Краткий вводный обзор Python-библиотек для data science

Сафин Руслан, технический директор Byndyusoft™

@razonrus

Для кого этот доклад?

Анализ данных

Pandas

```
import pandas as pd
```

IBM HR Analytics Employee Attrition & Performance

Predict attrition of your valuable employees

https://www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset

```
df = pd.read_csv('data/WA_Fn-UseC_-HR-Employee-Attrition.csv')
```

```
df.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNum
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNum
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7

5 rows × 35 columns

```
df.columns
```

```
Index([u'Age', u'Attrition', u'BusinessTravel', u'DailyRate', u'Department',
       u'DistanceFromHome', u'Education', u'EducationField', u'EmployeeCount',
       u'EmployeeNumber', u'EnvironmentSatisfaction', u'Gender', u'HourlyRate',
       u'JobInvolvement', u'JobLevel', u'JobRole', u'JobSatisfaction',
       u'MaritalStatus', u'MonthlyIncome', u'MonthlyRate',
       u'NumCompaniesWorked', u'Over18', u'OverTime', u'PercentSalaryHike',
       u'PerformanceRating', u'RelationshipSatisfaction', u'StandardHours',
       u'StockOptionLevel', u'TotalWorkingYears', u'TrainingTimesLastYear',
       u'WorkLifeBalance', u'YearsAtCompany', u'YearsInCurrentRole',
       u'YearsSinceLastPromotion', u'YearsWithCurrManager'],
      dtype='object')
```

```
df.shape
```

```
(1470, 35)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
Age                         1470 non-null int64
Attrition                   1470 non-null object
BusinessTravel              1470 non-null object
DailyRate                   1470 non-null int64
Department                  1470 non-null object
DistanceFromHome            1470 non-null int64
Education                   1470 non-null int64
EducationField              1470 non-null object
EmployeeCount               1470 non-null int64
EmployeeNumber              1470 non-null int64
EnvironmentSatisfaction     1470 non-null int64
Gender                      1470 non-null object
HourlyRate                  1470 non-null int64
JobInvolvement              1470 non-null int64
JobLevel                    1470 non-null int64
JobRole                     1470 non-null object
JobSatisfaction             1470 non-null int64
MaritalStatus               1470 non-null object
MonthlyIncome               1470 non-null int64
MonthlyRate                 1470 non-null int64
NumCompaniesWorked          1470 non-null int64
Over18                      1470 non-null object
OverTime                    1470 non-null object
PercentSalaryHike           1470 non-null int64
PerformanceRating           1470 non-null int64
RelationshipSatisfaction    1470 non-null int64
StandardHours               1470 non-null int64
StockOptionLevel            1470 non-null int64
TotalWorkingYears           1470 non-null int64
TrainingTimesLastYear       1470 non-null int64
WorkLifeBalance             1470 non-null int64
YearsAtCompany              1470 non-null int64
```

```
YearsInCurrentRole          1470 non-null int64
YearsSinceLastPromotion     1470 non-null int64
YearsWithCurrManager        1470 non-null int64
dtypes: int64(26), object(9)
memory usage: 402.0+ KB
```

```
df.describe()
```

```
.dataframe thead th {
    text-align: left;
}

.dataframe tbody tr th {
    vertical-align: top;
}
```

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	HourlyRate
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.000000	1470.000000	1470.000000
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.865306	2.721769	65.891156
std	9.135373	403.509100	8.106864	1.024165	0.0	602.024335	1.093082	20.329428
min	18.000000	102.000000	1.000000	1.000000	1.0	1.000000	1.000000	30.000000
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.250000	2.000000	48.000000
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.500000	3.000000	66.000000
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.750000	4.000000	83.750000
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.000000	4.000000	100.000000

8 rows × 26 columns

```
#s = df.describe().loc['std']
```

```
#s[s == 0]
```

```
df.describe(include=['object', 'bool'])
```

```
.dataframe thead th {
    text-align: left;
}

.dataframe tbody tr th {
    vertical-align: top;
}
```

	Attrition	BusinessTravel	Department	EducationField	Gender	JobRole	MaritalStatus	Over18	OverTime
count	1470	1470	1470	1470	1470	1470	1470	1470	1470
unique	2	3	3	6	2	9	3	1	2
top	No	Travel_Rarely	Research & Development	Life Sciences	Male	Sales Executive	Married	Y	No
freq	1233	1043	961	606	882	326	673	1470	1054

```
df = df.drop(['Over18', 'EmployeeCount', 'StandardHours'], axis=1)
```

```
df.shape
```

```
(1470, 32)
```

```
df['Attrition'].value_counts()
```

```
No     1233
Yes     237
Name: Attrition, dtype: int64
```

```
df['JobRole'].value_counts()
```

```
Sales Executive              326
Research Scientist           292
Laboratory Technician        259
Manufacturing Director       145
Healthcare Representative    131
Manager                      102
Sales Representative          83
Research Director             80
Human Resources               52
Name: JobRole, dtype: int64
```

```
df['JobRole'].value_counts(normalize=True)
```

```
Sales Executive              0.221769
Research Scientist           0.198639
Laboratory Technician        0.176190
Manufacturing Director       0.098639
Healthcare Representative    0.089116
Manager                      0.069388
Sales Representative         0.056463
Research Director            0.054422
Human Resources              0.035374
Name: JobRole, dtype: float64
```

```
df[df['Attrition'] == 'Yes'].mean()
```

```
Age                           33.607595
DailyRate                    750.362869
DistanceFromHome              10.632911
Education                      2.839662
EmployeeNumber              1010.345992
EnvironmentSatisfaction        2.464135
HourlyRate                    65.573840
JobInvolvement                 2.518987
JobLevel                       1.637131
JobSatisfaction                2.468354
MonthlyIncome               4787.092827
MonthlyRate                14559.308017
NumCompaniesWorked             2.940928
PercentSalaryHike             15.097046
PerformanceRating              3.156118
RelationshipSatisfaction       2.599156
StockOptionLevel               0.527426
TotalWorkingYears              8.244726
TrainingTimesLastYear          2.624473
WorkLifeBalance                2.658228
YearsAtCompany                 5.130802
YearsInCurrentRole             2.902954
YearsSinceLastPromotion        1.945148
```

```
YearsWithCurrManager           2.852321
dtype: float64
```

```
df[df['Attrition'] == 'Yes']['YearsWithCurrManager'].mean()
```

```
2.852320675105485
```

```
df[df['Attrition'] == 'No']['YearsWithCurrManager'].mean()
```

```
4.367396593673966
```

```
df[(df['MaritalStatus'] != 'Married') & (df['BusinessTravel'] == 'Travel_Frequently')]['Attrition'].value_counts(normalize=True)
```

```
No     0.685535
Yes    0.314465
Name: Attrition, dtype: float64
```

```
df['Attrition'].value_counts(normalize=True)
```

```
No     0.838776
Yes    0.161224
Name: Attrition, dtype: float64
```

Группировка

```
df.groupby(['Attrition'])[['DistanceFromHome','YearsWithCurrManager']].describe(percentiles=[])
```

```
.dataframe thead th {
    text-align: left;
}

.dataframe tbody tr th {
    vertical-align: top;
}
```

	DistanceFromHome						YearsWithCurrManager					
	count	**mean**	**std**	**min**	**50%**	**max**	**count**	**mean**	**std**	**min**	**50%**	**max**
Attrition												
No	1233.0	8.915653	8.012633	1.0	7.0	29.0	1233.0	4.367397	3.594116	0.0	3.0	17.0
Yes	237.0	10.632911	8.452525	1.0	9.0	29.0	237.0	2.852321	3.143349	0.0	2.0	14.0

Сводные таблицы

```
pd.crosstab(df['Attrition'], df['MaritalStatus'])
```

```
.dataframe thead th {
    text-align: left;
}
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

MaritalStatus	Divorced	Married	Single
Attrition			
No	294	589	350
Yes	33	84	120

```
pd.crosstab(df['Attrition'], df['MaritalStatus'], normalize=True)
```

```
.dataframe thead th {
    text-align: left;
}

.dataframe tbody tr th {
    vertical-align: top;
}
```

