Banking Churn Analysis

Author: Cennen Del Rosario
Date: September 20, 2020

In [626]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from lifelines import KaplanMeierFitter
from lifelines.statistics import (logrank_test,
                                  pairwise_logrank_test,
                                  multivariate_logrank_test,
                                  survival_difference_at_fixed_point_in_time_test)
import warnings
warnings.filterwarnings("ignore")
plt.style.use('seaborn')
```

1. Data Exploration

In [58]:
```
class_df = pd.read_csv('data/bankchurn_class.csv')
transaction_df = pd.read_csv('data/bankchurn_transactions.csv')
transaction_df['fulldatewithtime'] = pd.to_datetime(transaction_df['fulldatewithtime'])
class_df.iloc[10:15,:]
```

Out[58]:

	account_id	net_amount	amount_credited	amount_debited	n_transactions	churn_flag
10	A00000011	-6412,5	10927,3	17339,8	15	0
11	A00000012	3295,0	18034,8	14739,8	17	0
12	A00000013	-362,2	20613,6	20975,8	16	1
13	A00000014	4658,1	44788,9	40130,8	15	0
14	A00000015	2430,7	51490,5	49059,8	17	0

In [59]:
```
transaction_df.head()
```

Out[59]:

	account_id	transaction_type	amount	balance	fulldatewithtime
0	A00002505	Debit	2240.0	22026.9	2017-01-01 14:29:43
1	A00002498	Debit	600.0	17459.7	2017-01-01 12:51:08
2	A00002732	Debit	2400.0	103486.4	2017-01-01 09:00:25
3	A00002733	Debit	360,0	31725,6	2017-01-01 10:42:12
4	A00002740	Debit	2200.0	16872.0	2017-01-01 10:52:50

In [221]:
```
from datetime import datetime, timedelta

def adjust_datetime(dt):
    dt = dt.split('T')
    date_ = pd.to_datetime(dt[0])
    hr, min, sec = dt[1].split(':')

    if sec == '60':
        min = int(min) + 1
        min = str(min) if min >= 10 else ''.join(['0',str(min)])
        sec = '00'

    if min in ['60','61']:
       hr = int(hr) + 1
       if hr >= 24:
          date_ = date_ + timedelta(days=1)
          hr = '00'
       else:
          hr = str(hr) if hr >= 10 else ''.join(['0',str(hr)])
       min = str(int(min) - 60) if (int(min) - 60) >= 10 else ''.join(['0', str(int(min) - 60)])

    return str(date_.date()) + ' ' + ':'.join([hr, min, sec])
```

In [224]:
```
target_accounts = transaction_df['account_id'].sort_values().unique()

transaction_full_df = pd.read_csv('data/completedtrans.csv')
date_time = transaction_full_df['fulldatewithtime'].apply(lambda dt: adjust_datetime(dt))
transaction_full_df['fulldatewithtime'] = pd.to_datetime(date_time)
transaction_full_df = transaction_full_df.sort_values(by=['account_id', 'fulldatewithtime'])

accounts_condition = transaction_full_df['account_id'].isin(target_accounts)
start_time_condition = transaction_full_df['fulldate'] >= '2017-01-01'
end_time_condition = transaction_full_df['fulldate'] < '2018-01-01'

transaction_full_df = transaction_full_df[accounts_condition & start_time_condition & end_time_condition]
transaction_full_df = transaction_full_df.sort_values(by=['account_id','fulldate']).reset_index(drop=True)
```

In [227]:
```
transaction_full_df.head()
```

Out[227]:

	Unnamed: 0	trans_id	account_id	type	operation	amount	balance	k_symbol	bank	account	year	month	day	fulldate	fulltime	fulldatewithtime
0	453275	T00000074	A00000001	Debit	Remittance to Another Bank	2452.0	10207.9	Household	State Street Corp.	87144583.0	2017	1	5	2017-01-05	08:25:51	2017-01-05 08:25:51
1	462534	T00000220	A00000001	Credit	Credit in Cash	1200.0	11407.9	NaN	NaN	NaN	2017	1	12	2017-01-12	10:31:34	2017-01-12 10:31:34
2	463978	T00000117	A00000001	Debit	Cash Withdrawal	630,0	14456,9	NaN	NaN	NaN	2017	1	13	2017-01-13	13:43:31	2017-01-13 13:43:31
3	463979	T00000026	A00000001	Credit	Collection from Another Bank	3679.0	15086,9	NaN	JPMorgan Chase	41403269.0	2017	1	13	2017-01-13	16:26:43	2017-01-13 16:26:43
4	468985	T00000119	A00000001	Debit	Cash Withdrawal	390.0	14066.9	NaN	NaN	NaN	2017	1	19	2017-01-19	12:58:26	2017-01-19 12:58:26

In [358]:
```
def get_last_inactivity(acct_id):
    account_txn = transaction_full_df[transaction_full_df['account_id'] == acct_id].reset_index(drop=True)
    account_txn = account_txn.sort_values(by=['fulldatewithtime'])
    churn_flag = class_df[class_df['account_id']==acct_id]['churn_flag'].values[0]
    last_txn_index = 0

