Table of Contents

1 1. Loading the libraries

In [3]:
```
import pandas as pd
import numpy as np
from pandas import Series
import os
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
%matplotlib inline
```

2 2. Loading the dataset

In [4]:
```
churnActive = pd.read_csv('churn-active.csv')
```

In [5]:
```
churnActive.head()
```

Out[5]:

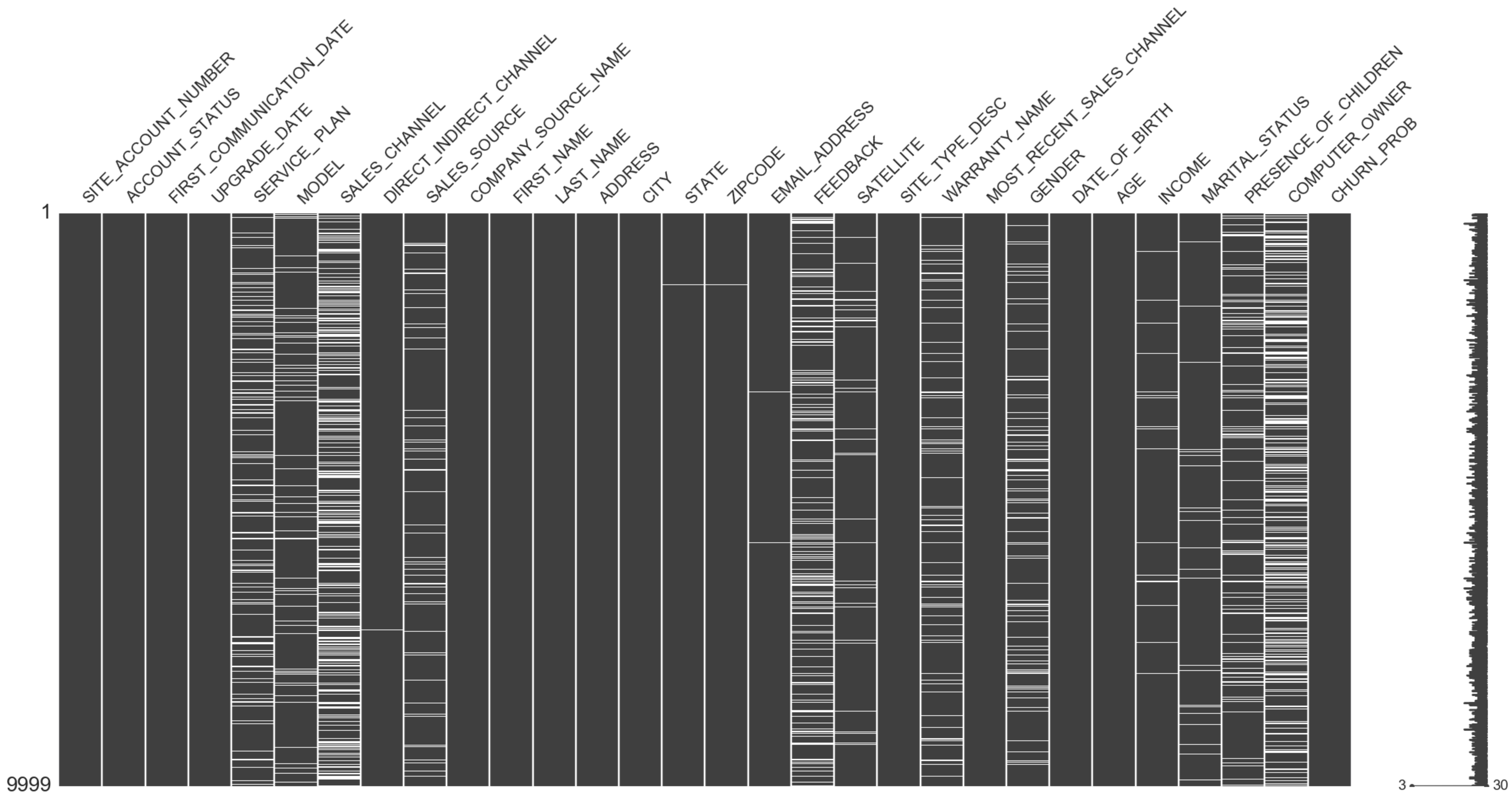
	SITE_ACCOUNT_NUMBER	ACCOUNT_STATUS	FIRST_COMMUNICATION_DATE	UPGRADE_DATE	SERVICE_PLAN	MODEL	SALES_CHANNEL	DIRECT_INDIRECT_CHANNEL	SA
0	AMR-11112345	Active	11/27/2001	12/14/2003	NaN	HN9000	Vars	Indirect	UI
1	AMR-14837287	Active	5/2/2003	8/8/2006	Pro	HN7000S	Sales Agents	Indirect	N(
2	AMR-14837803	Active	1/28/2001	3/17/2004	Power150	HN9000	Call Center	Indirect	N(
3	AMR-14837821	Active	11/20/2001	3/27/2005	Power150	DW6000	Sales Agents	Direct	Na
4	AMR-14839297	Active	2/27/2002	6/14/2004	Home	HN7000S	Call Center	Indirect	TA

5 rows × 29 columns

In [6]:
```
churnActive['CHURN_PROB'] = 0
```

3 3. Visualizing Missing Data

```
In [7]: msno.matrix(churnActive)
```



4 4. Data Preprocessing

```
In [8]: churnActive.ACCOUNT_STATUS.replace(["Inactive", "Active"], 1, inplace=True)
        churnActive.ACCOUNT_STATUS.replace("nan", 1, inplace=True)
```

```
In [9]: churnActive.ACCOUNT_STATUS = churnActive.ACCOUNT_STATUS.astype(int)
```

```
In [10]: churnActive.ACCOUNT_STATUS.unique()
```

```
Out[10]: array([1], dtype=int64)
```

```
In [11]: churnActive.columns.values[4] = 'SERVICE_PLAN'
```

In [12]:
```
churnActive.columns
```

Out[12]:
```
Index([u'SITE_ACCOUNT_NUMBER', u'ACCOUNT_STATUS', u'FIRST_COMMUNICATION_DATE',
       u'UPGRADE_DATE', u'SERVICE_PLAN', u'MODEL', u'SALES_CHANNEL',
       u'DIRECT_INDIRECT_CHANNEL', u'SALES_SOURCE', u'COMPANY_SOURCE_NAME',
       u'FIRST_NAME', u'LAST_NAME', u'ADDRESS', u'CITY', u'STATE', u'ZIPCODE',
       u'EMAIL_ADDRESS', u'FEEDBACK', u'SATELLITE', u'SITE_TYPE_DESC',
       u'WARRANTY_NAME', u'MOST_RECENT_SALES_CHANNEL', u'GENDER',
       u'DATE_OF_BIRTH', u'AGE', u'INCOME', u'MARITAL_STATUS',
       u'PRESENCE_OF_CHILDREN', u'COMPUTER_OWNER', u'CHURN_PROB'],
      dtype='object')
```

