Area          1250
New York City         338
Boston                260
Los Angeles           186
London                146
Seattle               143
Washington, D.C.       85
Chicago                85
Austin                 84
San Diego              77
Name: region, dtype: int64
```

Out[137]:

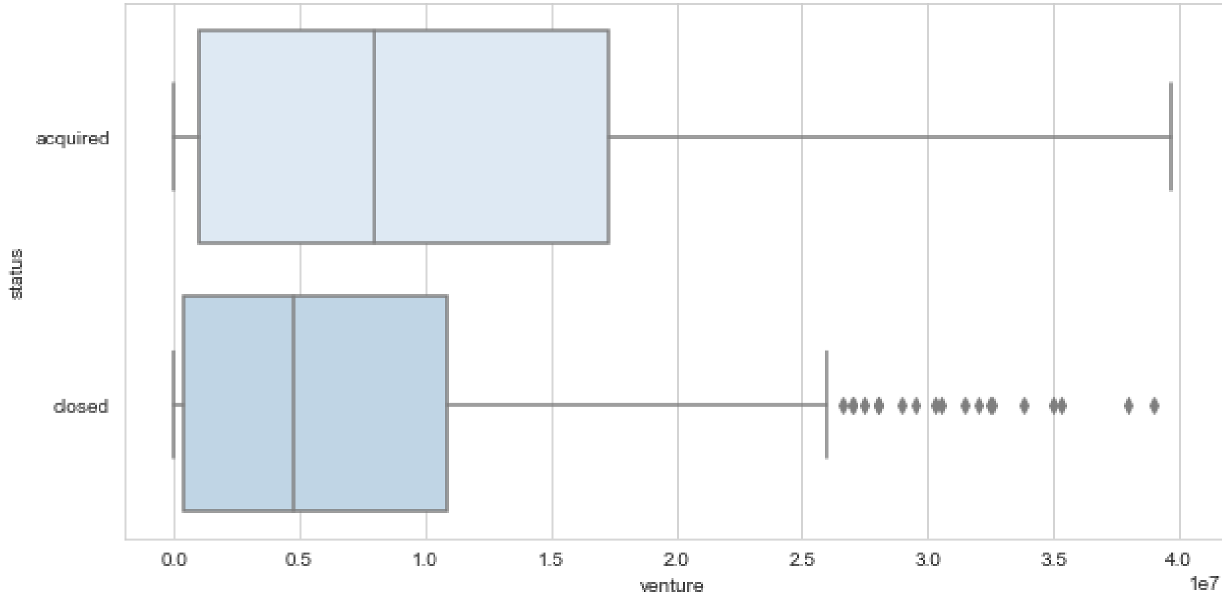
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 [Text(0, 0, 'other'),
  Text(1, 0, 'SF Bay Area'),
  Text(2, 0, 'New York City'),
  Text(3, 0, 'Boston'),
  Text(4, 0, 'Los Angeles'),
  Text(5, 0, 'London'),
  Text(6, 0, 'Seattle'),
  Text(7, 0, 'Washington, D.C.'),
  Text(8, 0, 'Chicago'),
  Text(9, 0, 'Austin'),
  Text(10, 0, 'San Diego')])
```

