In [10]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as sf
import numpy as np
```

*We use data on credit card holders

In [11]:
```
credit = pd.read_csv(r'C:\Users\Admin\OneDrive\Рабочий стол\DataScience\
```

In [12]:
```
credit.head()
```

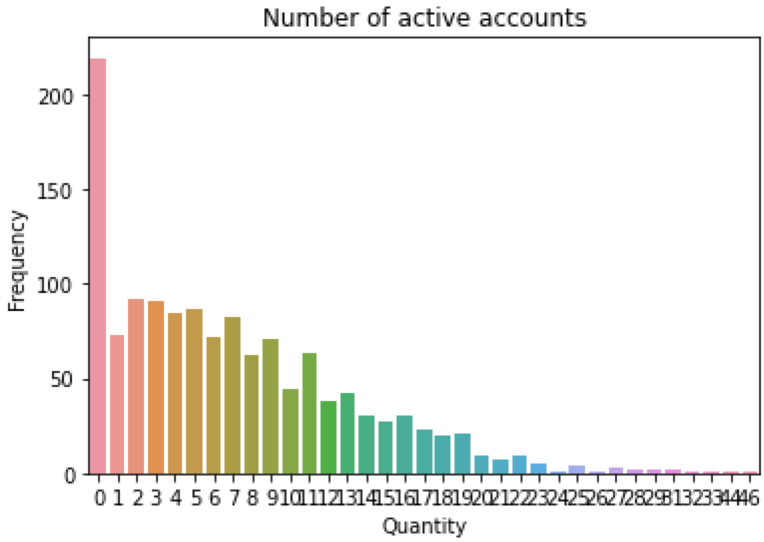
Out[12]:

	card	reports	age	income	share	expenditure	owner	selfemp	dependents	m
0	yes	0	37.66667	4.5200	0.033270	124.983300	yes	no	3	
1	yes	0	33.25000	2.4200	0.005217	9.854167	no	no	3	
2	yes	0	33.66667	4.5000	0.004156	15.000000	yes	no	4	
3	yes	0	30.50000	2.5400	0.065214	137.869200	no	no	0	
4	yes	0	32.16667	9.7867	0.067051	546.503300	yes	no	2	

*let's see the distribution of the number of active accounts

In [13]:
```
sns.countplot(x='active', data = credit)
plt.xlabel('Quantity')
plt.ylabel('Frequency')
plt.title('Number of active accounts')
```

Out[13]: Text(0.5, 1.0, 'Number of active accounts')



*Making the Poisson model

In [14]:
```
pols = sf.glm('active ~ age + income + expenditure + C(owner) + C(selfem
pols.summary()
```

Out[14]: Generalized Linear Model Regression Results

Dep. Variable:	active	**No. Observations:**	1319
Model:	GLM	**Df Residuals:**	1313
Model Family:	Poisson	**Df Model:**	5
Link Function:	log	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-5614.4
Date:	Thu, 28 Oct 2021	**Deviance:**	7142.4
Time:	01:57:29	**Pearson chi2:**	6.89e+03
No. Iterations:	5		
Covariance Type:	nonrobust		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	1.4000	0.037	37.440	0.000	1.327	1.473
C(owner)[T.yes]	0.4033	0.023	17.409	0.000	0.358	0.449
C(selfemp)[T.yes]	0.0141	0.040	0.355	0.723	-0.064	0.092
age	0.0062	0.001	5.648	0.000	0.004	0.008
income	0.0361	0.006	5.752	0.000	0.024	0.048
expenditure	2.925e-05	3.75e-05	0.779	0.436	-4.43e-05	0.000

According to the model result that "expenditure" as well as "selfemp" don't affect the number of active accounts (p-value). But "income", "age" and "owner"(so-called possession of property) have an influence on a number of active accounts.

Unfortunately, the Poisson model exerts problem with an overdispersion that is able significantly to distort outcomes

Let's count the dispersion model

In [15]:
```
pols.pearson_chi2/pols.df_resid
```

Out[15]: 5.249939996924244

For the Poisson model, this value should be close to 1. A different distribution is needed

Negative binomial distribution

Spread from (0; + infinity) u - average, 0 / a (shape / variance)and alpha is considered as a dispersion indicator. It affects final result coincidence that why is highly recommended to set it from (0,1 up to 2) to decrease the standard error.

In [16]:
```
neg = sf.glm('active ~ age + income + expenditure + C(owner) + C(selfemp
             family = sm.families.NegativeBinomial(alpha = 0.15)).fit()
neg.summary()
```

Out[16]: Generalized Linear Model Regression Results

Dep. Variable:	active	**No. Observations:**	1319
Model:	GLM	**Df Residuals:**	1313
Model Family:	NegativeBinomial	**Df Model:**	5

Link Function:	log	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-4429.3
Date:	Thu, 28 Oct 2021	**Deviance:**	3940.9
Time:	01:57:40	**Pearson chi2:**	3.44e+03
No. Iterations:	5		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	1.3690	0.054	25.206	0.000	1.263	1.475
C(owner)[T.yes]	0.3984	0.033	11.996	0.000	0.333	0.463
C(selfemp)[T.yes]	0.0055	0.059	0.094	0.925	-0.110	0.121
age	0.0069	0.002	4.263	0.000	0.004	0.010
income	0.0393	0.010	4.099	0.000	0.021	0.058
expenditure	2.445e-05	5.63e-05	0.434	0.664	-8.59e-05	0.000

*Compared to the Poison model we can see that data changed a bit but interpretation left the same. What about the current dispersion?

In [17]:
```
neg.pearson_chi2/neg.df_resid
```

Out[17]:
```
2.619678072487012
```

Let's compare the models using the Akaike information criterion (AIC)

The lower the criterion, the better the model.

In [18]:
```
print(pols.aic)
print(neg.aic)
```