MaritalStatus	Divorced	Married	Single
Attrition			
No	0.200000	0.400680	0.238095
Yes	0.022449	0.057143	0.081633

```
df.pivot_table(['DailyRate','Education','TotalWorkingYears'],
['Department'], aggfunc='mean')
```

```
.dataframe thead th {
    text-align: left;
}

.dataframe tbody tr th {
    vertical-align: top;
}
```

	DailyRate	Education	TotalWorkingYears
Department			
Human Resources	751.539683	2.968254	11.555556
Research & Development	806.851197	2.899063	11.342352
Sales	800.275785	2.934978	11.105381

```
pd.crosstab(df['Attrition'], df['BusinessTravel'], margins=True)
```

```
.dataframe thead th {
    text-align: left;
}

.dataframe tbody tr th {
    vertical-align: top;
}
```

BusinessTravel	Non-Travel	Travel_Frequently	Travel_Rarely	All
Attrition				
No	138	208	887	1233

BusinessTravel	Non-Travel	Travel_Frequently	Travel_Rarely	All
Attrition				
Yes	12	69	156	237
All	150	277	1043	1470

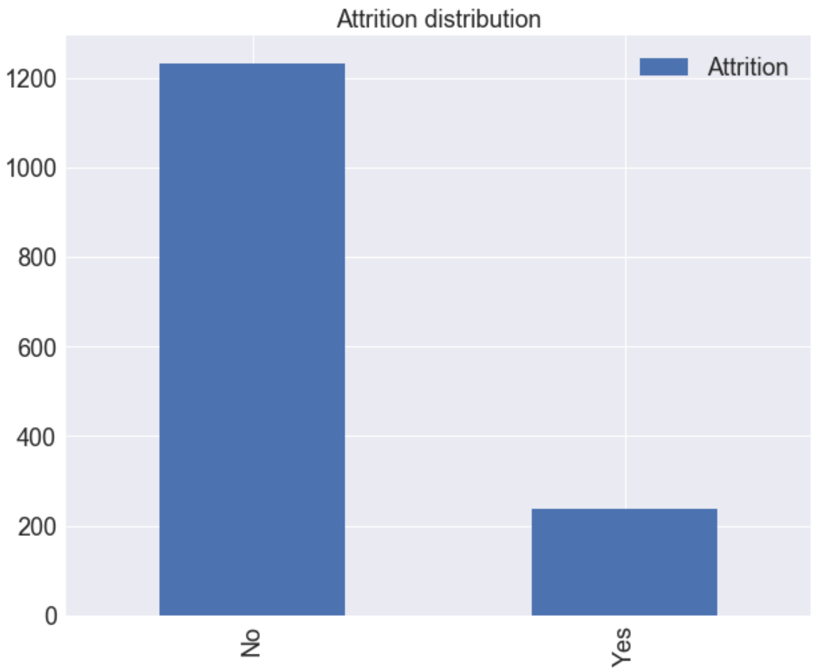
Визуализация данных

Seaborn и Matplotlib

```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
#графики в svg выглядят более четкими
#%config InlineBackend.figure_format = 'svg'

#увеличим дефолтный размер графиков
from pylab import rcParams
rcParams['figure.figsize'] = 10, 8;
```

```
BIGGER_SIZE = 18

plt.rc('font', size=BIGGER_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=BIGGER_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=BIGGER_SIZE)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=BIGGER_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=BIGGER_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=BIGGER_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
```

```
df['Attrition'].value_counts().plot(kind='bar', label='Attrition')
plt.legend()
plt.title('Attrition distribution');
```



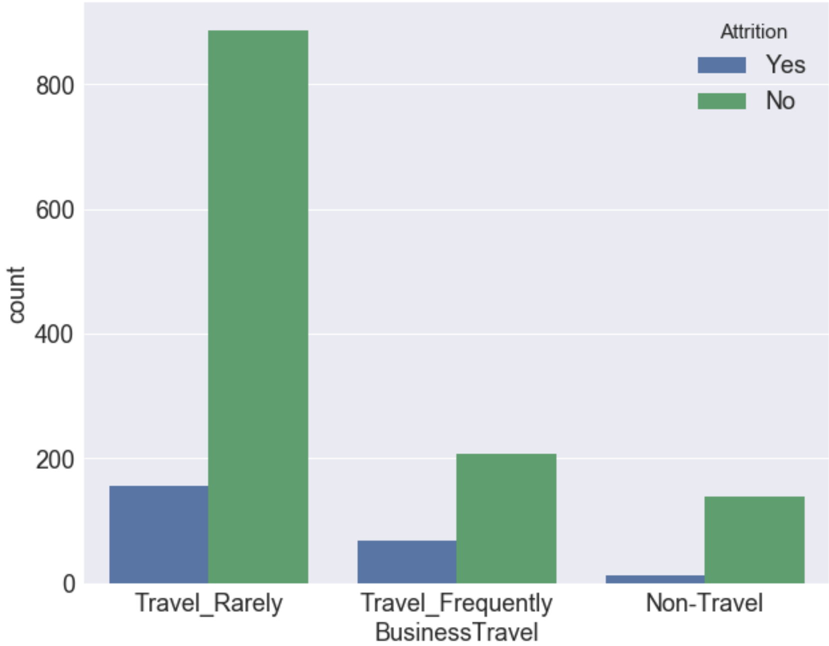

```
pd.crosstab(df['Attrition'], df['BusinessTravel'], margins=True)
```

```
.dataframe thead th {
    text-align: left;
}
```

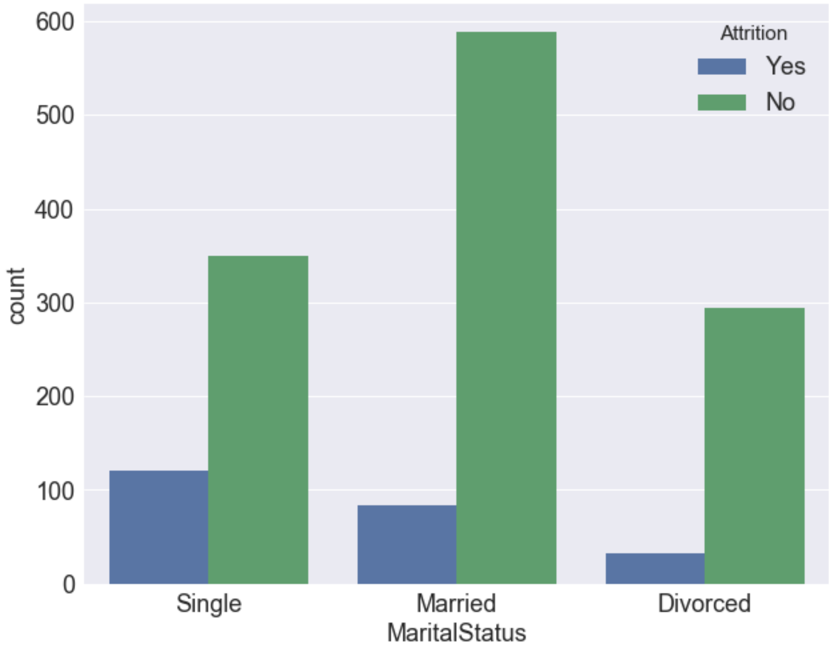
```
.dataframe tbody tr th {
    vertical-align: top;
}
```

BusinessTravel	Non-Travel	Travel_Frequently	Travel_Rarely	All
Attrition				
No	138	208	887	1233
Yes	12	69	156	237
All	150	277	1043	1470

```
sns.countplot(x='BusinessTravel', hue='Attrition', data=df);
```




```
sns.countplot(x='MaritalStatus', hue='Attrition', data=df);
```