In [13]:
```
churnActive = churnActive.drop(['SITE_ACCOUNT_NUMBER', 'FIRST_COMMUNICATION_DATE', 'UPGRADE_DATE'], axis=1)
```

In [14]:
```
churnActive.head()
```

Out[14]:

	ACCOUNT_STATUS	SERVICE_PLAN	MODEL	SALES_CHANNEL	DIRECT_INDIRECT_CHANNEL	SALES_SOURCE	COMPANY_SOURCE_NAME	FIRST_NAME	LAST_NAME
0	1	NaN	HN9000	Vars	Indirect	UNKNOWN	HNS Customers	Ace	FUTRELL
1	1	NaN	HN7000S	Sales Agents	Indirect	NCC	HNS Customers	Robert	Fitzpatrick
2	1	NaN	HN9000	Call Center	Indirect	NCC	HNS Customers	EDWARD	Vipperman
3	1	NaN	DW6000	Sales Agents	Direct	NaN	HNS Customers	Dannya	Jyotinagaram
4	1	NaN	HN7000S	Call Center	Indirect	TAG	HNS Customers	Joannea	BARNOSKY

5 rows × 27 columns

In [15]:
```
churnClosed = pd.read_csv('churn-closed.csv')
```

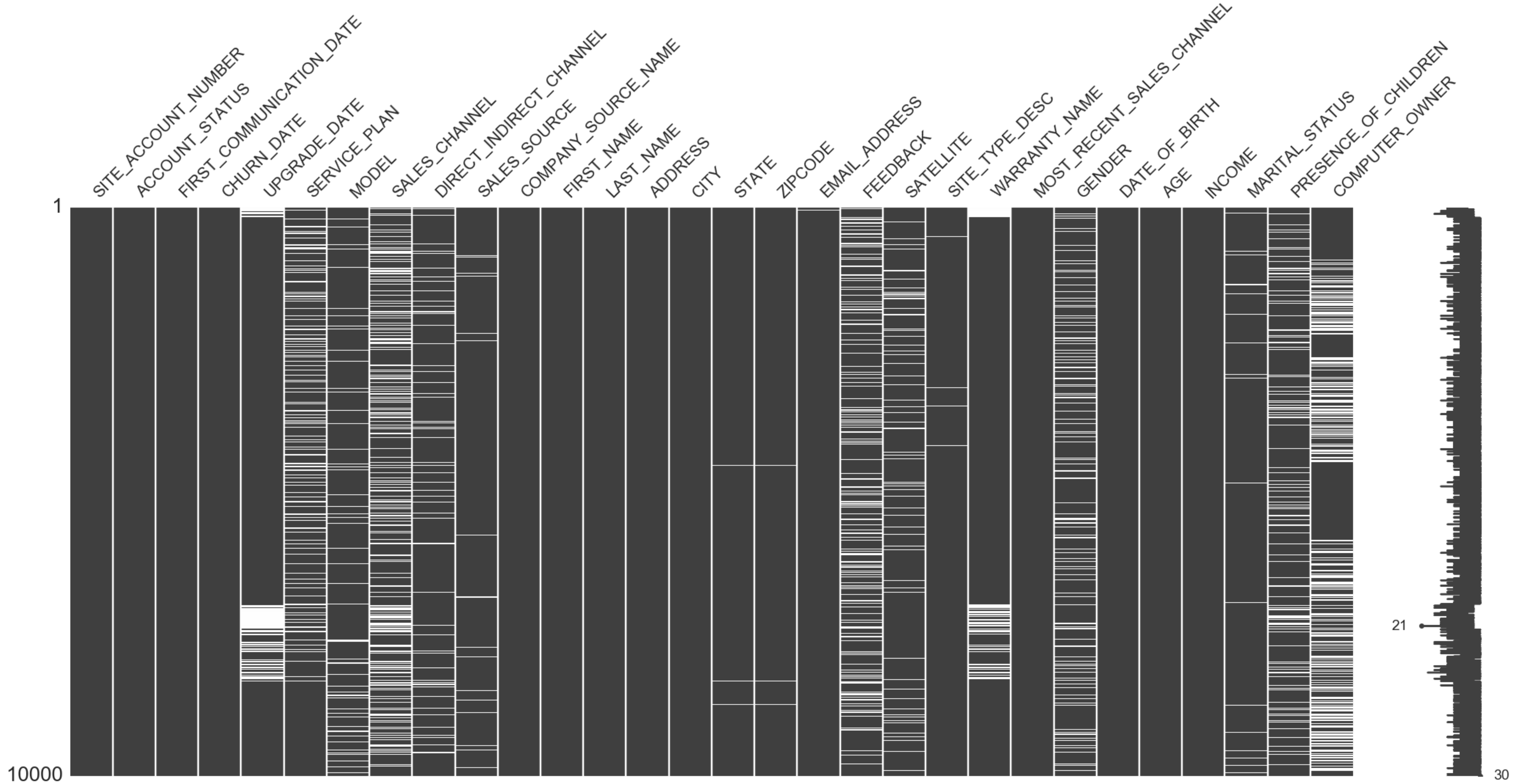
In [16]:
```
churnClosed.head()
```

Out[16]:

	SITE_ACCOUNT_NUMBER	ACCOUNT_STATUS	FIRST_COMMUNICATION_DATE	CHURN_DATE	UPGRADE_DATE	SERVICE_PLAN	MODEL	SALES_CHANNEL	DIRECT_INDI
0	AMR-53205840	Closed	1/18/2002	4/2/2006	1/3/2004	Professional	DW6000	NaN	Indirect
1	AMR-53383136	Closed	1/2/2002	7/9/2006	7/28/2004	SO	DW6000	Retail/Others	Indirect
2	AMR-53608585	Closed	1/6/2002	12/30/2008	4/27/2005	NaN	DW6000	Retail/Others	Indirect
3	AMR-53610590	Closed	1/5/2002	8/19/2004	NaN	SO	DW4000	NaN	Indirect
4	AMR-53612154	Closed	1/5/2002	8/17/2006	11/12/2005	Professional	DW7000	Retail/Others	Indirect

