National Charitable Organisation Project

[1. Import Libraries]\ [2. Import Data]\ [3. Data Description & Exploration]

- Data Shape
- Columns & dtype
- Peek at the Data
- Missing Values %
- Statistics
- Correlations (numeric variables)
- Data Distributions (Histograms)

[4. Data Preparation]

- Dropped Columns
- Handle Missing Values
- Prepare Data for Scaling
- Scaling: StandardScaler()
- Scaling: RobustScaler()

[5. About Algorithms]

- Set-up the test harness to use 10-fold cross validation
- Build Models:
  - Logistic Regression (LR)
  - Linear Discriminant Analysis (LDA)
  - K-Nearest Neighbors (KNN)
  - Classification and Regression Trees (CART)
  - Random Forest Decision Tree (RFTree)
  - Gradient Boosting (GrB)
  - Gaussian Naive Bayes (NB)
  - Support Vector Machines (SVM)
  - Deep Learning (Deep)

[6. Evaluate Some Algorithms]

- 6.1.a Create a Validation Dataset with Standard Scaled Data
- 6.2.a Build Models, Make and Evaluate Predictions on different models with Standard Scaled Data
- 6.1.b Create a Validation Dataset with Robust Scaled Data
- 6.2.b Build Models, Make and Evaluate Predictions on different models with Robust Scaled Data
- 6.3 Choose the best model

[7. Train the Final Machine Learning Model]\ [8. Save the Load Final Machine Learning Model]\ [9. Data Preparation of New Data]\ [10. Make Predictions]

1. Import Libraries

Versions of the Libraries

In [1]:
```
# Python version
import sys
print('Python: {}'.format(sys.version))
# numpy
import numpy
print('numpy: {}'.format(numpy.__version__))
# pandas
import pandas
print('pandas: {}'.format(pandas.__version__))
# scipy
import scipy
print('scipy: {}'.format(scipy.__version__))
# matplotlib
import matplotlib
print('matplotlib: {}'.format(matplotlib.__version__))
# seaborn
import seaborn
print('seaborn: {}'.format(seaborn.__version__))
# scikit-learn
import sklearn
print('sklearn: {}'.format(sklearn.__version__))
# joblib
import joblib
print('joblib: {}'.format(joblib.__version__))
```

```
Python: 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
numpy: 1.18.1
pandas: 1.0.1
scipy: 1.4.1
matplotlib: 3.1.3
seaborn: 0.10.0
sklearn: 0.22.1
joblib: 0.14.1
```

- Python: 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
- numpy: 1.18.1
- pandas: 1.0.1
- scipy: 1.4.1
- matplotlib: 3.1.3
- seaborn: 0.10.0
- sklearn: 0.22.1
- joblib: 0.14.1

Import Libraries

In [2]:
```
import numpy as np
import pandas as pd
from pandas import read_csv
from pandas.plotting import scatter_matrix
pd.set_option('display.max_columns', None) # Set it to None to display all columns in
the dataframe
pd.set_option('display.width', None) # Width of the display in characters. If set to
 None and pandas will correctly auto-detect the width
pd.set_option('display.max_colwidth', None) # The maximum width in characters of a co
lumn in the repr of a pandas data structure
pd.options.mode.chained_assignment = None # switch off pandas warning
from scipy import stats  # library of statistical functions
import matplotlib.pyplot as plt
%matplotlib inline
#plt.rcParams['figure.figsize'] = (14,12) # to change the charts size
import seaborn as sns      # for drawing attractive and informative statistical graph
ics
import time
from sklearn import metrics
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
import joblib
```

Correlations conditional formatting

In [3]:
```
def correlations_conditional_formatting(value):
  """
  Colors elements in a dateframe
  green if positive and red if
  negative. Does not color NaN
  values.
  """

  if value < -0.9:
    color = 'red'
  elif value > 0.9:
    color = 'green'
  else:
    color = 'gainsboro'

  return 'color: %s' % color
```

[Table of Content](#)

2. Import Data

In [3]:
```
donor_data = pd.read_csv('Donor Raw Data_ML with Python.csv')              # Historica
l Data
prospective_data=pd.read_csv('Prospective Donor_ML with Python.csv')      # New Conta
ct List
```

Table of Content

3. Data Description & Exploration

Data shape

In [5]:
```
print('Donor Shape (Historical Data) : ',donor_data.shape)                   # Histori
cal Data
print('Prospective Shape (New Contact List) : ',prospective_data.shape)     # New Con
tact List
```

```
Donor Shape (Historical Data) :  (19372, 50)
Prospective Shape (New Contact List) :  (2148, 48)
```

Donor Columns and Data Types

In [6]:
```
print()
print('|Column Name Donors            |Type     |','\n')
donor_data.dtypes
```

```
|Column Name Donors            |Type     |
```

Out[6]:
```
TARGET_B                          int64
TARGET_D                        float64
CONTROL_NUMBER                    int64
MONTHS_SINCE_ORIGIN               int64
DONOR_AGE                       float64
IN_HOUSE                          int64
URBANICITY                       object
SES                              object
CLUSTER_CODE                     object
HOME_OWNER                       object
DONOR_GENDER                     object
INCOME_GROUP                    float64
PUBLISHED_PHONE                   int64
OVERLAY_SOURCE                   object
MOR_HIT_RATE                      int64
WEALTH_RATING                   float64
MEDIAN_HOME_VALUE                 int64
MEDIAN_HOUSEHOLD_INCOME           int64
PCT_OWNER_OCCUPIED                int64
PER_CAPITA_INCOME                 int64
PCT_ATTRIBUTE1                    int64
PCT_ATTRIBUTE2                    int64
PCT_ATTRIBUTE3                    int64
PCT_ATTRIBUTE4                    int64
PEP_STAR                          int64
RECENT_STAR_STATUS                int64
RECENCY_STATUS_96NK              object
FREQUENCY_STATUS_97NK             int64
RECENT_RESPONSE_PROP            float64
RECENT_AVG_GIFT_AMT             float64
RECENT_CARD_RESPONSE_PROP       float64
RECENT_AVG_CARD_GIFT_AMT        float64
RECENT_RESPONSE_COUNT             int64
RECENT_CARD_RESPONSE_COUNT        int64
MONTHS_SINCE_LAST_PROM_RESP     float64
LIFETIME_CARD_PROM                int64
LIFETIME_PROM                     int64
LIFETIME_GIFT_AMOUNT            float64
LIFETIME_GIFT_COUNT               int64
LIFETIME_AVG_GIFT_AMT           float64
LIFETIME_GIFT_RANGE             float64
LIFETIME_MAX_GIFT_AMT           float64
LIFETIME_MIN_GIFT_AMT           float64
LAST_GIFT_AMT                   float64
CARD_PROM_12                      int64
NUMBER_PROM_12                    int64
MONTHS_SINCE_LAST_GIFT            int64
MONTHS_SINCE_FIRST_GIFT           int64
FILE_AVG_GIFT                   float64
FILE_CARD_GIFT                    int64
dtype: object
```

Prospective Columns and Data Types

In [7]:
```
print()
print('|Column Name Prospective         |Type     |','\n')
prospective_data.dtypes
```

```
|Column Name Prospective         |Type     |
```

Out[7]:
```
CONTROL_NUMBER                     int64
MONTHS_SINCE_ORIGIN                int64
DONOR_AGE                        float64
IN_HOUSE                           int64
URBANICITY                        object
SES                               object
CLUSTER_CODE                      object
HOME_OWNER                        object
DONOR_GENDER                      object
INCOME_GROUP                     float64
PUBLISHED_PHONE                    int64
OVERLAY_SOURCE                    object
MOR_HIT_RATE                       int64
WEALTH_RATING                    float64
MEDIAN_HOME_VALUE                  int64
MEDIAN_HOUSEHOLD_INCOME            int64
PCT_OWNER_OCCUPIED                 int64
PER_CAPITA_INCOME                  int64
PCT_ATTRIBUTE1                     int64
PCT_ATTRIBUTE2                     int64
PCT_ATTRIBUTE3                     int64
PCT_ATTRIBUTE4                     int64
PEP_STAR                           int64
RECENT_STAR_STATUS                 int64
RECENCY_STATUS_96NK               object
FREQUENCY_STATUS_97NK              int64
RECENT_RESPONSE_PROP             float64
RECENT_AVG_GIFT_AMT              float64
RECENT_CARD_RESPONSE_PROP        float64
RECENT_AVG_CARD_GIFT_AMT         float64
RECENT_RESPONSE_COUNT              int64
RECENT_CARD_RESPONSE_COUNT         int64
MONTHS_SINCE_LAST_PROM_RESP      float64
LIFETIME_CARD_PROM                 int64
LIFETIME_PROM                      int64
LIFETIME_GIFT_AMOUNT             float64
LIFETIME_GIFT_COUNT                int64
LIFETIME_AVG_GIFT_AMT            float64
LIFETIME_GIFT_RANGE              float64
LIFETIME_MAX_GIFT_AMT            float64
LIFETIME_MIN_GIFT_AMT            float64
LAST_GIFT_AMT                    float64
CARD_PROM_12                       int64
NUMBER_PROM_12                     int64
MONTHS_SINCE_LAST_GIFT             int64
MONTHS_SINCE_FIRST_GIFT            int64
FILE_AVG_GIFT                    float64
FILE_CARD_GIFT                     int64
dtype: object
```

Peek at the Data

In [8]:
```
print('\n','Donor Raw Data_ML with Python.csv       HEAD')
donor_data.head()
```

```
Donor Raw Data_ML with Python.csv       HEAD
```

Out[8]:

	TARGET_B	TARGET_D	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBA
0	0	NaN	5	101	87.0	0	
1	1	10.0	12	137	79.0	0	
2	0	NaN	37	113	75.0	0	
3	0	NaN	38	92	NaN	0	
4	0	NaN	41	101	74.0	0	

In [9]:
```
print('\n','Donor Raw Data_ML with Python.csv       TAIL')
donor_data.tail()
```

```
Donor Raw Data_ML with Python.csv       TAIL
```

Out[9]:

	TARGET_B	TARGET_D	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE
19367	0	NaN	191687	89	66.0	1
19368	0	NaN	191710	137	77.0	1
19369	0	NaN	191746	29	NaN	1
19370	0	NaN	191775	129	78.0	1
19371	1	150.0	191779	29	70.0	0

In [10]:
```
print('\n','Prospective Raw Data_ML with Python.csv       HEAD')
prospective_data.head()
```

```
Prospective Raw Data_ML with Python.csv       HEAD
```

Out[10]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER
0	139	101	NaN	0	R	2	
1	142	137	NaN	0	R	2	
2	282	17	30.0	0	T	1	
3	368	137	75.0	0	U	1	
4	387	5	NaN	0	T	2	

Donor Raw Data_ML with Python **(Historical Data)** and Prospective Donor_ML with Python **(New Contact List)** have practically the same structure.

In Prospective Donor_ML with Python **(New Contact List)** are missing two columns:

- TARGET_B: which it will be created and named "Prediction" and populated with the model predictions
- TARGET_D: which it will be dropped also in the historical data

Missing Values

In [11]:

```python
# missing value counts in each of these columns
miss = donor_data.isnull().sum()/len(donor_data)
miss = miss[miss > 0]
miss.sort_values(ascending=False,inplace=True)
# miss
# convert 'miss' from Series to DataFrame
miss_DF= miss.to_frame()
miss_DF
```

Out[11]:

	0
TARGET_D	0.750000
WEALTH_RATING	0.454780
DONOR_AGE	0.247522
INCOME_GROUP	0.226719
MONTHS_SINCE_LAST_PROM_RESP	0.012699

In [12]:
```
#count of missing values
print('|Column Name Donor            |Type    |','\n')
donor_data.isna().sum()
```

```
|Column Name Donor            |Type    |
```

Out[12]:
```
TARGET_B                          0
TARGET_D                      14529
CONTROL_NUMBER                    0
MONTHS_SINCE_ORIGIN               0
DONOR_AGE                      4795
IN_HOUSE                          0
URBANICITY                        0
SES                               0
CLUSTER_CODE                      0
HOME_OWNER                        0
DONOR_GENDER                      0
INCOME_GROUP                   4392
PUBLISHED_PHONE                   0
OVERLAY_SOURCE                    0
MOR_HIT_RATE                      0
WEALTH_RATING                  8810
MEDIAN_HOME_VALUE                 0
MEDIAN_HOUSEHOLD_INCOME           0
PCT_OWNER_OCCUPIED                0
PER_CAPITA_INCOME                 0
PCT_ATTRIBUTE1                    0
PCT_ATTRIBUTE2                    0
PCT_ATTRIBUTE3                    0
PCT_ATTRIBUTE4                    0
PEP_STAR                          0
RECENT_STAR_STATUS                0
RECENCY_STATUS_96NK               0
FREQUENCY_STATUS_97NK             0
RECENT_RESPONSE_PROP              0
RECENT_AVG_GIFT_AMT               0
RECENT_CARD_RESPONSE_PROP         0
RECENT_AVG_CARD_GIFT_AMT          0
RECENT_RESPONSE_COUNT             0
RECENT_CARD_RESPONSE_COUNT        0
MONTHS_SINCE_LAST_PROM_RESP     246
LIFETIME_CARD_PROM                0
LIFETIME_PROM                     0
LIFETIME_GIFT_AMOUNT              0
LIFETIME_GIFT_COUNT               0
LIFETIME_AVG_GIFT_AMT             0
LIFETIME_GIFT_RANGE               0
LIFETIME_MAX_GIFT_AMT             0
LIFETIME_MIN_GIFT_AMT             0
LAST_GIFT_AMT                     0
CARD_PROM_12                      0
NUMBER_PROM_12                    0
MONTHS_SINCE_LAST_GIFT            0
MONTHS_SINCE_FIRST_GIFT           0
FILE_AVG_GIFT                     0
FILE_CARD_GIFT                    0
dtype: int64
```

In [13]:
```python
print('|Column Name Prospective        |Type     |','\n')
prospective_data.isna().sum()
```

```
|Column Name Prospective        |Type     |
```

Out[13]:
```
CONTROL_NUMBER                   0
MONTHS_SINCE_ORIGIN              0
DONOR_AGE                      529
IN_HOUSE                         0
URBANICITY                       0
SES                              0
CLUSTER_CODE                     0
HOME_OWNER                       0
DONOR_GENDER                     0
INCOME_GROUP                   481
PUBLISHED_PHONE                  0
OVERLAY_SOURCE                   0
MOR_HIT_RATE                     0
WEALTH_RATING                 1006
MEDIAN_HOME_VALUE                0
MEDIAN_HOUSEHOLD_INCOME          0
PCT_OWNER_OCCUPIED               0
PER_CAPITA_INCOME                0
PCT_ATTRIBUTE1                   0
PCT_ATTRIBUTE2                   0
PCT_ATTRIBUTE3                   0
PCT_ATTRIBUTE4                   0
PEP_STAR                         0
RECENT_STAR_STATUS               0
RECENCY_STATUS_96NK              0
FREQUENCY_STATUS_97NK            0
RECENT_RESPONSE_PROP             0
RECENT_AVG_GIFT_AMT              0
RECENT_CARD_RESPONSE_PROP        0
RECENT_AVG_CARD_GIFT_AMT         0
RECENT_RESPONSE_COUNT            0
RECENT_CARD_RESPONSE_COUNT       0
MONTHS_SINCE_LAST_PROM_RESP     26
LIFETIME_CARD_PROM               0
LIFETIME_PROM                    0
LIFETIME_GIFT_AMOUNT             0
LIFETIME_GIFT_COUNT              0
LIFETIME_AVG_GIFT_AMT            0
LIFETIME_GIFT_RANGE              0
LIFETIME_MAX_GIFT_AMT            0
LIFETIME_MIN_GIFT_AMT            0
LAST_GIFT_AMT                    0
CARD_PROM_12                     0
NUMBER_PROM_12                   0
MONTHS_SINCE_LAST_GIFT           0
MONTHS_SINCE_FIRST_GIFT          0
FILE_AVG_GIFT                    0
FILE_CARD_GIFT                   0
dtype: int64
```

Statistics

In [3]:
```python
# statistics

donor_statistics = donor_data.describe()
```

In [4]:
```python
donor_statistics
```

Out[4]:

	TARGET_B	TARGET_D	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOU
count	19372.000000	4843.000000	19372.000000	19372.000000	14577.000000	19372.000
mean	0.250000	15.624344	96546.225377	73.409973	58.919051	0.073
std	0.433024	12.445137	55830.643871	41.255574	16.669382	0.260
min	0.000000	1.000000	5.000000	5.000000	0.000000	0.000
25%	0.000000	10.000000	48289.000000	29.000000	47.000000	0.000
50%	0.000000	13.000000	96937.000000	65.000000	60.000000	0.000
75%	0.250000	20.000000	145429.500000	113.000000	73.000000	0.000
max	1.000000	200.000000	191779.000000	137.000000	87.000000	1.000

In [9]:
```python
donor_statistics.to_csv('donor_statistics.csv')
```

In [5]:
```python
donor_statistics_transposed = donor_statistics.transpose()
```

In [6]:

```
donor_statistics_transposed
```

Out[6]:

	count	mean	std	min	25%	50%	
TARGET_B	19372.0	0.250000	0.433024	0.00	0.000	0.000	
TARGET_D	4843.0	15.624344	12.445137	1.00	10.000	13.000	
CONTROL_NUMBER	19372.0	96546.225377	55830.643871	5.00	48289.000	96937.000	1
MONTHS_SINCE_ORIGIN	19372.0	73.409973	41.255574	5.00	29.000	65.000	
DONOR_AGE	14577.0	58.919051	16.669382	0.00	47.000	60.000	
IN_HOUSE	19372.0	0.073198	0.260469	0.00	0.000	0.000	
INCOME_GROUP	14980.0	3.907543	1.864796	1.00	2.000	4.000	
PUBLISHED_PHONE	19372.0	0.497729	0.500008	0.00	0.000	0.000	
MOR_HIT_RATE	19372.0	3.361656	9.503481	0.00	0.000	0.000	
WEALTH_RATING	10562.0	5.005397	2.815386	0.00	3.000	5.000	
MEDIAN_HOME_VALUE	19372.0	1079.871929	960.753448	0.00	518.000	747.000	
MEDIAN_HOUSEHOLD_INCOME	19372.0	341.970215	164.207807	0.00	232.000	311.000	
PCT_OWNER_OCCUPIED	19372.0	69.698999	21.711019	0.00	60.000	76.000	
PER_CAPITA_INCOME	19372.0	15857.334452	8710.630390	0.00	10869.000	13816.500	
PCT_ATTRIBUTE1	19372.0	1.029011	4.918297	0.00	0.000	0.000	
PCT_ATTRIBUTE2	19372.0	30.573921	11.421471	0.00	25.000	31.000	
PCT_ATTRIBUTE3	19372.0	29.603293	15.120360	0.00	20.000	29.000	
PCT_ATTRIBUTE4	19372.0	32.852467	17.839765	0.00	21.000	32.000	
PEP_STAR	19372.0	0.504439	0.499993	0.00	0.000	1.000	
RECENT_STAR_STATUS	19372.0	0.931138	2.545585	0.00	0.000	0.000	
FREQUENCY_STATUS_97NK	19372.0	1.983998	1.099346	1.00	1.000	2.000	
RECENT_RESPONSE_PROP	19372.0	0.190127	0.113947	0.00	0.105	0.167	
RECENT_AVG_GIFT_AMT	19372.0	15.365396	10.167485	0.00	10.000	14.000	
RECENT_CARD_RESPONSE_PROP	19372.0	0.230808	0.186230	0.00	0.100	0.200	
RECENT_AVG_CARD_GIFT_AMT	19372.0	11.685470	10.834120	0.00	5.000	10.140	
RECENT_RESPONSE_COUNT	19372.0	3.043103	2.046401	0.00	2.000	3.000	
RECENT_CARD_RESPONSE_COUNT	19372.0	1.730539	1.535521	0.00	1.000	1.000	
MONTHS_SINCE_LAST_PROM_RESP	19126.0	19.038900	3.415559	-12.00	17.000	18.000	
LIFETIME_CARD_PROM	19372.0	18.668078	8.558778	2.00	11.000	18.000	
LIFETIME_PROM	19372.0	47.570514	22.950158	5.00	28.000	47.000	
LIFETIME_GIFT_AMOUNT	19372.0	104.425716	105.722460	15.00	42.000	79.000	
LIFETIME_GIFT_COUNT	19372.0	9.979765	8.688163	1.00	4.000	8.000	
LIFETIME_AVG_GIFT_AMT	19372.0	12.858338	8.787758	1.36	8.000	11.200	
LIFETIME_GIFT_RANGE	19372.0	11.587876	15.116893	0.00	5.000	10.000	
LIFETIME_MAX_GIFT_AMT	19372.0	19.208808	16.101128	5.00	12.000	16.000	
LIFETIME_MIN_GIFT_AMT	19372.0	7.620932	7.959786	0.00	3.000	5.000	
LAST_GIFT_AMT	19372.0	16.584199	11.977558	0.00	10.000	15.000	
CARD_PROM_12	19372.0	5.367128	1.264205	0.00	5.000	6.000	
NUMBER_PROM_12	19372.0	12.901869	4.642072	2.00	11.000	12.000	
MONTHS_SINCE_LAST_GIFT	19372.0	18.191152	4.033065	4.00	16.000	18.000	
MONTHS_SINCE_FIRST_GIFT	19372.0	69.482088	37.568169	15.00	33.000	65.000	

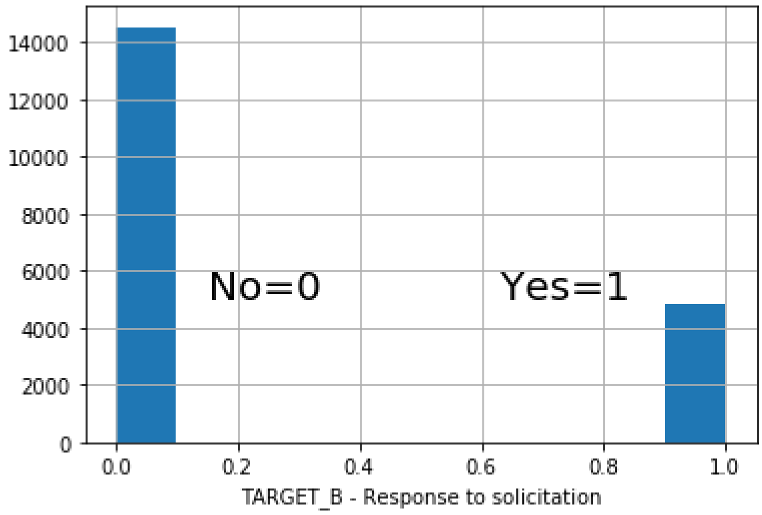
	count	mean	std	min	25%	50%
FILE_AVG_GIFT	19372.0	12.858338	8.787758	1.36	8.000	11.200
FILE_CARD_GIFT	19372.0	5.273591	4.607063	0.00	2.000	4.000

In [7]:
```
donor_statistics_transposed.to_csv('donor_statistics_transposed.csv')
```

In [4]:
```
#skewness
print ("The skewness of TARGET_B is {:.2f}".format(donor_data['TARGET_B'].skew()))
```

```
The skewness of TARGET_B is 1.15
```

