In [190]:

```
from lifetimes.utils import summary_data_from_transaction_data

import numpy as np
import pandas as pd
```

In [191]:

```
import lifetime_value as ltv
```

a.- Transform data as RFM

In [192]:

```
import pandas as pd
df1 = pd.read_csv('transactions_company_10000.csv', sep=',')
df1 = df1[df1['purchaseamount']>0]
```

In [193]:

```
df1.head()
```

Out[193]:

	id	chain	dept	category	company	brand	date	productsize	productmeasure	purchasequantity	purchaseamount
0	86246	205	97	9753	10000	0	2012-03-02	1.0	CT	1	0.69
2	86246	205	0	0	10000	0	2012-03-17	0.0	NaN	4	5.81
4	86246	205	0	0	10000	0	2012-03-24	0.0	NaN	1	5.79
6	86246	205	97	9753	10000	0	2012-03-28	1.0	CT	1	1.18
7	86246	205	0	0	10000	0	2012-03-31	0.0	NaN	3	37.89

In [94]:

```
import pandas as pd
df12 = pd.read_csv('customer_level_data_company_10000.csv', sep=',')
df12.head()
```

Out[94]:

	id	calibration_value	chain	dept	category	brand	productmeasure	holdout_value	log_calibration_value	label
0	86246	0.69	205	97	9753	0	CT	322.73	-0.371064	322.73
1	86252	4.69	205	0	0	0	UNKNOWN	310.04	1.545433	310.04
2	12262064	0.99	95	97	9753	0	CT	11.73	-0.010050	11.73
3	12277270	1.99	95	0	0	0	UNKNOWN	139.27	0.688135	139.27
4	12332190	1.00	95	97	9753	0	CT	11.72	0.000000	11.72

In [194]:

```
from lifetimes.utils import summary_data_from_transaction_data

xaction_RFM = summary_data_from_transaction_data(
    df1,
    'id',
    'date',
    'purchaseamount',
    observation_period_end='2013-07-28')

xaction_RFM.head()
```

Out[194]:

id	frequency	recency	T	monetary_value
86246	105.0	417.0	513.0	8.578190
86252	84.0	389.0	513.0	10.165119
12262064	6.0	401.0	438.0	3.170000
12277270	24.0	458.0	508.0	7.982500
12332190	4.0	154.0	353.0	2.930000

In [96]:

```
xaction_RFM.describe()
```

Out[96]:

	frequency	recency	T	monetary_value
count	234385.000000	234385.000000	234385.000000	234385.000000
mean	7.185140	236.455835	401.247985	3.708001
std	11.323179	148.576623	112.695688	29.026119
min	0.000000	0.000000	5.000000	0.000000
25%	1.000000	107.000000	335.000000	1.108000
50%	4.000000	268.000000	443.000000	2.486667
75%	9.000000	360.000000	491.000000	4.306667
max	509.000000	512.000000	513.000000	8463.085000

In [195]:

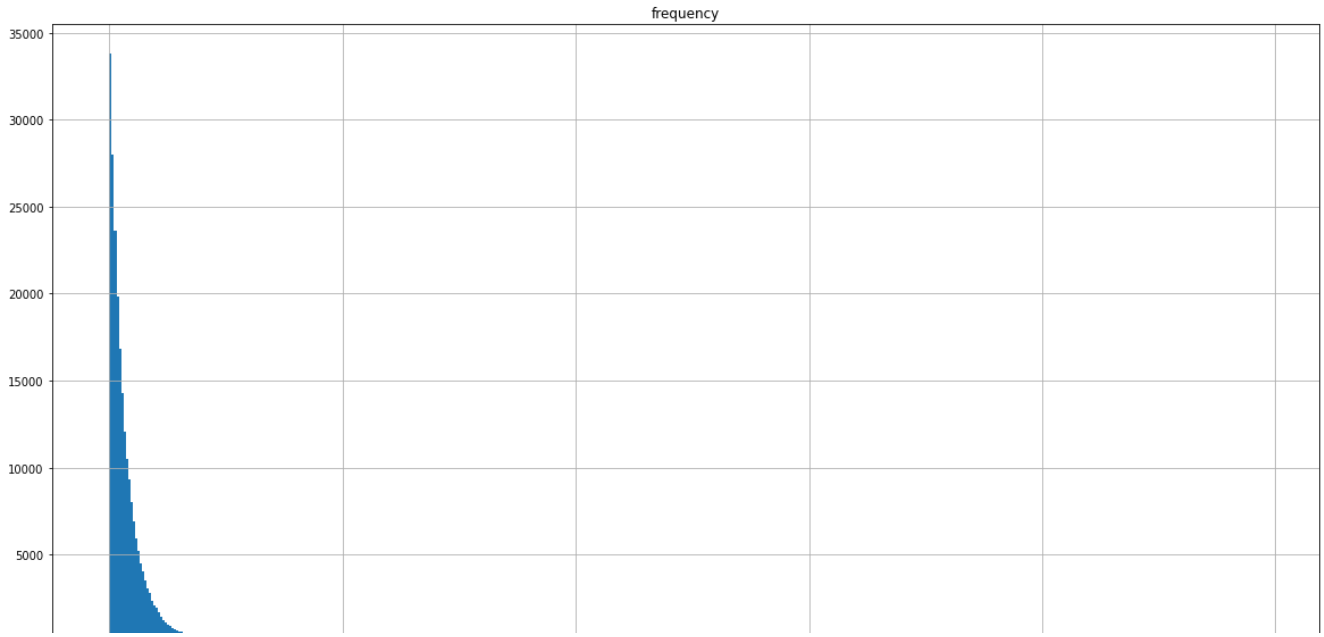
```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (20,10)

import pandas as pd
plot_freq = xaction_RFM[xaction_RFM['frequency']<500]

plot_freq.hist(column='frequency', bins=500)
```

Out[195]:

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



3.) Training Model to det. 'Alive-ness'

For small samples sizes, the parameters can get implausibly large, so by adding an l2 penalty the likelihood, we can control how large these parameters can be. This is implemented as setting as positive `penalizer_coef` in the initialization of the model. In typical applications, penalizers on the order of 0.001 to 0.1 are effective.