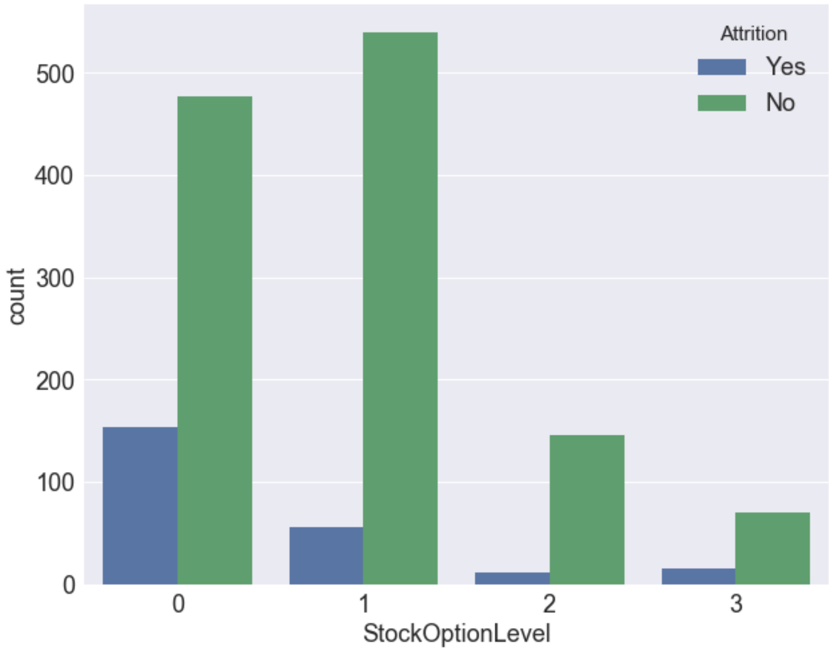



```
pd.crosstab(df['Attrition'], df['StockOptionLevel'], margins=True)
```

```
.dataframe thead th {
    text-align: left;
}

.dataframe tbody tr th {
    vertical-align: top;
}
```

StockOptionLevel	0	1	2	3	All
Attrition					
No	477	540	146	70	1233
Yes	154	56	12	15	237
All	631	596	158	85	1470

```
sns.countplot(x='StockOptionLevel', hue='Attrition', data=df);
```




```
#df['Risk'] = ((df['MaritalStatus'] != 'Married') & (df['BusinessTravel'] == 'Travel_Frequently') &(df['StockOptionLevel'] ==
0)).astype('int')
```

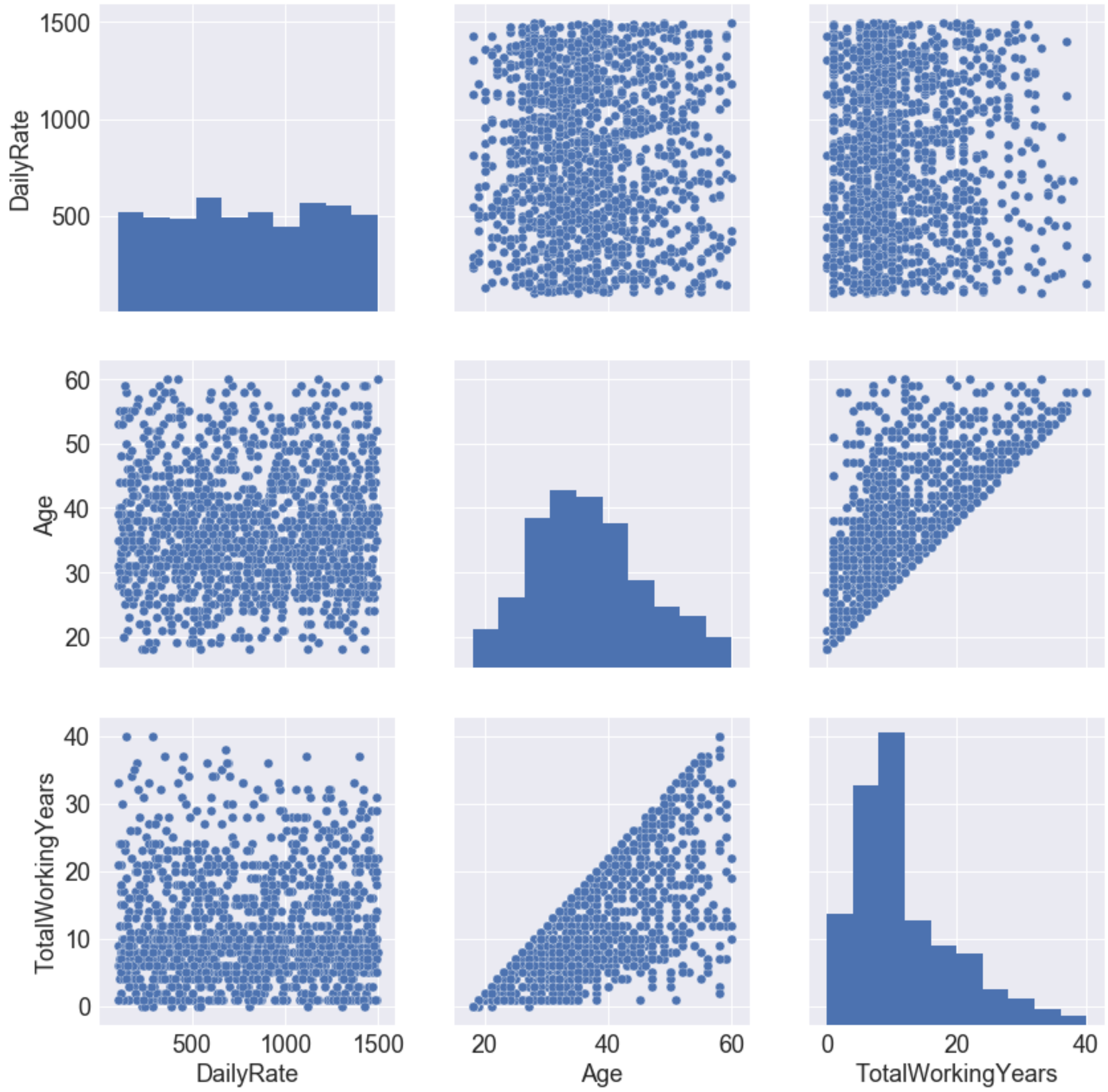
```
#df[['Risk','MaritalStatus','BusinessTravel', 'StockOptionLevel']].head(6)
```

```
#pd.crosstab(df['Attrition'], df['Risk'])
```

```
#sns.countplot(x='Risk', hue='Attrition', data=df);
```

Scatter plot matrix

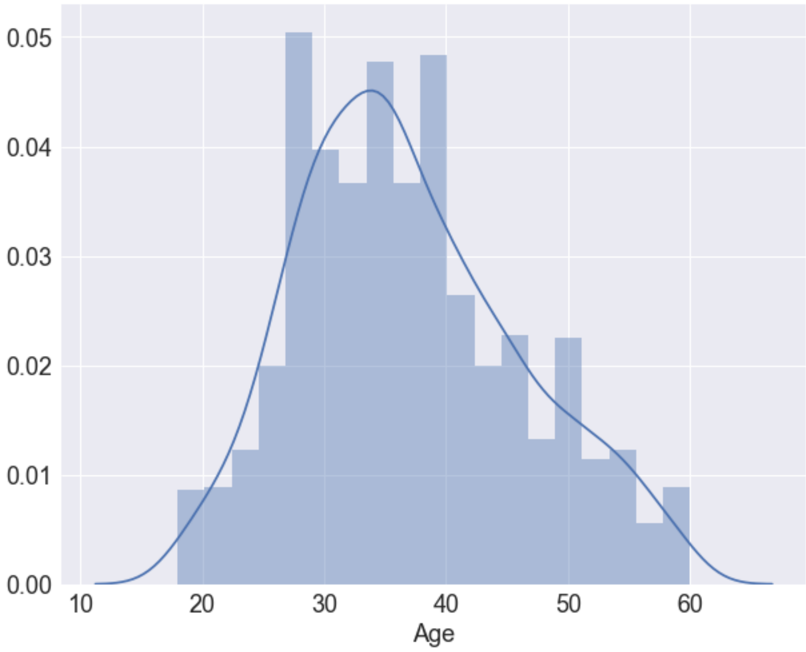
```
cols = ['DailyRate','Age','TotalWorkingYears']
sns_plot = sns.pairplot(df[cols], size = 4)
sns_plot.savefig('pairplot.png')
```



Гистограмма и KDE (kernel density estimation)

