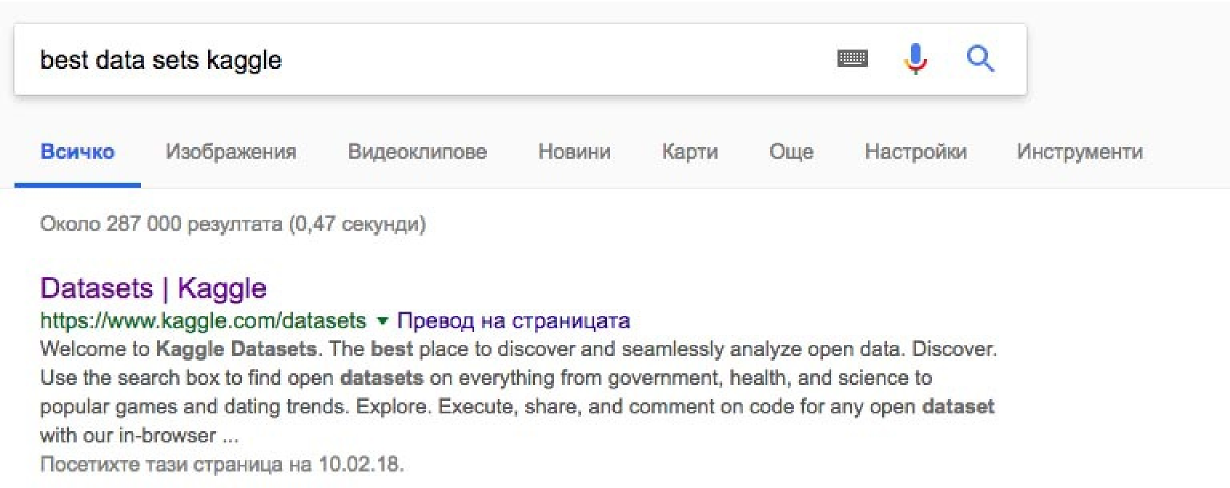
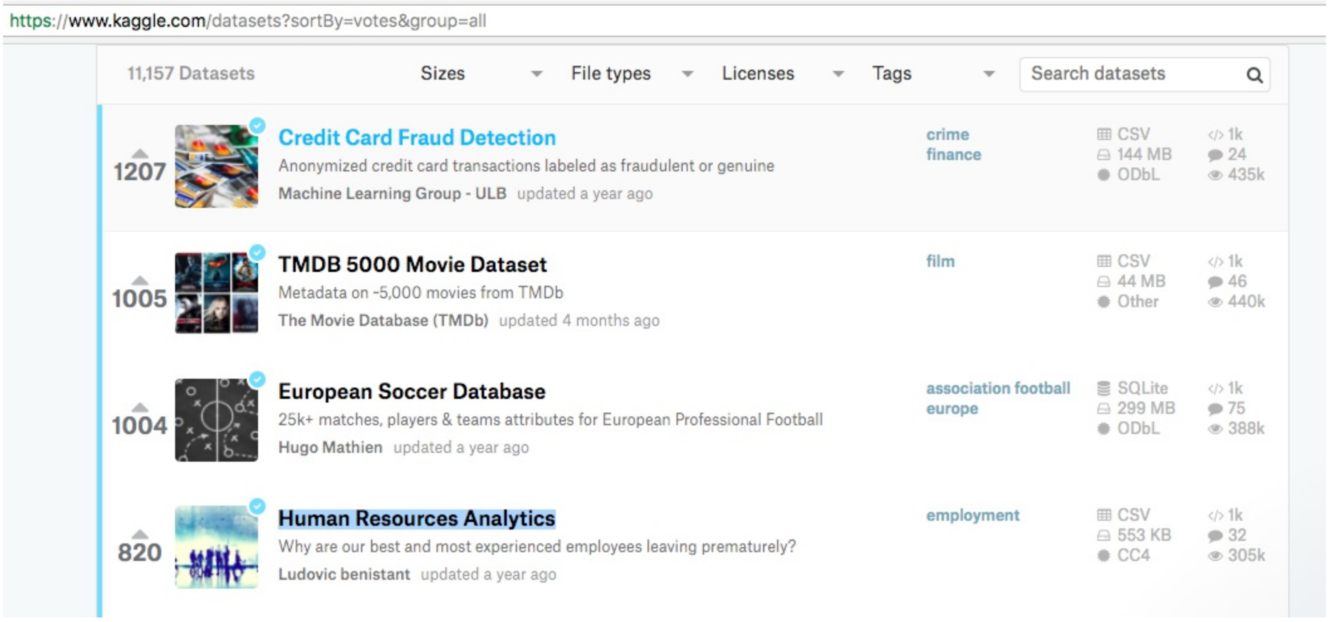
Как започна всичко





Избрахме data set, сега да разгледаме какво съдържа...