In [196]:

```
from lifetimes import BetaGeoFitter

I2 = 0.0001

bgf = BetaGeoFitter(penalizer_coef = I2)
bgf.fit(xaction_RFM['frequency'], xaction_RFM['recency'], xaction_RFM['T'])
```

Out[196]:

```
<lifetimes.BetaGeoFitter: fitted with 234385 subjects, a: 1.04, alpha: 45.52, b: 12.48, r: 0.99>
```

a.- Visualizing Model Frequency/Recency Matrix

Recency_Frequency Plot

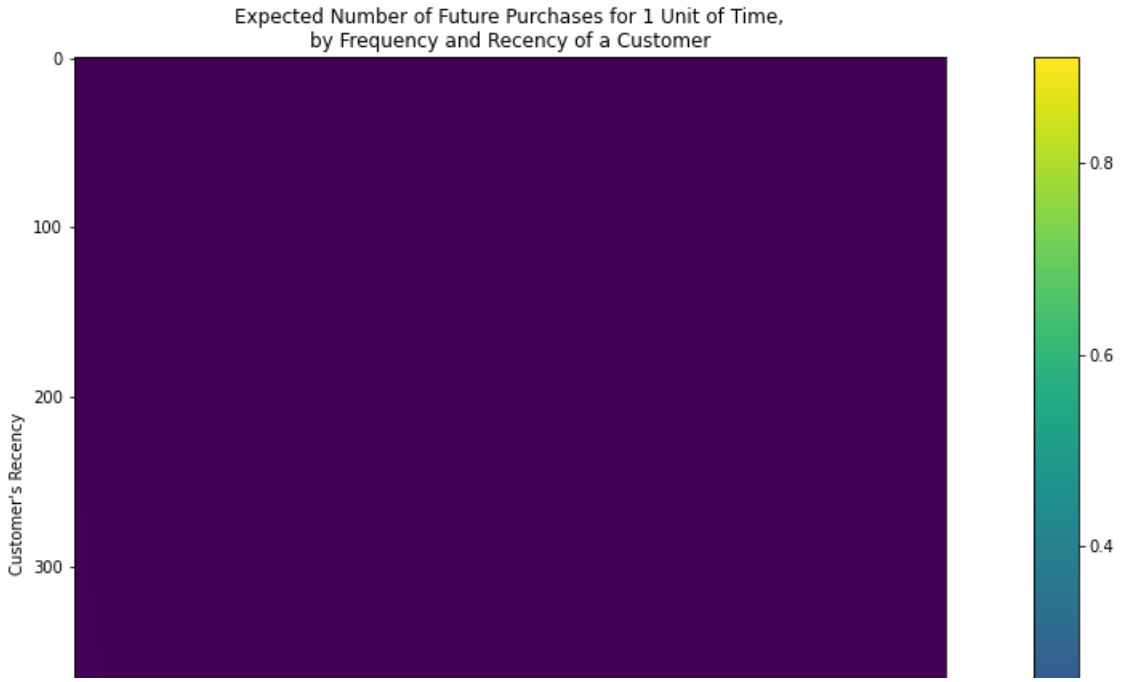
In [197]:

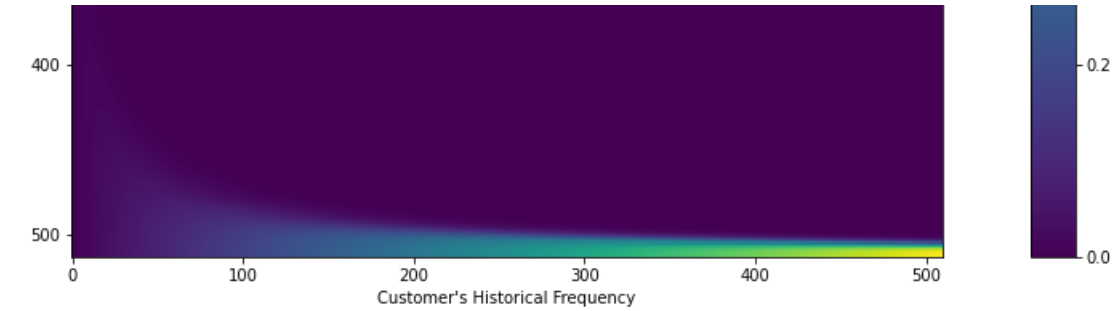
```
from lifetimes.plotting import plot_frequency_recency_matrix
plt.rcParams["figure.figsize"] = (20,10)
plot_frequency_recency_matrix(bgf, T=1)
```

```
/home/temor/miniconda3/envs/RFM/lib/python3.8/site-
packages/lifetimes/fitters/beta_geo_fitter.py:256: RuntimeWarning: overflow encountered in
double_scalars
  denominator = 1 + (x > 0) * (a / (b + x - 1)) * ((alpha + T) / (alpha + recency)) ** (r + x)
```

Out[197]:

```
<AxesSubplot:title={'center':'Expected Number of Future Purchases for 1 Unit of Time,\nby
Frequency and Recency of a Customer'}, xlabel="Customer's Historical Frequency",
ylabel="Customer's Recency">
```