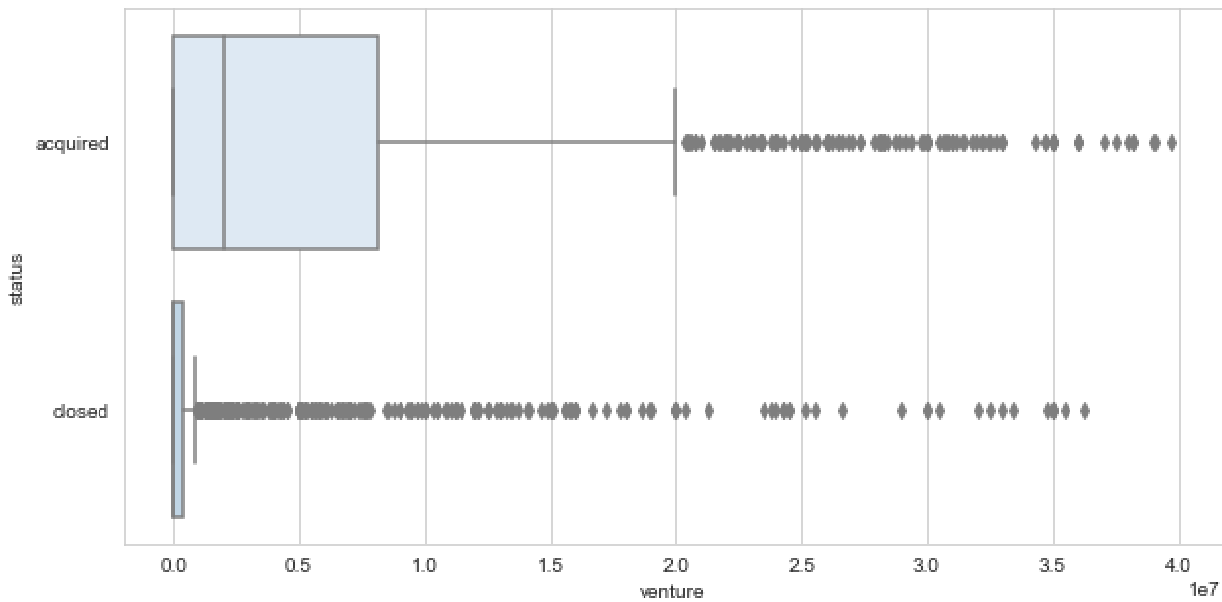


In [138…

```
plt.figure(figsize=(10,5))
sns.boxplot(x=df_box['venture'][df_box['founded_year'] < 2005],
            y=df_box['status'][df_box['founded_year'] < 2005])

plt.figure(figsize=(10,5))
sns.boxplot(x=df_box['venture'][df_box['founded_year'] >= 2005],
            y=df_box['status'][df_box['founded_year'] >= 2005])
```

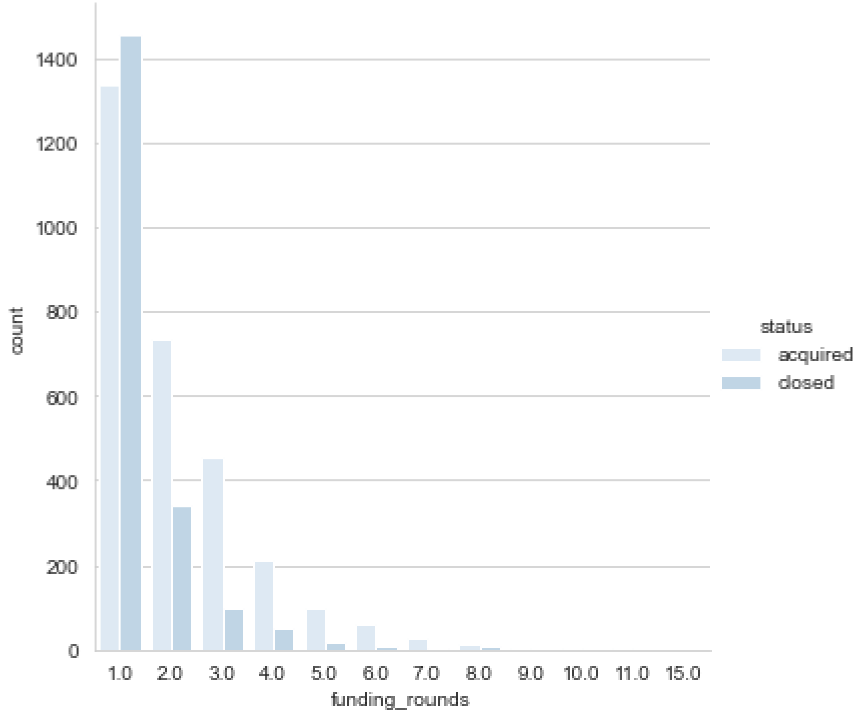
Out[138]: <AxesSubplot:xlabel='venture', ylabel='status'>