    # determine last transaction date
    delta_t = account_txn['fulldatewithtime'].diff().apply(lambda t: t.days)
    delta_t[0] = 0
    account_txn['time_diff'] = delta_t
    max_idle_time = account_txn['time_diff'].max()
    idle_txn = account_txn[account_txn['time_diff'] == max_idle_time]

    if churn_flag > 0:
        last_txn_index = idle_txn.index.max()
        if last_txn_index > 0:
            last_txn_index = last_txn_index - 1
    else:
        last_txn_index = len(account_txn)-1

    assigned_last_txn = account_txn.iloc[last_txn_index,]
    last_txn_time = assigned_last_txn['fulldatewithtime']

    # determine first transaction date
    first_txn_time = account_txn['fulldatewithtime'][0]

    # compute features
    account_txn = account_txn.iloc[0:last_txn_index+1,]
    amount_debited, amount_credited, n_transactions, net_amount = 0, 0, 0, 0

    credit_txns = account_txn[account_txn['type'] == 'Credit']
    debit_txns = account_txn[account_txn['type'] == 'Debit']

    amount_debited = debit_txns['amount'].sum()
    amount_credited = credit_txns['amount'].sum()
    n_transactions = len(account_txn)
    net_amount = assigned_last_txn['balance']
    ave_balance = account_txn['balance'].mean()
    ave_debit = debit_txns['amount'].mean()
    ave_credit = credit_txns['amount'].mean()

    return {'account_id': acct_id, 'amount_debited' : amount_debited,
            'amount_credited' : amount_credited, 'net_amount' : net_amount,
            'n_transactions':n_transactions, 'churn_flag': churn_flag,
            'first_trans': first_txn_time, 'last_trans': last_txn_time,
            'max_idle_time':max_idle_time,
            'ave_balance': ave_balance, 'ave_debit': ave_debit, 'ave_credit': ave_credit}
```

In [359]:
```
class_df = pd.DataFrame([get_last_inactivity(id) for id in target_accounts])
class_df.iloc[10:15,]
```

Out[359]:

	account_id	amount_debited	amount_credited	net_amount	n_transactions	churn_flag	first_trans	last_trans	max_idle_time	ave_balance	ave_debit	ave_credit
10	A00000011	34559.2	43526.7	37889.1	63	0	2017-01-03 11:16:55	2017-12-31 12:03:48	18.0	32283.622222	886.133333	1813.612500
11	A00000012	23454.4	49167.7	25713.2	34	0	2017-04-15 13:00:36	2017-12-31 12:45:07	31.0	22980.682353	1234.442105	3277.846667
12	A00000013	0.0	14606.0	14606.0	3	1	2017-08-17 14:21:33	2017-10-09 10:53:12	31.0	7803.000000	NaN	4868.666667
13	A00000014	178963.4	193606.3	39611.0	54	0	2017-01-10 12:35:04	2017-12-31 15:40:42	16.0	38857.583333	5965.446667	8075.262500
14	A00000015	220179.2	223827.8	37027.2	74	0	2017-01-03 11:17:27	2017-12-31 13:01:55	20.0	38100.168486	4403.584000	9317.816667

In [85]:
```
print("{} ({}%) total churns, {} ({}%) not churn".format(
      sum(class_df['churn_flag'] == 1), round(100*sum(class_df['churn_flag'] == 1)/len(class_df),4),
      sum(class_df['churn_flag'] == 0), round(100*sum(class_df['churn_flag'] == 0)/len(class_df),4)))
```