In [20]:
```
donor_data['TARGET_B'].hist()
plt.xlabel('TARGET_B - Response to solicitation')
plt.text(0.15, 5000, 'No=0                  Yes=1',dict(size=20))
plt.show()
```



separate variables into numeric and categorical data

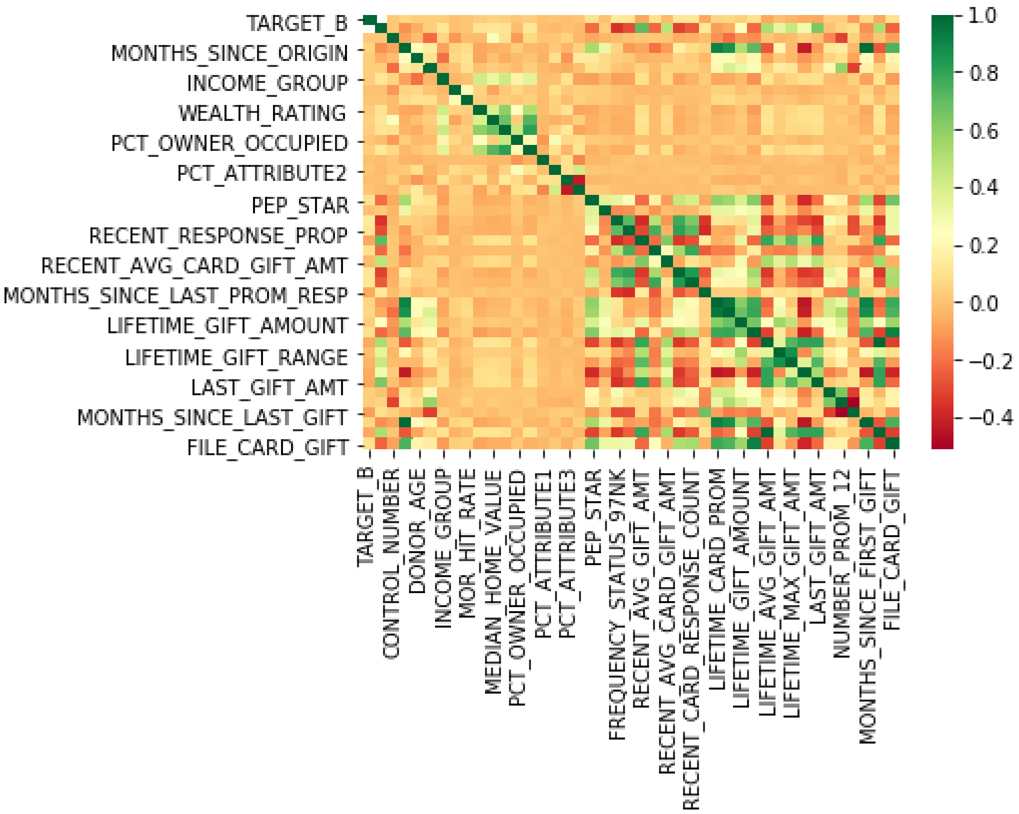
In [18]:
```
#numeric data
numeric_data = donor_data.select_dtypes(include=[np.number])

#categorical data
cat_data = donor_data.select_dtypes(exclude=[np.number])

print ("There are {} numeric and {} categorical columns in donor data (historical dat
a).".
       format(numeric_data.shape[1],cat_data.shape[1]))
```

```
There are 43 numeric and 7 categorical columns in donor data (historical data).
```

correlations before Data Preparation

In [19]:
```
#correlation plot
corr_before_Data_Preparation = numeric_data.corr()
sns.heatmap(corr_before_Data_Preparation, cmap='RdYlGn')
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1ebda1984c8>



In [20]:

```
corr_before_Data_Preparation.style.applymap(correlations_conditional_formatting)
```

Out[20]:

	TARGET_B	TARGET_D	CONTROL_NUMBER	MONTHS_SINCE_ORIGI
TARGET_B	1.000000	nan	0.024607	0.06279
TARGET_D	nan	1.000000	0.021113	-0.12685
CONTROL_NUMBER	0.024607	0.021113	1.000000	-0.07787
MONTHS_SINCE_ORIGIN	0.062795	-0.126858	-0.077874	1.00000
DONOR_AGE	0.036949	-0.056139	-0.007441	0.23617
IN_HOUSE	0.040964	0.038842	-0.213938	0.15596
INCOME_GROUP	0.039932	0.126074	0.062979	-0.08244
PUBLISHED_PHONE	-0.003219	-0.002797	-0.110440	0.06562
MOR_HIT_RATE	0.012689	0.012679	-0.014747	0.07895
WEALTH_RATING	0.034742	0.114298	-0.019119	-0.07519
MEDIAN_HOME_VALUE	0.050377	0.126180	0.258975	-0.04749
MEDIAN_HOUSEHOLD_INCOME	0.038190	0.118255	0.104717	-0.03708
PCT_OWNER_OCCUPIED	0.015720	-0.007265	-0.099438	0.03622
PER_CAPITA_INCOME	0.041528	0.135409	0.075765	-0.02553
PCT_ATTRIBUTE1	-0.003648	-0.005814	0.012211	-0.03577
PCT_ATTRIBUTE2	0.008815	-0.021628	0.018996	0.02604
PCT_ATTRIBUTE3	-0.010106	0.000733	0.051788	-0.03965
PCT_ATTRIBUTE4	0.010067	-0.019237	-0.023189	0.04887
PEP_STAR	0.105389	-0.215399	-0.067940	0.53429
RECENT_STAR_STATUS	-0.001475	0.036037	-0.030397	0.31866
FREQUENCY_STATUS_97NK	0.137343	-0.358655	-0.018009	0.05813
RECENT_RESPONSE_PROP	0.118343	-0.292625	0.002186	-0.10350
RECENT_AVG_GIFT_AMT	-0.074668	0.707250	0.016568	-0.07978
RECENT_CARD_RESPONSE_PROP	0.100902	-0.221056	0.019523	-0.19789
RECENT_AVG_CARD_GIFT_AMT	-0.016935	0.477654	0.011152	-0.09871
RECENT_RESPONSE_COUNT	0.128762	-0.312299	-0.052220	0.17136
RECENT_CARD_RESPONSE_COUNT	0.126241	-0.256911	-0.041656	0.08803
MONTHS_SINCE_LAST_PROM_RESP	-0.066744	0.133834	-0.004883	0.04881
LIFETIME_CARD_PROM	0.065585	-0.097675	-0.120422	0.91206
LIFETIME_PROM	0.067846	-0.057903	-0.207466	0.86034
LIFETIME_GIFT_AMOUNT	0.041378	0.247667	-0.113607	0.50998
LIFETIME_GIFT_COUNT	0.100018	-0.220247	-0.132218	0.71491
LIFETIME_AVG_GIFT_AMT	-0.067107	0.516724	0.011250	-0.26047
LIFETIME_GIFT_RANGE	-0.006354	0.338629	-0.030587	0.20525
LIFETIME_MAX_GIFT_AMT	-0.036990	0.443145	-0.011010	-0.01695
LIFETIME_MIN_GIFT_AMT	-0.062756	0.295783	0.035819	-0.42411
LAST_GIFT_AMT	-0.068220	0.645388	0.000237	-0.09920
CARD_PROM_12	0.038947	0.006996	-0.143549	0.13067
NUMBER_PROM_12	0.039967	0.064494	-0.316467	0.14914
MONTHS_SINCE_LAST_GIFT	-0.089854	0.090215	0.046103	-0.02765
MONTHS_SINCE_FIRST_GIFT	0.066514	-0.127522	-0.086687	0.98782

	TARGET_B	TARGET_D	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN
FILE_AVG_GIFT	-0.067107	0.516724	0.011250	-0.26047
FILE_CARD_GIFT	0.105552	-0.229592	-0.083132	0.74342

In [21]:
```
corr_before_Data_Preparation[(corr_before_Data_Preparation < -0.9) | (corr_before_Dat
a_Preparation > 0.9)].style.applymap(correlations_conditional_formatting)
```

Out[21]:

	TARGET_B	TARGET_D	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN
TARGET_B	1.000000	nan	nan	nan
TARGET_D	nan	1.000000	nan	nan
CONTROL_NUMBER	nan	nan	1.000000	nan
MONTHS_SINCE_ORIGIN	nan	nan	nan	1.00000
DONOR_AGE	nan	nan	nan	nan
IN_HOUSE	nan	nan	nan	nan
INCOME_GROUP	nan	nan	nan	nan
PUBLISHED_PHONE	nan	nan	nan	nan
MOR_HIT_RATE	nan	nan	nan	nan
WEALTH_RATING	nan	nan	nan	nan
MEDIAN_HOME_VALUE	nan	nan	nan	nan
MEDIAN_HOUSEHOLD_INCOME	nan	nan	nan	nan
PCT_OWNER_OCCUPIED	nan	nan	nan	nan
PER_CAPITA_INCOME	nan	nan	nan	nan
PCT_ATTRIBUTE1	nan	nan	nan	nan
PCT_ATTRIBUTE2	nan	nan	nan	nan
PCT_ATTRIBUTE3	nan	nan	nan	nan
PCT_ATTRIBUTE4	nan	nan	nan	nan
PEP_STAR	nan	nan	nan	nan
RECENT_STAR_STATUS	nan	nan	nan	nan
FREQUENCY_STATUS_97NK	nan	nan	nan	nan
RECENT_RESPONSE_PROP	nan	nan	nan	nan
RECENT_AVG_GIFT_AMT	nan	nan	nan	nan
RECENT_CARD_RESPONSE_PROP	nan	nan	nan	nan
RECENT_AVG_CARD_GIFT_AMT	nan	nan	nan	nan
RECENT_RESPONSE_COUNT	nan	nan	nan	nan
RECENT_CARD_RESPONSE_COUNT	nan	nan	nan	nan
MONTHS_SINCE_LAST_PROM_RESP	nan	nan	nan	nan
LIFETIME_CARD_PROM	nan	nan	nan	0.91206
LIFETIME_PROM	nan	nan	nan	nan
LIFETIME_GIFT_AMOUNT	nan	nan	nan	nan
LIFETIME_GIFT_COUNT	nan	nan	nan	nan
LIFETIME_AVG_GIFT_AMT	nan	nan	nan	nan
LIFETIME_GIFT_RANGE	nan	nan	nan	nan
LIFETIME_MAX_GIFT_AMT	nan	nan	nan	nan
LIFETIME_MIN_GIFT_AMT	nan	nan	nan	nan
LAST_GIFT_AMT	nan	nan	nan	nan
CARD_PROM_12	nan	nan	nan	nan
NUMBER_PROM_12	nan	nan	nan	nan
MONTHS_SINCE_LAST_GIFT	nan	nan	nan	nan
MONTHS_SINCE_FIRST_GIFT	nan	nan	nan	0.98782

	TARGET_B	TARGET_D	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN
FILE_AVG_GIFT	nan	nan	nan	nan
FILE_CARD_GIFT	nan	nan	nan	nan

In [22]:
```
corr_before_Data_Preparation.to_csv('corr_before_Data_Preparation.csv')
```

In [23]:
```
print (corr_before_Data_Preparation['TARGET_B'].sort_values(ascending=False))
```

```
TARGET_B                        1.000000
FREQUENCY_STATUS_97NK           0.137343
RECENT_RESPONSE_COUNT           0.128762
RECENT_CARD_RESPONSE_COUNT      0.126241
RECENT_RESPONSE_PROP            0.118343
FILE_CARD_GIFT                  0.105552
PEP_STAR                        0.105389
RECENT_CARD_RESPONSE_PROP       0.100902
LIFETIME_GIFT_COUNT             0.100018
LIFETIME_PROM                   0.067846
MONTHS_SINCE_FIRST_GIFT         0.066514
LIFETIME_CARD_PROM              0.065585
MONTHS_SINCE_ORIGIN             0.062795
MEDIAN_HOME_VALUE               0.050377
PER_CAPITA_INCOME               0.041528
LIFETIME_GIFT_AMOUNT            0.041378
IN_HOUSE                        0.040964
NUMBER_PROM_12                  0.039967
INCOME_GROUP                    0.039932
CARD_PROM_12                    0.038947
MEDIAN_HOUSEHOLD_INCOME         0.038190
DONOR_AGE                       0.036949
WEALTH_RATING                   0.034742
CONTROL_NUMBER                  0.024607
PCT_OWNER_OCCUPIED              0.015720
MOR_HIT_RATE                    0.012689
PCT_ATTRIBUTE4                  0.010067
PCT_ATTRIBUTE2                  0.008815
RECENT_STAR_STATUS             -0.001475
PUBLISHED_PHONE                -0.003219
PCT_ATTRIBUTE1                 -0.003648
LIFETIME_GIFT_RANGE            -0.006354
PCT_ATTRIBUTE3                 -0.010106
RECENT_AVG_CARD_GIFT_AMT       -0.016935
LIFETIME_MAX_GIFT_AMT          -0.036990
LIFETIME_MIN_GIFT_AMT          -0.062756
MONTHS_SINCE_LAST_PROM_RESP    -0.066744
FILE_AVG_GIFT                  -0.067107
LIFETIME_AVG_GIFT_AMT          -0.067107
LAST_GIFT_AMT                  -0.068220
RECENT_AVG_GIFT_AMT            -0.074668
MONTHS_SINCE_LAST_GIFT         -0.089854
TARGET_D                             NaN
Name: TARGET_B, dtype: float64
```

Data Distributions (Histograms)

In [24]:
```
donor_data.hist(figsize=(22,18))
plt.show()
```

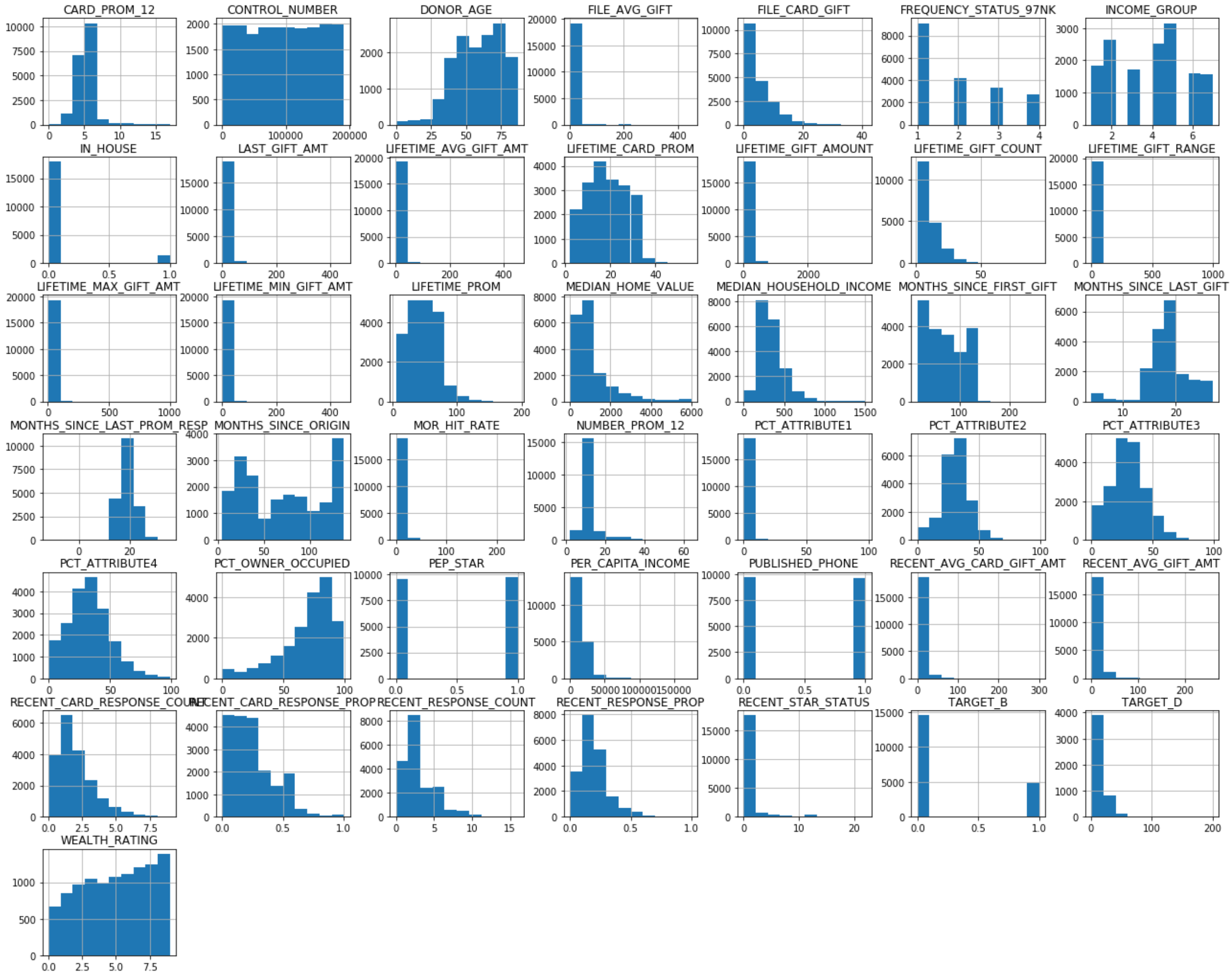


Table of Content

4. Data Preparation

Dropped Columns:

Column Name	Reason
TARGET_D	missing values 75%
CONTROL_NUMBER	unique values
HOME_OWNER	missing values 45%
INCOME_GROUP	missing values 23%
OVERLAY_SOURCE	missing values 23%
WEALTH_RATING	missing values 45%

In [21]:
```
# A new donor_dataset with dropped: 'TARGET_D', 'CONTROL_NUMBER' , 'HOME_OWNER' , 'IN
COME_GROUP', OVERLAY_SOURCE
#                                     , 'WEALTH_RATING'

column_list = ['TARGET_B', 'MONTHS_SINCE_ORIGIN',
       'DONOR_AGE', 'IN_HOUSE', 'URBANICITY', 'SES', 'CLUSTER_CODE',
       'DONOR_GENDER', 'PUBLISHED_PHONE',
       'MOR_HIT_RATE', 'MEDIAN_HOME_VALUE',
       'MEDIAN_HOUSEHOLD_INCOME', 'PCT_OWNER_OCCUPIED', 'PER_CAPITA_INCOME',
       'PCT_ATTRIBUTE1', 'PCT_ATTRIBUTE2', 'PCT_ATTRIBUTE3', 'PCT_ATTRIBUTE4',
       'PEP_STAR', 'RECENT_STAR_STATUS', 'RECENCY_STATUS_96NK',
       'FREQUENCY_STATUS_97NK', 'RECENT_RESPONSE_PROP', 'RECENT_AVG_GIFT_AMT',
       'RECENT_CARD_RESPONSE_PROP', 'RECENT_AVG_CARD_GIFT_AMT',
       'RECENT_RESPONSE_COUNT', 'RECENT_CARD_RESPONSE_COUNT',
       'MONTHS_SINCE_LAST_PROM_RESP', 'LIFETIME_CARD_PROM', 'LIFETIME_PROM',
       'LIFETIME_GIFT_AMOUNT', 'LIFETIME_GIFT_COUNT', 'LIFETIME_AVG_GIFT_AMT',
       'LIFETIME_GIFT_RANGE', 'LIFETIME_MAX_GIFT_AMT', 'LIFETIME_MIN_GIFT_AMT',
       'LAST_GIFT_AMT', 'CARD_PROM_12', 'NUMBER_PROM_12',
       'MONTHS_SINCE_LAST_GIFT', 'MONTHS_SINCE_FIRST_GIFT', 'FILE_AVG_GIFT',
       'FILE_CARD_GIFT']
donor_dataset = donor_data[column_list]
```

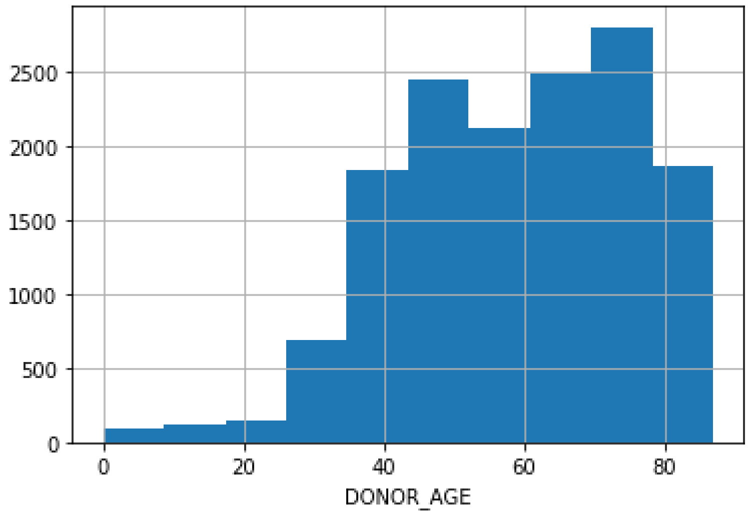
In [22]:
```
print(donor_dataset.shape)
```

```
(19372, 44)
```

[DONOR_AGE]

Handle missing values

In [23]:
```
donor_dataset['DONOR_AGE'].hist()
plt.xlabel('DONOR_AGE')
plt.show()
```



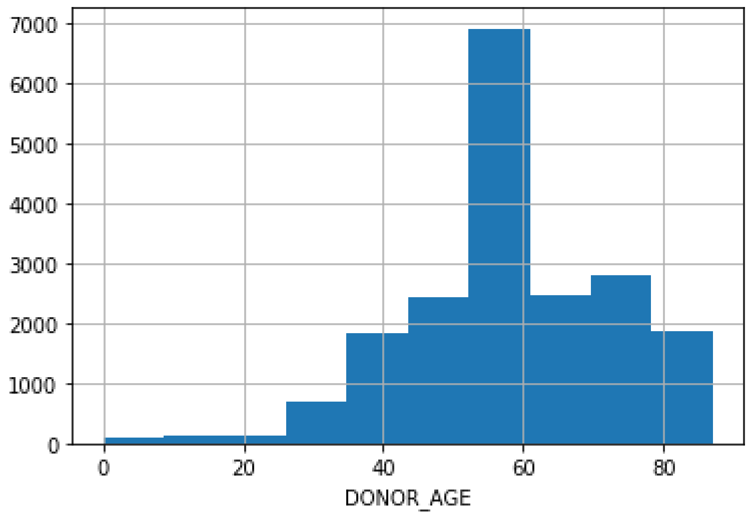
In [24]:
```
# deal with age missing values

# calculate the mean and the median for the whole population

median_DONOR_AGE = donor_dataset['DONOR_AGE'].median()
print('Median = ',median_DONOR_AGE)
mean_DONOR_AGE = donor_dataset['DONOR_AGE'].mean()
print('Mean = ',mean_DONOR_AGE)
```