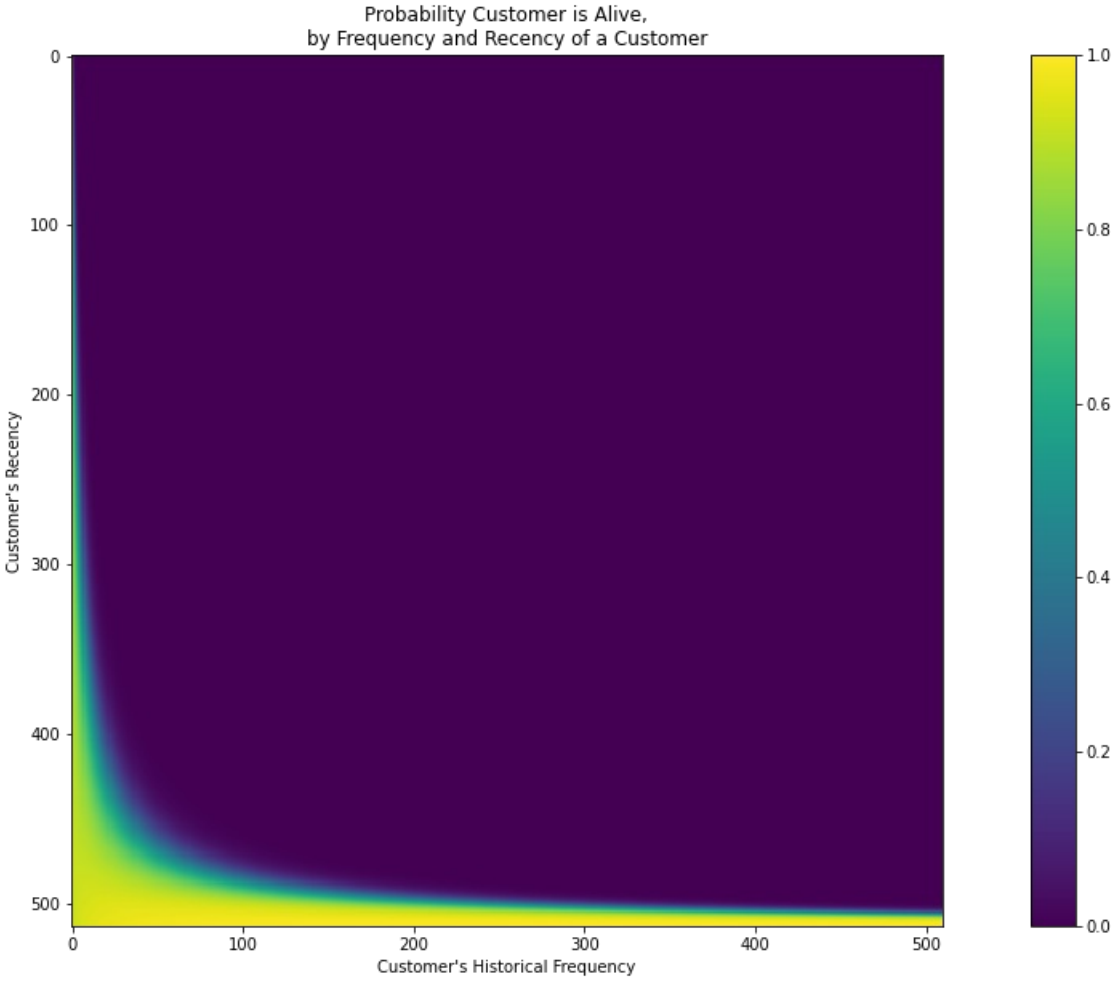
'Still Alive' Plot

In [198]:

```
from lifetimes.plotting import plot_probability_alive_matrix
plt.rcParams["figure.figsize"] = (20,10)
plot_probability_alive_matrix(bgf)
```

Out[198]:

```
<AxesSubplot:title={'center':'Probability Customer is Alive,\nby Frequency and Recency of a
Customer'}, xlabel="Customer's Historical Frequency", ylabel="Customer's Recency">
```



b.- Rank customer from Best to Worst

The expected number of repeat purchases in the **next day** using their history.

In [199]:

```
t = 1  #day(s)
xaction_RFM['predicted_purchase'] = bgf.conditional_expected_number_of_purchases_up_to_time(
```

```
    t,
    xaction_RFM['frequency'],
    xaction_RFM['recency'],
    xaction_RFM['T']
)
```

In [200]:

```
t = 3  #day(s)
xaction_RFM['predicted_purchase'] = bgf.conditional_expected_number_of_purchases_up_to_time(
    t,
    xaction_RFM['frequency'],
    xaction_RFM['recency'],
    xaction_RFM['T']
)


xaction_RFM.sort_values(by='predicted_purchase').head()
```

Out[200]:

id	frequency	recency	T	monetary_value	predicted_purchase
3288731255	377.0	377.0	513.0	5707.329920	1.172251e-43
3388822242	374.0	377.0	513.0	2876.808021	2.665434e-43
3388824436	379.0	379.0	513.0	2987.222005	4.078203e-43
347580535	378.0	379.0	513.0	245.244894	5.337666e-43
349776602	376.0	379.0	513.0	178.703697	9.143215e-43

The expected number of repeat purchases over the **next 3-days** using their history.

c.- Split Data (training & validation)

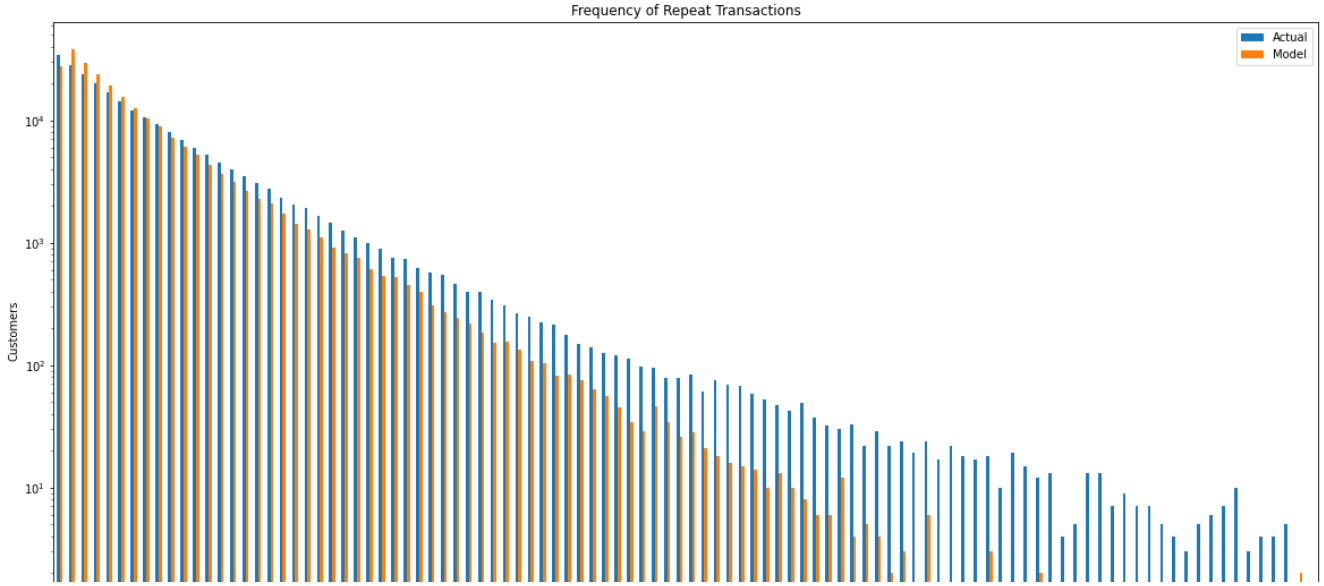
Split data into a **training** (calibration) period and a **holdout** (observation) period. Then, train the BG/NBD model and evaluate performance with four plots as outlined by Peter Fader (26:10)

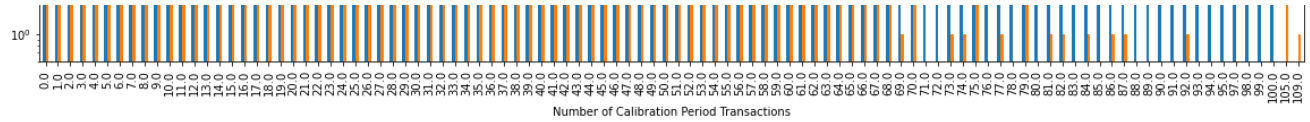
Calibration period histogram

Compares the purchases between actual and what model's predicts. The more similar in values between the two, the better.