5 rows × 30 columns

In [17]:
```
msno.matrix(churnClosed)
```



In [18]:
```
churnClosed.ACCOUNT_STATUS.unique()
```

Out[18]:
```
array(['Closed', '#REF!'], dtype=object)
```

In [19]:
```
churnClosed.ACCOUNT_STATUS.replace("Closed", 0, inplace=True)
churnClosed.ACCOUNT_STATUS.replace("#REF!", 0, inplace=True)
```

In [20]:
```
churnClosed.ACCOUNT_STATUS = churnClosed.ACCOUNT_STATUS.astype(int)
```

In [21]:
```
churnClosed.ACCOUNT_STATUS.unique()
```

Out[21]:
```
array([0], dtype=int64)
```

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

Out[22]:
```
Index([u'SITE_ACCOUNT_NUMBER', u'ACCOUNT_STATUS', u'FIRST_COMMUNICATION_DATE',
       u'CHURN_DATE', u'UPGRADE_DATE', u'SERVICE_PLAN', u'MODEL',
       u'SALES_CHANNEL', u'DIRECT_INDIRECT_CHANNEL', u'SALES_SOURCE',
       u'COMPANY_SOURCE_NAME', u'FIRST_NAME', u'LAST_NAME', u'ADDRESS',
       u'CITY', u'STATE', u'ZIPCODE', u'EMAIL_ADDRESS', u'FEEDBACK',
       u'SATELLITE', u'SITE_TYPE_DESC', u'WARRANTY_NAME',
       u'MOST_RECENT_SALES_CHANNEL', u'GENDER', u'DATE_OF_BIRTH', u'AGE',
       u'INCOME', u'MARITAL_STATUS', u'PRESENCE_OF_CHILDREN',
       u'COMPUTER_OWNER'],
      dtype='object')
```

In [23]:
```
churnClosed.CHURN_DATE = pd.to_datetime(churnClosed.CHURN_DATE)
```

In [24]:
```
churnClosed.CHURN_DATE = churnClosed.CHURN_DATE.fillna(0)
```

In [25]:
```
churnClosed.CHURN_DATE = churnClosed.CHURN_DATE.dt.to_period('M')
```

In [26]:
```
churnClosed.FIRST_COMMUNICATION_DATE = pd.to_datetime(churnClosed.FIRST_COMMUNICATION_DATE)
```

In [27]:
```
churnClosed.FIRST_COMMUNICATION_DATE = churnClosed.FIRST_COMMUNICATION_DATE.fillna(0)
```

In [28]:
```
churnClosed.FIRST_COMMUNICATION_DATE = churnClosed.FIRST_COMMUNICATION_DATE.dt.to_period('M')
```

In [29]:
```
churnClosed['DELTA'] = churnClosed.CHURN_DATE - churnClosed.FIRST_COMMUNICATION_DATE
```

In [30]:
```
churnClosed['DELTA'] = churnClosed.DELTA.astype(int)
```

In [31]:
```
churnClosed['DELTA'].head()
```

Out[31]:
```
0    51
1    54
2    83
3    31
4    55
Name: DELTA, dtype: int32
```

In [32]:
```
churnClosed['CHURN_PROB'] = 1/churnClosed['DELTA']
```

In [33]:
```
churnClosed['CHURN_PROB'] = churnClosed.CHURN_PROB.replace('inf', 1)
```

In [34]:
```
churnClosed.head()
```

Out[34]:

	SITE_ACCOUNT_NUMBER	ACCOUNT_STATUS	FIRST_COMMUNICATION_DATE	CHURN_DATE	UPGRADE_DATE	SERVICE_PLAN	MODEL	SALES_CHANNEL	DIRECT_INDI
0	AMR-53205840	0	2002-01	2006-04	1/3/2004	Professional	DW6000	NaN	Indirect
1	AMR-53383136	0	2002-01	2006-07	7/28/2004	SO	DW6000	Retail/Others	Indirect
2	AMR-53608585	0	2002-01	2008-12	4/27/2005	NaN	DW6000	Retail/Others	Indirect
3	AMR-53610590	0	2002-01	2004-08	NaN	SO	DW4000	NaN	Indirect
4	AMR-53612154	0	2002-01	2006-08	11/12/2005	Professional	DW7000	Retail/Others	Indirect

5 rows × 32 columns

In [35]:
```
churnClosed = churnClosed.drop(['SITE_ACCOUNT_NUMBER', 'FIRST_COMMUNICATION_DATE', 'CHURN_DATE', 'UPGRADE_DATE', 'DELTA'], axis=1)
```

In [36]:
```
churnClosed.head()
```

Out[36]:

	ACCOUNT_STATUS	SERVICE_PLAN	MODEL	SALES_CHANNEL	DIRECT_INDIRECT_CHANNEL	SALES_SOURCE	COMPANY_SOURCE_NAME	FIRST_NAME	LAST_NAME
0	0	Professional	DW6000	NaN	Indirect	Perfect 10	HNS Customers	Rennisa	Branson
1	0	SO	DW6000	Retail/Others	Indirect	ValueElectronics	HNS Customers	Howard	Dachs
2	0	NaN	DW6000	Retail/Others	Indirect	TAG	HNS Customers	Pavel	Groisman
3	0	SO	DW4000	NaN	Indirect	TAG	HNS Customers	Stoneysmita	Stoneysmith
4	0	Professional	DW7000	Retail/Others	Indirect	Perfect 10	HNS Customers	Bryana	Emilio

5 rows × 27 columns

In [37]:
```
churn = pd.concat([churnActive, churnClosed])
```

In [38]:
```
churn = churn.drop("DATE_OF_BIRTH", axis=1)
```

In [39]:
```
churn.head()
```

Out[39]:

	ACCOUNT_STATUS	SERVICE_PLAN	MODEL	SALES_CHANNEL	DIRECT_INDIRECT_CHANNEL	SALES_SOURCE	COMPANY_SOURCE_NAME	FIRST_NAME	LAST_NAME
0	1	NaN	HN9000	Vars	Indirect	UNKNOWN	HNS Customers	Ace	FUTRELL
1	1	NaN	HN7000S	Sales Agents	Indirect	NCC	HNS Customers	Robert	Fitzpatrick
2	1	NaN	HN9000	Call Center	Indirect	NCC	HNS Customers	EDWARD	Vipperman
3	1	NaN	DW6000	Sales Agents	Direct	NaN	HNS Customers	Dannya	Jyotinagaram
4	1	NaN	HN7000S	Call Center	Indirect	TAG	HNS Customers	Joannea	BARNOSKY