```
Median =  60.0
Mean =  58.91905055909995
```

Because the **distribution is not normal**, NaN values will be replaced with the **median**.

In [25]:
```
donor_dataset['DONOR_AGE'].fillna(median_DONOR_AGE, inplace = True)
```

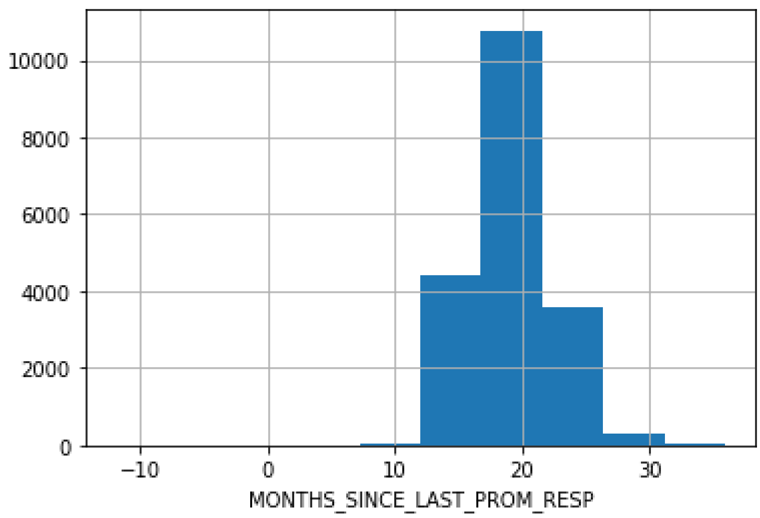
In [26]:
```
donor_dataset['DONOR_AGE'].hist()
plt.xlabel('DONOR_AGE')
plt.show()
```



[MONTHS_SINCE_LAST_PROM_RESP]

Handle missing values

In [27]:
```
donor_dataset['MONTHS_SINCE_LAST_PROM_RESP'].hist()
plt.xlabel('MONTHS_SINCE_LAST_PROM_RESP')
plt.show()
```



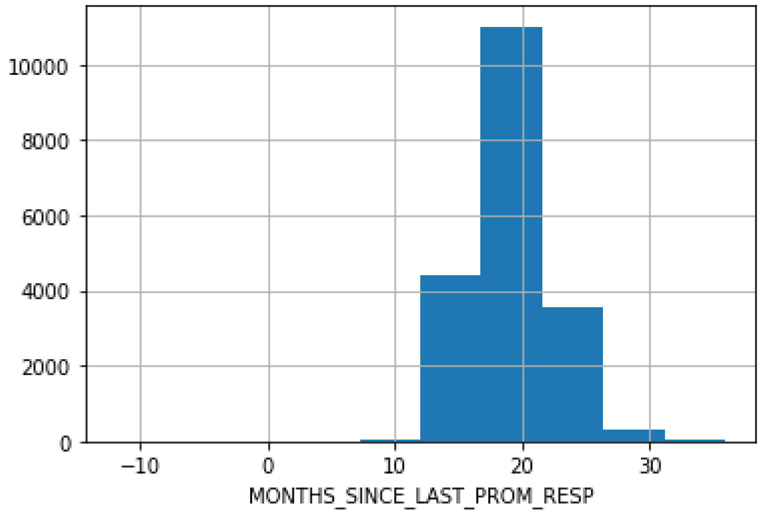
In [28]:
```
median_MONTHS_SINCE_LAST_PROM_RESP = donor_dataset['MONTHS_SINCE_LAST_PROM_RESP'].med
ian()
print('Median = ',median_MONTHS_SINCE_LAST_PROM_RESP)
mean_MONTHS_SINCE_LAST_PROM_RESP = donor_dataset['MONTHS_SINCE_LAST_PROM_RESP'].mean
()
print('Mean = ',mean_MONTHS_SINCE_LAST_PROM_RESP)
```

```
Median =  18.0
Mean =  19.038899926801214
```

Because the distribution is quite **normal**, NaN values will be replaced with the **mean**.

In [29]:
```
# replace NaN with the mean

donor_dataset['MONTHS_SINCE_LAST_PROM_RESP'].fillna(mean_MONTHS_SINCE_LAST_PROM_RESP,
inplace = True)
```

In [30]:
```
donor_dataset['MONTHS_SINCE_LAST_PROM_RESP'].hist()
plt.xlabel('MONTHS_SINCE_LAST_PROM_RESP')
plt.show()
```

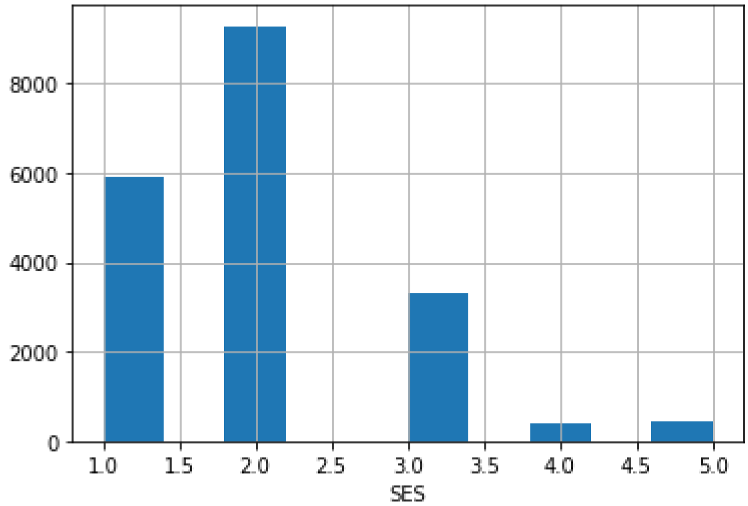


[SES]

Str values '1','2','3','4' replaced with numbers and '?' with number 5.

In [31]:
```
donor_dataset['SES'].replace('1',1,inplace=True)
donor_dataset['SES'].replace('2',2,inplace=True)
donor_dataset['SES'].replace('3',3,inplace=True)
donor_dataset['SES'].replace('4',4,inplace=True)
donor_dataset['SES'].replace('?',5,inplace=True)
```

In [32]:
```
donor_dataset['SES'].hist()
plt.xlabel('SES')
plt.show()
print(donor_dataset['SES'].dtype)
donor_dataset.head(5)
```




```
int64
```

Out[32]:

	TARGET_B	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE
0	0	101	87.0	0	?	5	.
1	1	137	79.0	0	R	2	45
2	0	113	75.0	0	S	1	11
3	0	92	60.0	0	U	2	04
4	0	101	74.0	0	R	2	49

[CLUSTER_CODE]

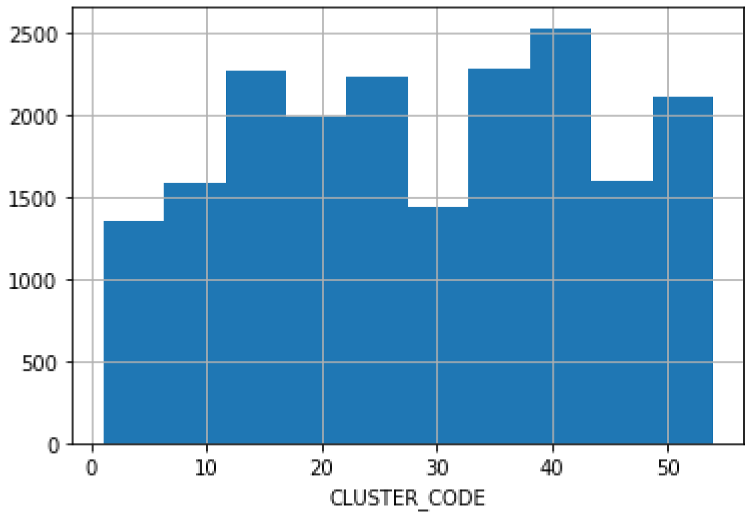
Replaced ' .' with the number '54' and after str values with numbers.

In [33]:
```
donor_dataset['CLUSTER_CODE'].replace(' .','54',inplace=True)
```

In [34]:
```
donor_dataset['CLUSTER_CODE'] = donor_dataset['CLUSTER_CODE'].astype(int)
```

```
In [35]: print(donor_dataset['CLUSTER_CODE'].dtype)
         donor_dataset['CLUSTER_CODE'].hist()
         plt.xlabel('CLUSTER_CODE')
         plt.show()
```

int32



[DONOR_GENDER]

Dropped rows with gender inputted wrongly.

```
In [36]: donor_dataset.shape
```

Out[36]: (19372, 44)

```
In [37]: donor_dataset['DONOR_GENDER'].unique()
```

Out[37]: array(['M', 'F', 'U', 'A'], dtype=object)

```
In [38]: # Get names of indexes for which column ['DONOR_GENDER'] has value 'U' and 'A'
         indexNames_donor_U = donor_dataset[donor_dataset['DONOR_GENDER'] == 'U'].index    # U
         1017 values
         indexNames_donor_A = donor_dataset[donor_dataset['DONOR_GENDER'] == 'A'].index    # A
         1 value

         # Delete these row indexes from dataFrame
         donor_dataset.drop(indexNames_donor_U, inplace=True)
         donor_dataset.drop(indexNames_donor_A, inplace=True)
```

```
In [39]: donor_dataset.shape
```

Out[39]: (18354, 44)

```
In [40]: 19372-18354
```

Out[40]: 1018

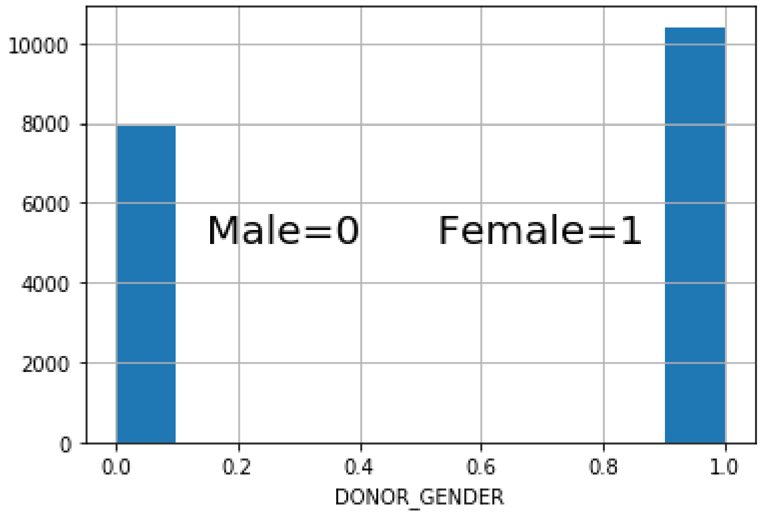
```
In [41]: donor_dataset.to_csv('donor_prepared_for_analysis.csv',index=None)
```

Replaced gender values with 1 for female, 0 for male with a categorical endoding.

In [45]:
```
donor_dataset['DONOR_GENDER'] = np.where(donor_dataset['DONOR_GENDER']=='F',1,0) # ca
tegorical encoding
```

In [46]:
```
print(donor_dataset.shape)
donor_dataset['DONOR_GENDER'].hist()
plt.xlabel('DONOR_GENDER')
plt.text(0.15, 5000, 'Male=0      Female=1',dict(size=20))
plt.show()
donor_dataset.head()
```

(18354, 44)



Out[46]:

	TARGET_B	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE
0	0	101	87.0	0	?	5	54
1	1	137	79.0	0	R	2	45
2	0	113	75.0	0	S	1	11
3	0	92	60.0	0	U	2	4
4	0	101	74.0	0	R	2	49

[URBANICITY]

Replaced:

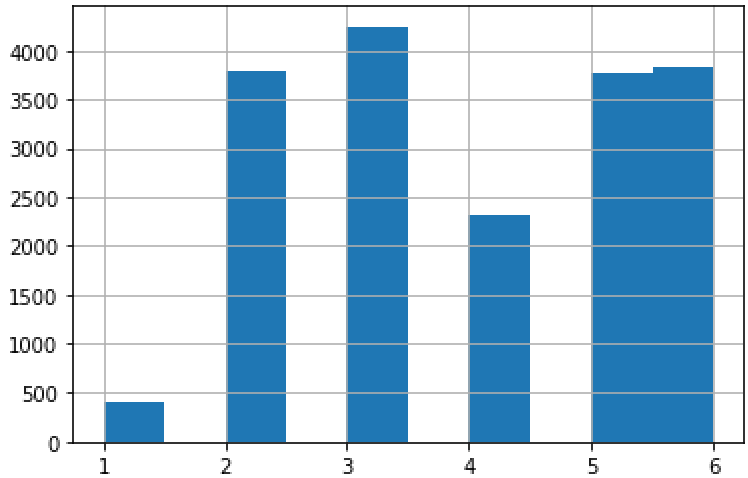
- ? -> 1
- R -> 2
- S -> 3
- U -> 4
- T -> 5
- C -> 6

In [47]:
```
donor_dataset['URBANICITY'].replace('C',6,inplace=True)  # C = City
donor_dataset['URBANICITY'].replace('T',5,inplace=True)  # T = Town
donor_dataset['URBANICITY'].replace('U',4,inplace=True)  # U = Urban
donor_dataset['URBANICITY'].replace('S',3,inplace=True)  # S = Suburban
donor_dataset['URBANICITY'].replace('R',2,inplace=True)  # R = Rural
donor_dataset['URBANICITY'].replace('?',1,inplace=True)  # ? = Unknown
donor_dataset.head()
```

Out[47]:

	TARGET_B	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE
0	0	101	87.0	0	1	5	54
1	1	137	79.0	0	2	2	45
2	0	113	75.0	0	3	1	11
3	0	92	60.0	0	4	2	4
4	0	101	74.0	0	2	2	49

In [48]:
```
donor_dataset['URBANICITY'].hist()
plt.show()
```



RECENCY_STATUS_96NK

Replaced:

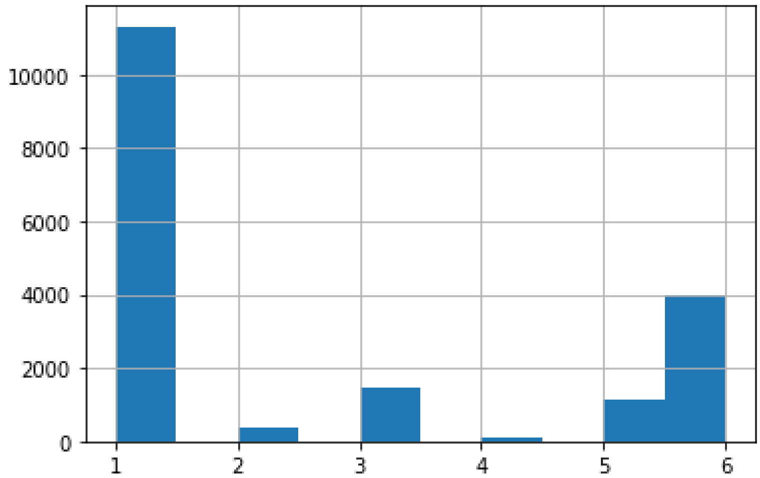
- A -> 1
- E -> 2
- F -> 3
- L -> 4
- N -> 5
- S -> 6

In [49]:
```
donor_dataset['RECENCY_STATUS_96NK'].replace('A',1,inplace=True)
donor_dataset['RECENCY_STATUS_96NK'].replace('E',2,inplace=True)
donor_dataset['RECENCY_STATUS_96NK'].replace('F',3,inplace=True)
donor_dataset['RECENCY_STATUS_96NK'].replace('L',4,inplace=True)
donor_dataset['RECENCY_STATUS_96NK'].replace('N',5,inplace=True)
donor_dataset['RECENCY_STATUS_96NK'].replace('S',6,inplace=True)
donor_dataset.head()
```

Out[49]:

	TARGET_B	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE
0	0	101	87.0	0	1	5	54
1	1	137	79.0	0	2	2	45
2	0	113	75.0	0	3	1	11
3	0	92	60.0	0	4	2	4
4	0	101	74.0	0	2	2	49

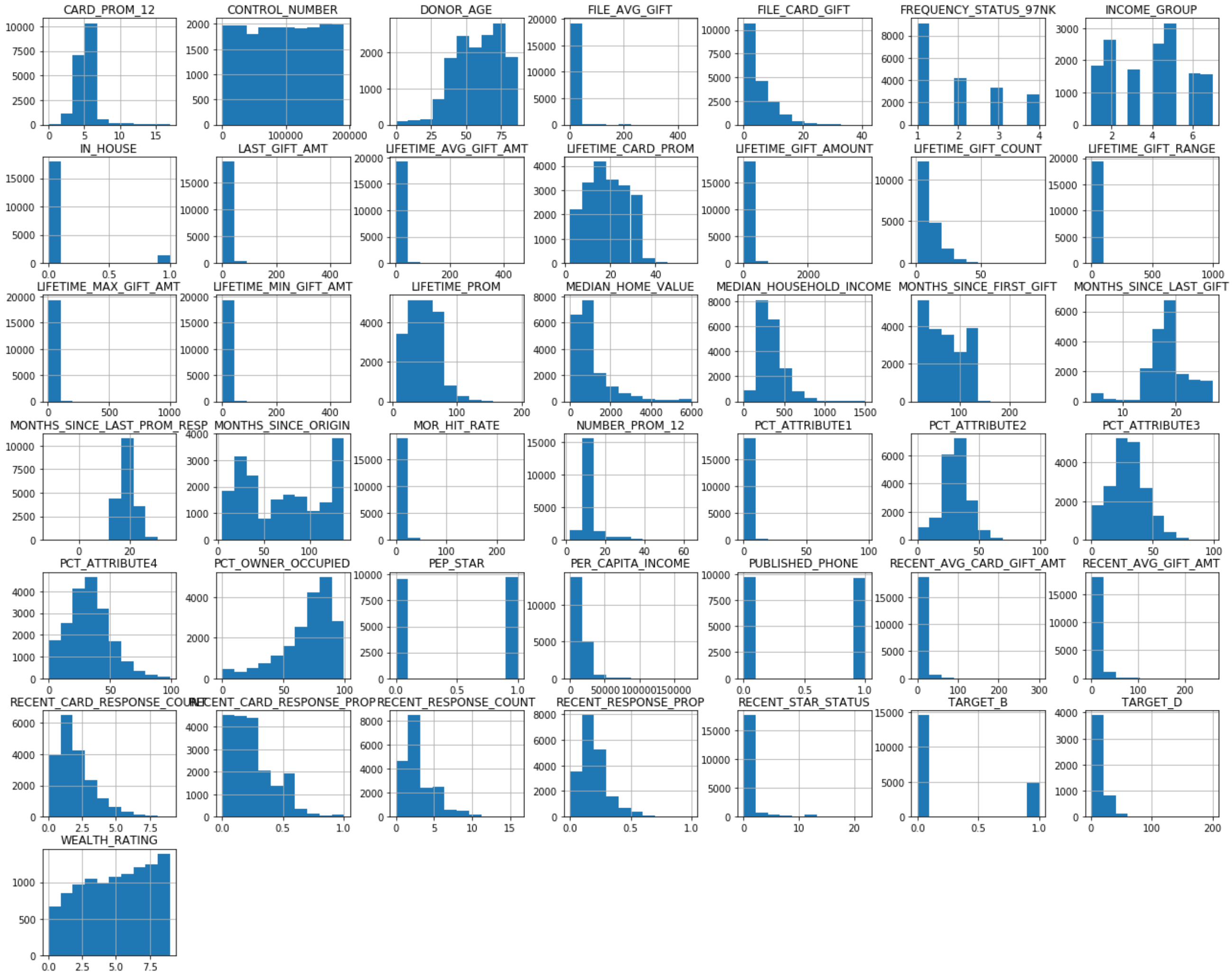
In [50]:
```
donor_dataset['RECENCY_STATUS_96NK'].hist()
plt.show()
```



Data Distributions (Histograms) - Before Data Preparation

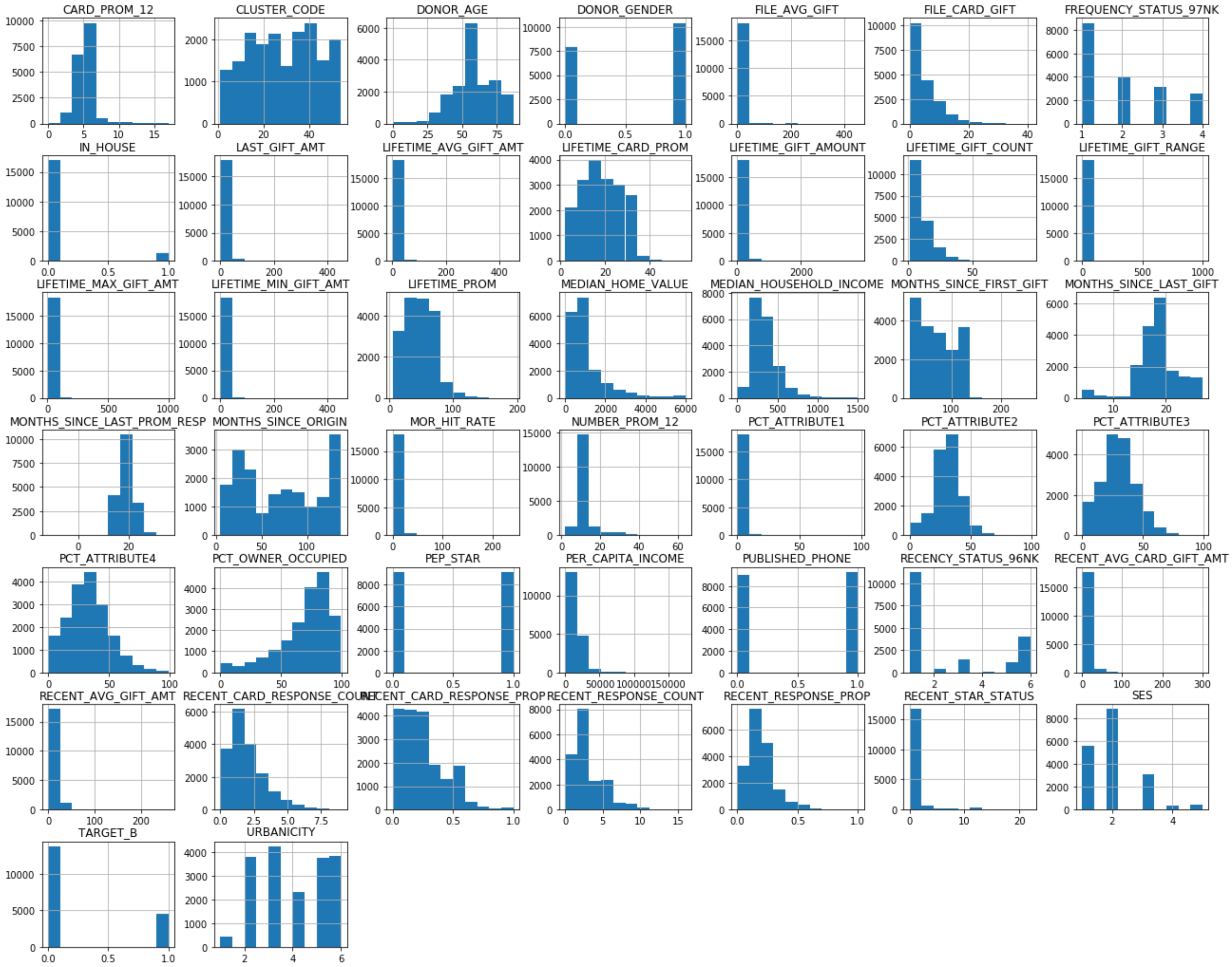
In [51]:

```
donor_data.hist(figsize=(22,18))
plt.show()
```



Data Distributions (Histograms) - After Data Preparation

In [52]:
```
donor_dataset.hist(figsize=(22,18))
plt.show()
```

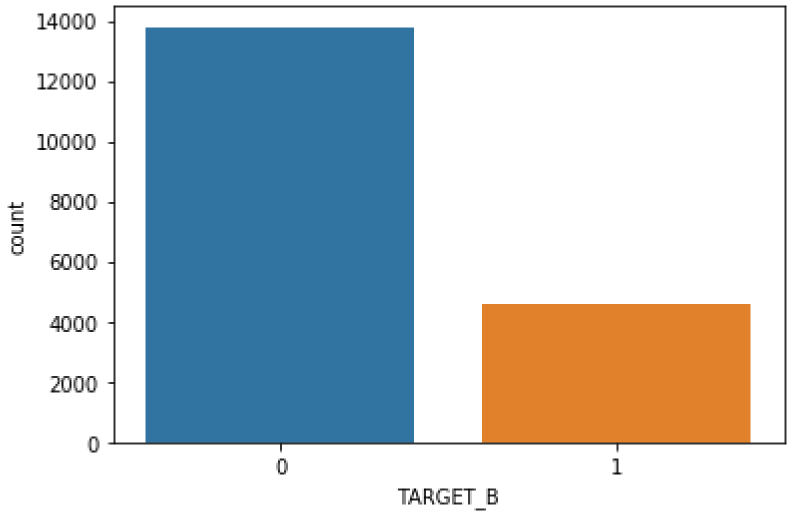


[TARGET_B]

In [53]:
```
print('The shape is: ', donor_dataset.shape)
print('0=no 1=yes','\n',donor_dataset.groupby('TARGET_B').size())
sns.countplot(donor_dataset['TARGET_B'])
```