Полетата в набора от данни включват:

- Ниво на удовлетвореност
- Последна оценка
- Брой проекти
- Средни месечни часове
- Времето, прекарано в компанията
- Дали са имали трудова злополука
- Дали са имали повишение през последните 5 години
- Отдел (column sales)
- Заплата
- Дали служителят е напуснал

За реализацията на проекта ще използвам python и блиблиотеки като pandas, nupy, sklearn и други

```
pip install numpy scipy matplotlib ipython scikit-learn pandas pillow mglearn jupyter
```

In [42]:

```
import sys

import sklearn
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import mglearn
import seaborn as sns
from IPython.display import display
from sklearn.model_selection import train_test_split


%matplotlib inline
# pd.options.display.max_rows = 15
pd.options.mode.chained_assignment = None  # default='warn'
# pd.options.mode.chained_assignment = 'warn'  # default=
```

Нека да заредим данните и да ги разделим на train и validate множества

In [43]:

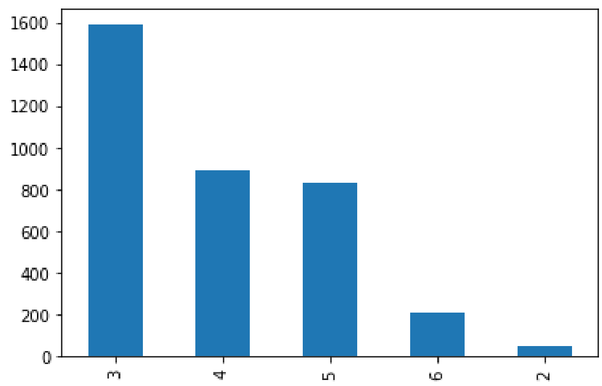
```
data = pd.read_csv('HR_comma_sep.csv')
data_no_left = data.drop(labels=['left'], axis=1)
data_left = data['left']
x_train, x_val, y_train, y_val = train_test_split(data_no_left, data_left, test_size=0.15, random_state
=4330)
print("Размери на всичките данни",data.shape)
print("Размери на train",x_train.shape)
print("Размери на validate",x_val.shape)
```

```
Размери на всичките данни (14999, 10)
Размери на train (12749, 9)
Размери на validate (2250, 9)
```

За начало да видим колко време средно прекарват служителите в компанията

In [44]:

```
data[data.left == True].time_spend_company.value_counts().plot(kind='bar');
```



Вижда се, че най-често хората напускат след 3-та си година в компанията

In [45]:

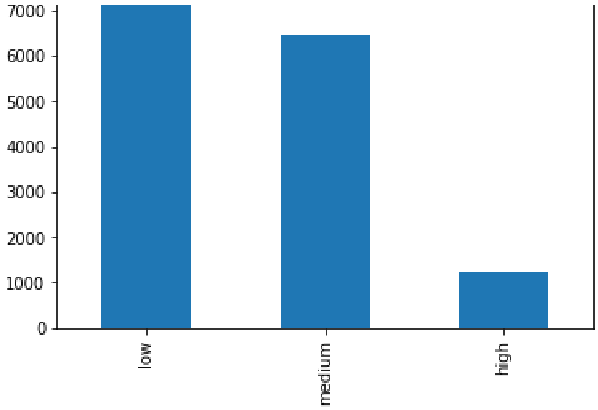
```
data.dtypes[data.dtypes == 'object']
```

Out[45]:

```
sales      object
salary     object
dtype: object
```

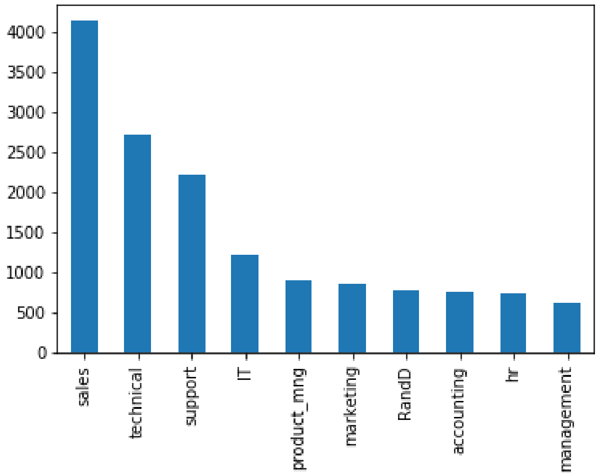
In [46]:

```
data.salary.value_counts().plot(kind='bar');
```