5 rows × 26 columns

In [40]:
```
churn.INCOME = churn.INCOME.replace(to_replace= '#REF!', value=15)
churn.INCOME = churn.INCOME.replace(to_replace= 'A', value=11)
churn.INCOME = churn.INCOME.replace(to_replace= 'B', value=12)
churn.INCOME = churn.INCOME.replace(to_replace= 'C', value=13)
churn.INCOME = churn.INCOME.replace(to_replace= 'D', value=14)
churn.INCOME = churn.INCOME.fillna(15)
churn.INCOME = churn.INCOME.astype(dtype=int)
```

In [41]:
```
churn['AGE'] = churn.AGE.fillna(0)
churn.AGE = churn.AGE.replace(to_replace= '#REF!', value=0)
churn.AGE = churn.AGE.replace(to_replace= '.', value=0)
churn.AGE = churn.AGE.astype(dtype=int)
```

In [42]:
```
churn.MARITAL_STATUS.unique()
```

Out[42]:
```
array(['1', '0', '2', nan, '#REF!', '.'], dtype=object)
```

In [43]:
```
churn['MARITAL_STATUS'] = churn.MARITAL_STATUS.fillna(3)
churn.MARITAL_STATUS = churn.MARITAL_STATUS.replace(to_replace= '#REF!', value=3)
churn.MARITAL_STATUS = churn.MARITAL_STATUS.replace(to_replace= '.', value=3)
churn.MARITAL_STATUS = churn.MARITAL_STATUS.astype(dtype=int)
```

In [44]:
```
churn.PRESENCE_OF_CHILDREN.unique()
```

Out[44]:
```
array(['0', '1', nan, '#REF!', 0.0, 1.0], dtype=object)
```

In [45]:
```
churn['PRESENCE_OF_CHILDREN'] = churn.PRESENCE_OF_CHILDREN.fillna(2)
churn.PRESENCE_OF_CHILDREN = churn.PRESENCE_OF_CHILDREN.replace(to_replace= '#REF!', value=2)
churn.PRESENCE_OF_CHILDREN = churn.PRESENCE_OF_CHILDREN.replace(to_replace= '.', value=2)
churn.PRESENCE_OF_CHILDREN = churn.PRESENCE_OF_CHILDREN.astype(dtype=int)
```

In [46]:
```
churn.COMPUTER_OWNER.unique()
```
Out[46]:
```
array(['N', nan, 'Y'], dtype=object)
```

In [47]:
```
churn['COMPUTER_OWNER'] = churn.COMPUTER_OWNER.fillna(2)
churn.COMPUTER_OWNER = churn.COMPUTER_OWNER.replace(to_replace= 'Y', value=1)
churn.COMPUTER_OWNER = churn.COMPUTER_OWNER.replace(to_replace= 'N', value=0)
churn.COMPUTER_OWNER = churn.COMPUTER_OWNER.astype(dtype=int)
```

In [48]:
```
churn_objects = churn.select_dtypes(include=[object])
```

In [49]:
```
churn_objects.head()
```
Out[49]:

	SERVICE_PLAN	MODEL	SALES_CHANNEL	DIRECT_INDIRECT_CHANNEL	SALES_SOURCE	COMPANY_SOURCE_NAME	FIRST_NAME	LAST_NAME	ADDRESS	CITY
0	NaN	HN9000	Vars	Indirect	UNKNOWN	HNS Customers	Ace	FUTRELL	TRIPLE CANTHOOK LN 0 0 0	GERMA
1	NaN	HN7000S	Sales Agents	Indirect	NCC	HNS Customers	Robert	Fitzpatrick	RR JOE 62 0 0 0	WEST B
2	NaN	HN9000	Call Center	Indirect	NCC	HNS Customers	EDWARD	Vipperman	69330 1 RD WAY 0 0	MERRY
3	NaN	DW6000	Sales Agents	Direct	NaN	HNS Customers	Dannya	Jyotinagaram	PO ROCK ST 421 0 0	ZANES
4	NaN	HN7000S	Call Center	Indirect	TAG	HNS Customers	Joannea	BARNOSKY	626 NW RD 0 0 0	CALLIC

In [50]:
```
churn_objects.columns
```
Out[50]:
```
Index([u'SERVICE_PLAN', u'MODEL', u'SALES_CHANNEL', u'DIRECT_INDIRECT_CHANNEL',
       u'SALES_SOURCE', u'COMPANY_SOURCE_NAME', u'FIRST_NAME', u'LAST_NAME',
       u'ADDRESS', u'CITY', u'STATE', u'ZIPCODE', u'EMAIL_ADDRESS',
       u'FEEDBACK', u'SATELLITE', u'SITE_TYPE_DESC', u'WARRANTY_NAME',
       u'MOST_RECENT_SALES_CHANNEL'],
      dtype='object')
```

In [51]:
```
churn_objects = churn_objects.fillna('Not Available')
churn_objects = churn_objects.replace(to_replace= '#REF!', value='Not Available')
churn_objects = churn_objects.replace(to_replace= '.', value='Not Available')
```

In [52]:
```
churn_numerics = churn.select_dtypes(exclude=[object])
```

In [53]:
```
churn_numerics.head()
```

Out[53]:

	ACCOUNT_STATUS	GENDER	AGE	INCOME	MARITAL_STATUS	PRESENCE_OF_CHILDREN	COMPUTER_OWNER	CHURN_PROB
0	1	2.0	69	0	1	0	0	0.0
1	1	2.0	43	4	1	0	0	0.0
2	1	2.0	58	4	1	0	0	0.0
3	1	NaN	22	6	1	1	2	0.0
4	1	1.0	36	7	0	0	2	0.0

In [54]:
```
churn_numerics.columns
```

Out[54]:
```
Index([u'ACCOUNT_STATUS', u'GENDER', u'AGE', u'INCOME', u'MARITAL_STATUS',
       u'PRESENCE_OF_CHILDREN', u'COMPUTER_OWNER', u'CHURN_PROB'],
      dtype='object')
```