```
The shape is:  (18354, 44)
0=no 1=yes
 TARGET_B
0    13783
1     4571
dtype: int64
```

Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x1ebd7ad99c8>



TARGET_B (response to sollicitation) 0 = No , 1 = Yes

- Of these 18354 data points, 13783 are labeled as 0 and 4571 as 1:
  - Unbalanced DataSet

Prepare Data for Scaling

In [54]:
```
donor_dataset
```
Out[54]:

	TARGET_B	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_COI
0	0	101	87.0	0	1	5	
1	1	137	79.0	0	2	2	
2	0	113	75.0	0	3	1	
3	0	92	60.0	0	4	2	
4	0	101	74.0	0	2	2	
...	...	...	...	...	...	...	
19367	0	89	66.0	1	4	1	
19368	0	137	77.0	1	6	1	
19369	0	29	60.0	1	3	1	
19370	0	129	78.0	1	1	5	
19371	1	29	70.0	0	1	5	

18354 rows × 44 columns

In [55]:
```
TARGET_B = donor_dataset['TARGET_B']
print(TARGET_B.shape, type(TARGET_B))
TARGET_B
```
```
(18354,) <class 'pandas.core.series.Series'>
```
Out[55]:
```
0        0
1        1
2        0
3        0
4        0
        ..
19367    0
19368    0
19369    0
19370    0
19371    1
Name: TARGET_B, Length: 18354, dtype: int64
```
In [56]:
```
TARGET_B_np = TARGET_B.to_numpy()
print(TARGET_B_np.shape, type(TARGET_B_np))
TARGET_B_np
```
```
(18354,) <class 'numpy.ndarray'>
```
Out[56]:
```
array([0, 1, 0, ..., 0, 0, 1], dtype=int64)
```

In [57]:
```python
TARGET_B_df = pd.DataFrame(TARGET_B_np)
TARGET_B_df.columns = ['TARGET_B']
print(TARGET_B_df.shape, type(TARGET_B_df))
TARGET_B_df
```

```
(18354, 1) <class 'pandas.core.frame.DataFrame'>
```

Out[57]:

	TARGET_B
0	0
1	1
2	0
3	0
4	0
...	...
18349	0
18350	0
18351	0
18352	0
18353	1

18354 rows × 1 columns

In [58]:
```python
donor_dataset2 = donor_dataset.drop('TARGET_B',axis=1)
```

In [59]:
```python
print(donor_dataset2.shape)
donor_dataset2.tail()
```

```
(18354, 43)
```

Out[59]:

	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE	DONOR_
19367	89	66.0	1	4	1	3	
19368	137	77.0	1	6	1	24	
19369	29	60.0	1	3	1	11	
19370	129	78.0	1	1	5	54	
19371	29	70.0	0	1	5	54	

In [60]:
```
print(donor_dataset2.columns)
```

```
Index(['MONTHS_SINCE_ORIGIN', 'DONOR_AGE', 'IN_HOUSE', 'URBANICITY', 'SES',
       'CLUSTER_CODE', 'DONOR_GENDER', 'PUBLISHED_PHONE', 'MOR_HIT_RATE',
       'MEDIAN_HOME_VALUE', 'MEDIAN_HOUSEHOLD_INCOME', 'PCT_OWNER_OCCUPIED',
       'PER_CAPITA_INCOME', 'PCT_ATTRIBUTE1', 'PCT_ATTRIBUTE2',
       'PCT_ATTRIBUTE3', 'PCT_ATTRIBUTE4', 'PEP_STAR', 'RECENT_STAR_STATUS',
       'RECENCY_STATUS_96NK', 'FREQUENCY_STATUS_97NK', 'RECENT_RESPONSE_PROP',
       'RECENT_AVG_GIFT_AMT', 'RECENT_CARD_RESPONSE_PROP',
       'RECENT_AVG_CARD_GIFT_AMT', 'RECENT_RESPONSE_COUNT',
       'RECENT_CARD_RESPONSE_COUNT', 'MONTHS_SINCE_LAST_PROM_RESP',
       'LIFETIME_CARD_PROM', 'LIFETIME_PROM', 'LIFETIME_GIFT_AMOUNT',
       'LIFETIME_GIFT_COUNT', 'LIFETIME_AVG_GIFT_AMT', 'LIFETIME_GIFT_RANGE',
       'LIFETIME_MAX_GIFT_AMT', 'LIFETIME_MIN_GIFT_AMT', 'LAST_GIFT_AMT',
       'CARD_PROM_12', 'NUMBER_PROM_12', 'MONTHS_SINCE_LAST_GIFT',
       'MONTHS_SINCE_FIRST_GIFT', 'FILE_AVG_GIFT', 'FILE_CARD_GIFT'],
      dtype='object')
```

- **TARGET_B_df**
- **donor_dataset2**

Scaling

Standard Scaler

In [61]:
```
# create StandardScaler() object
scaler_standard = preprocessing.StandardScaler()

# transform data and store it in scaled_standard
scaled_standard = scaler_standard.fit_transform(donor_dataset2)

# convert scaled_standard to DataFrame
scaled_standard_df = pd.DataFrame(scaled_standard, columns=['MONTHS_SINCE_ORIGIN', 'DONOR_AGE', 'IN_HOUSE', 'URBANICITY', 'SES',
       'CLUSTER_CODE', 'DONOR_GENDER', 'PUBLISHED_PHONE', 'MOR_HIT_RATE',
       'MEDIAN_HOME_VALUE', 'MEDIAN_HOUSEHOLD_INCOME', 'PCT_OWNER_OCCUPIED',
       'PER_CAPITA_INCOME', 'PCT_ATTRIBUTE1', 'PCT_ATTRIBUTE2',
       'PCT_ATTRIBUTE3', 'PCT_ATTRIBUTE4', 'PEP_STAR', 'RECENT_STAR_STATUS',
       'RECENCY_STATUS_96NK', 'FREQUENCY_STATUS_97NK', 'RECENT_RESPONSE_PROP',
       'RECENT_AVG_GIFT_AMT', 'RECENT_CARD_RESPONSE_PROP',
       'RECENT_AVG_CARD_GIFT_AMT', 'RECENT_RESPONSE_COUNT',
       'RECENT_CARD_RESPONSE_COUNT', 'MONTHS_SINCE_LAST_PROM_RESP',
       'LIFETIME_CARD_PROM', 'LIFETIME_PROM', 'LIFETIME_GIFT_AMOUNT',
       'LIFETIME_GIFT_COUNT', 'LIFETIME_AVG_GIFT_AMT', 'LIFETIME_GIFT_RANGE',
       'LIFETIME_MAX_GIFT_AMT', 'LIFETIME_MIN_GIFT_AMT', 'LAST_GIFT_AMT',
       'CARD_PROM_12', 'NUMBER_PROM_12', 'MONTHS_SINCE_LAST_GIFT',
       'MONTHS_SINCE_FIRST_GIFT', 'FILE_AVG_GIFT', 'FILE_CARD_GIFT'])
```

In [62]:
```
print(scaled_standard_df.shape)
scaled_standard_df.tail()
```

(18354, 43)

Out[62]:

	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE	DOI
18349	0.390331	0.468646	3.533507	0.059045	-1.116013	-1.723905	
18350	1.555826	1.220779	3.533507	1.379840	-1.116013	-0.303621	
18351	-1.066538	0.058392	3.533507	-0.601352	-1.116013	-1.182844	
18352	1.361577	1.289154	3.533507	-1.922147	3.477169	1.725357	
18353	-1.066538	0.742149	-0.283005	-1.922147	3.477169	1.725357	

In [63]:
```
donor_dataset_scaled_standard = TARGET_B_df.join(scaled_standard_df, how='right')
```

In [64]:
```
donor_dataset_scaled_standard
```

Out[64]:

	TARGET_B	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER
0	0	0.681705	1.904536	-0.283005	-1.922147	3.477169	1
1	1	1.555826	1.357530	-0.283005	-1.261750	0.032283	1
2	0	0.973078	1.084027	-0.283005	-0.601352	-1.116013	-1
3	0	0.463174	0.058392	-0.283005	0.059045	0.032283	-1
4	0	0.681705	1.015651	-0.283005	-1.261750	0.032283	1
...	...	...	...	...	...	...	
18349	0	0.390331	0.468646	3.533507	0.059045	-1.116013	-1
18350	0	1.555826	1.220779	3.533507	1.379840	-1.116013	-0
18351	0	-1.066538	0.058392	3.533507	-0.601352	-1.116013	-1
18352	0	1.361577	1.289154	3.533507	-1.922147	3.477169	1
18353	1	-1.066538	0.742149	-0.283005	-1.922147	3.477169	1

18354 rows × 44 columns

Robust Scaler

In [65]:

```
# create RobustScaler() object
scaler_robust = preprocessing.RobustScaler()

# transform data and store it in scaled_robust
scaled_robust = scaler_robust.fit_transform(donor_dataset2)

# convert scaled_robust to DataFrame
scaled_robust_df = pd.DataFrame(scaled_robust, columns=['MONTHS_SINCE_ORIGIN', 'DONOR
_AGE', 'IN_HOUSE', 'URBANICITY', 'SES',
        'CLUSTER_CODE', 'DONOR_GENDER', 'PUBLISHED_PHONE', 'MOR_HIT_RATE',
        'MEDIAN_HOME_VALUE', 'MEDIAN_HOUSEHOLD_INCOME', 'PCT_OWNER_OCCUPIED',
        'PER_CAPITA_INCOME', 'PCT_ATTRIBUTE1', 'PCT_ATTRIBUTE2',
        'PCT_ATTRIBUTE3', 'PCT_ATTRIBUTE4', 'PEP_STAR', 'RECENT_STAR_STATUS',
        'RECENCY_STATUS_96NK', 'FREQUENCY_STATUS_97NK', 'RECENT_RESPONSE_PROP',
        'RECENT_AVG_GIFT_AMT', 'RECENT_CARD_RESPONSE_PROP',
        'RECENT_AVG_CARD_GIFT_AMT', 'RECENT_RESPONSE_COUNT',
        'RECENT_CARD_RESPONSE_COUNT', 'MONTHS_SINCE_LAST_PROM_RESP',
        'LIFETIME_CARD_PROM', 'LIFETIME_PROM', 'LIFETIME_GIFT_AMOUNT',
        'LIFETIME_GIFT_COUNT', 'LIFETIME_AVG_GIFT_AMT', 'LIFETIME_GIFT_RANGE',
        'LIFETIME_MAX_GIFT_AMT', 'LIFETIME_MIN_GIFT_AMT', 'LAST_GIFT_AMT',
        'CARD_PROM_12', 'NUMBER_PROM_12', 'MONTHS_SINCE_LAST_GIFT',
        'MONTHS_SINCE_FIRST_GIFT', 'FILE_AVG_GIFT', 'FILE_CARD_GIFT'])
```

In [66]:

```
print(scaled_robust_df.shape)
scaled_robust_df.tail()
```

(18354, 43)

Out[66]:

	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE	DONOR_
18349	0.285714	0.315789	1.0	0.0	-1.0	-1.00	
18350	0.857143	0.894737	1.0	1.0	-1.0	-0.16	
18351	-0.428571	0.000000	1.0	-0.5	-1.0	-0.68	
18352	0.761905	0.947368	1.0	-1.5	3.0	1.04	
18353	-0.428571	0.526316	0.0	-1.5	3.0	1.04	

In [67]:

```
donor_dataset_scaled_robust = TARGET_B_df.join(scaled_robust_df, how='right')
```

In [68]:
```
donor_dataset_scaled_robust
```

Out[68]:

	TARGET_B	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_COI
0	0	0.428571	1.421053	0.0	-1.5	3.0	1.
1	1	0.857143	1.000000	0.0	-1.0	0.0	0.
2	0	0.571429	0.789474	0.0	-0.5	-1.0	-0.
3	0	0.321429	0.000000	0.0	0.0	0.0	-0.
4	0	0.428571	0.736842	0.0	-1.0	0.0	0.
...	...	...	...	...	...	...	
18349	0	0.285714	0.315789	1.0	0.0	-1.0	-1.
18350	0	0.857143	0.894737	1.0	1.0	-1.0	-0.
18351	0	-0.428571	0.000000	1.0	-0.5	-1.0	-0.
18352	0	0.761905	0.947368	1.0	-1.5	3.0	1.
18353	1	-0.428571	0.526316	0.0	-1.5	3.0	1.

18354 rows × 44 columns

Comparison

Standard Scaler

In [69]:
```
donor_dataset_scaled_standard
```

Out[69]:

	TARGET_B	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER
0	0	0.681705	1.904536	-0.283005	-1.922147	3.477169	1
1	1	1.555826	1.357530	-0.283005	-1.261750	0.032283	1
2	0	0.973078	1.084027	-0.283005	-0.601352	-1.116013	-1
3	0	0.463174	0.058392	-0.283005	0.059045	0.032283	-1
4	0	0.681705	1.015651	-0.283005	-1.261750	0.032283	1
...	...	...	...	...	...	...	
18349	0	0.390331	0.468646	3.533507	0.059045	-1.116013	-1
18350	0	1.555826	1.220779	3.533507	1.379840	-1.116013	-0
18351	0	-1.066538	0.058392	3.533507	-0.601352	-1.116013	-1
18352	0	1.361577	1.289154	3.533507	-1.922147	3.477169	1
18353	1	-1.066538	0.742149	-0.283005	-1.922147	3.477169	1

18354 rows × 44 columns

Robust Scaler

In [70]:
```
donor_dataset_scaled_robust
```

Out[70]:

	TARGET_B	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_COI
0	0	0.428571	1.421053	0.0	-1.5	3.0	1.
1	1	0.857143	1.000000	0.0	-1.0	0.0	0.
2	0	0.571429	0.789474	0.0	-0.5	-1.0	-0.
3	0	0.321429	0.000000	0.0	0.0	0.0	-0.
4	0	0.428571	0.736842	0.0	-1.0	0.0	0.
...	...	...	...	...	...	...	
18349	0	0.285714	0.315789	1.0	0.0	-1.0	-1.
18350	0	0.857143	0.894737	1.0	1.0	-1.0	-0.
18351	0	-0.428571	0.000000	1.0	-0.5	-1.0	-0.
18352	0	0.761905	0.947368	1.0	-1.5	3.0	1.
18353	1	-0.428571	0.526316	0.0	-1.5	3.0	1.

18354 rows × 44 columns

[Table of Content](#)

5. About Algorithms

Test Harness

Stratified 10-fold cross validation will be used to estimate model accuracy.

This will split the dataset into 10 parts, train on 9 and test on 1 and repeat for all combinations of train-test splits.

Stratified means that each fold or split of the dataset will aim to have the same distribution of example by class as exist in the whole training dataset.

The random seed will be set via the random_state argument to a fixed number to ensure that each algorithm is evaluated on the same splits of the training dataset.

The metric of **'accuracy'** will be used to evaluate models.

This is a ratio of the number of correctly predicted instances divided by the total number of instances in the dataset multiplied by 100 to give a percentage (e.g. 95% accurate).

Build Models

Not knowing which algorithm would be good for this project and what configuration, 9 different algorithms will be tested.

Algorithms:

- Logistic Regression (LR)
- Linear Discriminant Analysis (LDA)
- K-Nearest Neighbors (KNN)
- Classification and Regression Trees (CART)
- Random Forest Decision Tree (RFTree)
- Gradient Boosting (GrB)
- Gaussian Naive Bayes (NB)
- Support Vector Machines (SVM)
- Deep Learning (Deep)

This is a good mixture of **simple linear (LR and LDA)**, **nonlinear (KNN, CART, NB, SVM and ....)** algorithms.

Table of Content

6. Evaluate Some Algorithms

6.1.a Create a Validation Dataset with Standard Scaled Data

The loaded dataset will be split into two:

- 75% to train, evaluate and select among the models
- 25% as a validation dataset.

In [71]:
```
# Split-out validation dataset

X = donor_dataset_scaled_standard.drop('TARGET_B',axis=1)
y = donor_dataset_scaled_standard['TARGET_B']
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.25,
random_state=1)
```

In [72]:
```
print ('All Data: ', X.size)
print ('Train Size: ', X_train.size,'=', X_train.size/ X.size*100,'%')
print ('Test Size: ', X_validation.size, '=', X_validation.size/X.size*100,'%')
```

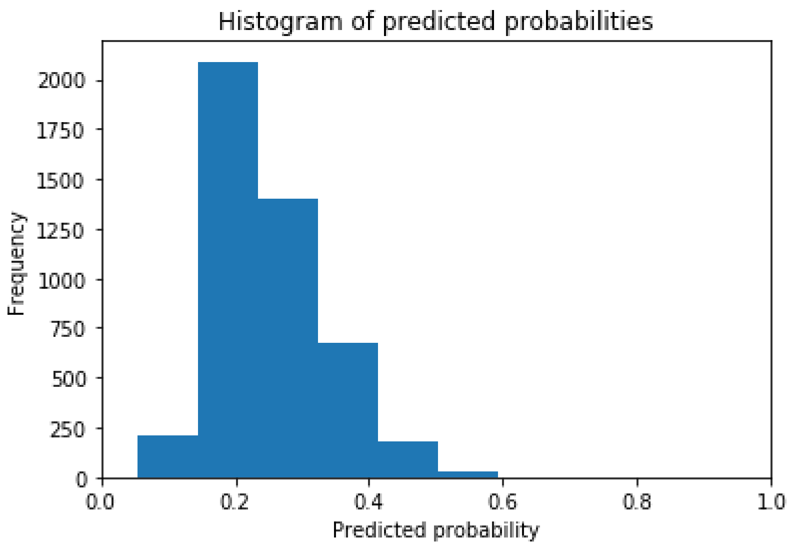
```
All Data:  789222
Train Size:  591895 = 74.99727579819113 %
Test Size:  197327 = 25.002724201808867 %
```

6.2.a Build Models, Make and Evaluate Predictions on different models with Standard Scaled Data

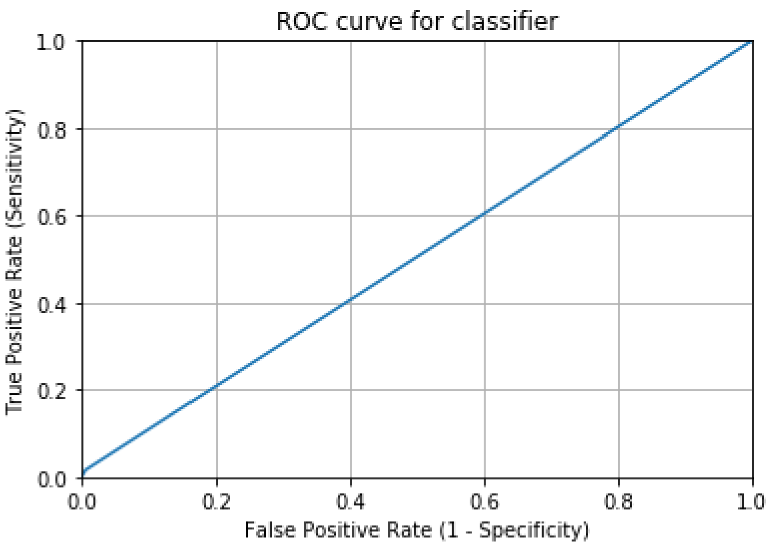
In [73]:
```python
# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('RFTree', RandomForestClassifier()))
models.append(('GrB', GradientBoostingClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
models.append(('Deep', MLPClassifier()))

# evaluate each model in turn
results = []
names = []
print()

for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accurac
y')
    results.append(cv_results)
    names.append(name)
    print('Estimate Model Accuracy: mean (std)','\n%s: %f (%f)' % (name, cv_results.m
ean(), cv_results.std()),'\n')


    # Make predictions on validation dataset

    model.fit(X_train, Y_train)
    predictions = model.predict(X_validation)

    # Evaluate predictions

    print("Prediction:", '%s' % (name))
    print('Accuracy Score:',accuracy_score(Y_validation, predictions))
    print()
    print('Confusion Matrix:\n',confusion_matrix(Y_validation, predictions))
    print('\n')
    print('Classification Report:\n',classification_report(Y_validation, predictions
))
    print()


    try:
        # store the predicted probabilities for class 1
        Y_pred_prob = model.predict_proba(X_validation)[:, 1]

    except AttributeError:
        print("An AttributeError has occurred. I can't show the histogram of predicte
d probabilities, ROC curve for classifier and AUC for the classifier.")

    else:

        # histogram of predicted probabilities

        print()
        print("Prediction probabilities distribution:", '%s' % (name))
        plt.hist(Y_pred_prob, bins=8)
        plt.xlim(0, 1)
        plt.title('Histogram of predicted probabilities')
        plt.xlabel('Predicted probability')
        plt.ylabel('Frequency')
        plt.show()
```

```
        # ROC curve for classifier
        # IMPORTANT: first argument is true values, second argument is predicted probabilities
        print()
        print("ROC curve for classifier:", '%s' % (name))
        fpr, tpr, thresholds = metrics.roc_curve(Y_validation, predictions)
        plt.plot(fpr, tpr)
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.0])
        plt.title('ROC curve for classifier')
        plt.xlabel('False Positive Rate (1 - Specificity)')
        plt.ylabel('True Positive Rate (Sensitivity)')
        plt.grid(True)
        plt.show()

        # IMPORTANT: first argument is true values, second argument is predicted probabilities
        print("AUC for classifier:", '%s' % (name),' = ', metrics.roc_auc_score(Y_validation, Y_pred_prob))


    print()
    print('XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX','\n')
```

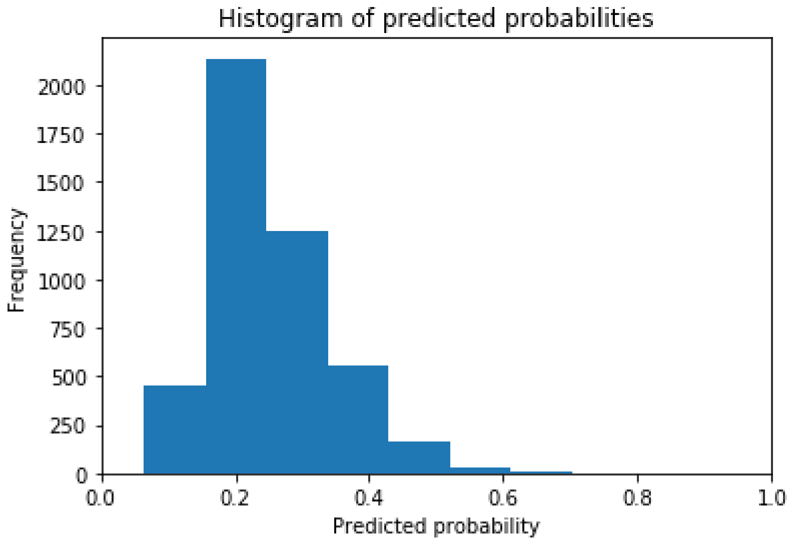
```
Estimate Model Accuracy: mean (std)
LR: 0.749001 (0.002130)

Prediction: LR
Accuracy Score: 0.7583351492699935

Confusion Matrix:
 [[3462   21]
 [1088   18]]


Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.99      0.86      3483
           1       0.46      0.02      0.03      1106

    accuracy                           0.76      4589
   macro avg       0.61      0.51      0.45      4589
weighted avg       0.69      0.76      0.66      4589