In [201]:

```
from lifetimes.plotting import plot_period_transactions
plot_period_transactions(bgf, max_frequency=100).set_yscale('log')
```





The model is fairly representative of the real data up until **third repeat** transactions. This proves that our model doesn't suck

Cumulative transaction plot

In [202]:

```
from lifetimes.utils import calibration_and_holdout_data

summary_cal_holdout = calibration_and_holdout_data(df1, 'id', 'date',
                                        calibration_period_end='2013-02-28',
                                        observation_period_end='2013-07-28' )
```

In [203]:

```
bgf.fit(summary_cal_holdout['frequency_cal'], summary_cal_holdout['recency_cal'],
summary_cal_holdout['T_cal'])
```

Out[203]:

```
<lifetimes.BetaGeoFitter: fitted with 225115 subjects, a: 0.01, alpha: 54.56, b: 1.58, r: 1.18>
```

In [204]:

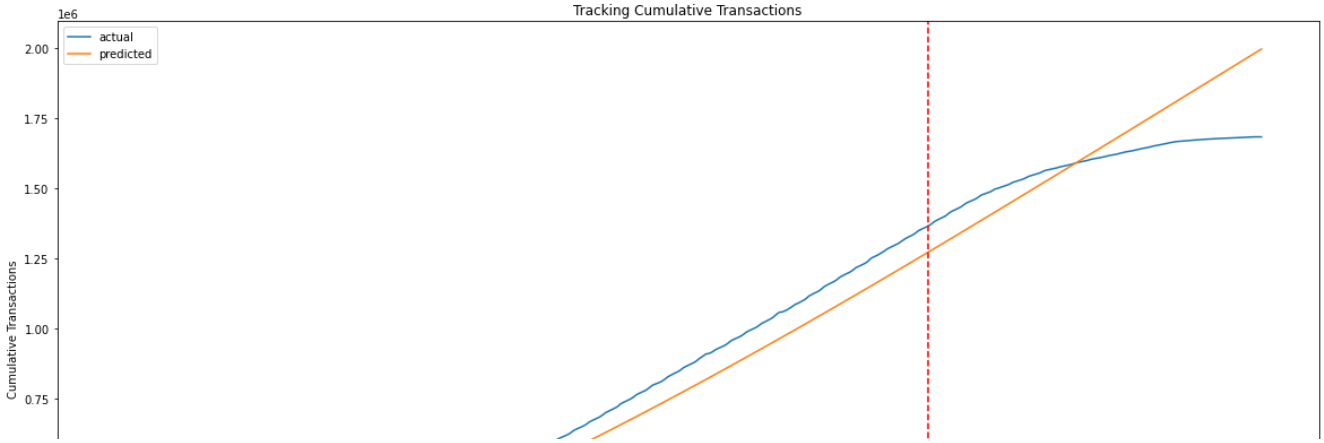
```
summary_cal_holdout.describe()
```

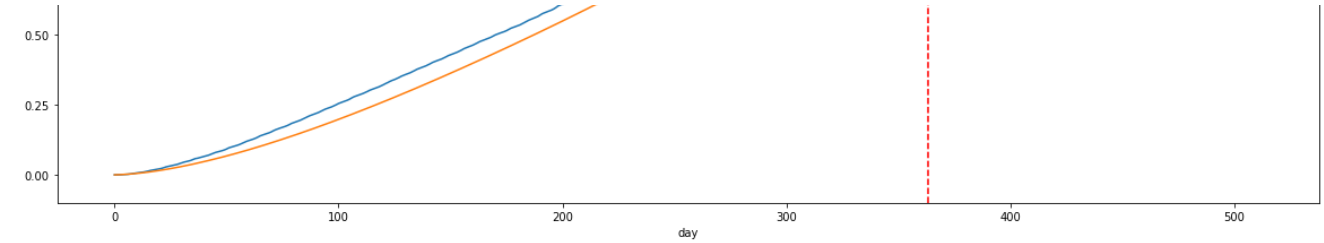
Out[204]:

	frequency_cal	recency_cal	T_cal	frequency_holdout	duration_holdout
count	225115.000000	225115.000000	225115.000000	225115.000000	225115.0
mean	6.059179	188.938320	263.665531	1.395114	150.0
std	9.712225	124.639341	96.259319	2.658672	0.0
min	0.000000	0.000000	0.000000	0.000000	150.0
25%	1.000000	69.000000	204.000000	0.000000	150.0
50%	3.000000	217.000000	299.000000	1.000000	150.0
75%	8.000000	302.000000	342.000000	2.000000	150.0
max	363.000000	363.000000	363.000000	149.000000	150.0

In [205]:

```
from lifetimes.plotting import plot_cumulative_transactions
plt.rcParams["figure.figsize"] = (20,10)
plot_cumulative_transactions(bgf, df1, 'date', 'id', 513, 363, freq='D');
```





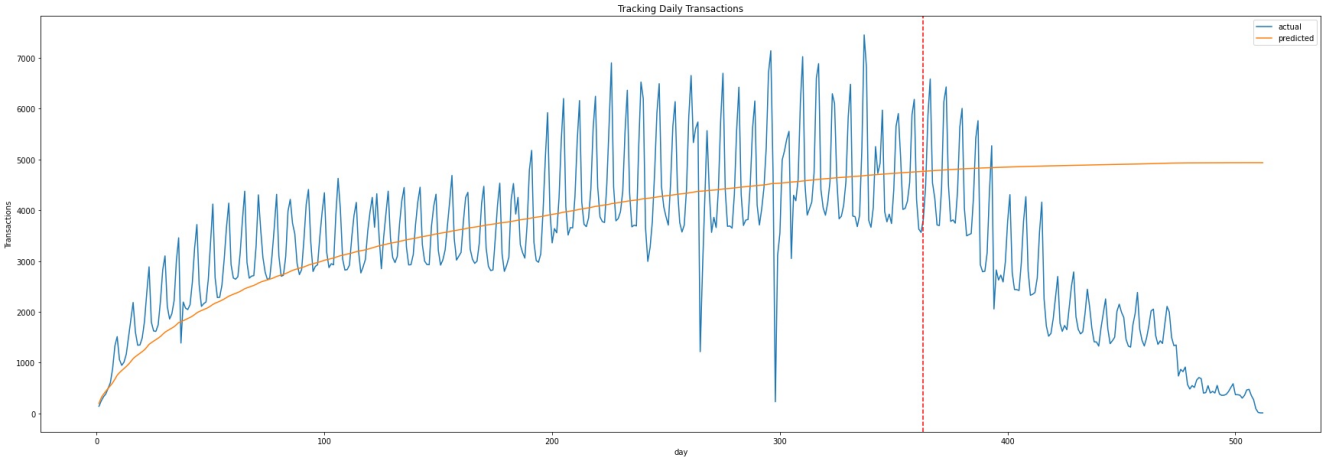
The red line represents the boundary between the calibration period on the left and the holdout period on the right.