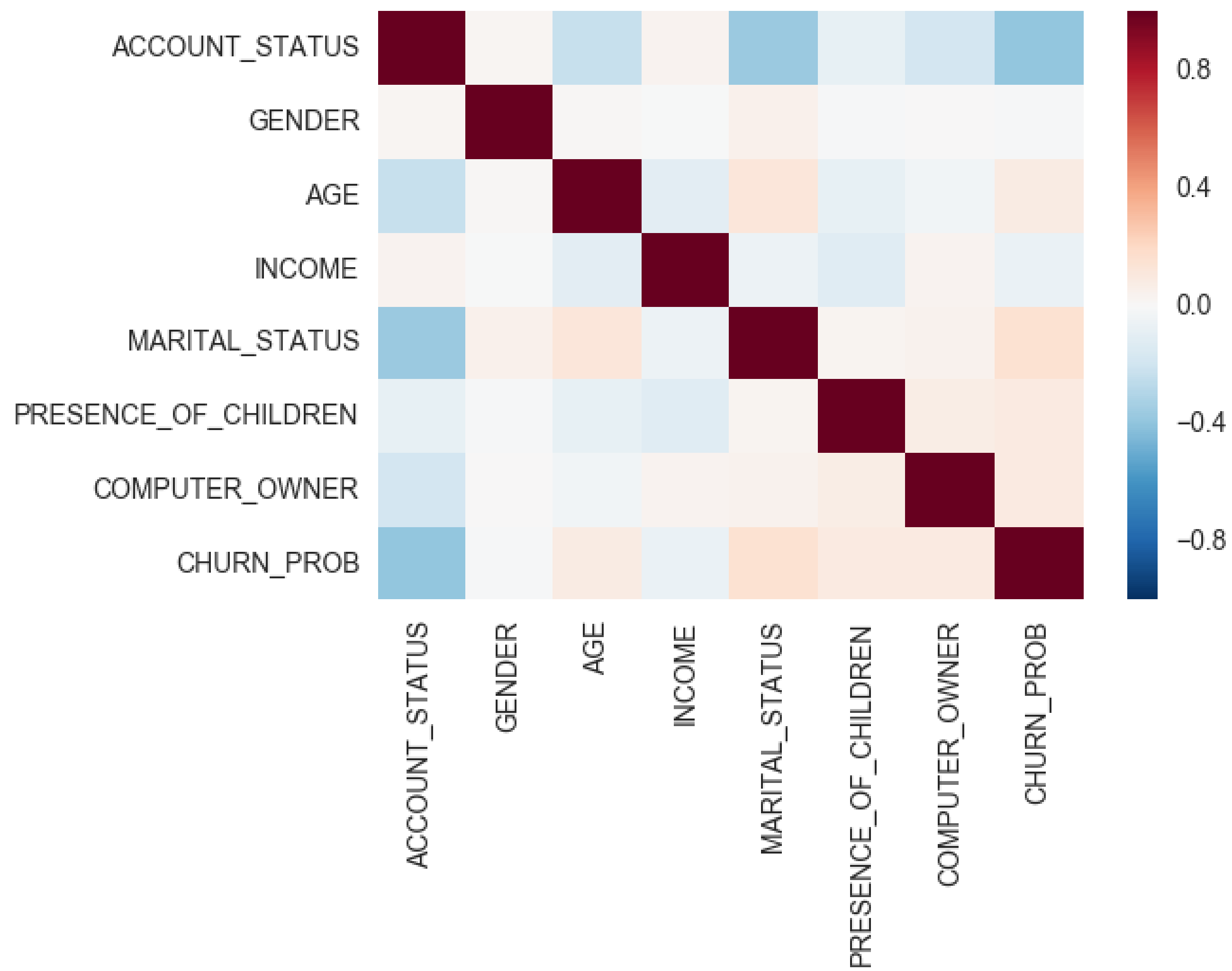
In [55]:
```
churn_numerics = churn_numerics.fillna(churn_numerics.median())
```

In [56]:
```
churn_numerics.GENDER = churn_numerics.GENDER.astype(int)
```

5 5. Correlation Matrix and Label Encoding

In [57]:
```
corr = churn_numerics.corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
```

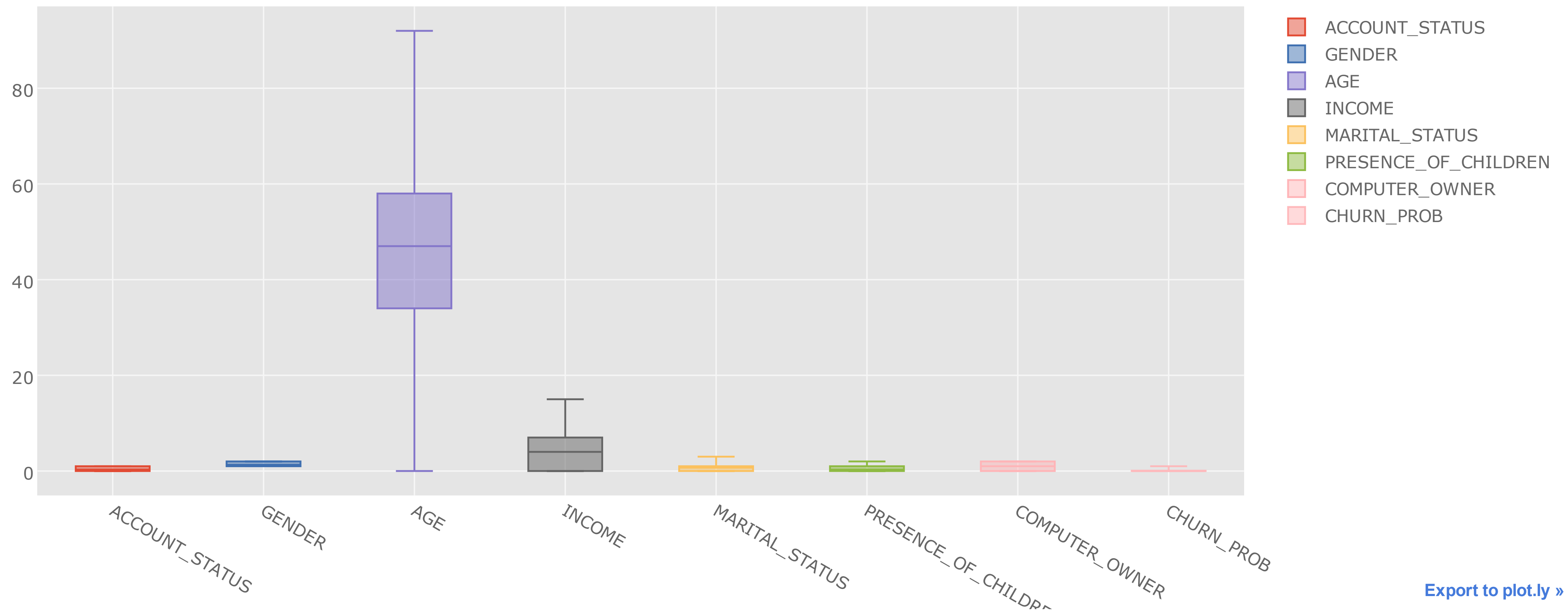
Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0xc7307b8>