```
531 (11.85%) total churns, 3950 (88.15%) not churn
```

In [360]:
```
class_df['Time'] = class_df['last_trans'] - class_df['first_trans']
class_df
```

Out[360]:

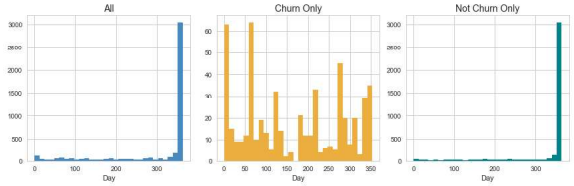
	account_id	amount_debited	amount_credited	net_amount	n_transactions	churn_flag	first_trans	last_trans	max_idle_time	ave_balance	ave_debit	ave_credit	Time
0	A00000001	47139,2	49275,1	14810,3	64	0	2017-01-05 08:25:51	2017-12-31 14:08:39	18,0	14099,793750	1274,032432	1825,003704	360 days 05:42:48
1	A00000002	281093,2	281578,0	53875,6	84	0	2017-01-01 08:46:37	2017-12-31 15:43:31	18,0	43722,405952	4846,434483	10829,923077	364 days 06:56:54
2	A00000003	31390,2	58139,7	26749,4	25	0	2017-07-07 16:55:15	2017-12-31 12:26:35	20,0	23092,900000	2853,654545	4152,835714	176 days 19:31:20
3	A00000004	72711,2	67644,8	19891,3	78	0	2017-01-01 15:19:04	2017-12-31 15:21:53	17,0	20180,684615	1346,503704	2818,533333	364 days 00:02:49
4	A00000005	8389,8	35950,9	27561,0	18	0	2017-05-30 15:13:40	2017-12-31 15:03:36	31,0	22244,094444	1198,542857	3268,263636	214 days 23:49:56
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4476	A00011333	506102,2	575286,5	88960,0	89	0	2017-01-06 15:28:01	2017-12-31 13:24:54	18,0	58198,873034	11769,818605	12506,228261	358 days 21:56:53
4477	A00011349	568642,0	552273,8	28332,0	93	0	2017-01-02 13:27:36	2017-12-31 13:42:47	18,0	52998,366667	8362,382353	22090,952000	363 days 00:15:11
4478	A00011359	356718,3	352500,4	32189,6	97	0	2017-01-04 12:20:25	2017-12-31 16:55:03	15,0	26056,784536	4886,552055	14687,516667	361 days 04:34:38
4479	A00011362	210579,2	214136,4	27848,2	118	0	2017-01-02 09:06:20	2017-12-31 12:31:00	19,0	26335,063559	2264,292473	8565,456000	363 days 03:24:40
4480	A00011382	363318,8	403828,5	67515,2	78	0	2017-01-10 15:59:19	2017-12-31 16:27:25	20,0	28908,342308	8650,447619	11217,458333	355 days 00:28:06

4481 rows × 13 columns

In [361]:
```
class_df['Days'] = class_df['Time'].apply(lambda t: t.days)
class_df['Weeks'] = (class_df['Time']/np.timedelta64(1,'W')).round(decimals=0).astype(int)
```

Time to Event Distributions

In [382]:
```
f,axes = plt.subplots(1,3, figsize=(12,4))
axes = axes.ravel()
plt.style.use('seaborn-whitegrid')

churn_data = class_df[class_df['churn_flag']==1]
no_churn_data = class_df[class_df['churn_flag']==0]

data = [class_df['Days'], churn_data['Days'], no_churn_data['Days']]
titles = ['All', 'Churn Only', 'Not Churn Only']
colors = ['#4889BF', '#ECAE3F', '#008489']

for idx, ax in enumerate(axes):
    ax.hist(data[idx], bins=30, color=colors[idx])
    ax.set_title(titles[idx], fontsize=14)
    ax.set_xlabel('Day')
plt.tight_layout()
```

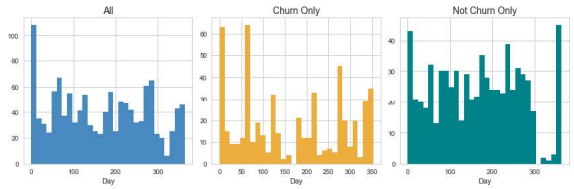


Resampling Data Set

In [494]:
```
from sklearn.utils import resample