As you can see, the BG/NBD model is not so greate in predicting cumulative transactions.

Incremental transaction plot

In [206]:

```
from lifetimes.plotting import plot_incremental_transactions
plt.rcParams["figure.figsize"] = (30,10)
plot_incremental_transactions(bgf, df1, 'date', 'id', 513, 363, freq='D');
```



This plot shows that the model does only a decent job capturing general trends in the data.

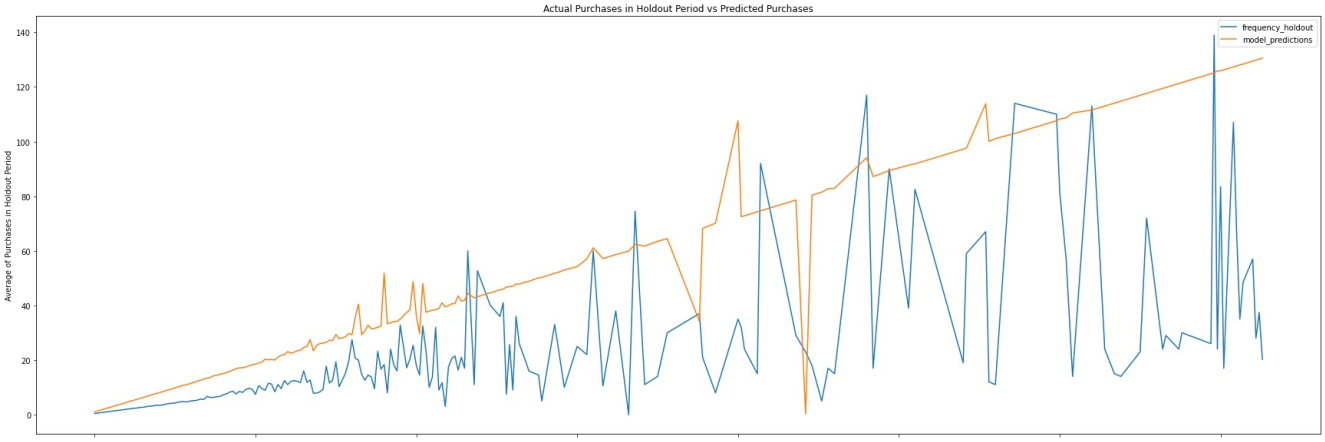
Conditional expectations plot

In [207]:

```
from lifetimes.plotting import plot_calibration_purchases_vs_holdout_purchases
plot_calibration_purchases_vs_holdout_purchases(bgf, summary_cal_holdout, n=500)
```

Out[207]:

```
<AxesSubplot:title={'center':'Actual Purchases in Holdout Period vs Predicted Purchases'},
xlabel='Purchases in calibration period', ylabel='Average of Purchases in Holdout Period'>
```



The model performs OK up to about 30-days, but increasingly diverges from the holdout data because of thie distribution of the data.

In [110]:

```
df1.groupby('id').size().value_counts()
```

Out[110]:

```
1        33393
2        27519
3        23239
4        19491
5        16567
         ...
1081         1
953          1
951          1
127          1
180          1
Length: 264, dtype: int64
```

d. Predictions

Single Customer Prediction

Let's pick some customer and predict what that indivudla's future purchase might look like:

In [208]:

```
t = 3  #predict purchases in 3-days
individual = xaction_RFM.loc[86252, :]     # customerID = '13694'
bgf.conditional_expected_number_of_purchases_up_to_time(
    t,
    individual['frequency'],
    individual['recency'],
    individual['T']
)
```

Out[208]:

```
2.302642421671075e-06
```

Single Customer probability Histories

Given a customer transaction history, we can calculate their historical probability of being alive, according to our trained model. For example:

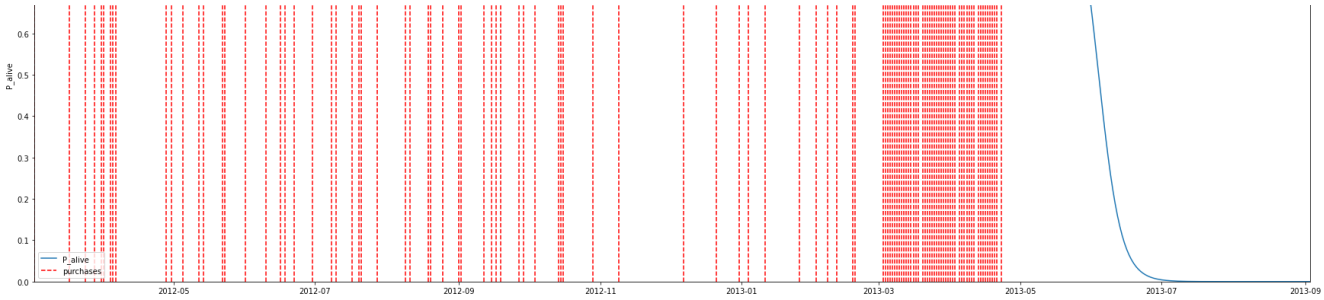
In [209]:

```
from lifetimes.plotting import plot_history_alive

customer_ID = 86246
days_since_birth = 550
sp_trans = df1.loc[df1['id'] == customer_ID]
plot_history_alive(bgf, days_since_birth, sp_trans, 'date')
```

Out[209]:

```
<AxesSubplot:title={'center':'History of P_alive'}, ylabel='P_alive'>
```



4.) Estimating CLV using Gamma-Gamma model

Until now, we have not taken into account the economics value of each transaction; rather, we've only been focusing on transactions' occurences.

In order to estimate CLV, we will use the `Gamma-Gama` submodel.

First, we shall create summary data from the transactional data that contains the economics values for each transaction (e.g., profits or revenues).

In [210]:

```
summary_with_money_value = xaction_RFM
returning_customers_summary = summary_with_money_value[summary_with_money_value['frequency']>0]
returning_customers_summary.head()
```

Out[210]:

id	frequency	recency	T	monetary_value	predicted_purchase
86246	105.0	417.0	513.0	8.578190	1.322366e-07
86252	84.0	389.0	513.0	10.165119	2.261019e-08
12262064	6.0	401.0	438.0	3.170000	3.921630e-02
12277270	24.0	458.0	508.0	7.982500	1.029548e-01
12332190	4.0	154.0	353.0	2.930000	1.201376e-02

NOTE

If computing the monetary value from your own data, note that it is the mean of a given customer's value, not the sum. monetary_value can be used to represent profit, or revenue, or any value as long as it is consistently calculated for each customer.

a.- Gamma-Gamma Model && Independence Assumptions

The model we are going to use to estimate the CLV for our data is called the Gamma-Gamma submodel, which relies upon an important assumption.