In [139…
```
# number of funding rounds
df['funding_rounds'].value_counts()

# acquired or closed by number of funding rounds

sns.catplot(x='funding_rounds',hue='status', data=df, kind="count")
```

Out[139]: <seaborn.axisgrid.FacetGrid at 0x7fc7555bef10>

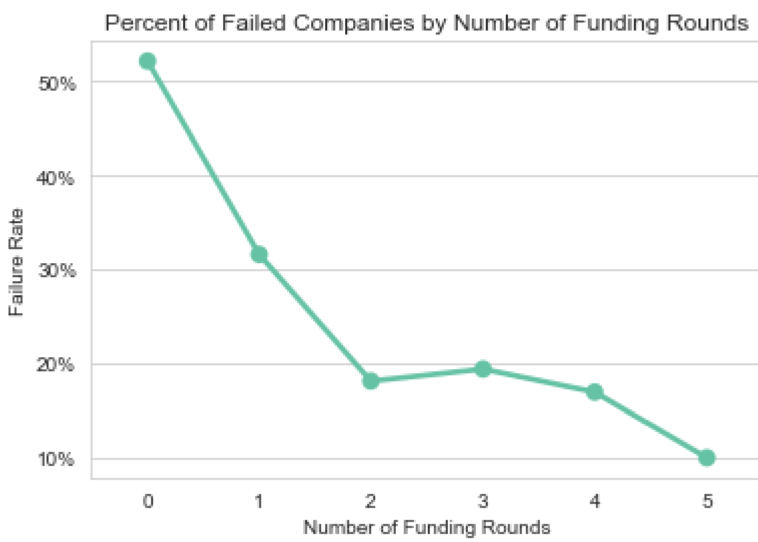


In [140…

```
encoder = LabelEncoder()

df_fr = df[['funding_rounds', 'status']]
df_fr['cat'] = pd.Series(encoder.fit_transform(df['status']))

df_fr

#df_fr.groupby('status').mean()
#df_fr.groupby('funding_rounds').sum()
df_fr_group = pd.DataFrame(df_fr[['funding_rounds', 'cat']].groupby('funding_ro

df_fr_group = df_fr_group.droplevel(axis=1, level=0).reset_index()
df_fr_group['pct_fail'] = df_fr_group['sum'] / df_fr_group['count']

display(df_fr_group)

sns.set_style("whitegrid")
sns.set_palette('Set2')
g = sns.pointplot(data=df_fr_group[:6], x="funding_rounds", y="pct_fail")
g.set(xlabel='Number of Funding Rounds',
      ylabel='Failure Rate',
      title='Percent of Failed Companies by Number of Funding Rounds')

vals = g.get_yticks()
g.set_yticklabels(['{:,.0%}'.format(x) for x in vals])

vals = g.get_xticks()
g.set_xticklabels(['{:,.0f}'.format(x) for x in vals])
```

	funding_rounds	count	sum	pct_fail
0	1.0	2793	1456	0.521303
1	2.0	1073	339	0.315937
2	3.0	552	100	0.181159
3	4.0	263	51	0.193916
4	5.0	118	20	0.169492
5	6.0	70	7	0.100000
6	7.0	32	5	0.156250
7	8.0	19	8	0.421053
8	9.0	5	0	0.000000
9	10.0	2	0	0.000000
10	11.0	2	0	0.000000
11	15.0	1	0	0.000000

Out[140]:
```
[Text(0, 0, '0'),
 Text(1, 0, '1'),
 Text(2, 0, '2'),
 Text(3, 0, '3'),
 Text(4, 0, '4'),
 Text(5, 0, '5')]
```