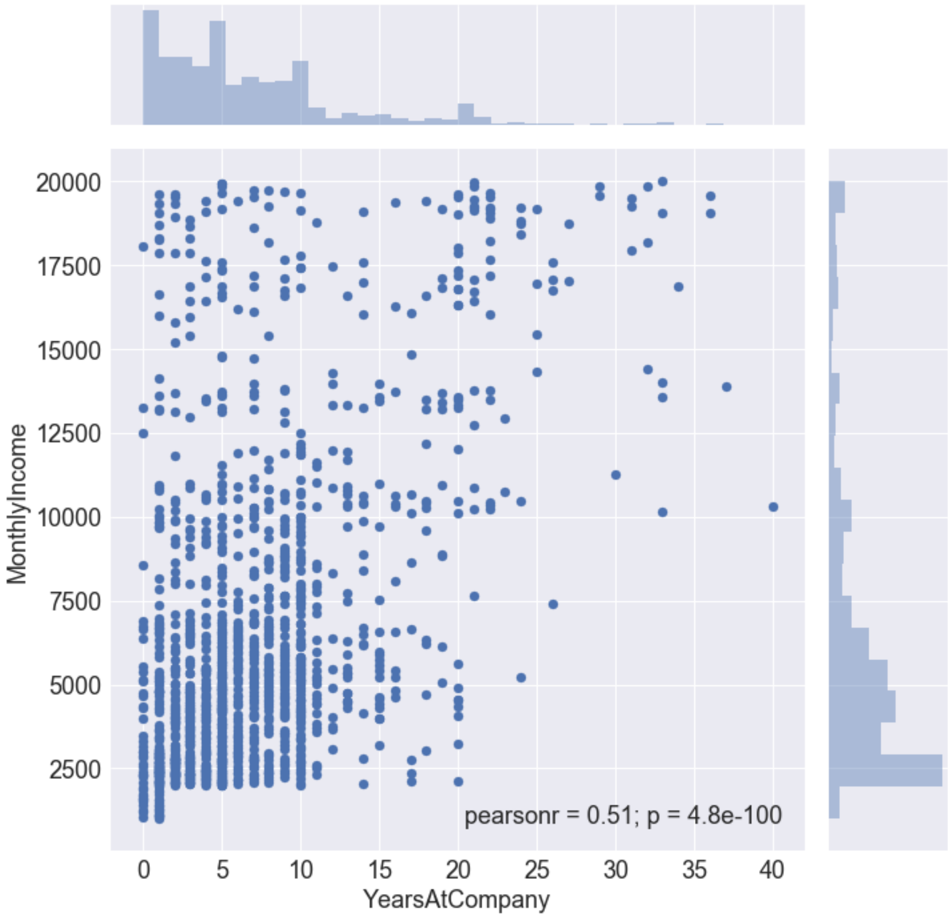
```
sns.distplot(df.Age)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10f61c88>
```




```
sns.jointplot(df.YearsAtCompany, df.MonthlyIncome, size =10)
```

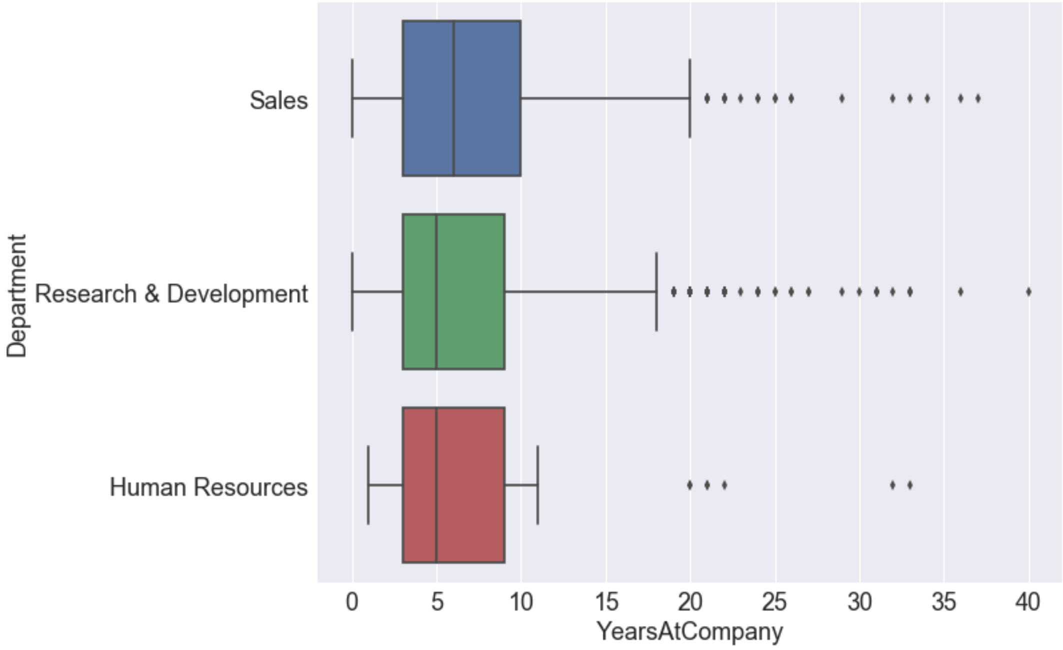
```
<seaborn.axisgrid.JointGrid at 0xfd3f2b0>
```



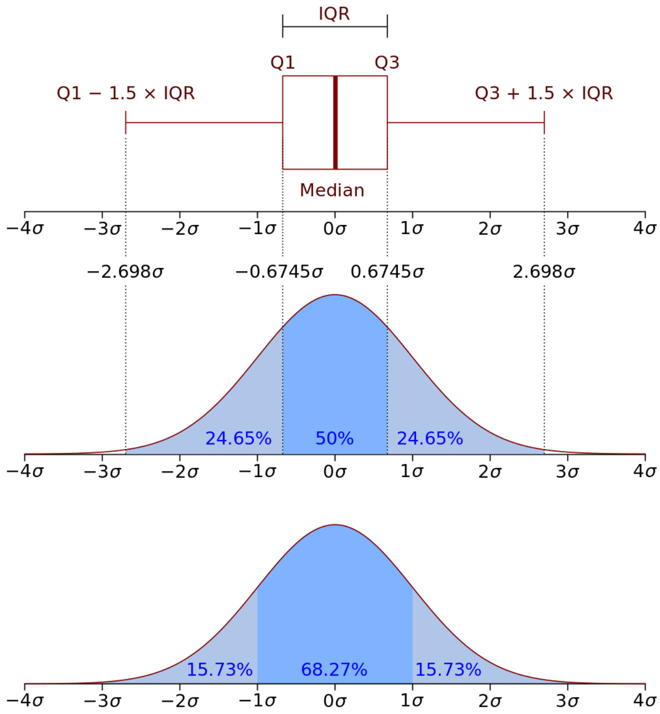
Box plot

```
sns.boxplot(y="Department", x="YearsAtCompany", data=df, orient="h")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1117af60>
```



Box plot состоит из коробки (поэтому он и называется box plot), усиков и точек. Коробка показывает интерквартильный размах распределения, то есть соответственно 25% (Q1) и 75% (Q3) перцентили. Черта внутри коробки обозначает медиану распределения. С коробкой разобрались, перейдем к усам. Усы отображают весь разброс точек кроме выбросов, то есть минимальные и максимальные значения, которые попадают в промежуток (Q1 - 1.5*IQR, Q3 + 1.5*IQR), где IQR = Q3 - Q1 — интерквартильный размах. Точками на графике обозначаются выбросы (outliers) — те значения, которые не вписываются в



промежуток значений, заданный усами графика.