no_churn_recent = no_churn_data[no_churn_data['Days'] < 300]
no_churn_frequent = no_churn_data[no_churn_data['Days'] >= 300]

not_churn_downsampled = resample(no_churn_frequent, replace = False, n_samples = 51, random_state = 12345)
len(no_churn_recent)
```

Out[494]: 649

In [524]:
```
f,axes = plt.subplots(1,3, figsize=(12,4))
axes = axes.ravel()
plt.style.use('seaborn-whitegrid')

no_churn_data = pd.concat([no_churn_recent, not_churn_downsampled]).sort_values(['account_id']).reset_index(drop=True)

data = [pd.concat([churn_data, no_churn_data])['Days'], churn_data['Days'], no_churn_data['Days']]
titles = ['All', 'Churn Only', 'Not Churn Only']
colors = ['#4889BF', '#ECAE3F', '#008489']

for idx, ax in enumerate(axes):
    ax.hist(data[idx], bins=30, color=colors[idx])
    ax.set_title(titles[idx], fontsize=14)
    ax.set_xlabel('Day')
plt.tight_layout()
```



2. Univariate Modelling

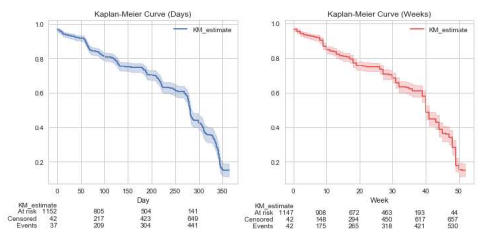
In [495]:
```
clean_accounts = pd.concat([churn_data, no_churn_recent, not_churn_downsampled]).sort_values(['account_id']).reset_index(drop=True)
clean_accounts.head(5)
```

Out[495]:

	account_id	amount_debited	amount_credited	net_amount	n_transactions	churn_flag	first_trans	last_trans	max_idle_time	ave_balance	ave_debit	ave_credit	Time	Days	Weeks
0	A00000003	31390,2	58139,7	26749,4	25	0	2017-07-07 16:55:15	2017-12-31 12:26:35	20,0	23092,900000	2853,654545	4152,835714	176 days 19:31:20	176	25
1	A00000005	8389,8	35950,9	27561,0	18	0	2017-05-30 15:13:40	2017-12-31 15:03:36	31,0	22244,094444	1198,542857	3268,263636	214 days 23:49:56	214	31
2	A00000012	23454,4	49167,7	25713,2	34	0	2017-04-15 13:00:36	2017-12-31 12:45:07	31,0	22980,682353	1234,442105	3277,846667	259 days 23:44:31	259	37
3	A00000013	0,0	14606,0	14606,0	3	1	2017-08-17 14:21:33	2017-10-09 10:53:12	31,0	7803,000000	NaN	4868,666667	52 days 20:31:39	52	8
4	A00000016	37000,0	75006,1	38006,1	10	0	2017-06-23 09:04:26	2017-12-31 11:33:26	17,0	30473,410000	18500,000000	9375,762500	99 days 02:29:00	99	14

In [496]:
```
from lifelines import KaplanMeierFitter

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))

time = clean_accounts['Days']
event = clean_accounts['churn_flag']

kmf = KaplanMeierFitter()
kmf.fit(time, event_observed=event)

kmf.plot(at_risk_counts=True, ax=ax1)
ax1.set_xlabel('Day')
plt.title('Kaplan-Meier Curve (Days)')

week = clean_accounts['Weeks']
kmf2 = KaplanMeierFitter()
kmf2.fit(week, event_observed=event)