In [58]:
```
import plotly.plotly as py
import cufflinks as cf
```

In [59]:
```
cf.set_config_file(offline=True, world_readable=True, theme='ggplot')
```

In [60]:
```
churn_numerics.iplot(kind='box', filename='cufflinks/box-plots')
```



In [61]:
```
from sklearn.preprocessing import LabelEncoder
```

In [62]:
```
churn_objects = churn_objects.apply(LabelEncoder().fit_transform)
```

In [63]:
```
churn_new = pd.concat([churn_numerics, churn_objects], axis=1)
```

In [64]:
```
churn_new.head()
```

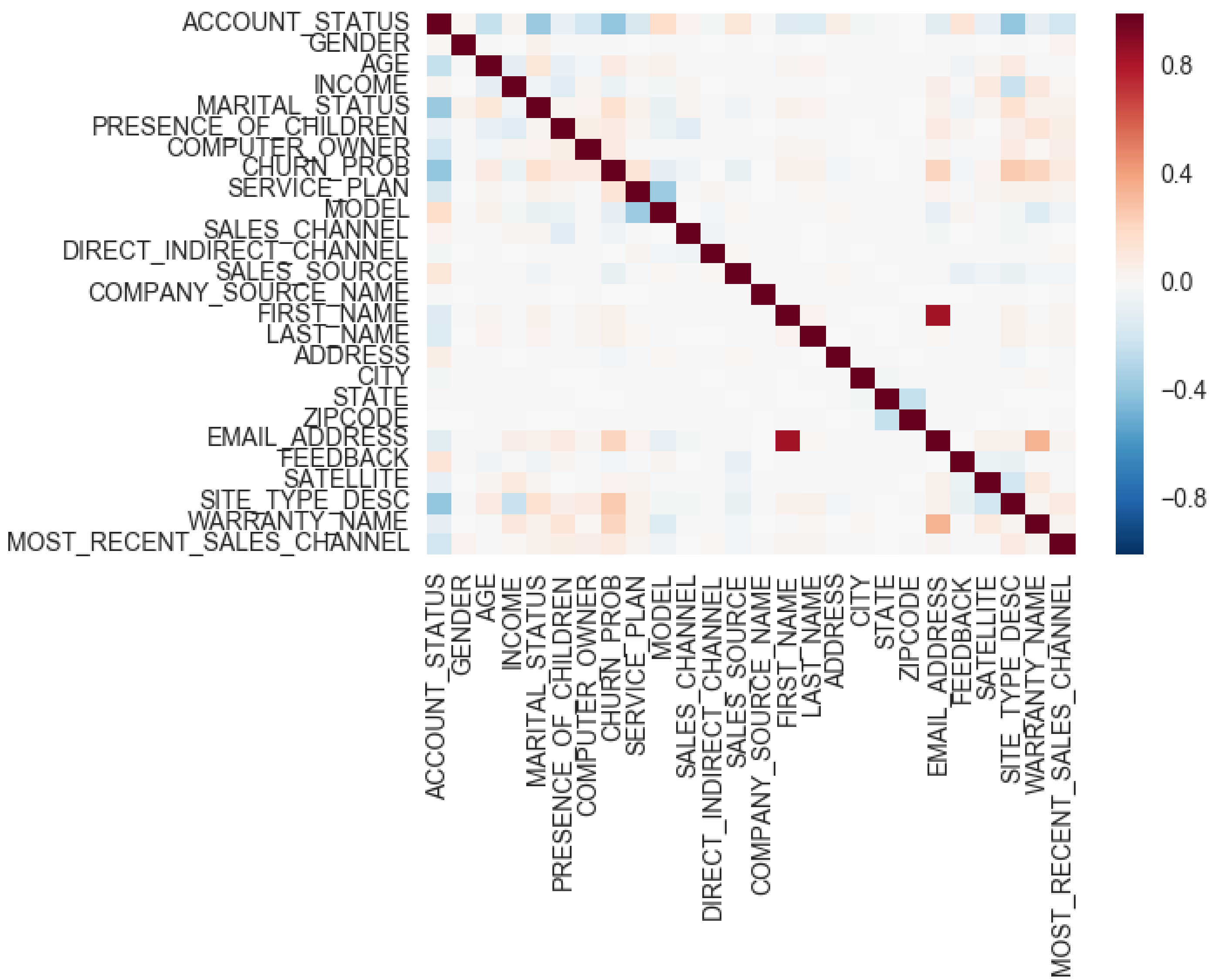
Out[64]:

	ACCOUNT_STATUS	GENDER	AGE	INCOME	MARITAL_STATUS	PRESENCE_OF_CHILDREN	COMPUTER_OWNER	CHURN_PROB	SERVICE_PLAN	MODEL	...	ADDRESS	C
0	1	2	69	0	1	0	0	0.0	4	5	...	10846	2
1	1	2	43	4	1	0	0	0.0	4	4	...	10747	7
2	1	2	58	4	1	0	0	0.0	4	5	...	8034	4
3	1	1	22	6	1	1	2	0.0	4	2	...	10481	8
4	1	1	36	7	0	0	2	0.0	4	4	...	7687	1

5 rows × 26 columns

In [65]:
```
corr2 = churn_new.corr()
sns.heatmap(corr2,
            xticklabels=corr2.columns.values,
            yticklabels=corr2.columns.values)
```

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x135590b8>



In [66]:
```
y = churn_new.ACCOUNT_STATUS
```

In [67]:
```
X = churn_new.ix[:, churn_new.columns != 'ACCOUNT_STATUS']
```

In [68]:
```
X_colnames = X.columns.tolist()
```

In [69]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [70]:
```
X = pd.DataFrame(X, columns=X_colnames)
```