Heat map

```
department_ef_mi = df.pivot_table(
                        index='Department',
                        columns='EducationField',
                        values='YearsInCurrentRole',
                        aggfunc='mean').fillna(0).applymap(float)
sns.heatmap(department_ef_mi, annot=True, fmt=".1f", linewidths=.5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11514b00>
```


```
df['Attrition'] = (df['Attrition']=='Yes').astype('int64')
```

```
corr_matrix = df.corr()
```

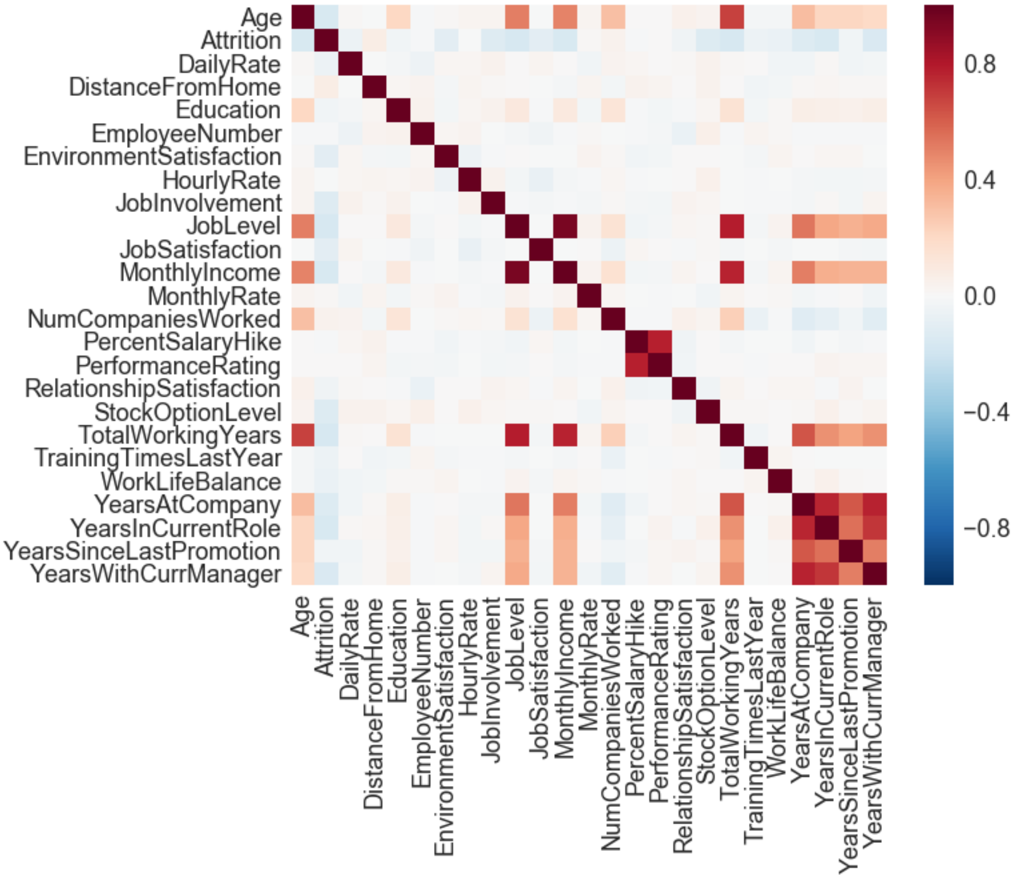
```
corr_matrix.head()
```

```
.dataframe thead th {
    text-align: left;
}

.dataframe tbody tr th {
    vertical-align: top;
}
```

	Age	Attrition	DailyRate	DistanceFromHome	Education	EmployeeNumber	EnvironmentSatisfaction	HourlyRate
Age	1.000000	-0.159205	0.010661	-0.001686	0.208034	-0.010145	0.010146	0.024287
Attrition	-0.159205	1.000000	-0.056652	0.077924	-0.031373	-0.010577	-0.103369	-0.006846
DailyRate	0.010661	-0.056652	1.000000	-0.004985	-0.016806	-0.050990	0.018355	0.023381
DistanceFromHome	-0.001686	0.077924	-0.004985	1.000000	0.021042	0.032916	-0.016075	0.031131
Education	0.208034	-0.031373	-0.016806	0.021042	1.000000	0.042070	-0.027128	0.016775

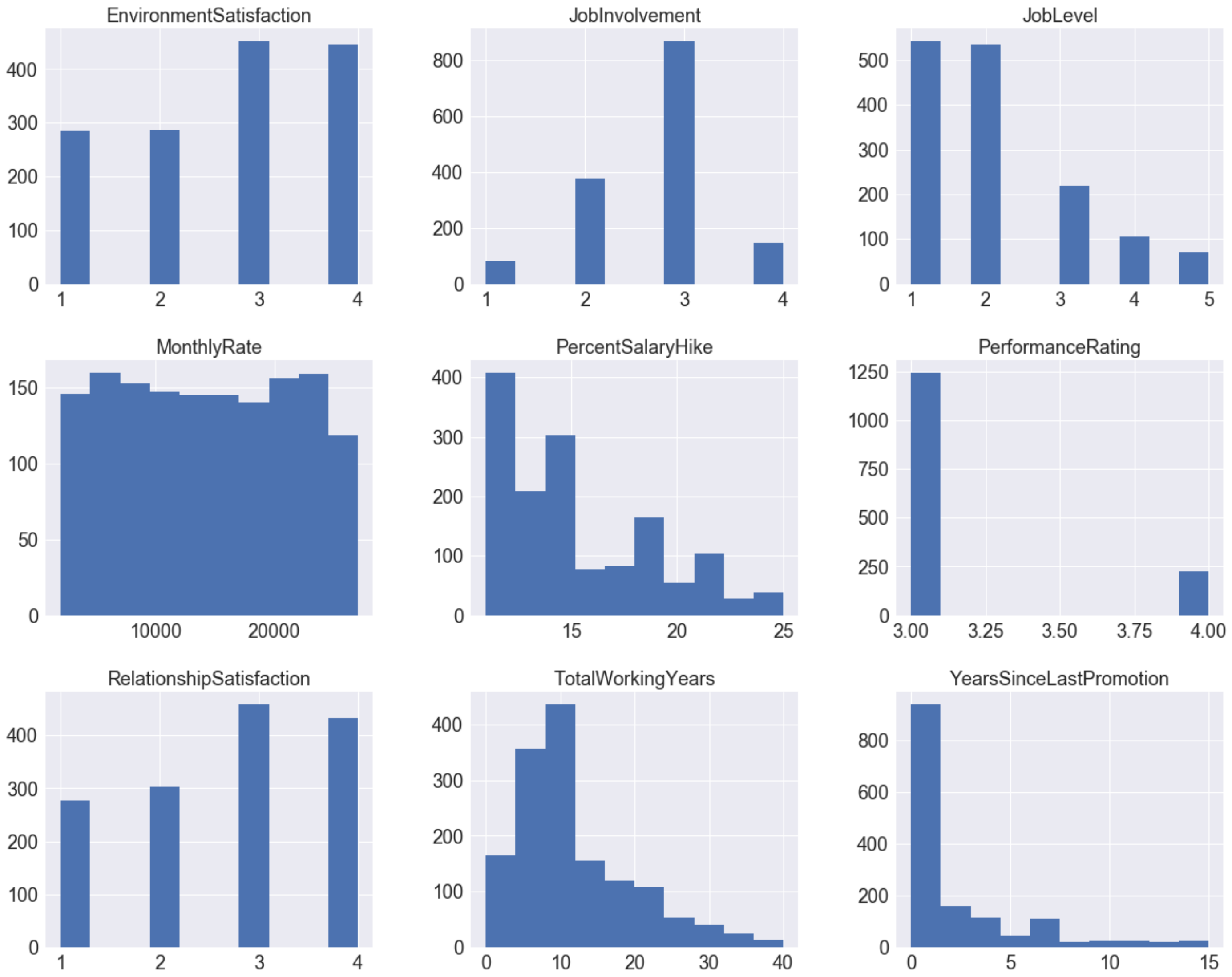
5 rows × 25 columns

```
sns.heatmap(corr_matrix);
```