kmf2.plot(at_risk_counts=True, ax=ax2, color='#E65656')
ax2.set_xlabel('Week')
plt.title('Kaplan-Meier Curve (Weeks)')
```

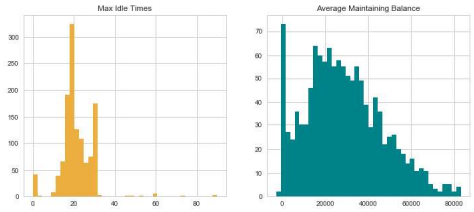
Out[496]: Text(0.5, 1.0, 'Kaplan-Meier Curve (Weeks)')



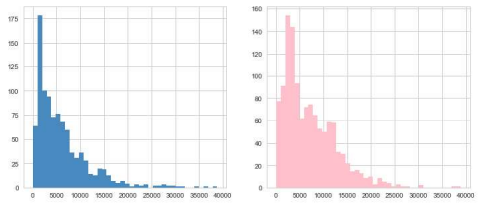
In [497]:
```
kmf.median_survival_time_
```

Out[497]:
```
282.0
```

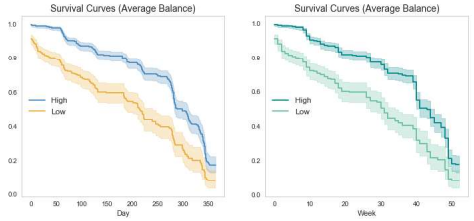
In [498]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,5))
colors = ['#4889BF', '#ECAE3F', '#008489', 'pink', 'lightblue']

ax1.set_title('Max Idle Times')
clean_accounts['max_idle_time'].hist(bins=40, ax=ax1, color=colors[1])
ax2.set_title('Average Maintaining Balance')
clean_accounts['ave_balance'].hist(bins=40, ax=ax2, color=colors[-3]);
```



In [499]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,5))

clean_accounts['ave_debit'].hist(bins=40, ax=ax1, color=colors[0])
clean_accounts['ave_credit'].hist(bins=40, ax=ax2, color=colors[3]);
```



In [500]:
```
# Segmentations
def segment_idle_times(t):
    idle_type = 'Long'
    if t < 19:
        idle_type = "Short"
    elif t < 22:
        idle_type = "Average"
    return idle_type

clean_accounts['Idle'] = [segment_idle_times(t) for t in clean_accounts['max_idle_time']]
clean_accounts['balance_type'] = ['High' if b > 20000 else 'Low' for b in clean_accounts['ave_balance']]

# Plot KM curves
colors = ['#4889BF', '#ECAE3F', '#008489', '#66BFA1', '#E65656']
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,5))
plt.style.use('seaborn-whitegrid')

kmf = KaplanMeierFitter()
for i, idle in enumerate(clean_accounts['balance_type'].unique()):
    flag = clean_accounts['balance_type'] == idle

    kmf.fit(time[flag], event_observed=event[flag], label=idle)
    kmf.plot(ax=ax1, color=colors[i], xlabel = "Day")
    ax1.grid(False)
    ax1.legend(loc = 'center left', fontsize = 12)
    ax1.set_title("Survival Curves (Average Balance)",fontsize=14)


kmf2 = KaplanMeierFitter()
for i, idle in enumerate(clean_accounts['balance_type'].unique()):
    flag = clean_accounts['balance_type'] == idle

    kmf2.fit(week[flag], event_observed=event[flag], label=idle)
    kmf2.plot(ax=ax2, color=colors[i + 2], xlabel = "Week")
    ax2.grid(False)
    ax2.legend(loc = 'center left', fontsize = 12)
    ax2.set_title("Survival Curves (Average Balance)", fontsize=14);
```



3. Survival Regression

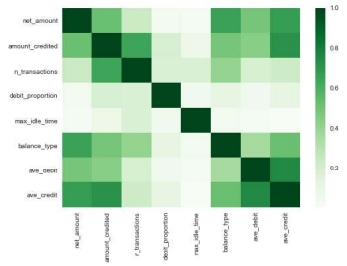
In [503]:
```
clean_accounts['debit_proportion'] = clean_accounts['amount_debited']/clean_accounts['net_amount']
clean_accounts['n_weekly_trans'] = clean_accounts['n_transactions']/clean_accounts['Weeks']
clean_accounts['balance_type'] = [1 if b == 'High' else 0 for b in clean_accounts['balance_type']]
clean_accounts.head(5)
```

Out[503]:

	account_id	amount_debited	amount_credited	net_amount	n_transactions	churn_flag	first_trans	last_trans	max_idle_time	ave_balance	ave_debit	ave_credit	Time	Days	Weeks	Idle	balance_type	debit_proportion	n_weekly_trans
0	A00000003	31390.2	58139.7	26749.4	25	0	2017-07-07 16:55:15	2017-12-31 12:26:35	20.0	23092.900000	2853.654545	4152.835714	176 days 19:31:20	176	25	Average	1	1.173492	1.000000
1	A00000005	8389.8	35950.9	27561.0	18	0	2017-05-30 15:13:40	2017-12-31 15:03:36	31.0	22244.094444	1198.542857	3268.263636	214 days 23:49:56	214	31	Long	1	0.304408	0.580645
2	A00000012	23454.4	49167.7	25713.2	34	0	2017-04-15 13:00:36	2017-12-31 12:45:07	31.0	22980.682353	1234.442105	3277.846667	259 days 23:44:31	259	37	Long	1	0.912154	0.918919
3	A00000013	0.0	14606.0	14606.0	3	1	2017-08-17 14:21:33	2017-10-09 10:53:12	31.0	7803.000000	NaN	4868.666667	52 days 20:31:39	52	8	Long	0	0.000000	0.375000
4	A00000016	37000.0	75006.1	38006.1	10	0	2017-09-23 09:04:28	2017-12-31 11:33:26	17.0	30473.410000	18500.000000	9375.762500	99 days 02:29:00	99	14	Short	1	0.973528	0.714286

In [504]:
```
import seaborn as sns