In [71]:
```
X.drop('CHURN_PROB', axis=1, inplace=True)
```

6 6. Principal Component Analysis

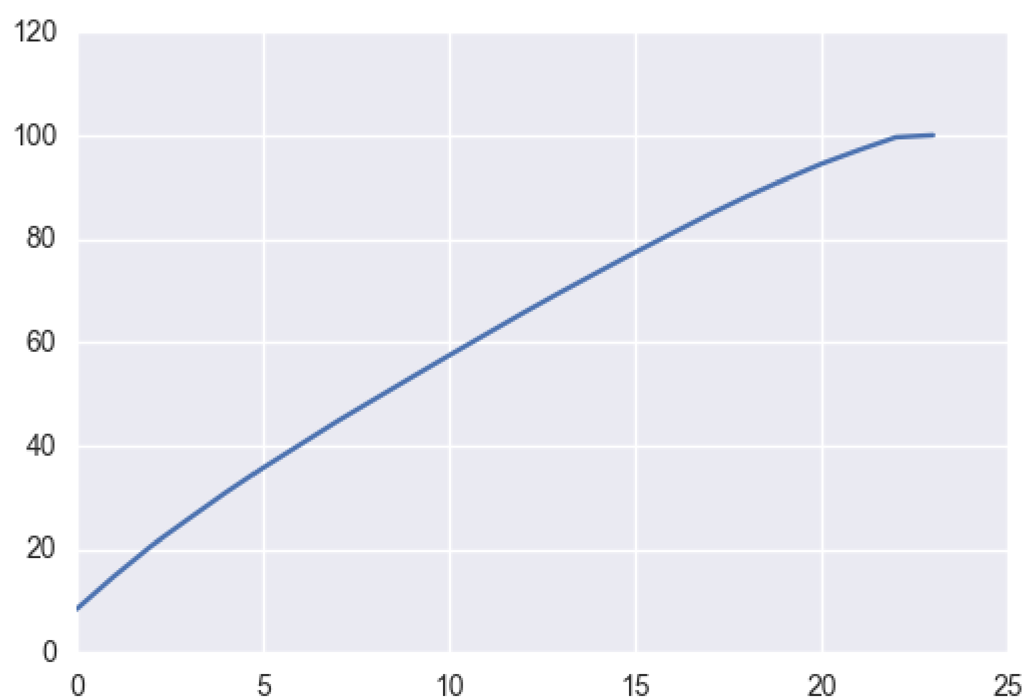
In [72]:
```
from sklearn.decomposition import PCA
pca_2 = PCA(n_components=24)
pca_2.fit(X)
```

Out[72]:
```
PCA(copy=True, iterated_power='auto', n_components=24, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

In [73]:
```
var1=np.cumsum(np.round(pca_2.explained_variance_ratio_, decimals=4)*100)
print var1
plt.plot(var1)
```

```
[   8.4    14.71   20.58   25.74   30.84   35.66   40.19   44.71   48.98
   53.23   57.44   61.58   65.71   69.7    73.57   77.4    81.12   84.78
   88.23   91.43   94.48   97.13   99.62  100.02]
```

Out[73]:
```
[<matplotlib.lines.Line2D at 0xca753c8>]
```



In [74]:
```
pca = PCA(n_components=19)
pca.fit(X)
X_reduced = pca.transform(X)
```

In [75]:
```
X_reduced = pd.DataFrame(X_reduced)
```

7 7. Building Models for Class Prediction

In [76]:
```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, random_state=1)
```

```
C:\Program Files\Anaconda2\lib\site-packages\sklearn\cross_validation.py:44: DeprecationWarning:

This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note
 that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
```

7.1 a. Logistic Regression

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

Out[77]:
```
RangeIndex(start=0, stop=19, step=1)
```

In [78]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

clf = LogisticRegression()
clf.fit(X_train, y_train)
y_train_pred = clf.predict(X_train)
```

In [79]:
```
accuracy_logistic_train = round(accuracy_score(y_train, y_train_pred), 3)
```

In [80]:
```
confusion_matrix(y_train, y_train_pred)
```

Out[80]:
```
array([[5778, 1772],
       [1557, 5892]])
```

In [81]:
```
y_test_pred = clf.predict(X_test)
accuracy_logistic_test = round(accuracy_score(y_test, y_test_pred), 3)
```

In [82]:
```
confusion_matrix(y_test, y_test_pred)
```

Out[82]:
```
array([[1885,  565],
       [ 535, 2015]])
```

7.2 b. Decision Tree

In [83]:
```
from sklearn import tree
```

In [84]:
```
dt = tree.DecisionTreeClassifier()
```

In [85]:
```
dt.fit(X_train, y_train)
```

Out[85]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

In [86]:
```
yhat_dt_train = dt.predict(X_train)
```

In [87]:
```
accuracy_dt_train = round(accuracy_score(y_train, yhat_dt_train), 3)
```

In [88]:
```
yhat_dt_test = dt.predict(X_test)
```

In [89]:
```
accuracy_dt_test = round(accuracy_score(y_test, yhat_dt_test), 3)
```

7.3 c. SVM

In [90]:
```
from sklearn.svm import SVC
```

In [91]:
```
svc = SVC()
```

In [92]:
```
svc.fit(X_train, y_train)
```

Out[92]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [93]:
```
yhat_svc_train = svc.predict(X_train)
```

In [94]:
```
accuracy_svc_train = round(accuracy_score(y_train, yhat_svc_train), 3)
```

In [95]:
```
yhat_svc_test = svc.predict(X_test)
```

In [96]:
```
accuracy_svc_test = round(accuracy_score(y_test, yhat_svc_test), 3)
```

7.4 d. Random Forest

In [97]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [98]:
```
rf = RandomForestClassifier()
```