In [141…
```
df[df['funding_rounds']==8]
```

Out[141]:

	name	market	funding_total_usd	status	country_code	state_code	reg
147	Aggregate Knowledge	Advertising	7.043011e+11	acquired	USA	CA	SF A
258	Appia	Advertising	4.042050e+11	acquired	USA	NC	ot
505	Biolex Therapeutics	Biotechnology	1.702206e+12	closed	USA	NC	ot
798	CipherMax	Security	1.401504e+12	closed	USA	CA	SF A
848	Cloudant	Enterprise Software	1.082050e+11	acquired	USA	MA	Bos
985	Cozi Group	other	2.095057e+11	closed	USA	WA	Sea
1147	Dilithium Networks	Mobile	9.096070e+11	closed	USA	CA	ot
1416	Extreme Enterprises	Enterprise Software	1.405500e+09	closed	USA	FL	ot
1523	Flurry	Mobile	7.032055e+11	acquired	USA	CA	SF A
2295	Laszlo Systems	other	3.098090e+11	acquired	USA	CA	SF A
2664	Moblyng	Games	1.091030e+11	closed	USA	CA	SF A
3136	PayScale	other	3.033086e+11	acquired	USA	WA	Sea
3253	Plextronics	Clean Technology	5.014047e+11	acquired	USA	PA	ot
3927	Socialtext	Enterprise Software	4.067070e+11	acquired	USA	CA	SF A
3941	SolFocus	Clean Technology	2.101400e+12	closed	USA	CA	SF A
3951	Solyndra	Manufacturing	1.056075e+14	closed	USA	CA	SF A
4602	Virident Systems	other	1.402303e+12	acquired	USA	CA	SF A
4793	Xobni	Software	4.017052e+11	acquired	USA	CA	SF A
4860	Zappos	Curated Web	6.027050e+11	acquired	USA	NV	ot

19 rows × 33 columns

In [142…

```
df_fr.groupby('status').mean()
#df_fr.groupby('funding_rounds').sum()
```

Out[142]:

status	funding_rounds	cat
acquired	2.125000	0
closed	1.449648	1

In [143…

```
df_time = df[['status', 'time_to_first_funding']]
df_time.groupby('status').mean()
```

Out[143]:

status	time_to_first_funding
acquired	3.740585
closed	1.685871

In [144…

```
sns.set_style("whitegrid")
sns.set_palette('Set2')

#category_order = ["U.S.", "International"]

g = sns.catplot(x='international',hue='status', data=df, kind="count")
g.set_xticklabels(["U.S.", "International"])
g.set(xlabel='Location', ylabel='Company Count')
g.fig.suptitle("Companies by Status & Location", y=1.02);
```

In [145…

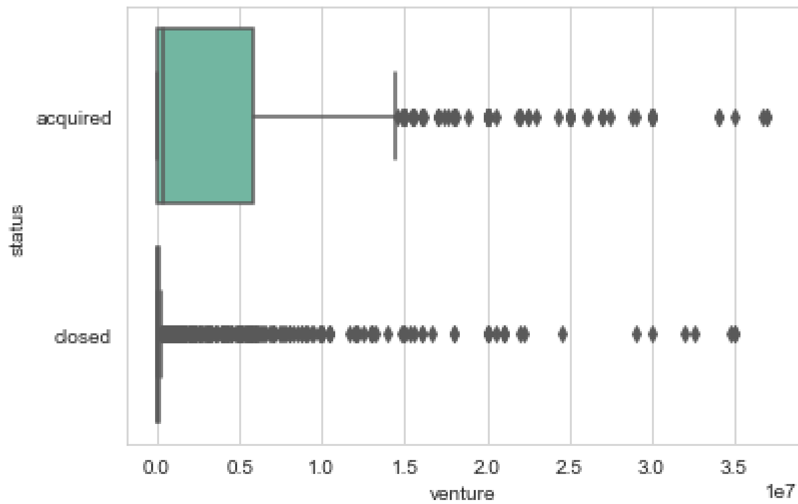
```
# ## region other in train / test, need to add to df

# sns.set_style("whitegrid")
# sns.set_palette('Set2')

# #category_order = ["U.S.", "International"]
```

```
# g = sns.catplot(x='region_other',hue='status', data=df, kind="count")
# g.set_xticklabels(["Top -- Region", "Other Region"])
# g.set(xlabel='Location', ylabel='Company Count')
# g.fig.suptitle("Companies by Status & Region", y=1.02);
```