col_names = ['net_amount', 'amount_credited', 'n_transactions', 'debit_proportion',
             'max_idle_time', 'balance_type', 'ave_debit', 'ave_credit', #'n_weekly_trans',
             'Days', 'Weeks', 'Age', 'churn_flag']

corr = clean_accounts[col_names[:-4]].corr()
corr = np.abs(corr)

# plot the heatmap
sns.heatmap(corr, cmap="Greens",
        xticklabels=corr.columns,
        yticklabels=corr.columns)
```

Out[504]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26f55ca0748>
```



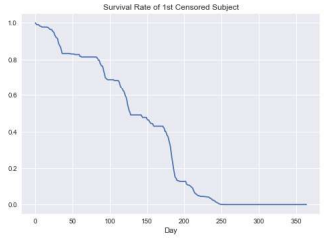
In [630]:
```
from lifelines import CoxPHFitter

form0 = " + ".join(['net_amount', 'amount_credited', 'n_transactions'])
form1 = " + ".join(['net_amount', 'amount_credited', 'n_transactions', 'max_idle_time'])
form2 = " + ".join(['n_transactions', 'amount_credited', 'max_idle_time', 'balance_type'])
form3 = " + ".join(['n_transactions', 'balance_type', 'max_idle_time'])
form4 = " + ".join(['n_transactions', 'balance_type', 'max_idle_time', 'ave_debit'])

cph = CoxPHFitter(penalizer=0.1)
cph.fit(clean_accounts.fillna(0.001), duration_col='Days', event_col='churn_flag',
        formula = form3)
cph.print_summary()
```

model	lifelines.CoxPHFitter
duration col	'Days'
event col	'churn_flag'
penalizer	0.1
l1 ratio	0
baseline estimation	breslow
number of observations	1231
number of events observed	531
partial log-likelihood	-3115.95
time fit was run	2020-09-21 07:19:37 UTC

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	z	p	-log2(p)
n_transactions	-0.03	0.97	0.00	-0.03	-0.03	0.97	0.97	-13.49	<0.005	135.26
balance_type	-0.46	0.63	0.09	-0.63	-0.30	0.53	0.74	-5.43	<0.005	24.11
max_idle_time	-0.04	0.96	0.01	-0.06	-0.03	0.95	0.97	-7.00	<0.005	38.47

Concordance	0.81
Partial AIC	6237.89
log-likelihood ratio test	265.05 on 3 df
-log2(p) of ll-ratio test	187.49

In [629]:
```
cph_new = cph.fit(clean_accounts, duration_col='Days', event_col='churn_flag',
        formula = form3)

# filter down to just censored subjects to predict remaining survival
censored_subjects = clean_accounts.loc[clean_accounts['churn_flag'] == 0]
censored_subjects_last_obs = censored_subjects['Days']

# predict new survival function
ps = cph_new.predict_survival_function(censored_subjects,
                                       conditional_after=censored_subjects_last_obs)

ps[20].plot(xlabel="Day", title="Survival Rate of 1st Censored Subject")

# predict median remaining life
pred_median_time = cph_new.predict_median(censored_subjects,
                                       conditional_after=censored_subjects_last_obs)
```