In [47]:

```
data.sales.value_counts().plot(kind='bar');
```



Виждаме 2 категориини колони, които биха могли да направят проблем при по-нататъчна работа с данните. Ще направим One Hot Encoding

In [48]:

```
encoded_x_train = x_train
encoded_x_train['low_salary'] = (x_train.salary == 'low').astype(float)
encoded_x_train['medium_salary'] = (x_train.salary == 'medium').astype(float)
encoded_x_train['high_salary'] = (x_train.salary == 'high').astype(float)
encoded_x_train = encoded_x_train.drop('salary', axis=1)

encoded_x_train['sales_sales'] = (x_train.sales == 'sales').astype(float)
encoded_x_train['technical_sales'] = (x_train.sales == 'technical').astype(float)
encoded_x_train['support_sales'] = (x_train.sales == 'support').astype(float)
encoded_x_train['IT_sales'] = (x_train.sales == 'IT').astype(float)
encoded_x_train['product_mng_sales'] = (x_train.sales == 'product_mng').astype(float)
encoded_x_train['marketing_sales'] = (x_train.sales == 'marketing_sales').astype(float)
encoded_x_train['RandD_sales'] = (x_train.sales == 'RandD').astype(float)
encoded_x_train['accounting_sales'] = (x_train.sales == 'accounting').astype(float)
encoded_x_train['hr_sales'] = (x_train.sales == 'hr').astype(float)
encoded_x_train['management_sales'] = (x_train.sales == 'management').astype(float)
encoded_x_train = encoded_x_train.drop('sales', axis=1)
```

In [49]:

```
encoded_x_val = x_val
encoded_x_val['low_salary'] = (x_val.salary == 'low').astype(float)
encoded_x_val['medium_salary'] = (x_val.salary == 'medium').astype(float)
encoded_x_val['high_salary'] = (x_val.salary == 'high').astype(float)
encoded_x_val = encoded_x_val.drop('salary', axis=1)


encoded_x_val['sales_sales'] = (encoded_x_val.sales == 'sales').astype(float)
encoded_x_val['technical_sales'] = (encoded_x_val.sales == 'technical').astype(float)
encoded_x_val['support_sales'] = (encoded_x_val.sales == 'support').astype(float)
encoded_x_val['IT_sales'] = (encoded_x_val.sales == 'IT').astype(float)
encoded_x_val['product_mng_sales'] = (encoded_x_val.sales == 'product_mng').astype(float)
```

```
encoded_x_val['marketing_sales'] = (encoded_x_val.sales == 'marketing_sales').astype(float)
encoded_x_val['RandD_sales'] = (encoded_x_val.sales == 'RandD').astype(float)
encoded_x_val['accounting_sales'] = (encoded_x_val.sales == 'accounting').astype(float)
encoded_x_val['hr_sales'] = (encoded_x_val.sales == 'hr').astype(float)
encoded_x_val['management_sales'] = (encoded_x_val.sales == 'management').astype(float)
encoded_x_val = encoded_x_val.drop('sales', axis=1)
```

In [50]:

```
encoded_x_train
```

Out[50]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accider
1397	0.43	0.53	2	146	3	0
11702	0.87	0.90	3	174	2	0
1485	0.11	0.88	7	253	4	0
10061	0.96	0.95	6	215	4	0
7964	0.91	0.61	3	255	3	0
10843	0.95	0.62	4	150	2	0
14787	0.48	0.78	2	198	2	0
5527	0.61	0.61	4	239	2	0
3459	0.63	0.62	5	212	6	0
9302	0.76	0.80	3	202	3	0
13500	0.98	0.55	3	166	6	1
13358	0.57	0.59	4	250	2	0
8259	0.94	0.78	3	218	2	1
10809	0.51	0.74	6	98	3	0
6192	0.98	0.63	3	135	3	0
4933	0.96	0.93	4	260	3	0
2249	0.90	0.48	4	204	3	0
149	0.39	0.50	2	147	3	0
8609	0.58	0.60	4	147	3	0
10846	0.48	0.51	3	136	3	0
10811	0.97	0.93	5	137	2	1
6017	0.82	0.59	3	178	2	0
1666	0.41	0.51	2	159	3	0
3712	0.94	0.95	4	155	3	0
1760	0.37	0.55	2	140	3	0
3276	0.93	0.65	4	212	4	0
6002	0.79	0.86	3	126	5	0
8356	0.95	0.43	6	283	2	0
4451	0.87	0.68	5	187	3	0
10743	0.61	0.73	3	252	3	0
...	...	...	...	...	...	...
4020	0.83	0.69	4	151	2	0
3335	0.74	0.85	5	135	2	0
1519	0.40	0.55	2	131	3	0
2329	0.66	0.77	2	171	2	0
5130	0.49	0.54	6	214	3	0