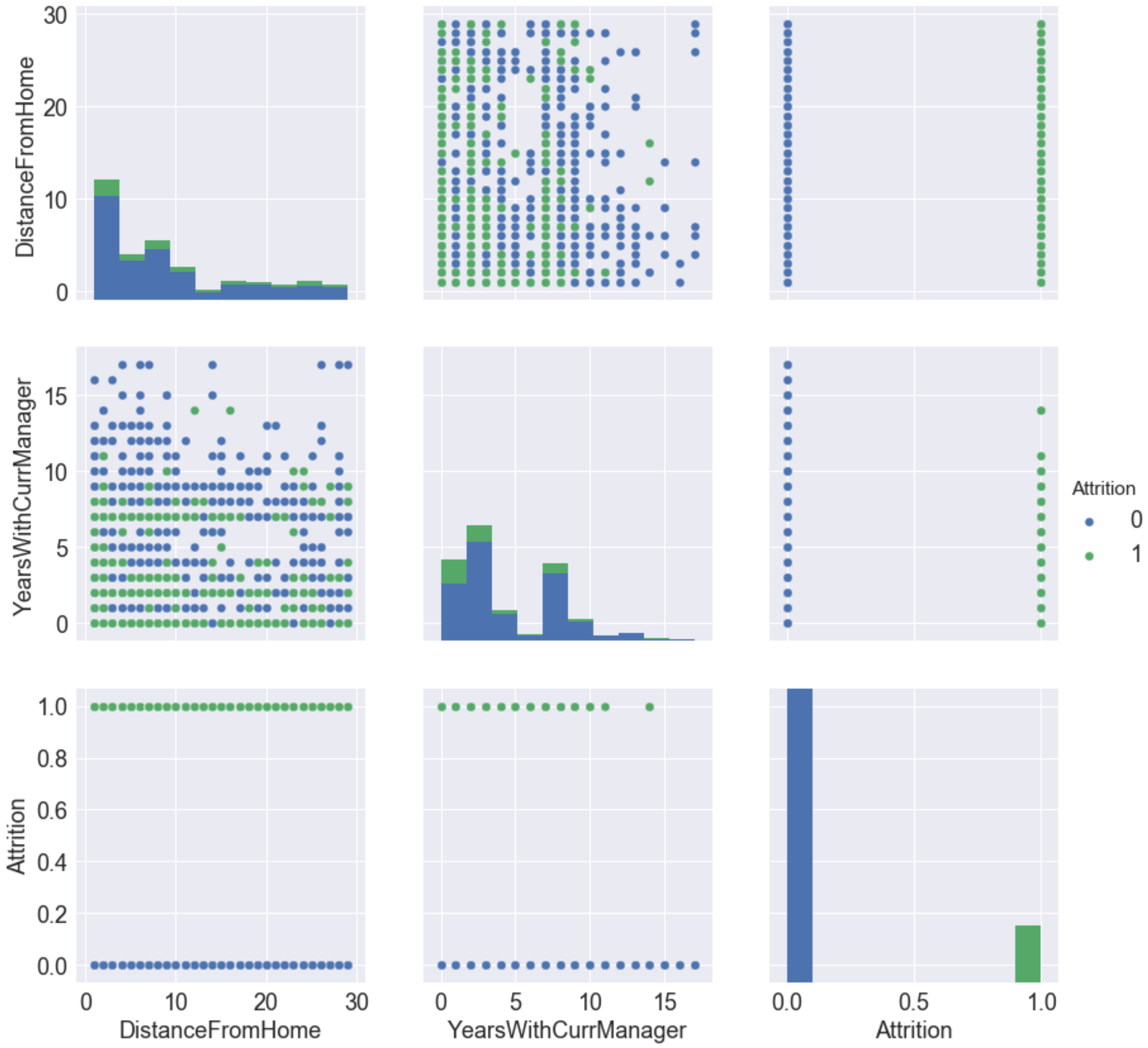



```
features = list(set(df.columns) - set(['BusinessTravel', 'Department',  'EducationField',
                                        'Gender',   'JobRole',    'MaritalStatus',
                                         'Over18', 'OverTime', 'Attrition']))
```

```
df[features[:9]].hist(figsize=(20,16));
```

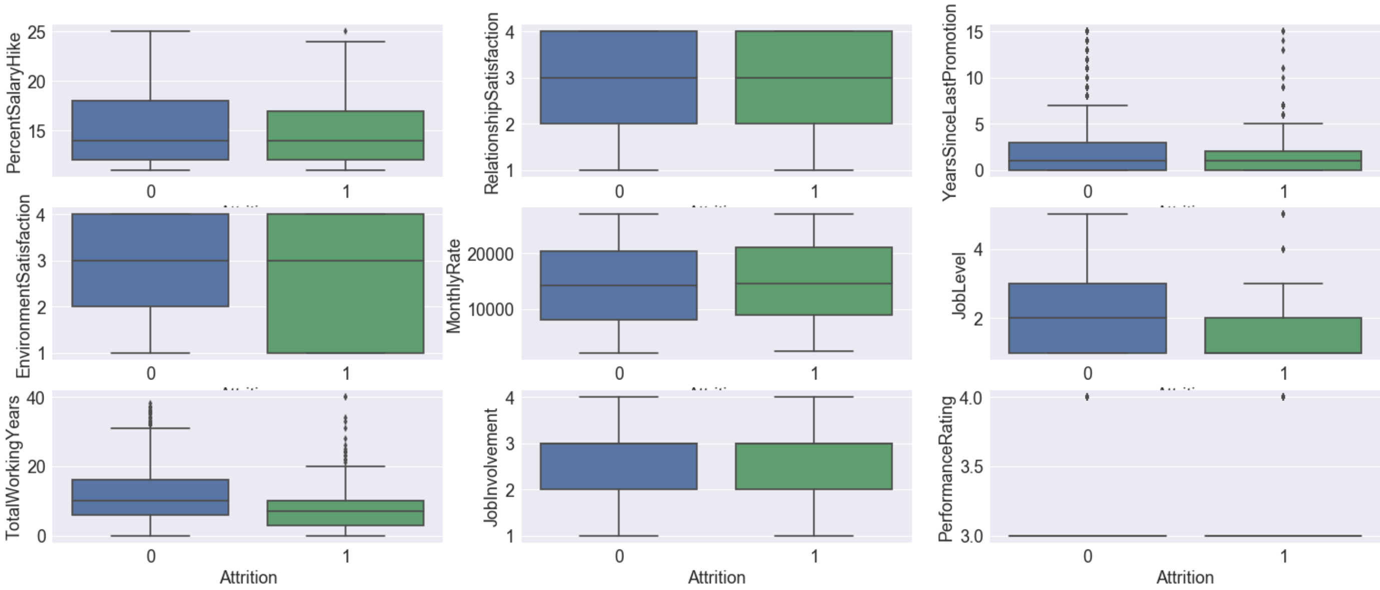



```
sns.pairplot(df[['DistanceFromHome','YearsWithCurrManager','Attrition']], hue='Attrition', size = 4);
```

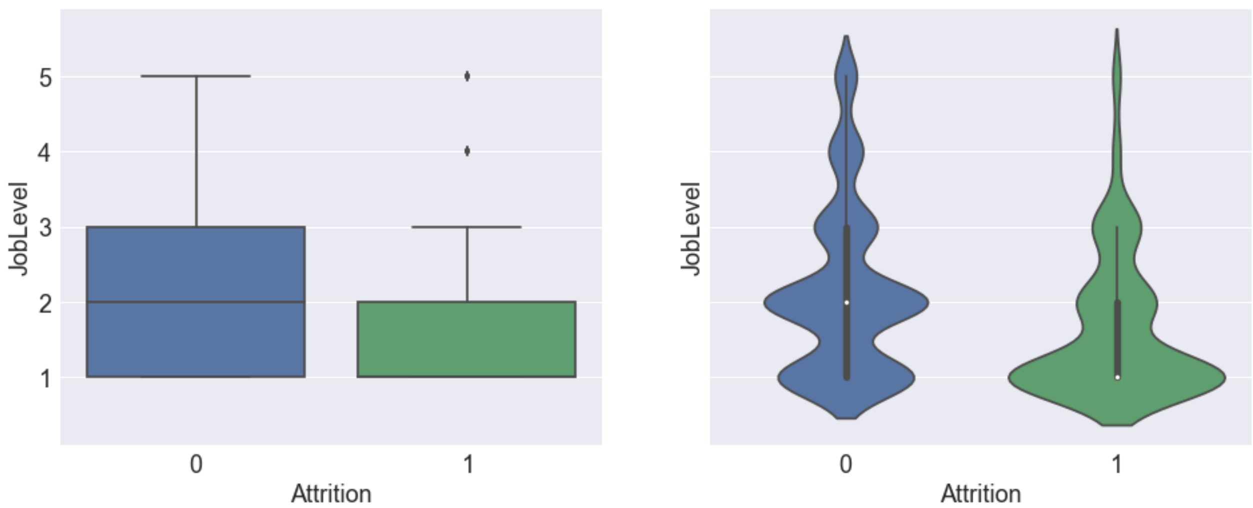



```
cnt = 3
fig, axes = plt.subplots(nrows=cnt, ncols=cnt, figsize=(25, 10))

for idx, feat in  enumerate(features[:9]):
    sns.boxplot(x='Attrition', y=feat, data=df, ax=axes[idx / cnt, idx % cnt])
    axes[idx / cnt, idx % cnt].legend()
    axes[idx / cnt, idx % cnt].set_xlabel('Attrition')
    axes[idx / cnt, idx % cnt].set_ylabel(feat);
```