The Gamma-Gamma submodel assumes that **there is no relationship between the monetary value and the purchase frequency.** In practice we need to check whether the Pearson correlation between the two vectors is close to 0 in order to use this model.

In [211]:

```
returning_customers_summary[['monetary_value', 'frequency']].corr()
```

Out[211]:

	monetary_value	frequency
monetary_value	1.000000	0.164569
frequency	0.164569	1.000000

Since there doesn't seem to be a high correlation between `Monetary Value` and `Purchase Frequency` (value of 0.015882), we can proceed in training our Gamma-Gamma submodel and predict the conditional, expected vaerage lifetime value of our customers.

b.- Train Gamma-Gamma model

In [213]:

```
from lifetimes import GammaGammaFitter

ggf = GammaGammaFitter(penalizer_coef = 0.001)
ggf.fit(
    returning_customers_summary['frequency'],
    returning_customers_summary['monetary_value']
)
ggf
```

Out[213]:

```
<lifetimes.GammaGammaFitter: fitted with 200592 subjects, p: 2.08, q: 2.13, v: 2.45>
```

c.- Est. Avg. Transaction Value

In [116]:

```
ggf.conditional_expected_average_profit(
    returning_customers_summary['frequency'],
    returning_customers_summary['monetary_value']
).head(11)
```

Out[116]:

```
id
86246        8.563895
86252       10.139489
12262064     3.283912
12277270     7.930869
12332190     3.118026
12524696     0.881253
12682470     2.144773
12996040     9.522469
13074629     3.545083
13089312     2.600518
13179265     1.957610
dtype: float64
```

In [154]:

```
x =
ggf.conditional_expected_average_profit(returning_customers_summary['frequency'],returning_customer
s_summary['monetary_value'])

GammaPred = pd.DataFrame(x)
GammaPred.to_csv('M_Pred_V2.csv')
```

In [214]:

```
print(
    "Expected conditional average revenue: %s, Average revenue: %s" % (
        ggf.conditional_expected_average_profit(
            returning_customers_summary['frequency'],
            returning_customers_summary['monetary_value']
        ).mean(),

        summary_with_money_value[summary_with_money_value['frequency']>0]['monetary_value'].mean()
    )
)
```

Expected conditional average revenue: 4.388770547535693, Average revenue: 4.332674204362177

In [215]:

```
bgf.fit(returning_customers_summary['frequency'], returning_customers_summary['recency'],
returning_customers_summary['T'])
print(ggf.customer_lifetime_value(
    bgf, #the model to use to predict the number of future transactions
    returning_customers_summary['frequency'],
    returning_customers_summary['recency'],
    returning_customers_summary['T'],
    returning_customers_summary['monetary_value'],
    time=5, # months
    discount_rate=0.01 # monthly discount rate ~ 12.7% annually
).head(10))
```

```
id
86246        0.000080
86252        0.000017
12262064     6.064081
12277270    34.824586
12332190     1.466677
12524696     2.926031
12682470     1.566076
12996040     5.084865
13074629     3.775058
13089312     2.198160
Name: clv, dtype: float64
```

In [216]:

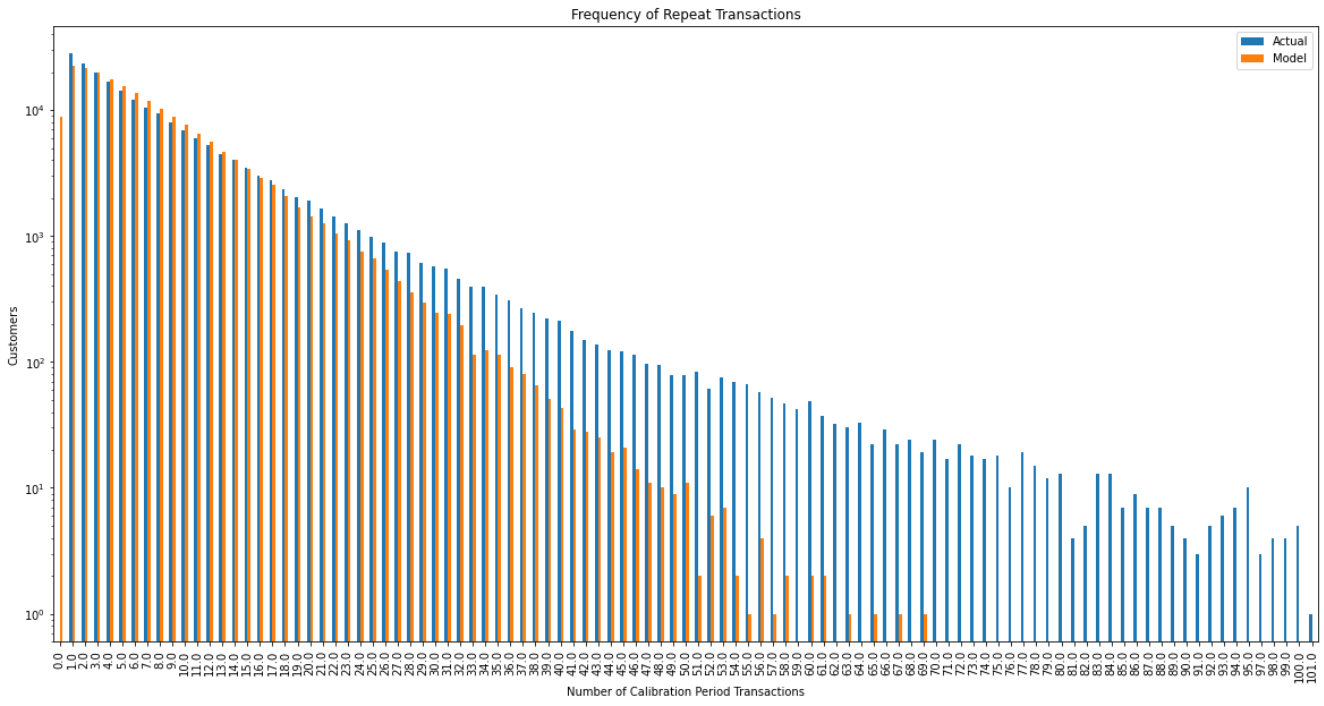
```
from lifetimes import BetaGeoFitter
bgf = BetaGeoFitter()
bgf.fit(returning_customers_summary['frequency'], returning_customers_summary['recency'],
returning_customers_summary['T'])
bgf
```