Prediction probabilities distribution: LR
```



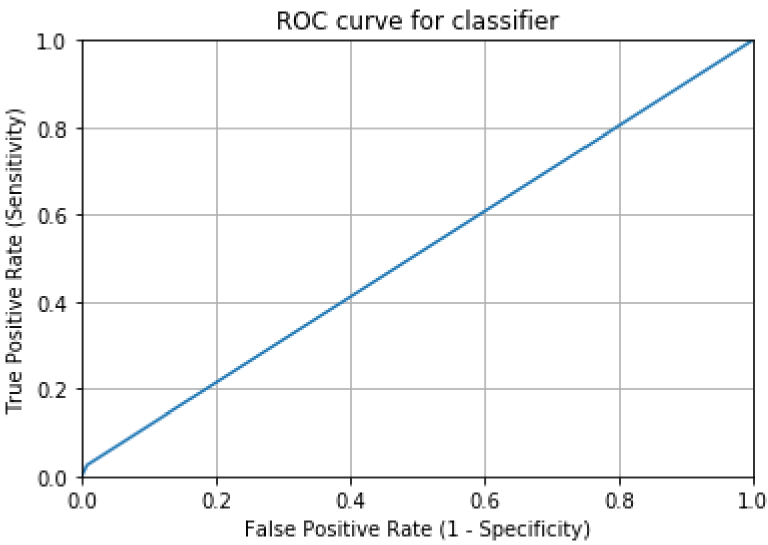

```
ROC curve for classifier: LR
```



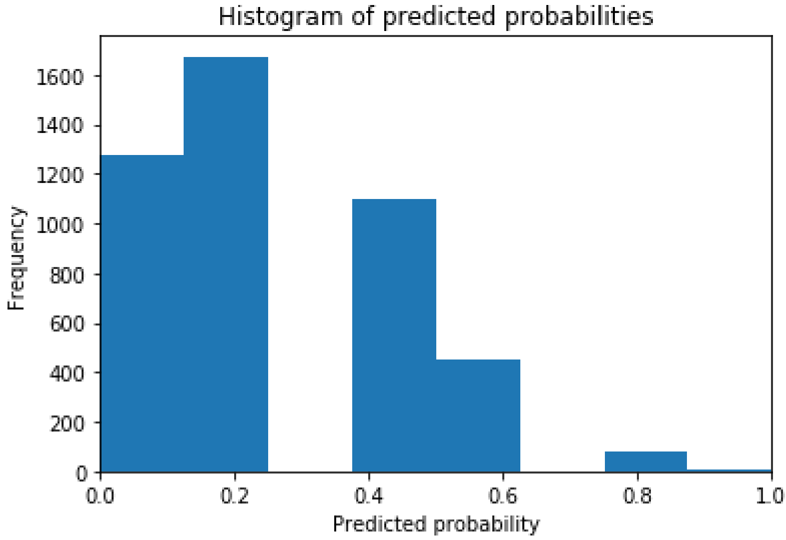
```
AUC for classifier: LR  =  0.6130037967934151

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
LDA: 0.748275 (0.003000)

Prediction: LDA
Accuracy Score: 0.7592067988668555

Confusion Matrix:
 [[3456   27]
 [1078   28]]


Classification Report:
               precision    recall  f1-score   support

           0       0.76      0.99      0.86      3483
           1       0.51      0.03      0.05      1106

    accuracy                           0.76      4589
   macro avg       0.64      0.51      0.46      4589
weighted avg       0.70      0.76      0.67      4589


Prediction probabilities distribution: LDA
```



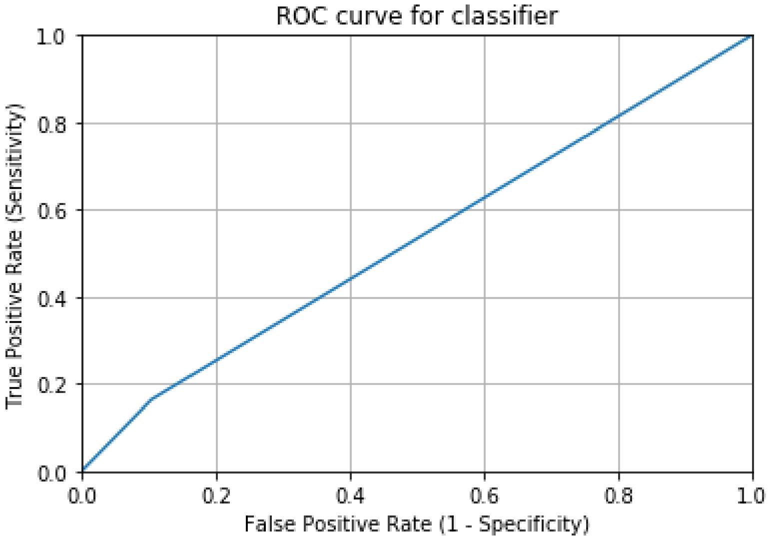

```
ROC curve for classifier: LDA
```



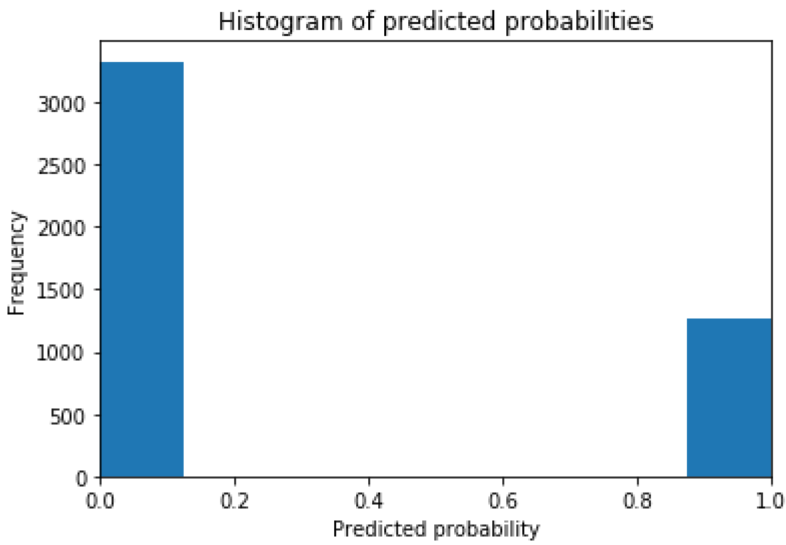
```
AUC for classifier: LDA  =  0.6120910711235508

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
KNN: 0.702579 (0.004318)

Prediction: KNN
Accuracy Score: 0.7193288298104162

Confusion Matrix:
 [[3120  363]
 [ 925  181]]


Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.90      0.83      3483
           1       0.33      0.16      0.22      1106

    accuracy                           0.72      4589
   macro avg       0.55      0.53      0.52      4589
weighted avg       0.67      0.72      0.68      4589


Prediction probabilities distribution: KNN
```



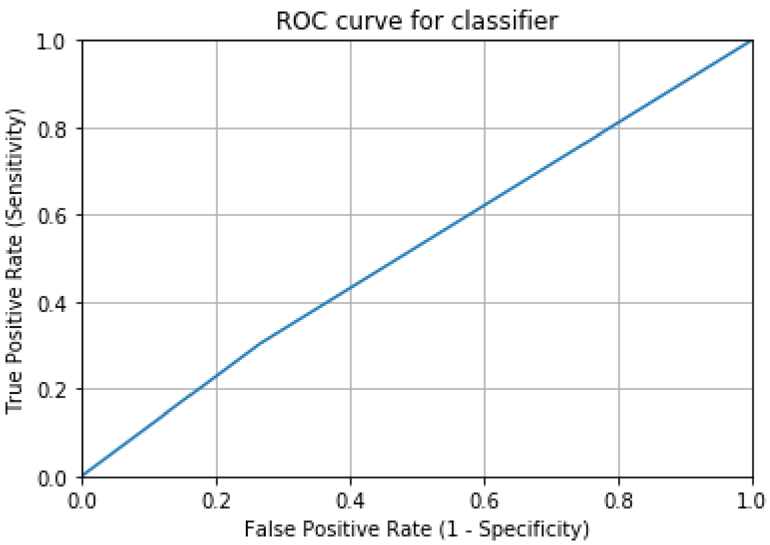

```
ROC curve for classifier: KNN
```



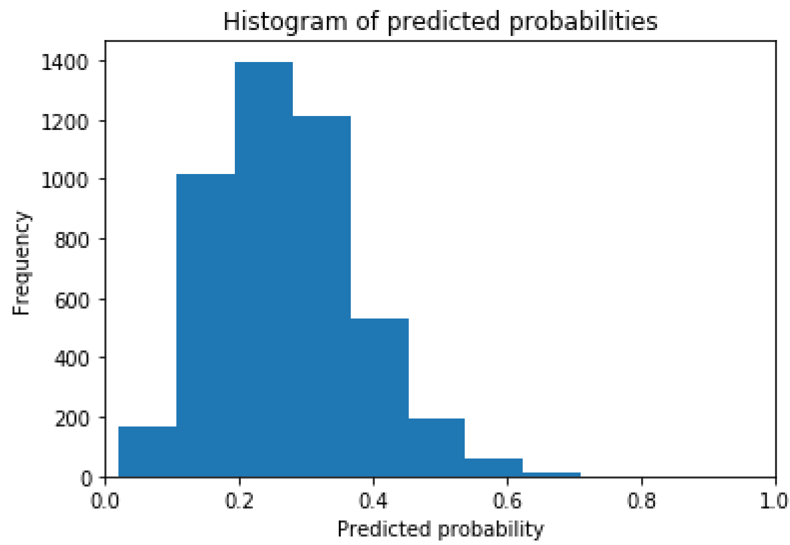
```
AUC for classifier: KNN  =  0.5405916310636161

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
CART: 0.630076 (0.012278)

Prediction: CART
Accuracy Score: 0.6302026585312704

Confusion Matrix:
 [[2556  927]
 [ 770  336]]


Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.73      0.75      3483
           1       0.27      0.30      0.28      1106

    accuracy                           0.63      4589
   macro avg       0.52      0.52      0.52      4589
weighted avg       0.65      0.63      0.64      4589



Prediction probabilities distribution: CART
```



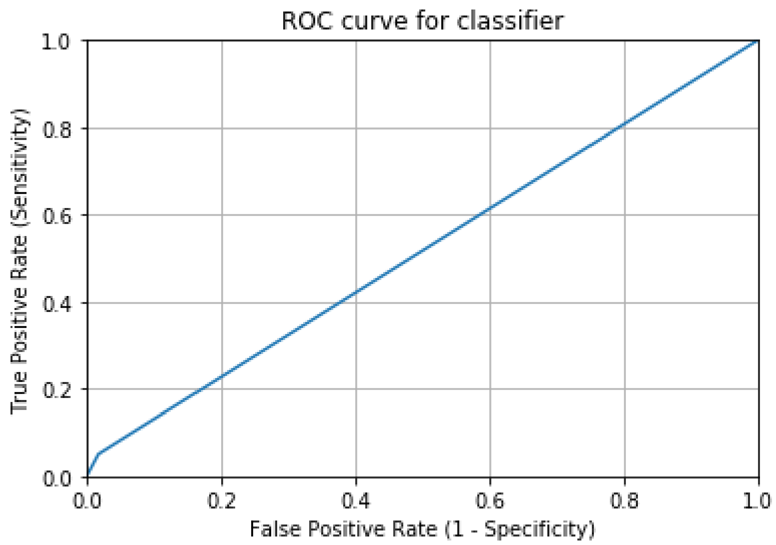

```
ROC curve for classifier: CART
```



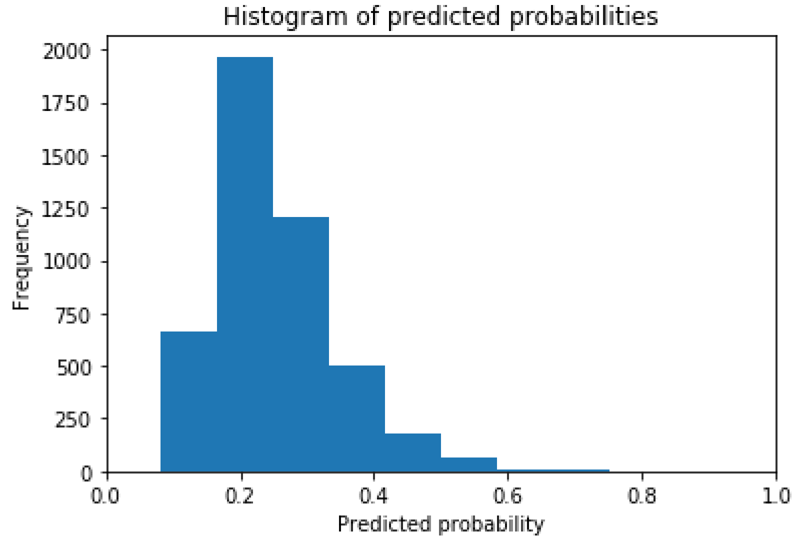
```
AUC for classifier: CART  =  0.5188237987766984

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
RFTree: 0.747185 (0.002265)

Prediction: RFTree
Accuracy Score: 0.7578993244715624

Confusion Matrix:
 [[3422   61]
 [1050   56]]


Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.98      0.86      3483
           1       0.48      0.05      0.09      1106

    accuracy                           0.76      4589
   macro avg       0.62      0.52      0.48      4589
weighted avg       0.70      0.76      0.68      4589


Prediction probabilities distribution: RFTree
```



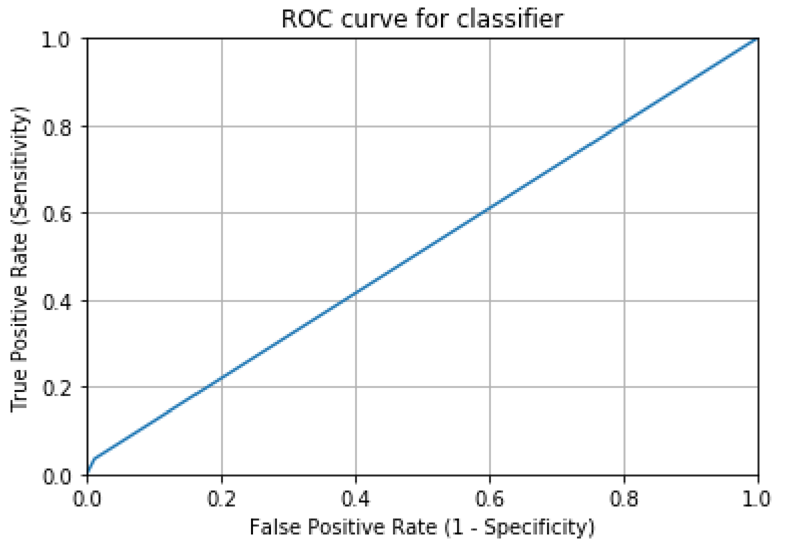

```
ROC curve for classifier: RFTree
```



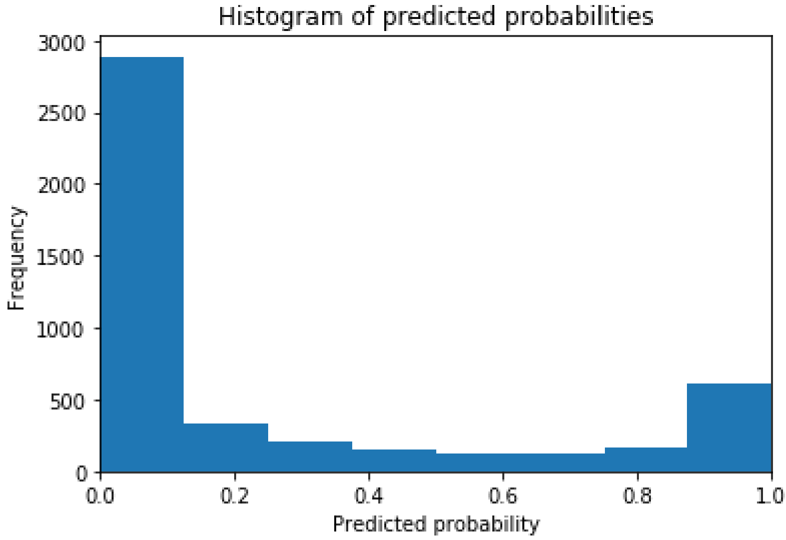
```
AUC for classifier: RFTree  =  0.5863815930541473

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
GrB: 0.749292 (0.003736)

Prediction: GrB
Accuracy Score: 0.758553061669209

Confusion Matrix:
 [[3442   41]
 [1067   39]]


Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.99      0.86      3483
           1       0.49      0.04      0.07      1106

    accuracy                           0.76      4589
   macro avg       0.63      0.51      0.46      4589
weighted avg       0.70      0.76      0.67      4589



Prediction probabilities distribution: GrB
```



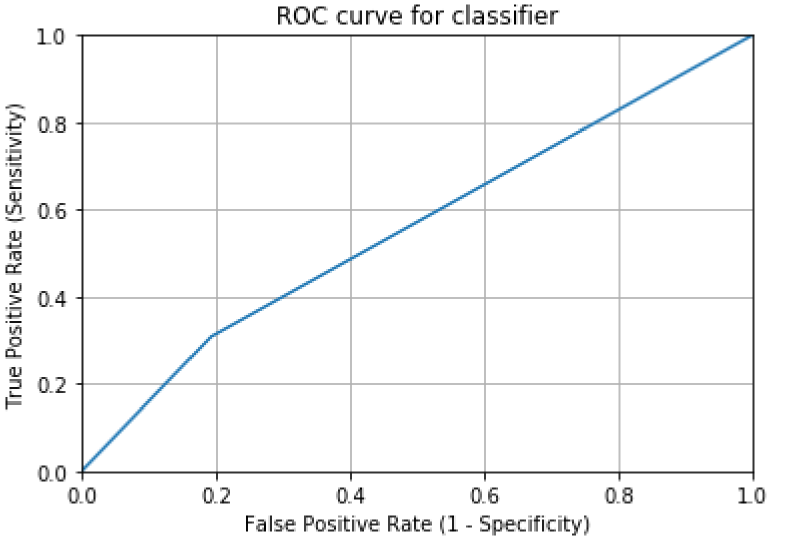

```
ROC curve for classifier: GrB
```



```
AUC for classifier: GrB  =  0.6184737908072222

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
NB: 0.676644 (0.013823)

Prediction: NB
Accuracy Score: 0.6868598823273044

Confusion Matrix:
 [[2811  672]
 [ 765  341]]


Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.81      0.80      3483
           1       0.34      0.31      0.32      1106

    accuracy                           0.69      4589
   macro avg       0.56      0.56      0.56      4589
weighted avg       0.68      0.69      0.68      4589



Prediction probabilities distribution: NB
```



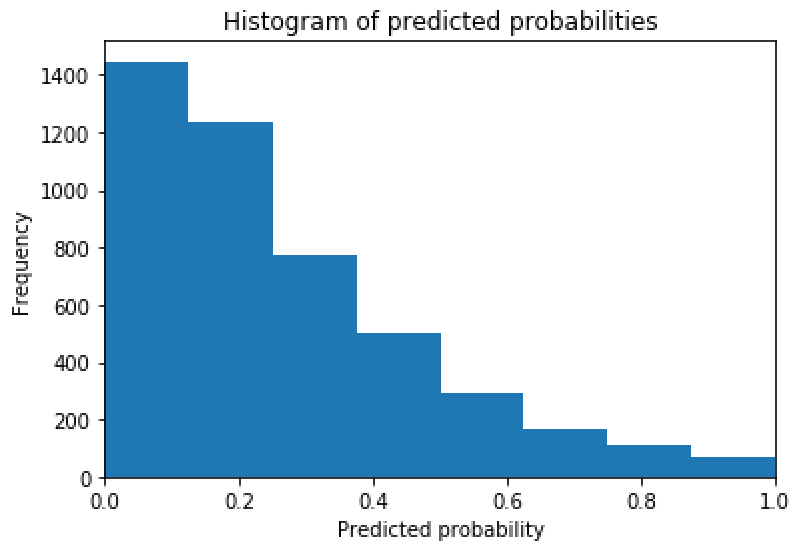

```
ROC curve for classifier: NB
```



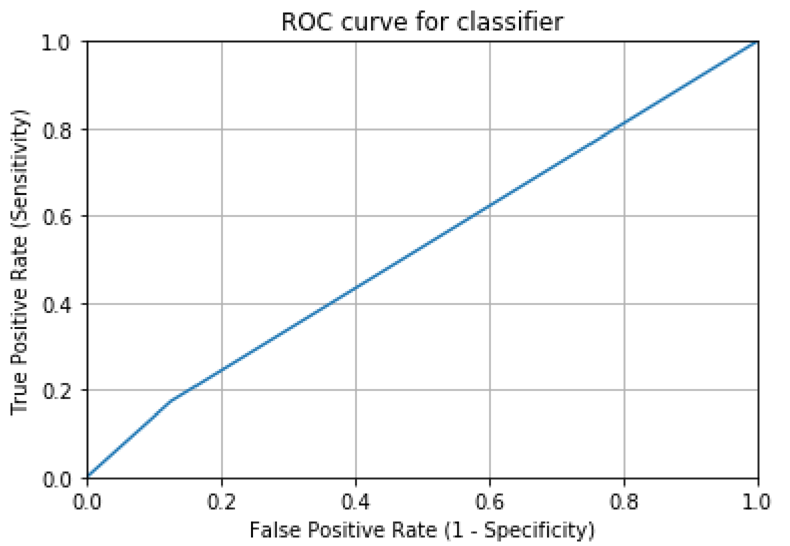
```
AUC for classifier: NB  =  0.596287885513673

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
SVM: 0.749001 (0.001396)

Prediction: SVM
Accuracy Score: 0.7596426236652866

Confusion Matrix:
 [[3471   12]
 [1091   15]]


Classification Report:
               precision    recall  f1-score   support

           0       0.76      1.00      0.86      3483
           1       0.56      0.01      0.03      1106

    accuracy                           0.76      4589
   macro avg       0.66      0.51      0.44      4589
weighted avg       0.71      0.76      0.66      4589


An AttributeError has occurred. I can't show the histogram of predicted probabilitie
s, ROC curve for classifier and AUC for the classifier.

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX
```

```
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Estimate Model Accuracy: mean (std)
Deep: 0.702069 (0.009081)


C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Prediction: Deep
Accuracy Score: 0.7053824362606232

Confusion Matrix:
 [[3045  438]
 [ 914  192]]


Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.87      0.82      3483
           1       0.30      0.17      0.22      1106

    accuracy                           0.71      4589
   macro avg       0.54      0.52      0.52      4589
weighted avg       0.66      0.71      0.67      4589



Prediction probabilities distribution: Deep
```




```
ROC curve for classifier: Deep
```



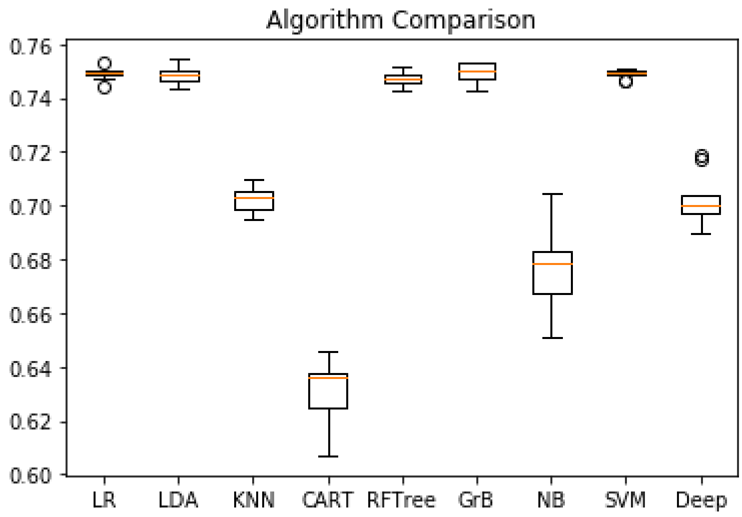

```
AUC for classifier: Deep  =  0.5457453121568517
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Plot of **the spread and the mean accuracy** of each model.