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_acciden
7723	1.00	0.84	3	215	2	0
7815	0.98	0.69	3	274	3	0
13293	0.74	0.73	3	156	8	0
7761	0.49	0.79	5	206	2	0
4865	0.85	0.79	3	217	2	0
10676	0.96	0.70	4	272	3	0
13092	0.50	0.95	5	137	3	0
2957	0.75	0.70	5	269	3	0
13957	0.67	0.49	3	247	10	0
6811	0.32	0.45	2	188	3	0
13584	0.42	0.45	3	227	3	0
759	0.41	0.55	2	151	3	0
8363	0.12	0.59	3	229	6	0
6922	0.48	0.41	5	286	3	0
735	0.83	0.99	5	258	5	0
3828	0.58	0.79	5	262	2	0
9559	0.15	0.95	4	173	5	1
11400	0.15	0.75	3	150	4	0
7002	0.68	0.75	5	243	3	1
3611	0.60	0.99	4	225	3	0
8987	0.89	1.00	4	226	2	1
831	0.73	1.00	5	274	5	0
78	0.43	0.56	2	157	3	0
4152	0.27	0.47	5	217	6	0

12749 rows × 20 columns

Ще пробвам няколко алгоритъма за машинно самообучение и ще видим как се представят

За начало - Логистична Регресия

In [51]:

```
from sklearn.linear_model import LogisticRegression
logReg = LogisticRegression().fit(encoded_x_train, y_train)

print("Резултат при трениране: {:.3f}".format(logReg.score(encoded_x_train, y_train)))
print("Резултат при тест:     {:.3f}".format(logReg.score(encoded_x_val, y_val)))
```

```
Резултат при трениране: 0.788
Резултат при тест:      0.799
```

И крос валидация

In [52]:

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score

# scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
# scores = cross_validate(logReg, encoded_x_train, y_train, scoring=scoring,cv=5, return_train_score=Fa
lse)

scores = cross_validate(LogisticRegression(), encoded_x_train, y_train, cv=5)
pd.DataFrame(scores)
```

Out[52]:

	fit_time	score_time	test_score	train_score
0	0.031227	0.001027	0.792941	0.787724
1	0.031315	0.000896	0.790196	0.789489
2	0.033378	0.001130	0.786667	0.783508
3	0.029690	0.001150	0.789020	0.789685
4	0.030631	0.001299	0.774421	0.790588

Не много добре, нека да регуляризираме.

In [53]:

```
scores = cross_validate(LogisticRegression(C=100), encoded_x_train, y_train, cv=5)
pd.DataFrame(scores)
```

Out[53]:

	fit_time	score_time	test_score	train_score
0	0.029225	0.001020	0.793333	0.787626
1	0.032824	0.000842	0.792157	0.789783
2	0.027321	0.000847	0.785098	0.783606
3	0.026708	0.001013	0.788627	0.790470
4	0.029726	0.000862	0.774421	0.790980

In [54]:

```
from sklearn.model_selection import GridSearchCV
search = GridSearchCV(LogisticRegression(), {'C': [1,10, 30, 50, 70, 100]})
search.fit(encoded_x_train, y_train)
pd.DataFrame(search.cv_results_)
```

Out[54]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	params	rank_test_score	split0_test
0	0.025445	0.002432	0.785944	0.788689	1	{'C': 1}	1	0.789647
1	0.024058	0.001356	0.785709	0.789317	10	{'C': 10}	2	0.789647
2	0.025343	0.001552	0.785552	0.789356	30	{'C': 30}	3	0.789647
3	0.025672	0.001377	0.785552	0.789356	50	{'C': 50}	3	0.789647
4	0.024519	0.001278	0.785552	0.789434	70	{'C': 70}	3	0.789647
5	0.024974	0.001288	0.785473	0.789434	100	{'C': 100}	6	0.789647

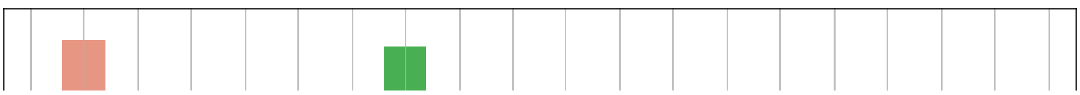
Няма положителна разлика.

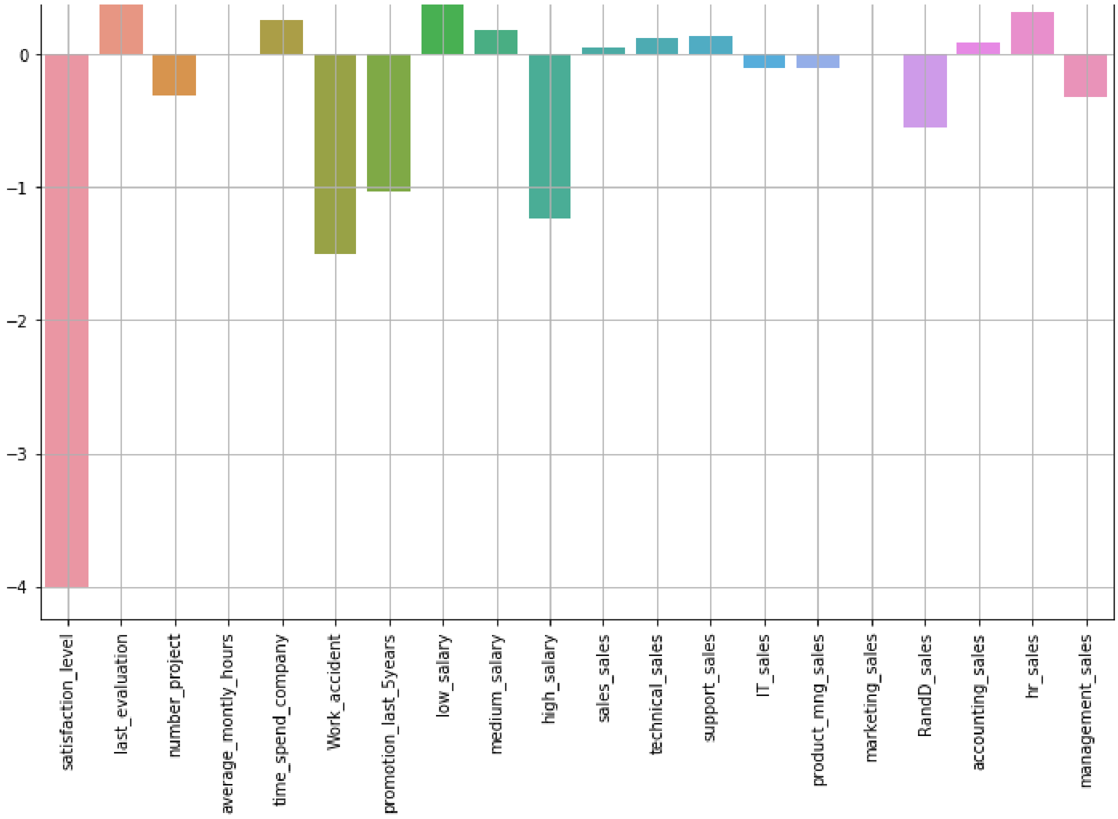
Нека да видим какви тегла е открил модела

In [55]:

```
def logistic_regression_features(X, model):
    plt.figure(figsize=(12,8))
    barplot = sns.barplot(x=X.columns, y=model.coef_[0], orient='vertical')
    plt.setp(barplot.get_xticklabels(), rotation=90)
    plt.grid(True)

logistic_regression_features(encoded_x_train, logReg)
```





Логично, нивото на удволетвореност указва най-голямо влияние. Следвао от инцидентите по време на работа, заплатата и от това дали работника е получавал повишение на скоро.

In [56]:

```
# logistic_regression_features(encoded_x_train.drop(labels=['satisfaction_level'], axis=1), logReg)
```

Нека да пробваме с друго - Decision Tree

In [57]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
scores = cross_validate(DecisionTreeClassifier(), encoded_x_train, y_train, cv=5)
pd.DataFrame(scores)
```

Out[57]:

	fit_time	score_time	test_score	train_score
0	0.034359	0.001546	0.974510	1.0
1	0.030791	0.001084	0.976078	1.0
2	0.028128	0.001079	0.975686	1.0
3	0.025988	0.001054	0.978431	1.0
4	0.029016	0.001113	0.976069	1.0

А с Random Forest?

In [58]:

```
from sklearn.ensemble import RandomForestClassifier
scores = cross_validate(RandomForestClassifier(), encoded_x_train, y_train, cv=5)
pd.DataFrame(scores)
```

Out[58]:

	fit_time	score_time	test_score	train_score
0	0.073817	0.006235	0.989412	0.997843

	fit_time	score_time	test_score	train_score
1	0.066990	0.005073	0.987451	0.998039
2	0.072401	0.004370	0.987059	0.997941
3	0.061216	0.005023	0.987451	0.998921
4	0.064365	0.005101	0.983523	0.998137

In [59]:

```
randF = RandomForestClassifier().fit(encoded_x_train, y_train)
print("Резултат при трениране: {:.3f}".format(randF.score(encoded_x_train, y_train)))
print("Резултат при тест:     {:.3f}".format(randF.score(encoded_x_val, y_val)))
```

```
Резултат при трениране: 0.998
Резултат при тест:     0.990
```