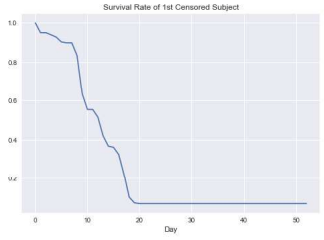
In [521]:
```
display = clean_accounts.iloc[0:10,[4,5,8,13,16]]
display['Remaining Life'] = pred_median_time[0:20]
display['Customer Lifetime'] = display['Days'] + display['Remaining Life']
display
```

Out[521]:

	n_transactions	churn_flag	max_idle_time	Days	balance_type	Remaining Life	Customer Lifetime
0	25	0	20.0	176	1	104.0	280.0
1	18	0	31.0	214	1	74.0	288.0
2	34	0	31.0	259	1	73.0	332.0
3	3	1	31.0	52	0	NaN	NaN
4	10	0	17.0	99	1	124.0	223.0
5	3	0	23.0	25	0	131.0	156.0
6	16	1	18.0	62	1	NaN	NaN
7	29	0	17.0	135	1	145.0	280.0
8	19	0	23.0	145	1	134.0	279.0
9	17	0	17.0	151	1	124.0	275.0

Fitting Cox PH Models (Weeks)

In [628]:
```
cph_new = cph.fit(clean_accounts, duration_col='Weeks', event_col='churn_flag',
        formula = form3)

# filter down to just censored subjects to predict remaining survival
censored_subjects = clean_accounts.loc[clean_accounts['churn_flag'] == 0]
censored_subjects_last_obs = censored_subjects['Weeks']

# predict new survival function
ps = cph_new.predict_survival_function(censored_subjects,
                                       conditional_after=censored_subjects_last_obs)

ps[1].plot(xlabel="Day", title="Survival Rate of 1st Censored Subject")

# predict median remaining life
pred_median_time = cph_new.predict_median(censored_subjects,
                                       conditional_after=censored_subjects_last_obs)
```



In [554]:
```
display = clean_accounts[['churn_flag', 'Weeks']]
display['Remaining Life'] = pred_median_time
display['Customer Lifetime'] = display['Weeks'] + display['Remaining Life']
display.head(20)
```

```
C:\Users\CD250050\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\CD250050\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

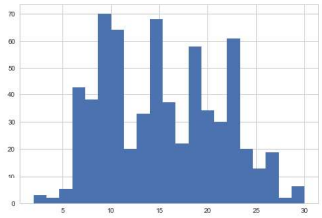
Out[554]:

	churn_flag	Weeks	Remaining Life	Customer Lifetime
0	0	25	15,0	40,0
1	0	31	13,0	44,0
2	0	37	11,0	48,0
3	1	8	NaN	NaN
4	0	14	18,0	32,0
5	0	4	22,0	26,0
6	1	9	NaN	NaN
7	0	19	21,0	40,0
8	0	21	19,0	40,0
9	0	22	18,0	40,0
10	0	52	inf	inf
11	0	17	14,0	31,0
12	0	38	3,0	41,0
13	0	42	5,0	47,0
14	1	4	NaN	NaN
15	1	27	NaN	NaN
16	1	8	NaN	NaN
17	1	31	NaN	NaN
18	1	27	NaN	NaN
19	0	40	4,0	44,0

In [581]:
```
def segment_remaining_life(remaining_life):
    type_ = 'Low'
    if remaining_life < 8:
        type_ = 'Extreme'
    elif remaining_life < 12:
        type_ = 'High'
    elif remaining_life < 16:
        type_ = 'Moderate'
    elif remaining_life < 24:
        type_ = 'Average'
    return type_
```

In [582]:
```
finite_life = display[(np.isfinite(display['Remaining Life'])) & (display['Remaining Life'].notnull())]['Remaining Life']
finite_life.hist(bins=21)
```

Out[582]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26f6790b3c8>
```



In [634]:
```
display['Risk'] = display['Remaining Life'].apply(lambda life: segment_remaining_life(life))
result = display.groupby(['Risk']).count()
result = pd.DataFrame({'Risk':result.index, 'Count':result['Remaining Life']}).reset_index(drop=True)
result = result.iloc[[3,0,4,2,1]]
result.plot.barh(x='Risk', y='Count', legend=False, title='Number of Observations per Risk',
                 color=['#ffffb2','#fecc5c','#fd8d3c','#f03b20','#bd0026']);
```