There is a **population of accuracy** measures for each algorithm because each algorithm was evaluated 10 times (via 10 fold-cross validation).

In [74]:
```
# Compare Algorithms
plt.boxplot(results, labels=names)
plt.title('Algorithm Comparison')
plt.show()
```



For Standard Scaled Data the chosen model is Deep Learning.

6.1.b Create a Validation Dataset with Robust Scaled Data

In [75]:
```
# Split-out validation dataset

X = donor_dataset_scaled_robust.drop('TARGET_B',axis=1)
y = donor_dataset_scaled_robust['TARGET_B']
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.25,
random_state=1)
```

In [76]:
```
print ('All Data: ', X.size)
print ('Train Size: ', X_train.size,'=', X_train.size/ X.size*100,'%')
print ('Test Size: ', X_validation.size, '=', X_validation.size/X.size*100,'%')
```

```
All Data:  789222
Train Size:  591895 = 74.99727579819113 %
Test Size:  197327 = 25.002724201808867 %
```

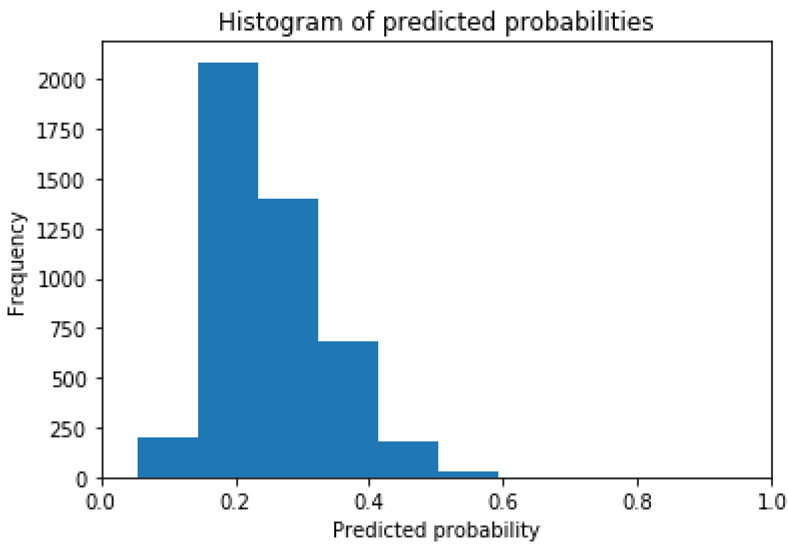
6.2.b Build Models, Make and Evaluate Predictions on different models with Robust Scaled Data

In [77]:

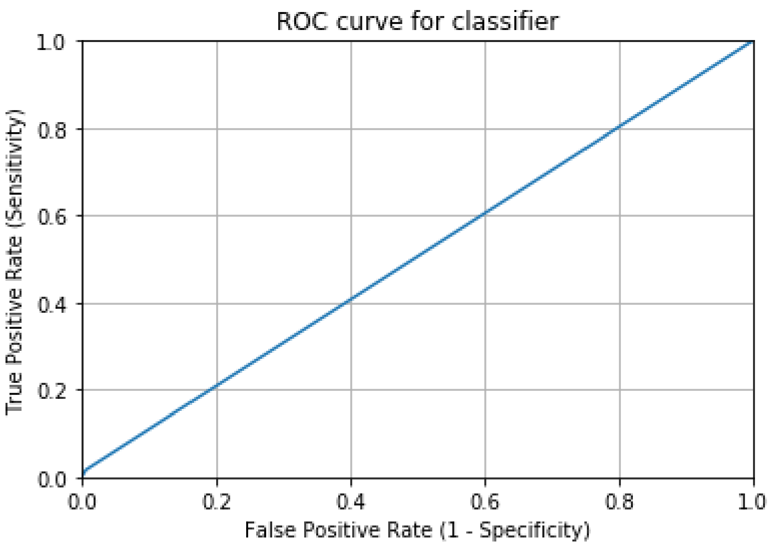
```
# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('RFTree', RandomForestClassifier()))
models.append(('GrB', GradientBoostingClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
models.append(('Deep', MLPClassifier()))

# evaluate each model in turn
results = []
names = []
print()

for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accurac
y')
    results.append(cv_results)
    names.append(name)
    print('Estimate Model Accuracy: mean (std)','\n%s: %f (%f)' % (name, cv_results.m
ean(), cv_results.std()),'\n')


    # Make predictions on validation dataset

    model.fit(X_train, Y_train)
    predictions = model.predict(X_validation)

    # Evaluate predictions

    print("Prediction:", '%s' % (name))
    print('Accuracy Score:',accuracy_score(Y_validation, predictions))
    print()
    print('Confusion Matrix:\n',confusion_matrix(Y_validation, predictions))
    print('\n')
    print('Classification Report:\n',classification_report(Y_validation, predictions
))
    print()


    try:
        # store the predicted probabilities for class 1
        Y_pred_prob = model.predict_proba(X_validation)[:, 1]

    except AttributeError:
        print("An AttributeError has occurred. I can't show the histogram of predicte
d probabilities, ROC curve for classifier and AUC for the classifier.")

    else:

        # histogram of predicted probabilities

        print()
        print("Prediction probabilities distribution:", '%s' % (name))
        plt.hist(Y_pred_prob, bins=8)
        plt.xlim(0, 1)
        plt.title('Histogram of predicted probabilities')
        plt.xlabel('Predicted probability')
        plt.ylabel('Frequency')
        plt.show()
```

```
        # ROC curve for classifier
        # IMPORTANT: first argument is true values, second argument is predicted prob
abilities
        print()
        print("ROC curve for classifier:", '%s' % (name))
        fpr, tpr, thresholds = metrics.roc_curve(Y_validation, predictions)
        plt.plot(fpr, tpr)
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.0])
        plt.title('ROC curve for classifier')
        plt.xlabel('False Positive Rate (1 - Specificity)')
        plt.ylabel('True Positive Rate (Sensitivity)')
        plt.grid(True)
        plt.show()

        # IMPORTANT: first argument is true values, second argument is predicted prob
abilities
        print("AUC for classifier:", '%s' % (name),' = ', metrics.roc_auc_score(Y_val
idation, Y_pred_prob))


    print()
    print('XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXX','\n')
```

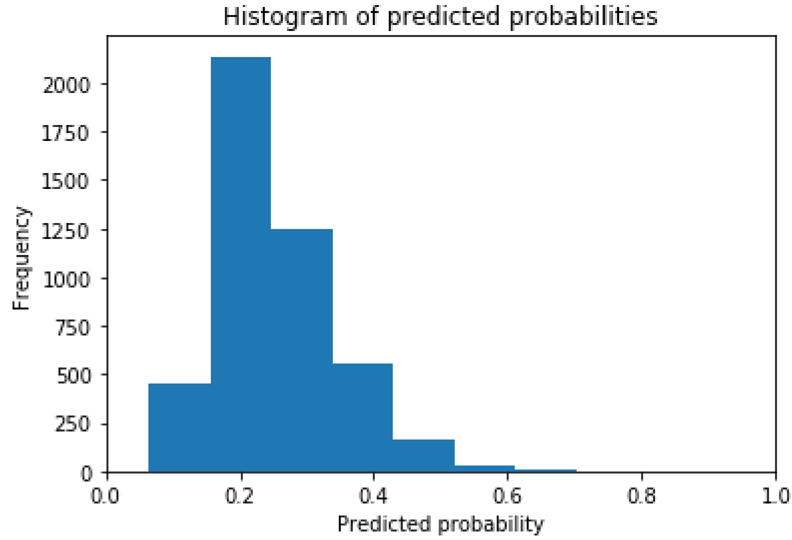
```
Estimate Model Accuracy: mean (std)
LR: 0.749001 (0.002130)

Prediction: LR
Accuracy Score: 0.7583351492699935

Confusion Matrix:
 [[3462   21]
 [1088   18]]


Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.99      0.86      3483
           1       0.46      0.02      0.03      1106

    accuracy                           0.76      4589
   macro avg       0.61      0.51      0.45      4589
weighted avg       0.69      0.76      0.66      4589


Prediction probabilities distribution: LR
```



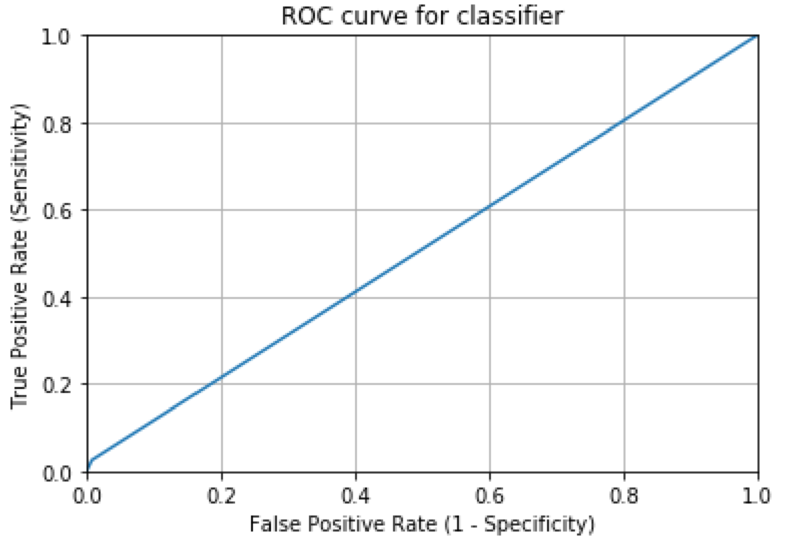

```
ROC curve for classifier: LR
```



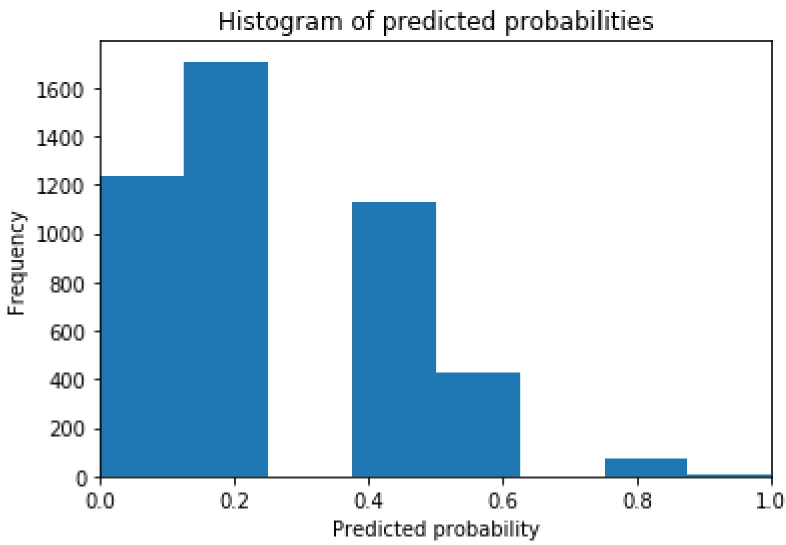
```
AUC for classifier: LR  =  0.6131335928215528

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
LDA: 0.748275 (0.003000)

Prediction: LDA
Accuracy Score: 0.7592067988668555

Confusion Matrix:
 [[3456   27]
 [1078   28]]


Classification Report:
               precision    recall  f1-score   support

           0       0.76      0.99      0.86      3483
           1       0.51      0.03      0.05      1106

    accuracy                           0.76      4589
   macro avg       0.64      0.51      0.46      4589
weighted avg       0.70      0.76      0.67      4589


Prediction probabilities distribution: LDA
```



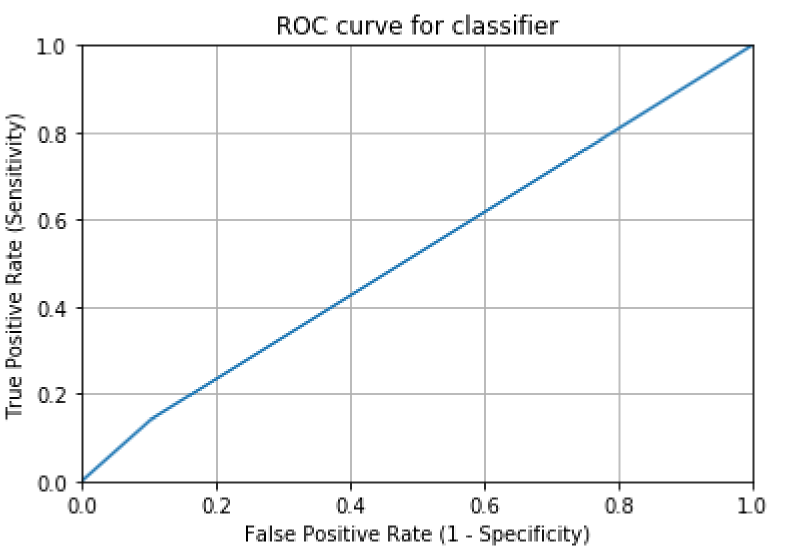

```
ROC curve for classifier: LDA
```



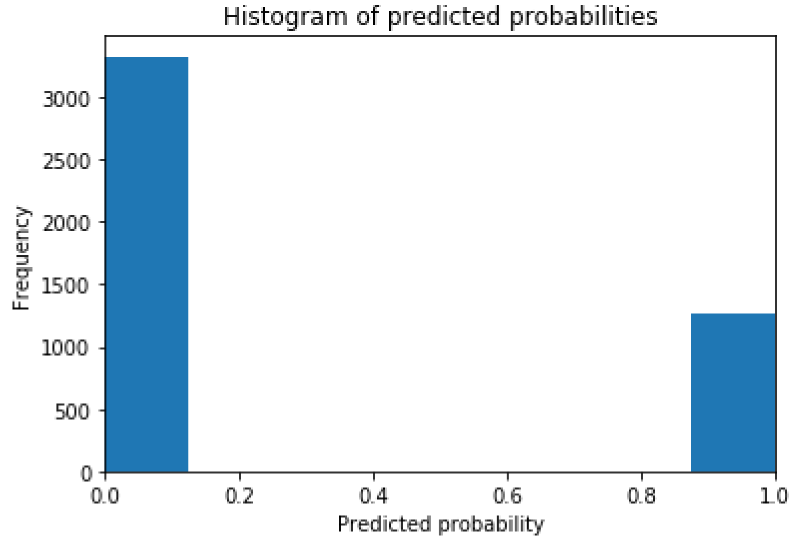
```
AUC for classifier: LDA  =  0.6120910711235508

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
KNN: 0.709699 (0.004861)

Prediction: KNN
Accuracy Score: 0.714970581826106

Confusion Matrix:
 [[3126  357]
 [ 951  155]]


Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.90      0.83      3483
           1       0.30      0.14      0.19      1106

    accuracy                           0.71      4589
   macro avg       0.53      0.52      0.51      4589
weighted avg       0.65      0.71      0.67      4589



Prediction probabilities distribution: KNN
```



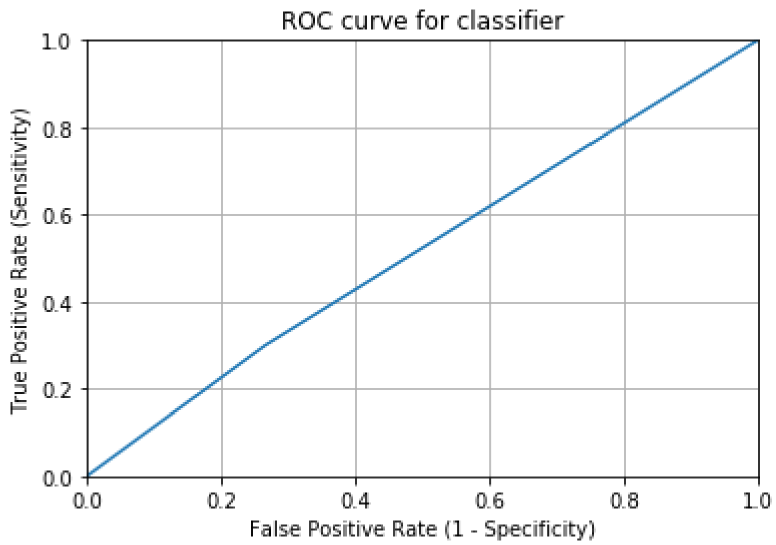

```
ROC curve for classifier: KNN
```



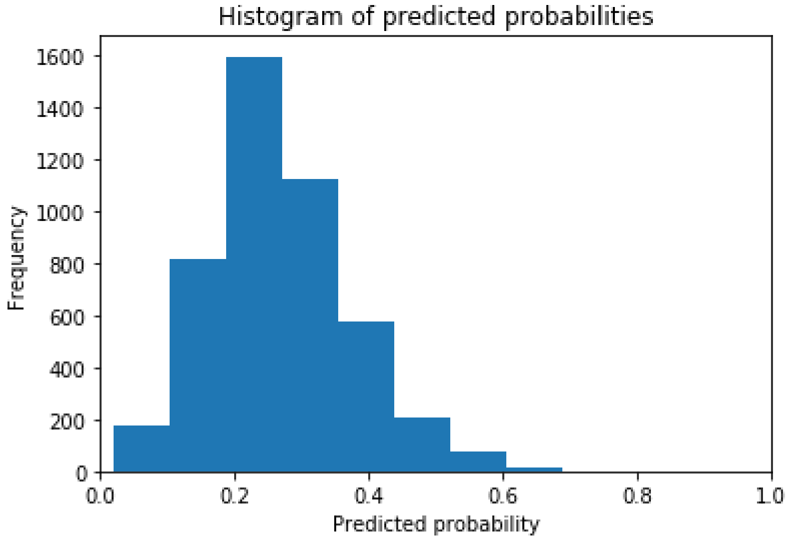
```
AUC for classifier: KNN  =  0.5514588035194452

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
CART: 0.633056 (0.009845)

Prediction: CART
Accuracy Score: 0.6291130965351929

Confusion Matrix:
 [[2555  928]
 [ 774  332]]


Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.73      0.75      3483
           1       0.26      0.30      0.28      1106

    accuracy                           0.63      4589
   macro avg       0.52      0.52      0.52      4589
weighted avg       0.65      0.63      0.64      4589


Prediction probabilities distribution: CART
```



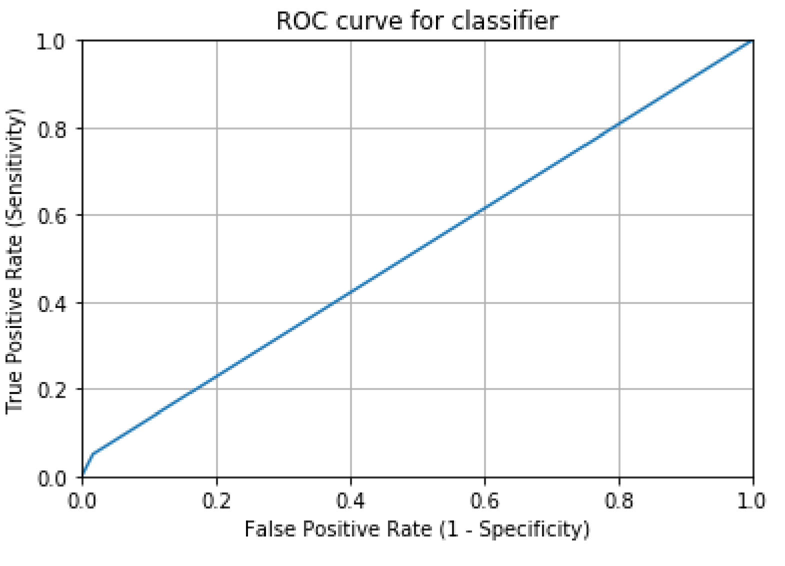

```
ROC curve for classifier: CART
```



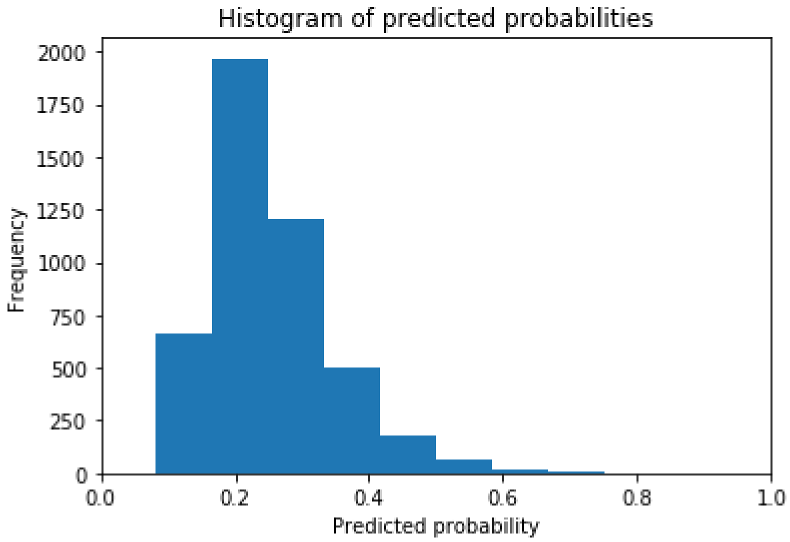
```
AUC for classifier: CART  =  0.5168719261055637

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
RFTree: 0.746023 (0.002387)

Prediction: RFTree
Accuracy Score: 0.7583351492699935

Confusion Matrix:
 [[3424   59]
 [1050   56]]


Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.98      0.86      3483
           1       0.49      0.05      0.09      1106

    accuracy                           0.76      4589
   macro avg       0.63      0.52      0.48      4589
weighted avg       0.70      0.76      0.68      4589



Prediction probabilities distribution: RFTree
```



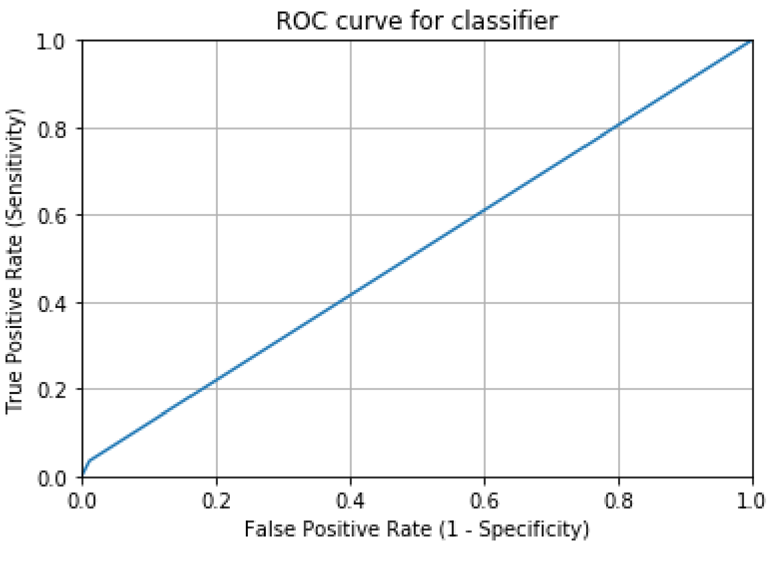

```
ROC curve for classifier: RFTree
```