Много добре, а със Support Vector Machine

In [60]:

```
from sklearn.svm import LinearSVC
from sklearn.svm import SVC

scores = cross_validate(SVC(), encoded_x_train, y_train, cv=5)
pd.DataFrame(scores)
```

Out[60]:

	fit_time	score_time	test_score	train_score
0	2.071727	0.202835	0.952157	0.953917
1	1.838029	0.194275	0.946667	0.956760
2	2.197947	0.264839	0.951765	0.956074
3	1.974786	0.218876	0.955294	0.953721
4	2.018188	0.197815	0.944292	0.955392

С регуляризация

In [61]:

```
scores = cross_validate(SVC(C=10, gamma=0.1), encoded_x_train, y_train, cv=5)
pd.DataFrame(scores)
```

Out[61]:

	fit_time	score_time	test_score	train_score
0	1.852807	0.177478	0.970196	0.987450
1	1.807484	0.155460	0.967843	0.988528
2	2.048003	0.199670	0.960000	0.989901
3	1.876012	0.185310	0.967843	0.988038
4	1.832783	0.169145	0.965477	0.989314

Добре, като за последно да пробваме и с невронни мрежи.

In [71]:

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop, Adam
```

Ще ползвам Sequential на kerass с активираща фунция [Relu]() и [Softmax]() за поседния слой

In [72]:

```
NNmodel = Sequential()
```

```
NNmodel = Sequential()
NNmodel.add(Dense(64, input_dim=encoded_x_train.shape[1], activation='relu'))
NNmodel.add(Dropout(0.5))
NNmodel.add(Dense(64, activation='relu'))
NNmodel.add(Dropout(0.5))
NNmodel.add(Dense(1, activation='sigmoid'))

NNmodel.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

NNmodel.fit(encoded_x_train, y_train,
          epochs=20,
          batch_size=128)
score = NNmodel.evaluate(encoded_x_val, y_val, batch_size=128)
pd.DataFrame(score)
```

```
Epoch 1/20
12749/12749 [==============================] - 1s 52us/step - loss: 3.9033 - acc: 0.7197
Epoch 2/20
12749/12749 [==============================] - 0s 28us/step - loss: 3.5837 - acc: 0.7370
Epoch 3/20
12749/12749 [==============================] - 0s 28us/step - loss: 2.0161 - acc: 0.6662
Epoch 4/20
12749/12749 [==============================] - 0s 27us/step - loss: 0.6974 - acc: 0.7241
Epoch 5/20
12749/12749 [==============================] - 0s 27us/step - loss: 0.5923 - acc: 0.7615
Epoch 6/20
12749/12749 [==============================] - 0s 34us/step - loss: 0.5809 - acc: 0.7630
Epoch 7/20
12749/12749 [==============================] - 0s 31us/step - loss: 0.5720 - acc: 0.7637
Epoch 8/20
12749/12749 [==============================] - 0s 27us/step - loss: 0.5711 - acc: 0.7641
Epoch 9/20
12749/12749 [==============================] - 0s 28us/step - loss: 0.5668 - acc: 0.7642
Epoch 10/20
12749/12749 [==============================] - 0s 28us/step - loss: 0.5622 - acc: 0.7643
Epoch 11/20
12749/12749 [==============================] - 0s 27us/step - loss: 0.5567 - acc: 0.7643
Epoch 12/20
12749/12749 [==============================] - 0s 27us/step - loss: 0.5537 - acc: 0.7643
Epoch 13/20
12749/12749 [==============================] - 0s 27us/step - loss: 0.5472 - acc: 0.7643
Epoch 14/20
12749/12749 [==============================] - 0s 26us/step - loss: 0.5241 - acc: 0.7643
Epoch 15/20
12749/12749 [==============================] - 0s 27us/step - loss: 0.4672 - acc: 0.7644
Epoch 16/20
12749/12749 [==============================] - 0s 27us/step - loss: 0.4297 - acc: 0.7663
Epoch 17/20
12749/12749 [==============================] - 0s 27us/step - loss: 0.4071 - acc: 0.7721
Epoch 18/20
12749/12749 [==============================] - 0s 27us/step - loss: 0.3912 - acc: 0.7835
Epoch 19/20
12749/12749 [==============================] - 0s 27us/step - loss: 0.3786 - acc: 0.7979
Epoch 20/20
12749/12749 [==============================] - 0s 28us/step - loss: 0.3629 - acc: 0.8080
2250/2250 [==============================] - 0s 26us/step
```

Out[72]:

	0
0	0.325747
1	0.885333

С друга архитектура

In [73]:

```
NNmodel = Sequential()
NNmodel.add(Dense(128, input_dim=encoded_x_train.shape[1], activation='relu'))
NNmodel.add(Dropout(0.5))
NNmodel.add(Dense(128, activation='relu'))
NNmodel.add(Dropout(0.5))
NNmodel.add(Dense(128, activation='relu'))
```

```python
NNmodel.add(Dropout(0.5))
NNmodel.add(Dense(1, activation='sigmoid'))

NNmodel.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

NNmodel.fit(encoded_x_train, y_train,
          epochs=20,
          batch_size=128)
score = NNmodel.evaluate(encoded_x_val, y_val, batch_size=128)
```

```
Epoch 1/20
12749/12749 [==============================] - 1s 76us/step - loss: 3.6353 - acc: 0.7381
Epoch 2/20
12749/12749 [==============================] - 1s 39us/step - loss: 1.8921 - acc: 0.6673
Epoch 3/20
12749/12749 [==============================] - 1s 51us/step - loss: 0.7266 - acc: 0.6986
Epoch 4/20
12749/12749 [==============================] - 1s 42us/step - loss: 0.6063 - acc: 0.7512
Epoch 5/20
12749/12749 [==============================] - 1s 42us/step - loss: 0.5847 - acc: 0.7622
Epoch 6/20
12749/12749 [==============================] - 1s 40us/step - loss: 0.5729 - acc: 0.7641
Epoch 7/20
12749/12749 [==============================] - 1s 41us/step - loss: 0.5680 - acc: 0.7641
Epoch 8/20
12749/12749 [==============================] - 1s 41us/step - loss: 0.5650 - acc: 0.7643
Epoch 9/20
12749/12749 [==============================] - 1s 40us/step - loss: 0.5618 - acc: 0.7643
Epoch 10/20
12749/12749 [==============================] - 1s 41us/step - loss: 0.5560 - acc: 0.7643
Epoch 11/20
12749/12749 [==============================] - 1s 41us/step - loss: 0.5391 - acc: 0.7643
Epoch 12/20
12749/12749 [==============================] - 1s 43us/step - loss: 0.4624 - acc: 0.7643
Epoch 13/20
12749/12749 [==============================] - 1s 41us/step - loss: 0.4045 - acc: 0.7703
Epoch 14/20
12749/12749 [==============================] - 1s 42us/step - loss: 0.3805 - acc: 0.8016
Epoch 15/20
12749/12749 [==============================] - 1s 41us/step - loss: 0.3675 - acc: 0.8198
Epoch 16/20
12749/12749 [==============================] - 1s 46us/step - loss: 0.3466 - acc: 0.8443
Epoch 17/20
12749/12749 [==============================] - 1s 43us/step - loss: 0.3408 - acc: 0.8499
Epoch 18/20
12749/12749 [==============================] - 1s 42us/step - loss: 0.3329 - acc: 0.8593
Epoch 19/20
12749/12749 [==============================] - 1s 42us/step - loss: 0.3230 - acc: 0.8635
Epoch 20/20
12749/12749 [==============================] - 1s 48us/step - loss: 0.3130 - acc: 0.8695
2250/2250 [==============================] - 0s 49us/step
```

С повече епохи

In [74]:

```python
NNmodel = Sequential()
NNmodel.add(Dense(128, input_dim=encoded_x_train.shape[1], activation='relu'))
NNmodel.add(Dropout(0.5))
NNmodel.add(Dense(128, activation='relu'))
NNmodel.add(Dropout(0.5))
NNmodel.add(Dense(128, activation='relu'))
NNmodel.add(Dropout(0.5))
NNmodel.add(Dense(1, activation='sigmoid'))

NNmodel.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

NNmodel.fit(encoded_x_train, y_train,
          epochs=50,
          batch_size=128)
score = NNmodel.evaluate(encoded_x_val, y_val, batch_size=128)
```