In [99]:
```
rf.fit(X_train, y_train)
```

Out[99]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
```

In [100]:
```
yhat_rf_train = rf.predict(X_train)
```

In [101]:
```
accuracy_rf_train = round(accuracy_score(y_train, yhat_rf_train), 3)
```

In [102]:
```
yhat_rf_test = rf.predict(X_test)
```

In [103]:
```
accuracy_rf_test = round(accuracy_score(y_test, yhat_rf_test), 3)
```

8 8. Building Models for Probability Prediction

8.1 a. Linear Regression

In [104]:
```
y_lin = churn_new.CHURN_PROB
```

In [105]:
```
X_lin = churn_new.ix[:, churn_new.columns != 'CHURN_PROB']
```

In [106]:
```
X_lin_colnames = X_lin.columns.tolist()
```

In [107]:
```
X_lin = scaler.fit_transform(X_lin)
```

In [108]:
```
X_lin = pd.DataFrame(X_lin, columns=X_lin_colnames)
```

In [109]:
```
X_lin.drop('ACCOUNT_STATUS', axis=1, inplace=True)
```

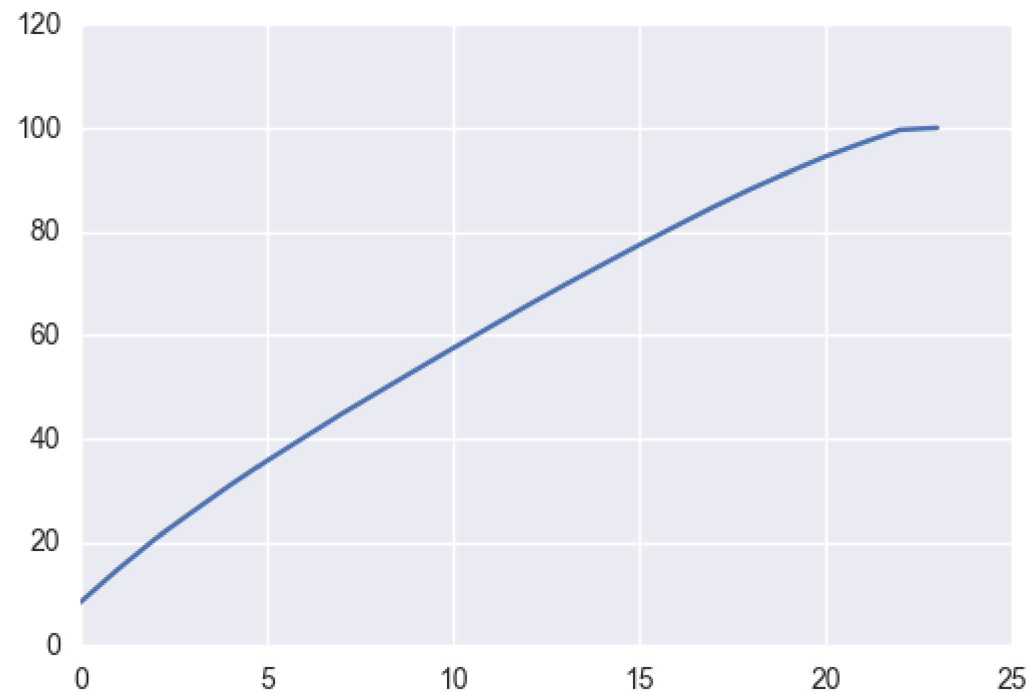
In [110]:
```
pca_3 = PCA(n_components=24)
pca_3.fit(X_lin)
```

Out[110]:
```
PCA(copy=True, iterated_power='auto', n_components=24, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

In [111]:
```
var1=np.cumsum(np.round(pca_3.explained_variance_ratio_, decimals=4)*100)
print var1
plt.plot(var1)
```

```
[   8.4    14.71   20.58   25.74   30.84   35.66   40.19   44.71   48.98
   53.23   57.44   61.58   65.71   69.7    73.57   77.4    81.12   84.78
   88.23   91.43   94.48   97.13   99.62  100.02]
```

Out[111]: [<matplotlib.lines.Line2D at 0xe8cf358>]



In [112]:
```
pca = PCA(n_components=19)
pca.fit(X_lin)
X_lin_reduced = pca.transform(X_lin)
```

In [113]:
```
X_lin_reduced = pd.DataFrame(X_lin_reduced)
```

In [114]:
```
from sklearn.cross_validation import train_test_split
X_lin_train, X_lin_test, y_lin_train, y_lin_test = train_test_split(X_lin_reduced, y_lin, random_state=1)
```

In [115]:
```
from sklearn import linear_model
from sklearn.cross_validation import cross_val_predict
```

In [116]:
```
lr = linear_model.LinearRegression()
LR = lr.fit(X_lin_train, y_lin_train)
```

In [117]:
```
yhat_lin_train = LR.predict(X_lin_train)
```

In [118]:
```
from sklearn import metrics
```

In [119]:
```
rmse_linear_train = np.sqrt(metrics.mean_squared_error(y_lin_train, yhat_lin_train))
```

In [120]:
```
yhat_lin_test = LR.predict(X_lin_test)
```

In [121]:
```
rmse_linear_test = np.sqrt(metrics.mean_squared_error(y_lin_test, yhat_lin_test))
```

8.2 b. SVR

In [122]:
```
from sklearn.svm import SVR
```

In [123]:
```
svr = SVR(kernel='linear')
```

In [124]:
```
svr.fit(X_lin_train, y_lin_train)
```

Out[124]:
```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
  kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

In [125]:
```
yhat_svr_train = svr.predict(X_lin_train)
```

In [126]:
```
rmse_svr_train = np.sqrt(metrics.mean_squared_error(y_lin_train, yhat_svr_train))
```

In [127]:
```
yhat_svr_test = LR.predict(X_lin_test)
```

In [128]:
```
rmse_svr_test = np.sqrt(metrics.mean_squared_error(y_lin_test, yhat_svr_test))
```

9 9. Model Selection

In [131]:
```
results_classification = {'Accuracy': [accuracy_logistic_train, accuracy_logistic_test, accuracy_dt_train, accuracy_dt_test, accuracy_svc_train, accuracy_svc
index = ['Logistic_Train', 'Logistic_Test', 'DT_Train', 'DT_Test', 'SVC_Train', 'SVC_Test', 'RF_Train', 'RF_Test']
results = pd.DataFrame(data=results_classification, index=index)
results
```

Out[131]:

	Accuracy
Logistic_Train	0.778
Logistic_Test	0.780
DT_Train	1.000
DT_Test	0.779
SVC_Train	0.932
SVC_Test	0.902
RF_Train	0.993
RF_Test	0.839

From the above table we see that the Support Vector Classification (SVC) Algorithm has the highest accuracy score as compared to the others. Hence, we choose SVC as the model of our choice to classify if a particular customer will churn or not.

In [132]:
```
results_regression = {'RMSE': [rmse_linear_train, rmse_linear_test, rmse_svr_train, rmse_svr_test]}
index2 = ['Linear_Train', 'Linear_Test', 'SVR_Train', 'SVR_Test']
results2 = pd.DataFrame(data=results_regression, index=index2)
results2
```

Out[132]:

	RMSE
Linear_Train	0.114169
Linear_Test	0.114344
SVR_Train	0.115404
SVR_Test	0.114344

From the above table we see that the Linear Regression Algorithm has the lowest Root Mean Square Error (RMSE) as compared to SVR. Hence, we choose Linear Regression as the model of our choice to predict the churn probability of a particular customer.

To encourage customers who are likely to churn in the near future (probability > 8.3%, within 1 year), I would offer them a discount (on call or internet usage) which will be valid for a year so that they remain with us for the year.

To check whether the offers had any effect on the customers, I would see if they had accepted the offer or not and if they are still our customers at the end of the year.