```
11240.817775454601
8870.632884339047
```

Negative binomial distribution does much better than Poisson

Another well-known model in data analysis is zero-inflated Poisson model

In [19]:
```
credit.owner = np.where(credit.owner == 'yes', 1, 0)
credit.selfemp = np.where(credit.selfemp == 'yes', 1, 0) #change the dat
Y = credit.active #Exogenous variable
X= credit.loc[:,['owner', 'selfemp', 'age', 'income', 'expenditure']] # 
X= sm.add_constant(X) # add a constant so that the model has an intercep
```

In [20]:
```
zeroinf = sm.ZeroInflatedPoisson(Y, X).fit(maxiter = 72, method ='ncg')
zeroinf.summary()
```

```
C:\Users\Admin\anaconda3\lib\site-packages\statsmodels\base\model.py:566:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. C
heck mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
Optimization terminated successfully.
         Current function value: 3.899929
         Iterations: 4
```

```
         Function evaluations: 6
         Gradient evaluations: 6
         Hessian evaluations: 4
```

Out[20]:

ZeroInflatedPoisson Regression Results

Dep. Variable:	active	**No. Observations:**	1319
Model:	ZeroInflatedPoisson	**Df Residuals:**	1313
Method:	MLE	**Df Model:**	5
Date:	Thu, 28 Oct 2021	**Pseudo R-squ.:**	-0.05893
Time:	01:57:59	**Log-Likelihood:**	-5144.0
converged:	True	**LL-Null:**	-4857.8
Covariance Type:	nonrobust	**LLR p-value:**	1.000

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
inflate_const	0.0999	0.055	1.812	0.070	-0.008	0.208
const	1.7967	0.037	48.172	0.000	1.724	1.870
owner	0.0017	0.023	0.073	0.942	-0.043	0.046
selfemp	0.0008	0.039	0.022	0.983	-0.075	0.077
age	0.0093	0.001	8.656	0.000	0.007	0.011
income	0.0019	0.007	0.280	0.779	-0.011	0.015
expenditure	0.0001	3.81e-05	2.691	0.007	2.79e-05	0.000

This model presented absolutely different output comparred to the other abovementioned approaches. According to p-value indicator we can deduce that 'income', 'selfemp', 'owner' do not affect any more but only 'age' and 'expenditure' have influence on number of active accounts.

In [21]:

```
print(pols.aic)
print(neg.aic)
print(zeroinf.aic)
```

```
11240.817775454601
8870.632884339047
10300.01303290444
```

The last model will be the zero-inflated Negative Binomial method:

In [22]:

```
zeroinf_2 = sm.ZeroInflatedNegativeBinomialP(Y, X).fit(maxiter = 50, met
zeroinf_2.summary()
```

```
Optimization terminated successfully.
         Current function value: 3.055914
         Iterations: 5
         Function evaluations: 9
         Gradient evaluations: 9
         Hessian evaluations: 5
```

Out[22]:

ZeroInflatedNegativeBinomialP Regression Results

Dep. Variable:	active	**No. Observations:**	1319
Model:	ZeroInflatedNegativeBinomialP	**Df Residuals:**	1313
Method:	MLE	**Df Model:**	5
Date:	Thu, 28 Oct 2021	**Pseudo R-squ.:**	-0.03018

Time:	01:58:12	**Log-Likelihood:**	-4030.7
converged:	True	**LL-Null:**	-3912.7
Covariance Type:	nonrobust	**LLR p-value:**	1.000

	coef	std err	z	P>|z|	[0.025	0.975]
inflate_const	-0.5152	0.058	-8.820	0.000	-0.630	-0.401
const	1.4543	0.087	16.765	0.000	1.284	1.624
owner	0.3590	0.050	7.112	0.000	0.260	0.458
selfemp	0.0029	0.090	0.032	0.974	-0.173	0.178
age	0.0097	0.003	3.757	0.000	0.005	0.015
income	0.0352	0.016	2.234	0.026	0.004	0.066
expenditure	5.109e-05	9.08e-05	0.563	0.573	-0.000	0.000
alpha	0.4517	0.032	14.330	0.000	0.390	0.514

Here the interpretation is similar to the original one. There is also the "alpha" parameter, which evaluates the excess of variance.

In [23]:

```
print(pols.aic)
print(neg.aic)
print(zeroinf.aic)
print(zeroinf_2.aic)
```

```
11240.817775454601
8870.632884339047
10300.01303290444
8073.499871710537
```

Zero-inflated Negative Binomial is the best one in comparison with the others only in this particular case.