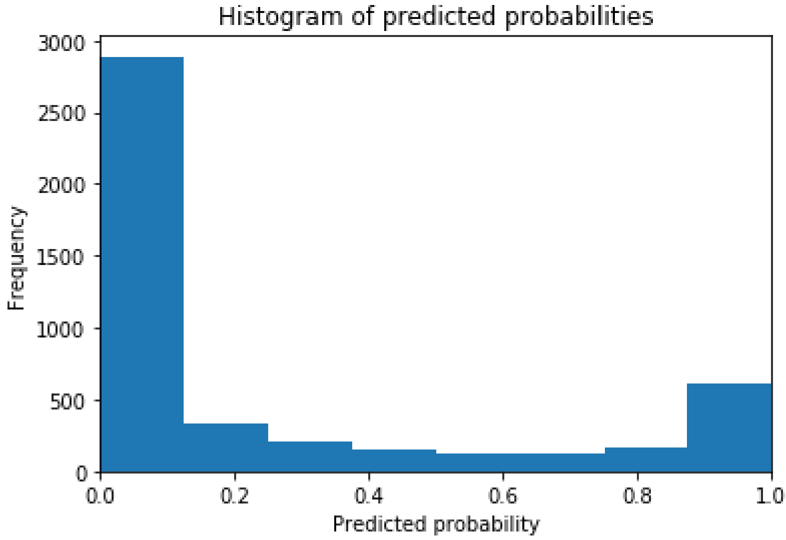
```
AUC for classifier: RFTree  =  0.5872257864211549

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
GrB: 0.749291 (0.003430)

Prediction: GrB
Accuracy Score: 0.758553061669209

Confusion Matrix:
 [[3442   41]
 [1067   39]]


Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.99      0.86      3483
           1       0.49      0.04      0.07      1106

    accuracy                           0.76      4589
   macro avg       0.63      0.51      0.46      4589
weighted avg       0.70      0.76      0.67      4589



Prediction probabilities distribution: GrB
```



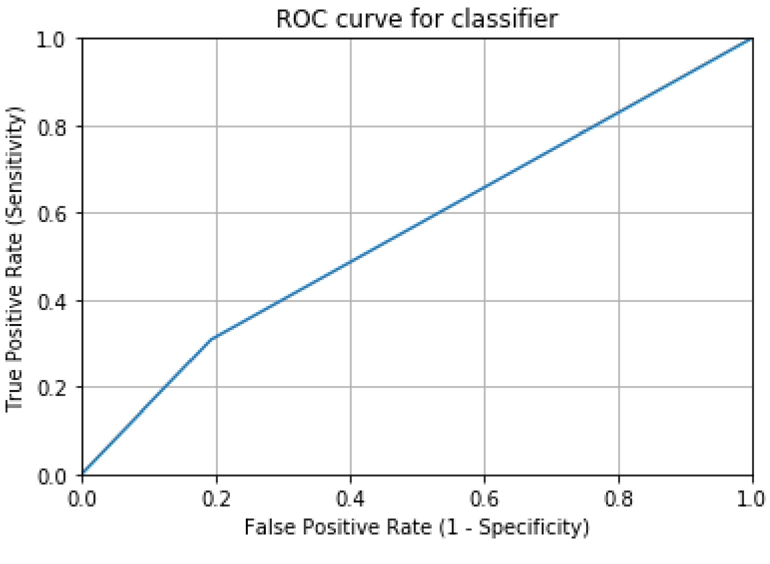

```
ROC curve for classifier: GrB
```



```
AUC for classifier: GrB  =  0.6185163639044514

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
NB: 0.676644 (0.013823)

Prediction: NB
Accuracy Score: 0.6868598823273044

Confusion Matrix:
 [[2811  672]
 [ 765  341]]


Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.81      0.80      3483
           1       0.34      0.31      0.32      1106

    accuracy                           0.69      4589
   macro avg       0.56      0.56      0.56      4589
weighted avg       0.68      0.69      0.68      4589



Prediction probabilities distribution: NB
```



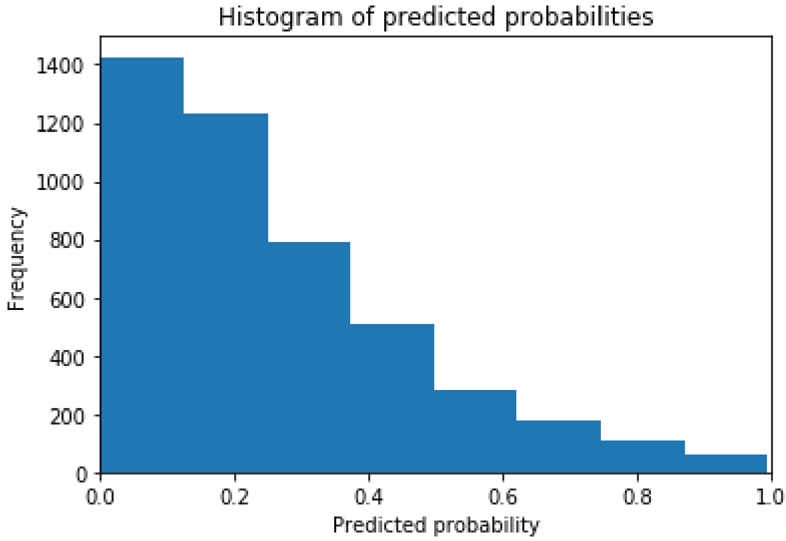

```
ROC curve for classifier: NB
```



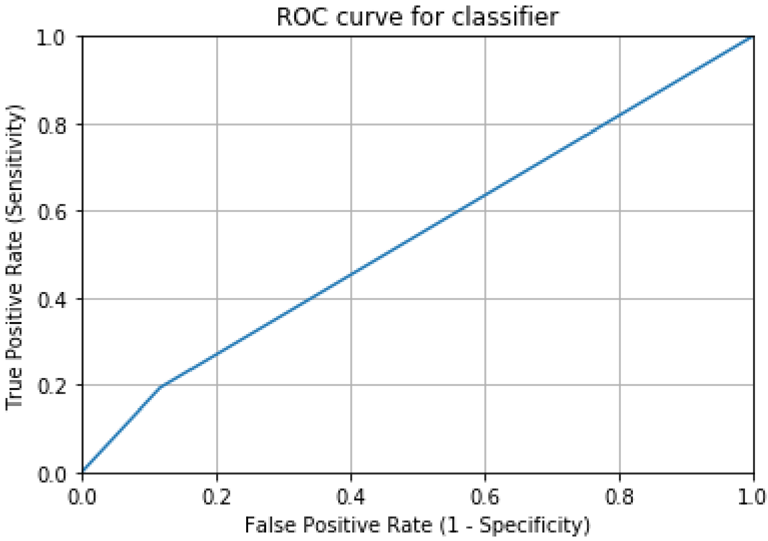
```
AUC for classifier: NB  =  0.596287885513673

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Estimate Model Accuracy: mean (std)
SVM: 0.747911 (0.001391)

Prediction: SVM
Accuracy Score: 0.7598605360645021

Confusion Matrix:
 [[3473   10]
 [1092   14]]


Classification Report:
               precision    recall  f1-score   support

           0       0.76      1.00      0.86      3483
           1       0.58      0.01      0.02      1106

    accuracy                           0.76      4589
   macro avg       0.67      0.50      0.44      4589
weighted avg       0.72      0.76      0.66      4589


An AttributeError has occurred. I can't show the histogram of predicted probabilitie
s, ROC curve for classifier and AUC for the classifier.

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX
```

```
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Estimate Model Accuracy: mean (std)
Deep: 0.709334 (0.008516)


C:\Users\aless\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_percep
tron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Prediction: Deep
Accuracy Score: 0.71671388101983

Confusion Matrix:
 [[3075  408]
 [ 892  214]]


Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.88      0.83      3483
           1       0.34      0.19      0.25      1106

    accuracy                           0.72      4589
   macro avg       0.56      0.54      0.54      4589
weighted avg       0.67      0.72      0.69      4589


Prediction probabilities distribution: Deep
```




```
ROC curve for classifier: Deep
```



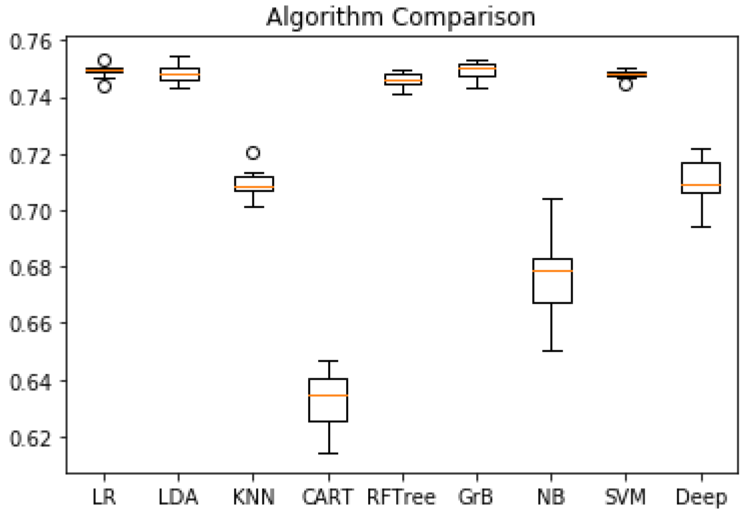

```
AUC for classifier: Deep  =  0.574323801632211
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXX

Plot of **the spread and the mean accuracy** of each model.

There is a **population of accuracy** measures for each algorithm because each algorithm was evaluated 10 times (via 10 fold-cross validation).

In [78]:
```
# Compare Algorithms
plt.boxplot(results, labels=names)
plt.title('Algorithm Comparison')
plt.show()
```



For Robust Scaled Data the chosen model is Deep Learning (Deep).

6.3 Choose the best model

The **Deep Learning (Deep)** model with **Robust Scaled Data** has been chosen, after the valuation of the number of False Positives , the Predictions Accuracy and the Precision.

Table of Content

7. Train the Final Machine Learning Model

In [42]:
```python
# Finalize a model by applying the chosen machine learning procedure on all data

X_final_training = donor_dataset_scaled_robust.drop('TARGET_B',axis=1)
y_final_training = donor_dataset_scaled_robust['TARGET_B']

model = MLPClassifier(max_iter=500)

print('start training')
start = time.time()

model.fit(X_final_training,y_final_training)

print('training_finished')
end = time.time()
print('time: ', end - start)
```

```
start training
training_finished
time:  77.68586158752441
```

Table of Content

8. Save the Final Machine Learning Model

In [ ]:
```python
# import joblib
```

In [43]:
```python
# save the model to disk
filename_joblib = 'Deep_finalized_model_saved_with_Joblib.sav'
joblib.dump(model, filename_joblib, compress=0)  # 0 or False is no compression. High
er value means more compression,
                                                  # but also slower read and write times. Usi
ng a value of 3 is often a good
                                                  # compromise. If compress is True, the comp
ression level used is 3.
```

Out[43]:
```
['Deep_finalized_model_saved_with_Joblib.sav']
```

Table of Content

9. Data Preparation of New Data

Dropped Columns:

- HOME_OWNER
- INCOME_GROUP
- OVERLAY_SOURCE
- WEALTH_RATING

In [42]:
```
# A new prospective_dataset with dropped: HOME_OWNER' , 'INCOME_GROUP', OVERLAY_SOURC
E
#                                    , 'WEALTH_RATING'

column_list_prospective = ['CONTROL_NUMBER','MONTHS_SINCE_ORIGIN',
       'DONOR_AGE', 'IN_HOUSE', 'URBANICITY', 'SES', 'CLUSTER_CODE',
       'DONOR_GENDER', 'PUBLISHED_PHONE',
       'MOR_HIT_RATE', 'MEDIAN_HOME_VALUE',
       'MEDIAN_HOUSEHOLD_INCOME', 'PCT_OWNER_OCCUPIED', 'PER_CAPITA_INCOME',
       'PCT_ATTRIBUTE1', 'PCT_ATTRIBUTE2', 'PCT_ATTRIBUTE3', 'PCT_ATTRIBUTE4',
       'PEP_STAR', 'RECENT_STAR_STATUS', 'RECENCY_STATUS_96NK',
       'FREQUENCY_STATUS_97NK', 'RECENT_RESPONSE_PROP', 'RECENT_AVG_GIFT_AMT',
       'RECENT_CARD_RESPONSE_PROP', 'RECENT_AVG_CARD_GIFT_AMT',
       'RECENT_RESPONSE_COUNT', 'RECENT_CARD_RESPONSE_COUNT',
       'MONTHS_SINCE_LAST_PROM_RESP', 'LIFETIME_CARD_PROM', 'LIFETIME_PROM',
       'LIFETIME_GIFT_AMOUNT', 'LIFETIME_GIFT_COUNT', 'LIFETIME_AVG_GIFT_AMT',
       'LIFETIME_GIFT_RANGE', 'LIFETIME_MAX_GIFT_AMT', 'LIFETIME_MIN_GIFT_AMT',
       'LAST_GIFT_AMT', 'CARD_PROM_12', 'NUMBER_PROM_12',
       'MONTHS_SINCE_LAST_GIFT', 'MONTHS_SINCE_FIRST_GIFT', 'FILE_AVG_GIFT',
       'FILE_CARD_GIFT']
prospective_dataset = prospective_data[column_list_prospective]
```

In [43]:
```
print(prospective_dataset.shape)
```

```
(2148, 44)
```

In [44]:
```
prospective_dataset.tail()
```

Out[44]:

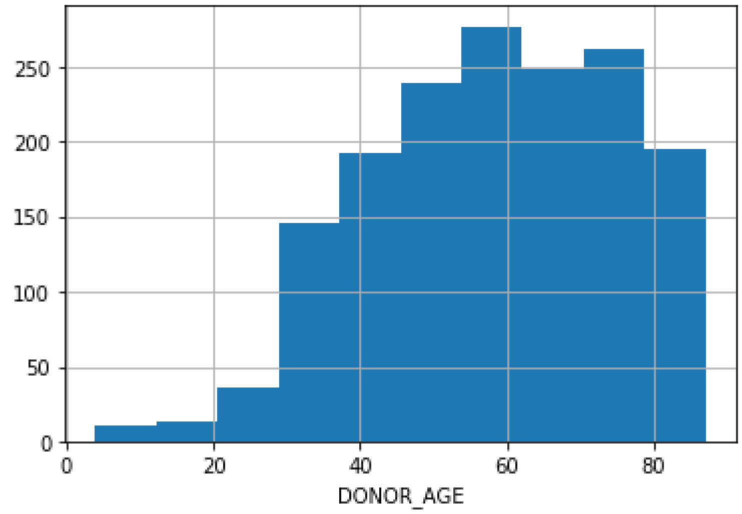
	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
2143	190842	101	47.0	1	C	1	
2144	191056	41	17.0	1	U	1	
2145	191164	89	55.0	0	?	?	
2146	191484	65	42.0	1	?	?	
2147	191710	137	77.0	1	C	1	

[DONOR_AGE]

Handle missing values

```
In [45]: prospective_dataset['DONOR_AGE'].hist()
         plt.xlabel('DONOR_AGE')
```

```
Out[45]: Text(0.5, 0, 'DONOR_AGE')
```




```
In [46]: # deal with age missing values

         # calculate the mean and the median for the whole population

         median_DONOR_AGE_prospective = prospective_dataset['DONOR_AGE'].median()
         print('Median = ',median_DONOR_AGE_prospective)
         mean_DONOR_AGE_prospective = prospective_dataset['DONOR_AGE'].mean()
         print('Mean = ',mean_DONOR_AGE_prospective)
```

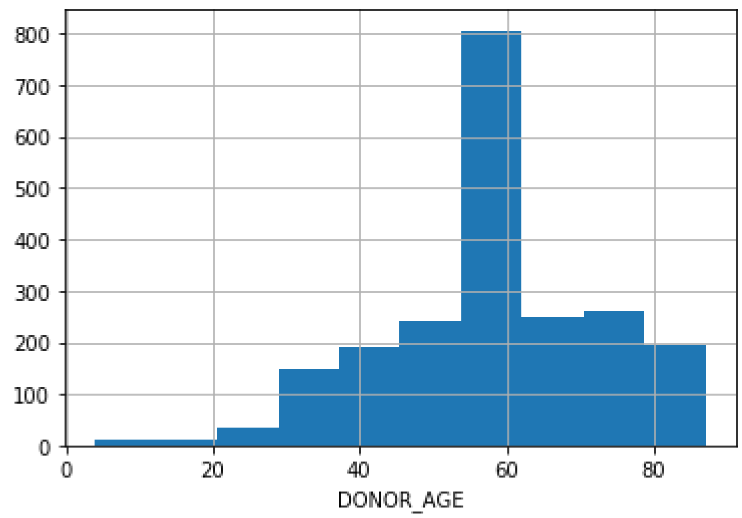
```
Median =  59.0
Mean =  58.18591723285979
```

Because the **distribution is not normal**, NaN values will be replaced with the **median**.

```
In [47]: prospective_dataset['DONOR_AGE'].fillna(median_DONOR_AGE_prospective, inplace = True)
```

```
In [48]: prospective_dataset['DONOR_AGE'].hist()
         plt.xlabel('DONOR_AGE')
```

```
Out[48]: Text(0.5, 0, 'DONOR_AGE')
```

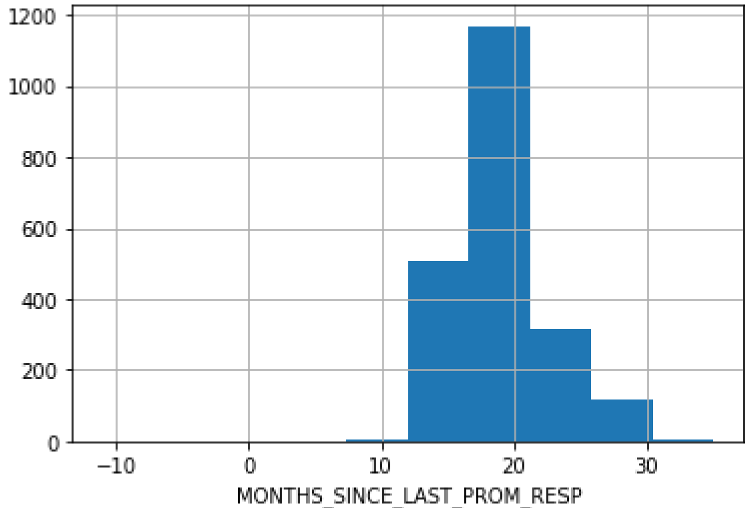


[MONTHS_SINCE_LAST_PROM_RESP]

Handle missing values

In [49]:
```
prospective_dataset['MONTHS_SINCE_LAST_PROM_RESP'].hist()
plt.xlabel('MONTHS_SINCE_LAST_PROM_RESP')
```

Out[49]:
```
Text(0.5, 0, 'MONTHS_SINCE_LAST_PROM_RESP')
```

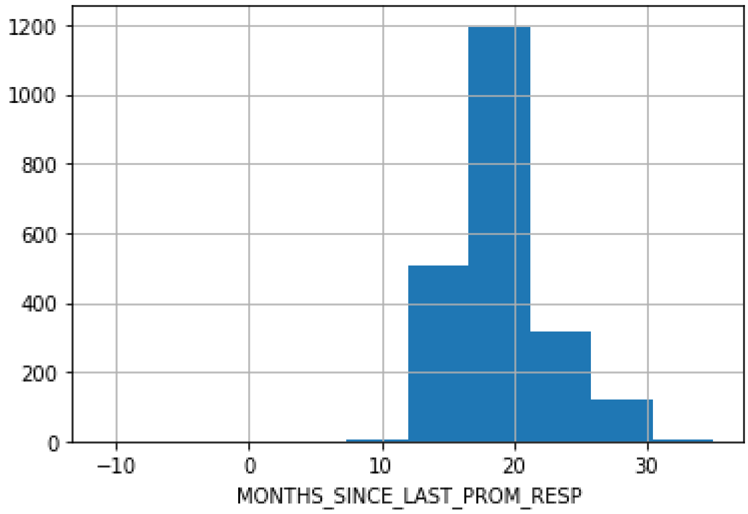


In [50]:
```
median_MONTHS_SINCE_LAST_PROM_RESP_prospective = prospective_dataset['MONTHS_SINCE_LA
ST_PROM_RESP'].median()
print('Median = ',median_MONTHS_SINCE_LAST_PROM_RESP_prospective)
mean_MONTHS_SINCE_LAST_PROM_RESP_prospective = prospective_dataset['MONTHS_SINCE_LAST
_PROM_RESP'].mean()
print('Mean = ',mean_MONTHS_SINCE_LAST_PROM_RESP_prospective)
```

```
Median =  18.0
Mean =  19.07775683317625
```

Because the distribution is quite **normal**, NaN values will be replaced with the **mean**.

In [51]:
```
# replace NaN with the mean

prospective_dataset['MONTHS_SINCE_LAST_PROM_RESP'].fillna(mean_MONTHS_SINCE_LAST_PROM
_RESP_prospective, inplace = True)
```

In [52]:
```
prospective_dataset['MONTHS_SINCE_LAST_PROM_RESP'].hist()
plt.xlabel('MONTHS_SINCE_LAST_PROM_RESP')
```
Out[52]:
```
Text(0.5, 0, 'MONTHS_SINCE_LAST_PROM_RESP')
```


[SES]

Str values '1','2','3','4' replaced with numbers and '?' with number 5.

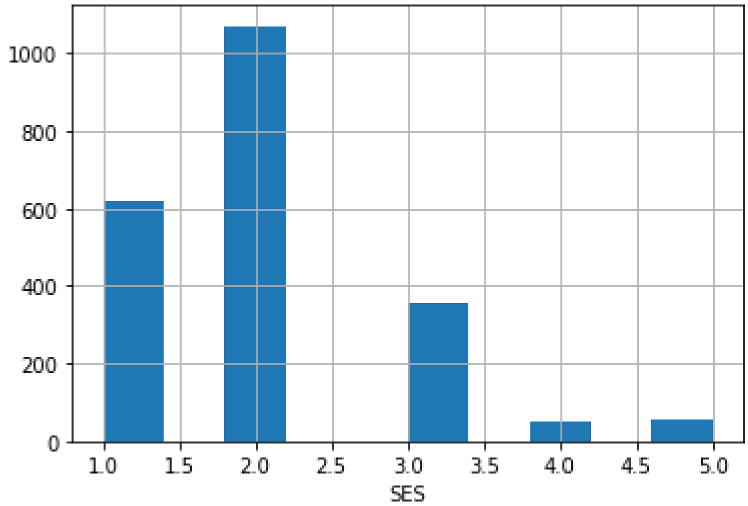
In [53]:
```
prospective_dataset['SES'].unique()
```
Out[53]:
```
array(['2', '1', '3', '?', '4'], dtype=object)
```
In [54]:
```
prospective_dataset['SES'].replace('1',1,inplace=True)
prospective_dataset['SES'].replace('2',2,inplace=True)
prospective_dataset['SES'].replace('3',3,inplace=True)
prospective_dataset['SES'].replace('4',4,inplace=True)
prospective_dataset['SES'].replace('?',5,inplace=True)
```

In [55]:
```python
prospective_dataset['SES'].hist()
plt.xlabel('SES')
print(prospective_dataset['SES'].dtype)
prospective_dataset.tail()
```