Out[216]:

```
<lifetimes.BetaGeoFitter: fitted with 200592 subjects, a: 16.75, alpha: 63.45, b: 268.28, r: 1.61>
```

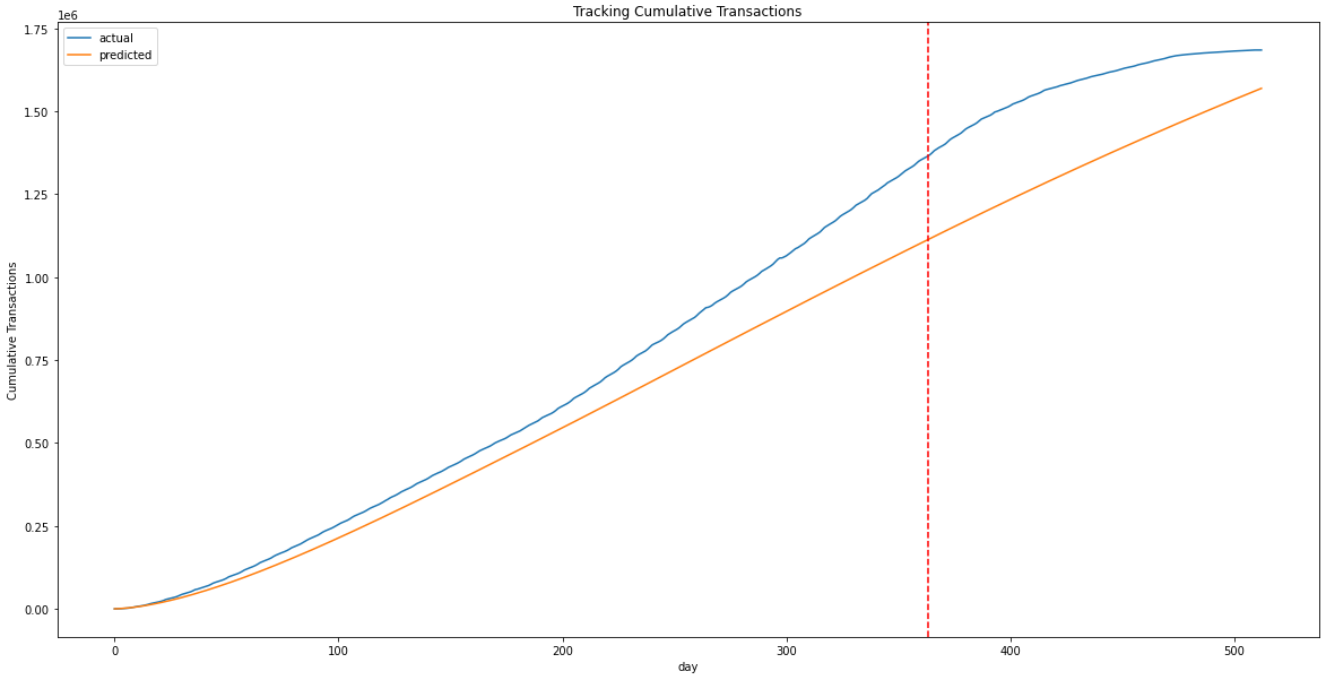
In [217]:

```
from lifetimes.plotting import plot_period_transactions
plt.rcParams["figure.figsize"] = (20,10)
plot_period_transactions(bgf, max_frequency=100).set_yscale('log')
```



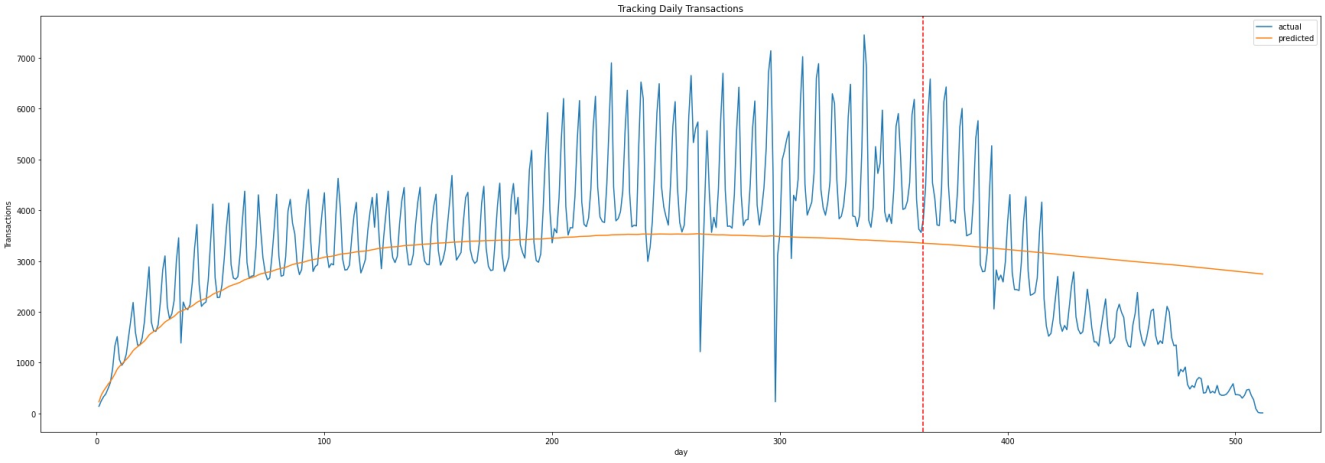
In [218]:

```
from lifetimes.plotting import plot_cumulative_transactions
plt.rcParams["figure.figsize"] = (20,10)
plot_cumulative_transactions(bgf, df1, 'date', 'id', 513, 363, freq='D');
```



In [219]:

```
from lifetimes.plotting import plot_incremental_transactions
plt.rcParams["figure.figsize"] = (30,10)
plot_incremental_transactions(bgf, df1, 'date', 'id', 513, 363, freq='D');
```