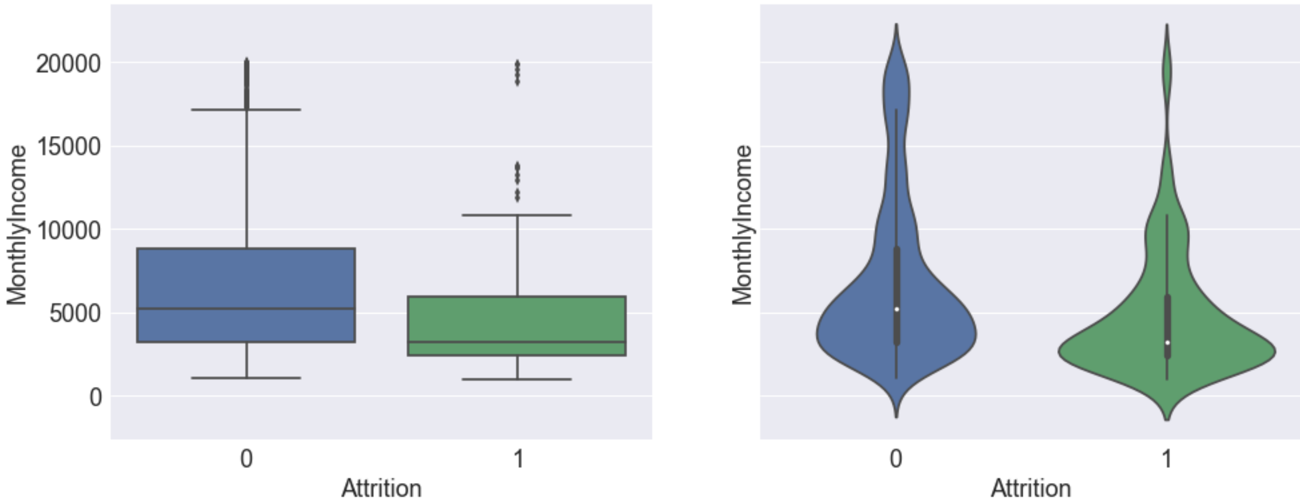
```
C:\Anaconda2\lib\site-packages\matplotlib\axes\_axes.py:545: UserWarning: No labelled objects found. Use label='...' kwarg on
individual plots.
  warnings.warn("No labelled objects found. "
```




```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(16,6))

sns.boxplot(x='Attrition', y='JobLevel', data=df, ax=axes[0]);
sns.violinplot(x='Attrition', y='JobLevel', data=df, ax=axes[1]);
```




```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(16,6))

sns.boxplot(x='Attrition', y='MonthlyIncome', data=df, ax=axes[0]);
sns.violinplot(x='Attrition', y='MonthlyIncome', data=df, ax=axes[1]);
```




```
#sns.countplot(x='JobLevel', hue='Attrition', data=df);
```

```
#_, axes = plt.subplots(1, 2, sharey=True, figsize=(16,6))

#sns.countplot(x='MaritalStatus', hue='Attrition', data=df, ax=axes[0]);
#sns.countplot(x='Department', hue='Attrition', data=df, ax=axes[1]);
```

t-SNE (t-distributed Stohastic Neighbor Embedding)

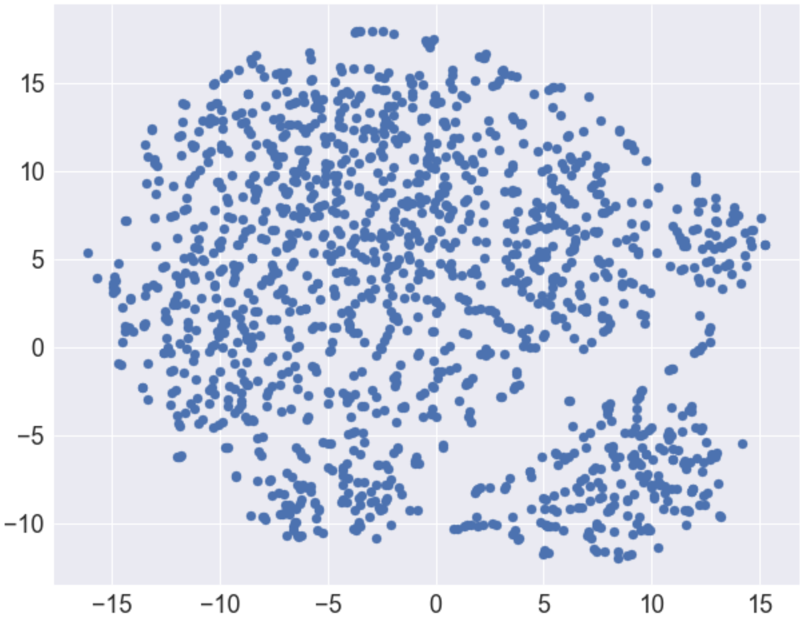
```
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
```

```
X = df.drop(['Attrition', 'JobRole', 'BusinessTravel', 'Department', 'EducationField', 'MaritalStatus'], axis=1)
X['Gender'] = pd.factorize(X['Gender'])[0]
X['OverTime'] = pd.factorize(X['OverTime'])[0]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

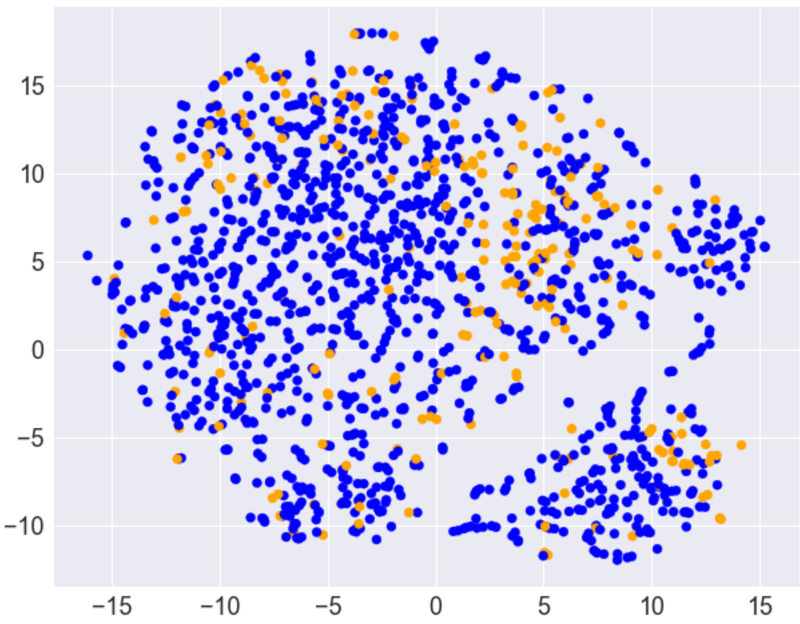
```
%%time
tsne = TSNE(random_state=17)
tsne_representation = tsne.fit_transform(X_scaled) #1min
```