```
int64
```

Out[55]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
2143	190842	101	47.0	1	C	1	
2144	191056	41	17.0	1	U	1	
2145	191164	89	55.0	0	?	5	
2146	191484	65	42.0	1	?	5	
2147	191710	137	77.0	1	C	1	



[CLUSTER_CODE]

Replaced ' .' with the number '54' and after str values with numbers.

In [56]:
```python
prospective_dataset['CLUSTER_CODE'].unique()
```

Out[56]:
```
array(['46', '43', '35', '02', '40', '37', '24', '11', '21', '16', '42',
       '27', '09', '12', '45', '26', '53', '25', '18', ' .', '29', '28',
       '14', '30', '39', '05', '20', '08', '17', '34', '07', '31', '13',
       '03', '36', '06', '04', '33', '49', '01', '50', '22', '23', '38',
       '41', '51', '10', '47', '32', '15', '44', '48', '19', '52'],
      dtype=object)
```

In [57]:
```python
prospective_dataset['CLUSTER_CODE'].replace(' .','54',inplace=True)
```

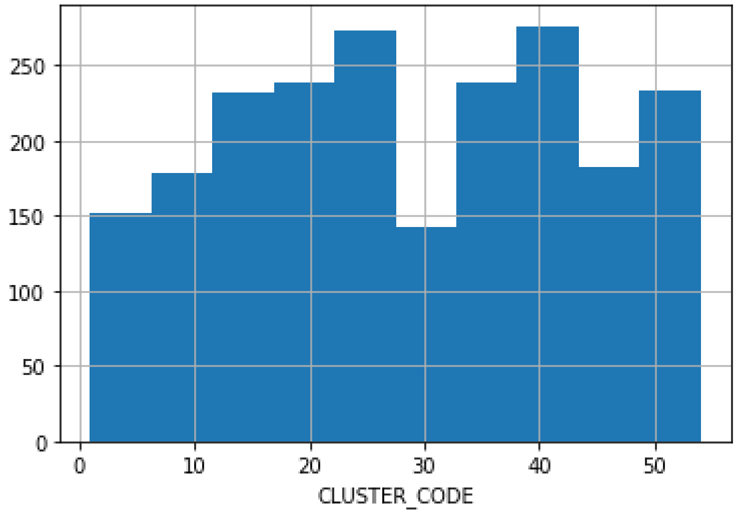
In [58]:
```python
prospective_dataset['CLUSTER_CODE'] = prospective_dataset['CLUSTER_CODE'].astype(int)
```

In [59]:
```python
prospective_dataset['CLUSTER_CODE'].hist()
plt.xlabel('CLUSTER_CODE')
print(prospective_dataset['CLUSTER_CODE'].dtype)
prospective_dataset.tail()
```

int32

Out[59]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
2143	190842	101	47.0	1	C	1	
2144	191056	41	17.0	1	U	1	
2145	191164	89	55.0	0	?	5	
2146	191484	65	42.0	1	?	5	
2147	191710	137	77.0	1	C	1	



[DONOR_GENDER]

Dropped rows with gender inputted wrongly.

In [60]:
```python
prospective_dataset['DONOR_GENDER'].unique()
```

Out[60]: array(['F', 'M', 'U'], dtype=object)

In [61]:
```python
print(prospective_dataset.shape)
prospective_dataset.tail()
```

(2148, 44)

Out[61]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
2143	190842	101	47.0	1	C	1	
2144	191056	41	17.0	1	U	1	
2145	191164	89	55.0	0	?	5	
2146	191484	65	42.0	1	?	5	
2147	191710	137	77.0	1	C	1	

In [62]:
```
# Get names of indexes for which column ['DONOR_GENDER'] has value 'U'
indexNames = prospective_dataset[prospective_dataset['DONOR_GENDER'] == 'U'].index

# Delete these row indexes from dataFrame
prospective_dataset.drop(indexNames, inplace=True)
```

In [63]:
```
print(prospective_dataset.shape)
prospective_dataset.tail()
```

```
(2041, 44)
```

Out[63]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
2143	190842	101	47.0	1	C	1	
2144	191056	41	17.0	1	U	1	
2145	191164	89	55.0	0	?	5	
2146	191484	65	42.0	1	?	5	
2147	191710	137	77.0	1	C	1	

In [64]:
```
prospective_dataset.to_csv('prospective_prepared_for_analysis.csv',index=None)
```

Replaced gender values with 1 for female, 0 for male with a categorical endoding.

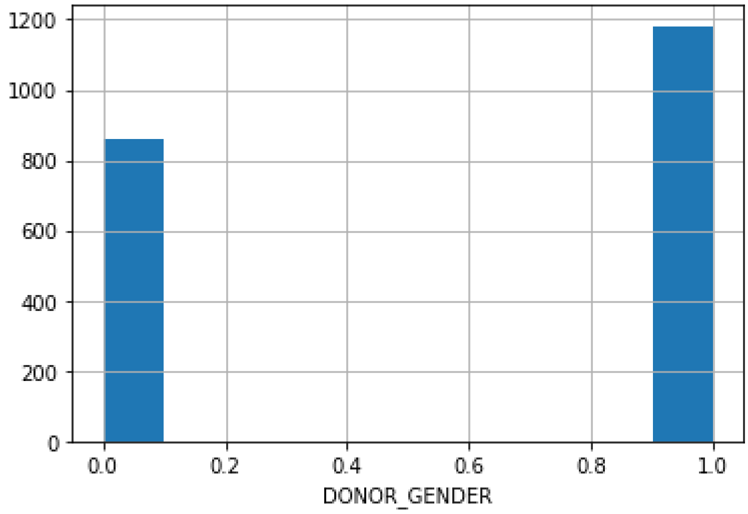
In [66]:
```
prospective_dataset['DONOR_GENDER'] = np.where(prospective_dataset['DONOR_GENDER']==
'F',1,0) # categorical encoding
```

In [67]:
```
print(prospective_dataset.shape)
prospective_dataset['DONOR_GENDER'].hist()
plt.xlabel('DONOR_GENDER')
prospective_dataset.tail()
```

```
(2041, 44)
```

Out[67]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
2143	190842	101	47.0	1	C	1	
2144	191056	41	17.0	1	U	1	
2145	191164	89	55.0	0	?	5	
2146	191484	65	42.0	1	?	5	
2147	191710	137	77.0	1	C	1	



[URBANICITY]

Replaced:

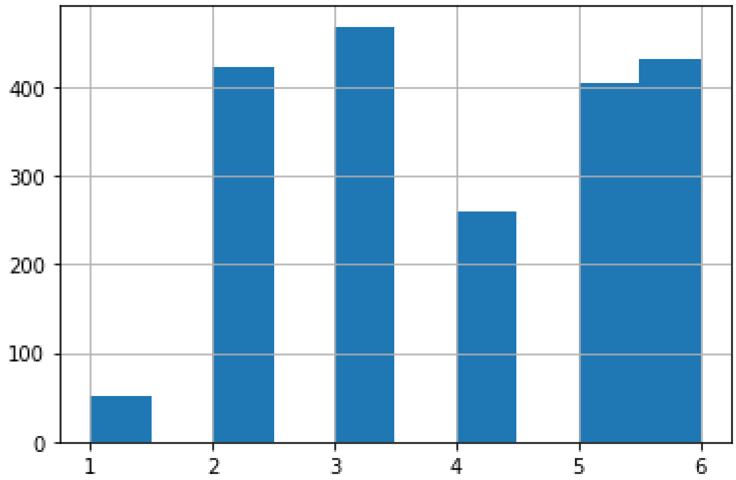
- ? -> 1
- R -> 2
- S -> 3
- U -> 4
- T -> 5
- C -> 6

In [68]:
```
prospective_dataset['URBANICITY'].replace('C',6,inplace=True)  # C = City
prospective_dataset['URBANICITY'].replace('T',5,inplace=True)  # T = Town
prospective_dataset['URBANICITY'].replace('U',4,inplace=True)  # U = Urban
prospective_dataset['URBANICITY'].replace('S',3,inplace=True)  # S = Suburban
prospective_dataset['URBANICITY'].replace('R',2,inplace=True)  # R = Rural
prospective_dataset['URBANICITY'].replace('?',1,inplace=True)  # ? = Unknown
prospective_dataset.tail()
```

Out[68]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
2143	190842	101	47.0	1	6	1	
2144	191056	41	17.0	1	4	1	
2145	191164	89	55.0	0	1	5	
2146	191484	65	42.0	1	1	5	
2147	191710	137	77.0	1	6	1	

In [69]:
```
prospective_dataset['URBANICITY'].hist()
plt.show()
```



RECENCY_STATUS_96NK

Replaced:

- A -> 1
- E -> 2
- F -> 3
- L -> 4
- N -> 5
- S -> 6

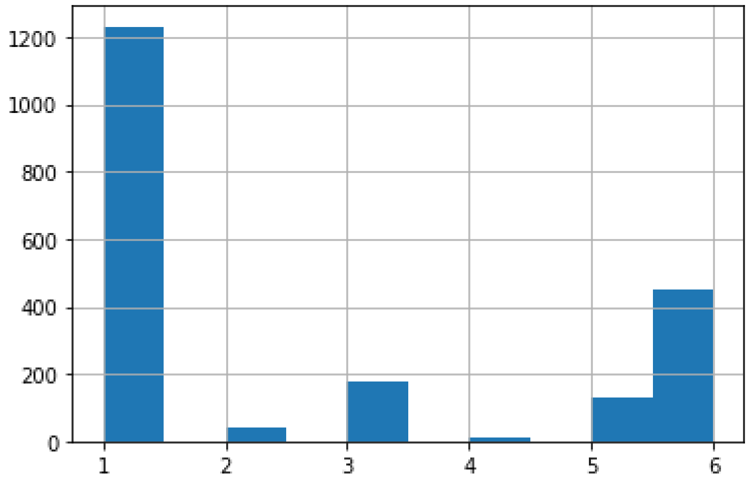
In [70]:
```
prospective_dataset['RECENCY_STATUS_96NK'].replace('A',1,inplace=True)
prospective_dataset['RECENCY_STATUS_96NK'].replace('E',2,inplace=True)
prospective_dataset['RECENCY_STATUS_96NK'].replace('F',3,inplace=True)
prospective_dataset['RECENCY_STATUS_96NK'].replace('L',4,inplace=True)
prospective_dataset['RECENCY_STATUS_96NK'].replace('N',5,inplace=True)
prospective_dataset['RECENCY_STATUS_96NK'].replace('S',6,inplace=True)
print(prospective_dataset.shape)
prospective_dataset.tail()
```

```
(2041, 44)
```

Out[70]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
2143	190842	101	47.0	1	6	1	
2144	191056	41	17.0	1	4	1	
2145	191164	89	55.0	0	1	5	
2146	191484	65	42.0	1	1	5	
2147	191710	137	77.0	1	6	1	

In [71]:
```
prospective_dataset['RECENCY_STATUS_96NK'].hist()
plt.show()
```



In [72]:
```
prospective_dataset.to_csv('Prospective Donor_ML with Python_PREPARED.csv', index=Non
e)
```

Prepare Data for Scaling

In [73]:
```
prospective_dataset
```

Out[73]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
0	139	101	59.0	0	2	2	
1	142	137	59.0	0	2	2	
2	282	17	30.0	0	5	1	
3	368	137	75.0	0	4	1	
4	387	5	59.0	0	5	2	
...	...	...	...	...	...	...	
2143	190842	101	47.0	1	6	1	
2144	191056	41	17.0	1	4	1	
2145	191164	89	55.0	0	1	5	
2146	191484	65	42.0	1	1	5	
2147	191710	137	77.0	1	6	1	

2041 rows × 44 columns

In [74]:
```
CONTROL_NUMBER = prospective_dataset['CONTROL_NUMBER']
print(CONTROL_NUMBER.shape, type(CONTROL_NUMBER))
CONTROL_NUMBER
```

```
(2041,) <class 'pandas.core.series.Series'>
```

Out[74]:
```
0          139
1          142
2          282
3          368
4          387
         ...  
2143    190842
2144    191056
2145    191164
2146    191484
2147    191710
Name: CONTROL_NUMBER, Length: 2041, dtype: int64
```

In [75]:
```
CONTROL_NUMBER_np = CONTROL_NUMBER.to_numpy()
print(CONTROL_NUMBER_np.shape, type(CONTROL_NUMBER_np))
CONTROL_NUMBER_np
```

```
(2041,) <class 'numpy.ndarray'>
```

Out[75]:
```
array([   139,    142,    282, ..., 191164, 191484, 191710], dtype=int64)
```

In [76]:
```
CONTROL_NUMBER_df = pd.DataFrame(CONTROL_NUMBER_np)
CONTROL_NUMBER_df.columns = ['CONTROL_NUMBER']
print(CONTROL_NUMBER_df.shape, type(CONTROL_NUMBER_df))
CONTROL_NUMBER_df
```

```
(2041, 1) <class 'pandas.core.frame.DataFrame'>
```

Out[76]:

	CONTROL_NUMBER
0	139
1	142
2	282
3	368
4	387
...	...
2036	190842
2037	191056
2038	191164
2039	191484
2040	191710

2041 rows × 1 columns

In [77]:
```
prospective_dataset2 = prospective_dataset.drop('CONTROL_NUMBER',axis=1)
```

In [78]:
```
print(prospective_dataset2.shape)
prospective_dataset2.tail()
```

```
(2041, 43)
```

Out[78]:

	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE	DONOR_G
2143	101	47.0	1	6	1	24	
2144	41	17.0	1	4	1	1	
2145	89	55.0	0	1	5	54	
2146	65	42.0	1	1	5	54	
2147	137	77.0	1	6	1	24	

In [79]:
```
print(prospective_dataset2.columns)
```

```
Index(['MONTHS_SINCE_ORIGIN', 'DONOR_AGE', 'IN_HOUSE', 'URBANICITY', 'SES',
       'CLUSTER_CODE', 'DONOR_GENDER', 'PUBLISHED_PHONE', 'MOR_HIT_RATE',
       'MEDIAN_HOME_VALUE', 'MEDIAN_HOUSEHOLD_INCOME', 'PCT_OWNER_OCCUPIED',
       'PER_CAPITA_INCOME', 'PCT_ATTRIBUTE1', 'PCT_ATTRIBUTE2',
       'PCT_ATTRIBUTE3', 'PCT_ATTRIBUTE4', 'PEP_STAR', 'RECENT_STAR_STATUS',
       'RECENCY_STATUS_96NK', 'FREQUENCY_STATUS_97NK', 'RECENT_RESPONSE_PROP',
       'RECENT_AVG_GIFT_AMT', 'RECENT_CARD_RESPONSE_PROP',
       'RECENT_AVG_CARD_GIFT_AMT', 'RECENT_RESPONSE_COUNT',
       'RECENT_CARD_RESPONSE_COUNT', 'MONTHS_SINCE_LAST_PROM_RESP',
       'LIFETIME_CARD_PROM', 'LIFETIME_PROM', 'LIFETIME_GIFT_AMOUNT',
       'LIFETIME_GIFT_COUNT', 'LIFETIME_AVG_GIFT_AMT', 'LIFETIME_GIFT_RANGE',
       'LIFETIME_MAX_GIFT_AMT', 'LIFETIME_MIN_GIFT_AMT', 'LAST_GIFT_AMT',
       'CARD_PROM_12', 'NUMBER_PROM_12', 'MONTHS_SINCE_LAST_GIFT',
       'MONTHS_SINCE_FIRST_GIFT', 'FILE_AVG_GIFT', 'FILE_CARD_GIFT'],
      dtype='object')
```

- **CONTROL_NUMBER_df**
- **prospective_dataset2**

Scaling

Robust Scaler

In [80]:
```
# create RobustScaler() object
scaler_robust = preprocessing.RobustScaler()

# transform data and store it in scaled_robust
scaled_robust = scaler_robust.fit_transform(prospective_dataset2)

# convert scaled_robust to DataFrame
scaled_robust_df = pd.DataFrame(scaled_robust, columns=['MONTHS_SINCE_ORIGIN', 'DONOR_AGE', 'IN_HOUSE', 'URBANICITY', 'SES',
       'CLUSTER_CODE', 'DONOR_GENDER', 'PUBLISHED_PHONE', 'MOR_HIT_RATE',
       'MEDIAN_HOME_VALUE', 'MEDIAN_HOUSEHOLD_INCOME', 'PCT_OWNER_OCCUPIED',
       'PER_CAPITA_INCOME', 'PCT_ATTRIBUTE1', 'PCT_ATTRIBUTE2',
       'PCT_ATTRIBUTE3', 'PCT_ATTRIBUTE4', 'PEP_STAR', 'RECENT_STAR_STATUS',
       'RECENCY_STATUS_96NK', 'FREQUENCY_STATUS_97NK', 'RECENT_RESPONSE_PROP',
       'RECENT_AVG_GIFT_AMT', 'RECENT_CARD_RESPONSE_PROP',
       'RECENT_AVG_CARD_GIFT_AMT', 'RECENT_RESPONSE_COUNT',
       'RECENT_CARD_RESPONSE_COUNT', 'MONTHS_SINCE_LAST_PROM_RESP',
       'LIFETIME_CARD_PROM', 'LIFETIME_PROM', 'LIFETIME_GIFT_AMOUNT',
       'LIFETIME_GIFT_COUNT', 'LIFETIME_AVG_GIFT_AMT', 'LIFETIME_GIFT_RANGE',
       'LIFETIME_MAX_GIFT_AMT', 'LIFETIME_MIN_GIFT_AMT', 'LAST_GIFT_AMT',
       'CARD_PROM_12', 'NUMBER_PROM_12', 'MONTHS_SINCE_LAST_GIFT',
       'MONTHS_SINCE_FIRST_GIFT', 'FILE_AVG_GIFT', 'FILE_CARD_GIFT'])
```

In [81]:
```
print(scaled_robust_df.shape)
scaled_robust_df.tail()
```

(2041, 43)

Out[81]:

	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE	DONOR_G
2036	0.428571	-0.666667	1.0	1.0	-1.0	-0.16	
2037	-0.285714	-2.333333	1.0	0.0	-1.0	-1.08	
2038	0.285714	-0.222222	0.0	-1.5	3.0	1.04	
2039	0.000000	-0.944444	1.0	-1.5	3.0	1.04	
2040	0.857143	1.000000	1.0	1.0	-1.0	-0.16	

In [82]:
```
prospective_dataset_scaled_robust = CONTROL_NUMBER_df.join(scaled_robust_df, how='rig
ht')
```

In [83]:
```
prospective_dataset_scaled_robust
```

Out[83]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
0	139	0.428571	0.000000	0.0	-1.0	0.0	
1	142	0.857143	0.000000	0.0	-1.0	0.0	
2	282	-0.571429	-1.611111	0.0	0.5	-1.0	
3	368	0.857143	0.888889	0.0	0.0	-1.0	
4	387	-0.714286	0.000000	0.0	0.5	0.0	
...	...	...	...	...	...	...	
2036	190842	0.428571	-0.666667	1.0	1.0	-1.0	
2037	191056	-0.285714	-2.333333	1.0	0.0	-1.0	
2038	191164	0.285714	-0.222222	0.0	-1.5	3.0	
2039	191484	0.000000	-0.944444	1.0	-1.5	3.0	
2040	191710	0.857143	1.000000	1.0	1.0	-1.0	

2041 rows × 44 columns

In [84]:
```
prospective_dataset_scaled_robust.to_csv('Prospective Donor_ML with Python_PREPARED_S
CALED.csv', index=None)
```

[Table of Content](#)

10. Make Predictions

In [85]:
```
# A new prospective_dataset with dropped: 'CONTROL_NUMBER'

column_list_prospective_without_CONTROL_NUMBER = ['MONTHS_SINCE_ORIGIN',
       'DONOR_AGE', 'IN_HOUSE', 'URBANICITY', 'SES', 'CLUSTER_CODE',
       'DONOR_GENDER', 'PUBLISHED_PHONE',
       'MOR_HIT_RATE', 'MEDIAN_HOME_VALUE',
       'MEDIAN_HOUSEHOLD_INCOME', 'PCT_OWNER_OCCUPIED', 'PER_CAPITA_INCOME',
       'PCT_ATTRIBUTE1', 'PCT_ATTRIBUTE2', 'PCT_ATTRIBUTE3', 'PCT_ATTRIBUTE4',
       'PEP_STAR', 'RECENT_STAR_STATUS', 'RECENCY_STATUS_96NK',
       'FREQUENCY_STATUS_97NK', 'RECENT_RESPONSE_PROP', 'RECENT_AVG_GIFT_AMT',
       'RECENT_CARD_RESPONSE_PROP', 'RECENT_AVG_CARD_GIFT_AMT',
       'RECENT_RESPONSE_COUNT', 'RECENT_CARD_RESPONSE_COUNT',
       'MONTHS_SINCE_LAST_PROM_RESP', 'LIFETIME_CARD_PROM', 'LIFETIME_PROM',
       'LIFETIME_GIFT_AMOUNT', 'LIFETIME_GIFT_COUNT', 'LIFETIME_AVG_GIFT_AMT',
       'LIFETIME_GIFT_RANGE', 'LIFETIME_MAX_GIFT_AMT', 'LIFETIME_MIN_GIFT_AMT',
       'LAST_GIFT_AMT', 'CARD_PROM_12', 'NUMBER_PROM_12',
       'MONTHS_SINCE_LAST_GIFT', 'MONTHS_SINCE_FIRST_GIFT', 'FILE_AVG_GIFT',
       'FILE_CARD_GIFT']
```

In [86]:
```
# Loading the model
loaded_model_joblib = joblib.load('Deep_finalized_model_saved_with_Joblib.sav')
```

In [87]:
```
# Making predictions on New Data
Xnew = prospective_dataset_scaled_robust[column_list_prospective_without_CONTROL_NUMB
ER]
ynew = loaded_model_joblib.predict(Xnew)
```

In [88]:
```
print(len(ynew))
print(ynew)
```

```
2041
[0 0 1 ... 0 1 0]
```

In [89]:
```
print(type(Xnew),type(ynew))
```

```
<class 'pandas.core.frame.DataFrame'> <class 'numpy.ndarray'>
```

In [90]:
```
# Convert the outcome in DataFrame
ynew_df = pd.DataFrame(ynew)
```

In [91]:
```
len(ynew_df)
```

Out[91]:
```
2041
```

In [92]:
```
ynew_df.isna().sum()
```

Out[92]:
```
0    0
dtype: int64
```

In [93]:
```
print(type(Xnew),type(ynew_df))
```

```
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.frame.DataFrame'>
```

In [94]:
```
ynew_df.tail(100)
```
Out[94]:

	0
1941	0
1942	1
1943	0
1944	0
1945	0
...	...
2036	0
2037	1
2038	0
2039	1
2040	0

100 rows × 1 columns

In [95]:
```
ynew_df.columns = ['Prediction']
ynew_df.tail()
```
Out[95]:

	Prediction
2036	0
2037	1
2038	0
2039	1
2040	0

In [96]:
```
ynew_df.to_csv('Predictions.csv', index=None)
```
In [97]:
```
X_data = pd.read_csv('Prospective Donor_ML with Python_PREPARED_SCALED.csv')
Y_data = pd.read_csv('Predictions.csv')
```
In [98]:
```
X_data.tail()
```
Out[98]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
2036	190842	0.428571	-0.666667	1.0	1.0	-1.0	
2037	191056	-0.285714	-2.333333	1.0	0.0	-1.0	
2038	191164	0.285714	-0.222222	0.0	-1.5	3.0	
2039	191484	0.000000	-0.944444	1.0	-1.5	3.0	
2040	191710	0.857143	1.000000	1.0	1.0	-1.0	

In [99]:
```
Y_data.tail()
```
Out[99]:

	Prediction
2036	0
2037	1
2038	0
2039	1
2040	0

In [100]:
```
Z = X_data.join(Y_data, how='right')
```
In [101]:
```
Z
```
Out[101]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUST
0	139	0.428571	0.000000	0.0	-1.0	0.0	
1	142	0.857143	0.000000	0.0	-1.0	0.0	
2	282	-0.571429	-1.611111	0.0	0.5	-1.0	
3	368	0.857143	0.888889	0.0	0.0	-1.0	
4	387	-0.714286	0.000000	0.0	0.5	0.0	
...	...	...	...	...	...	...	
2036	190842	0.428571	-0.666667	1.0	1.0	-1.0	
2037	191056	-0.285714	-2.333333	1.0	0.0	-1.0	
2038	191164	0.285714	-0.222222	0.0	-1.5	3.0	
2039	191484	0.000000	-0.944444	1.0	-1.5	3.0	
2040	191710	0.857143	1.000000	1.0	1.0	-1.0	

2041 rows × 45 columns

In [102]:
```
Z.to_csv('Prospective Donor_ML with Python_PREPARED_SCALED_with_Deep_PREDICTIONS.csv'
, index=None)
```
[Table of Content](#)