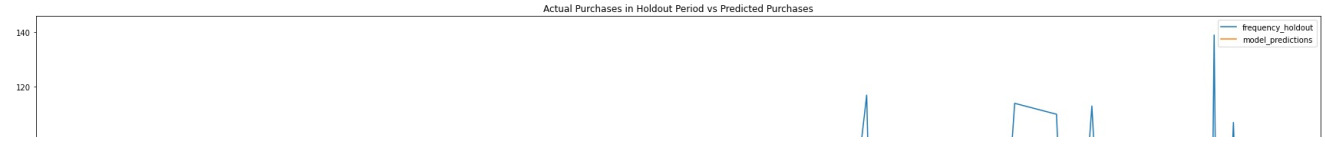


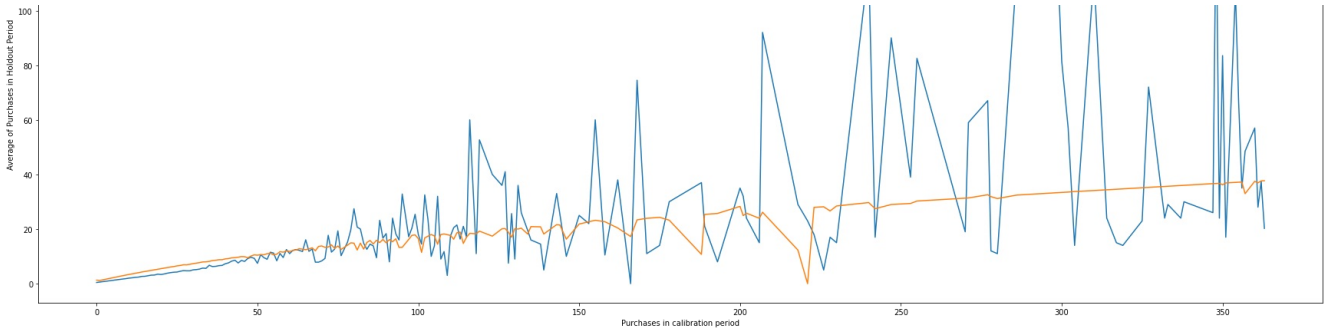
In [220]:

```
from lifetimes.plotting import plot_calibration_purchases_vs_holdout_purchases
plot_calibration_purchases_vs_holdout_purchases(bgf, summary_cal_holdout, n=500)
```

Out[220]:

```
<AxesSubplot:title={'center':'Actual Purchases in Holdout Period vs Predicted Purchases'},
xlabel='Purchases in calibration period', ylabel='Average of Purchases in Holdout Period'>
```





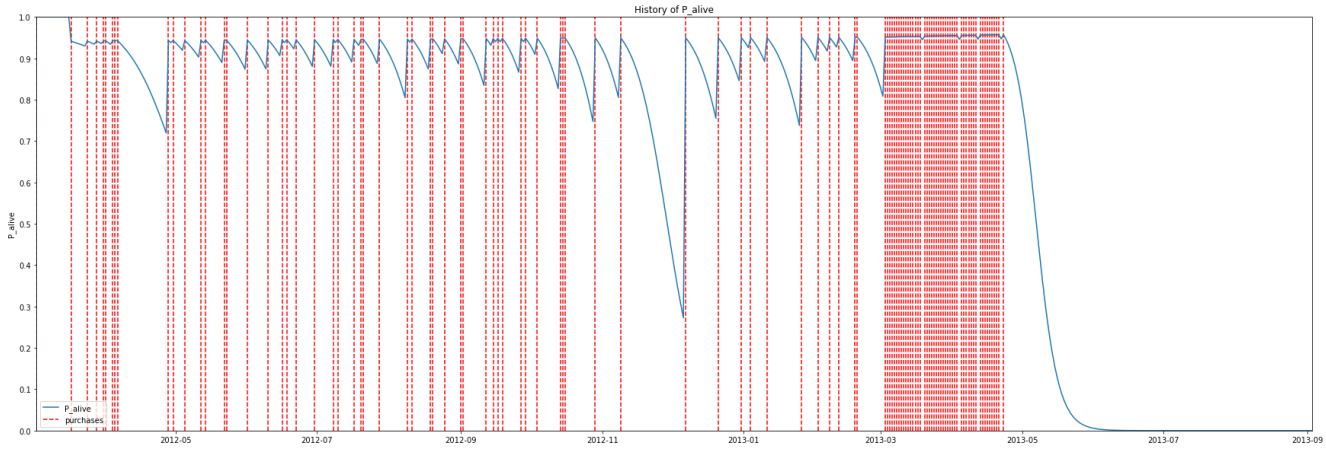
In [221]:

```
from lifetimes.plotting import plot_history_alive

customer_ID = 86246
days_since_birth = 550
sp_trans = df1.loc[df1['id'] == customer_ID]
plot_history_alive(bgf, days_since_birth, sp_trans, 'date')
```

Out[221]:

```
<AxesSubplot:title={'center':'History of P_alive'}, ylabel='P_alive'>
```



In [222]:

```
summary_with_money_value = xaction_RFM
returning_customers_summary = summary_with_money_value[summary_with_money_value['frequency']>0]
returning_customers_summary.head()
```

Out[222]:

id	frequency	recency	T	monetary_value	predicted_purchase
86246	105.0	417.0	513.0	8.578190	1.322366e-07
86252	84.0	389.0	513.0	10.165119	2.261019e-08
12262064	6.0	401.0	438.0	3.170000	3.921630e-02
12277270	24.0	458.0	508.0	7.982500	1.029548e-01
12332190	4.0	154.0	353.0	2.930000	1.201376e-02

In [223]:

```
returning_customers_summary[['monetary_value', 'frequency']].corr()
```

Out[223]:

	monetary_value	frequency
monetary_value	1.000000	0.164569

	monetary_value	frequency
monetary_value	1.000000	0.164569
frequency	0.164569	1.000000

In [ ]:

```
```

In [ ]:

```
x =
ggf.conditional_expected_average_profit(returning_customers_summary['frequency'],returning_customer
s_summary['monetary_value'])

GammaPred = pd.DataFrame(x)
GammaPred.to_csv('M_Pred_V2.csv')
```

In [153]:

```
Gamma_V = ggf.customer_lifetime_value(
    bgf,
    returning_customers_summary['frequency'],
    returning_customers_summary['recency'],
    returning_customers_summary['T'],
    returning_customers_summary['monetary_value'],
    time=150, # months
    discount_rate=0.01)

GammaPred = pd.DataFrame(Gamma_V)
GammaPred.to_csv('M_Pred_V2.csv')
```

In [160]:

```
from sklearn import metrics

xaction_RFM = xaction_RFM[xaction_RFM['frequency']>0]

# compute the R Square for model
print("R-Square:",metrics.r2_score(GammaPred[0], xaction_RFM['monetary_value']))

# calculate MAE using scikit-learn
print("MAE:",metrics.mean_absolute_error(GammaPred[0],xaction_RFM['monetary_value']))

#calculate mean squared error
print("MSE",metrics.mean_squared_error(GammaPred[0], xaction_RFM['monetary_value']))

# compute the RMSE of our predictions
print("RMSE:",np.sqrt(metrics.mean_squared_error(GammaPred[0], xaction_RFM['monetary_value'])))
```

```
R-Square: 0.9920897234035092
MAE: 0.3391864814291401
MSE 6.979518189427808
RMSE: 2.6418777771554476
```

In [ ]:

```
```