In [146…

```
# more than one funding round
df_ofr = df_box[df_box['one_funding_round']=='one']
df_mfr = df_box[df_box['one_funding_round']=='multiple']
sns.boxplot(x='venture', y='status', data=df_ofr)
```

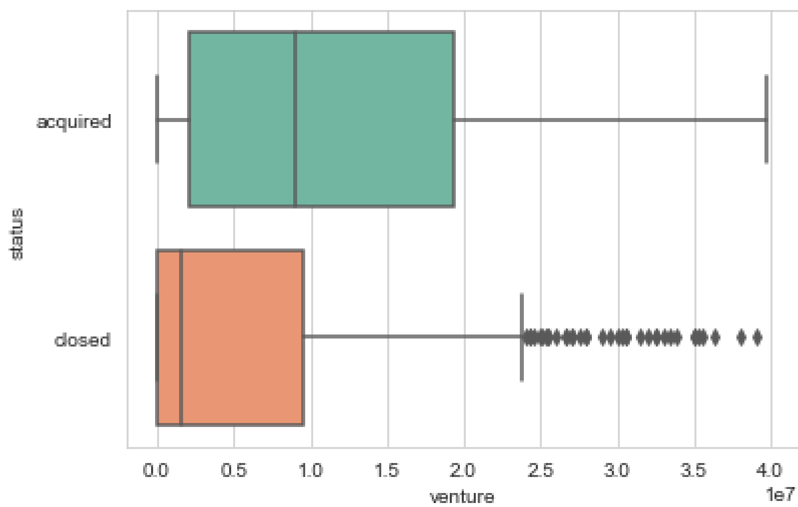
Out[146]: <AxesSubplot:xlabel='venture', ylabel='status'>



In [147…

```
sns.boxplot(x='venture', y='status', data=df_mfr)
```

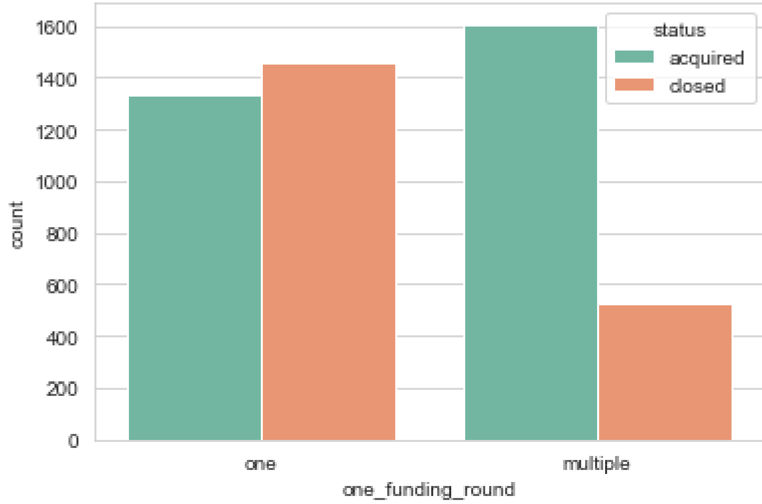
Out[147]: <AxesSubplot:xlabel='venture', ylabel='status'>



In [148…

```
sns.countplot(x='one_funding_round',hue='status', data=df, orient='v')
```

Out[148]: <AxesSubplot:xlabel='one_funding_round', ylabel='count'>



In [149…

```
### need something with funding_total_usd, time to first funding
```