```
Epoch 1/50
```

```
12749/12749 [==============================] - 1s 76us/step - loss: 3.3667 - acc: 0.7093
Epoch 2/50
12749/12749 [==============================] - 1s 42us/step - loss: 1.2866 - acc: 0.6612
Epoch 3/50
12749/12749 [==============================] - 1s 39us/step - loss: 0.6568 - acc: 0.7202
Epoch 4/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.5970 - acc: 0.7582
Epoch 5/50
12749/12749 [==============================] - 1s 43us/step - loss: 0.5780 - acc: 0.7637
Epoch 6/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.5754 - acc: 0.7638
Epoch 7/50
12749/12749 [==============================] - 1s 43us/step - loss: 0.5657 - acc: 0.7643
Epoch 8/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.5619 - acc: 0.7643
Epoch 9/50
12749/12749 [==============================] - 1s 42us/step - loss: 0.5592 - acc: 0.7643
Epoch 10/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.5549 - acc: 0.7643
Epoch 11/50
12749/12749 [==============================] - 1s 44us/step - loss: 0.5421 - acc: 0.7643
Epoch 12/50
12749/12749 [==============================] - 1s 46us/step - loss: 0.4762 - acc: 0.7643
Epoch 13/50
12749/12749 [==============================] - 1s 46us/step - loss: 0.4173 - acc: 0.7656
Epoch 14/50
12749/12749 [==============================] - 1s 44us/step - loss: 0.3954 - acc: 0.7796
Epoch 15/50
12749/12749 [==============================] - 1s 44us/step - loss: 0.3706 - acc: 0.8116
Epoch 16/50
12749/12749 [==============================] - 1s 44us/step - loss: 0.3612 - acc: 0.8278
Epoch 17/50
12749/12749 [==============================] - 1s 46us/step - loss: 0.3538 - acc: 0.8434
Epoch 18/50
12749/12749 [==============================] - 1s 42us/step - loss: 0.3402 - acc: 0.8496
Epoch 19/50
12749/12749 [==============================] - 0s 38us/step - loss: 0.3300 - acc: 0.8551
Epoch 20/50
12749/12749 [==============================] - 1s 44us/step - loss: 0.3253 - acc: 0.8650
Epoch 21/50
12749/12749 [==============================] - 1s 44us/step - loss: 0.3228 - acc: 0.8631
Epoch 22/50
12749/12749 [==============================] - 1s 44us/step - loss: 0.3099 - acc: 0.8747
Epoch 23/50
12749/12749 [==============================] - 1s 42us/step - loss: 0.3110 - acc: 0.8729
Epoch 24/50
12749/12749 [==============================] - 1s 43us/step - loss: 0.3129 - acc: 0.8720
Epoch 25/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.3052 - acc: 0.8771
Epoch 26/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.3008 - acc: 0.8780
Epoch 27/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.2968 - acc: 0.8782
Epoch 28/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.3002 - acc: 0.8816
Epoch 29/50
12749/12749 [==============================] - 1s 40us/step - loss: 0.2950 - acc: 0.8846
Epoch 30/50
12749/12749 [==============================] - 1s 43us/step - loss: 0.2933 - acc: 0.8818
Epoch 31/50
12749/12749 [==============================] - 1s 42us/step - loss: 0.2939 - acc: 0.8805
Epoch 32/50
12749/12749 [==============================] - 1s 42us/step - loss: 0.2944 - acc: 0.8805
Epoch 33/50
12749/12749 [==============================] - 1s 44us/step - loss: 0.2904 - acc: 0.8848
Epoch 34/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.2895 - acc: 0.8878
Epoch 35/50
12749/12749 [==============================] - 1s 43us/step - loss: 0.2896 - acc: 0.8868
Epoch 36/50
12749/12749 [==============================] - 1s 44us/step - loss: 0.2885 - acc: 0.8882
Epoch 37/50
12749/12749 [==============================] - 1s 48us/step - loss: 0.2899 - acc: 0.8851
Epoch 38/50
12749/12749 [==============================] - 1s 43us/step - loss: 0.2844 - acc: 0.8881
Epoch 39/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.2891 - acc: 0.8879
```

```
Epoch 40/50
12749/12749 [==============================] - 1s 39us/step - loss: 0.2832 - acc: 0.8909
Epoch 41/50
12749/12749 [==============================] - 1s 42us/step - loss: 0.2824 - acc: 0.8879
Epoch 42/50
12749/12749 [==============================] - 1s 40us/step - loss: 0.2885 - acc: 0.8876
Epoch 43/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.2813 - acc: 0.8906
Epoch 44/50
12749/12749 [==============================] - 1s 42us/step - loss: 0.2852 - acc: 0.8893
Epoch 45/50
12749/12749 [==============================] - 1s 43us/step - loss: 0.2872 - acc: 0.8869
Epoch 46/50
12749/12749 [==============================] - 1s 41us/step - loss: 0.2829 - acc: 0.8913
Epoch 47/50
12749/12749 [==============================] - 1s 45us/step - loss: 0.2803 - acc: 0.8896
Epoch 48/50
12749/12749 [==============================] - 1s 46us/step - loss: 0.2772 - acc: 0.8911
Epoch 49/50
12749/12749 [==============================] - 1s 45us/step - loss: 0.2753 - acc: 0.8940
Epoch 50/50
12749/12749 [==============================] - 1s 45us/step - loss: 0.2876 - acc: 0.8900
2250/2250 [==============================] - 0s 38us/step
```