```
Wall time: 17.4 s
```

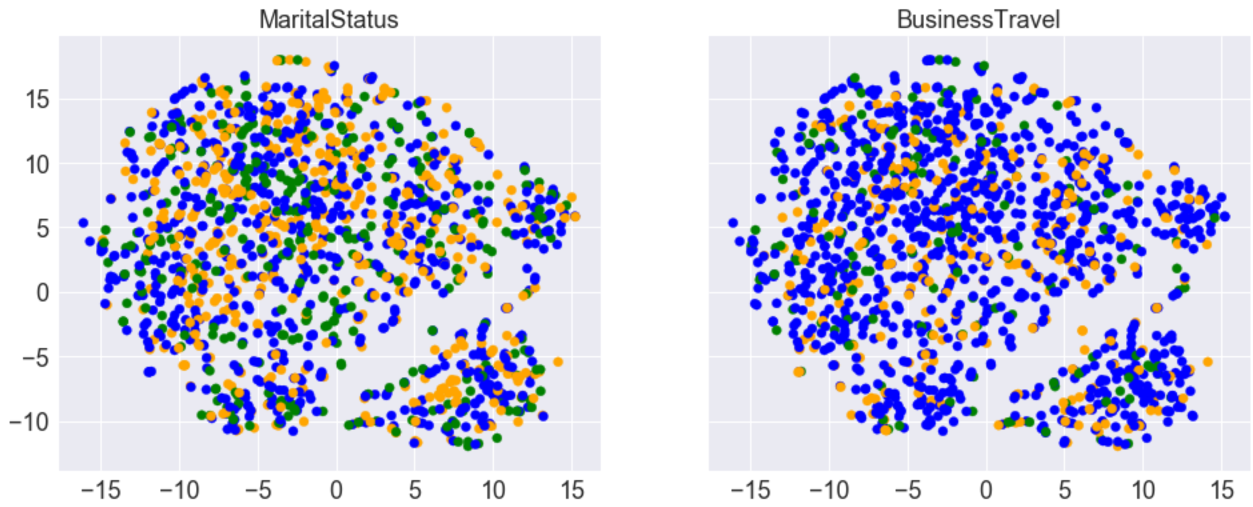
```
plt.scatter(tsne_representation[:, 0], tsne_representation[:, 1]);
```



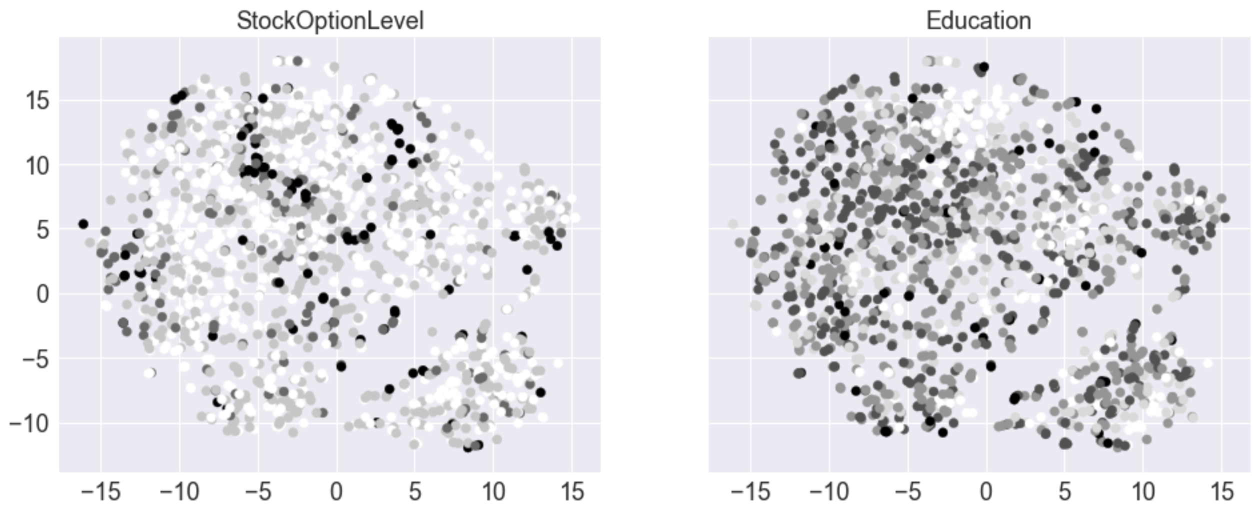

```
plt.scatter(tsne_representation[:, 0], tsne_representation[:, 1],
            c=df['Attrition'].map({0: 'blue', 1: 'orange'}));
```




```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(16,6))

axes[0].scatter(tsne_representation[:, 0], tsne_representation[:, 1],
            c=df['MaritalStatus'].map({'Married': 'blue', 'Single': 'orange', 'Divorced': 'green'}));
axes[1].scatter(tsne_representation[:, 0], tsne_representation[:, 1],
            c=df['BusinessTravel'].map({'Travel_Rarely': 'blue', 'Travel_Frequently': 'orange', 'Non-Travel': 'green'}));
axes[0].set_title('MaritalStatus');
axes[1].set_title('BusinessTravel');
```




```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(16,6))

axes[0].scatter(tsne_representation[:, 0], tsne_representation[:, 1],
            c=df['StockOptionLevel']);
axes[1].scatter(tsne_representation[:, 0], tsne_representation[:, 1],
            c=df['Education']);
axes[0].set_title('StockOptionLevel');
axes[1].set_title('Education');
```



Машинное обучение

Scikit-learn. Деревья решений и метод ближайших соседей

```
df['Department'] = pd.factorize(df['Department'])[0]
df['Gender'] = pd.factorize(df['Gender'])[0]
df['JobRole'] = pd.factorize(df['JobRole'])[0]
df['MaritalStatus'] = pd.factorize(df['MaritalStatus'])[0]
df['OverTime'] = pd.factorize(df['OverTime'])[0]
df['EducationField'] = pd.factorize(df['EducationField'])[0]
df['BusinessTravel'] = pd.factorize(df['BusinessTravel'])[0]
```

```
y = df['Attrition']
```

```
df.drop(['Attrition'], axis=1, inplace=True)
```

```
y.value_counts(normalize=True)
```

```
0    0.838776
1    0.161224
Name: Attrition, dtype: float64
```

```
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
X_train, X_holdout, y_train, y_holdout = train_test_split(df.values, y, test_size=0.3,
random_state=17)

tree = DecisionTreeClassifier(max_depth=5, random_state=17)
knn = KNeighborsClassifier(n_neighbors=10)

tree.fit(X_train, y_train)
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=10, p=2,
           weights='uniform')
```

```
from sklearn.metrics import accuracy_score
```

```
tree_pred = tree.predict(X_holdout)
accuracy_score(y_holdout, tree_pred)
```

```
0.83673469387755106
```

```
knn_pred = knn.predict(X_holdout)
accuracy_score(y_holdout, knn_pred)
```

```
0.8344671201814059
```

```
from sklearn.model_selection import GridSearchCV, cross_val_score
```

```
tree_params = {'max_depth': range(1,4),'max_features': range(10,20)}
```

```
tree_grid = GridSearchCV(tree, tree_params, cv=5, n_jobs=-1, verbose=True)
```

```
%%time
tree_grid.fit(X_train, y_train) #12sec
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Wall time: 5.06 s


[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:    4.6s finished
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            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=17, splitter='best'),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid={'max_features': [10, 11, 12, 13, 14, 15, 16, 17, 18, 19], 'max_depth': [1, 2, 3]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=True)
```

```
tree_grid.best_params_
```

```
{'max_depth': 3, 'max_features': 13}
```

```
tree_grid.best_score_
```

```
0.86297376093294464
```

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```
knn_pipe = Pipeline([('scaler', StandardScaler()), ('knn', KNeighborsClassifier(n_jobs=-1))])
```

```
knn_params = {'knn__n_neighbors': range(1, 10)}
```

```
knn_grid = GridSearchCV(knn_pipe, knn_params,
cv=5, n_jobs=-1,
verbose=True)
```

```
%%time
knn_grid.fit(X_train, y_train) #10sec
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Wall time: 7.97 s


[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed:    7.6s finished
C:\Anaconda2\lib\site-packages\sklearn\utils\validation.py:429: DataConversionWarning: Data with input dtype int64 was converted
to float64 by StandardScaler.
  warnings.warn(msg, _DataConversionWarning)




GridSearchCV(cv=5, error_score='raise',
       estimator=Pipeline(steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('knn',
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=-1, n_neighbors=5, p=2,
           weights='uniform'))]),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid={'knn__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9]},
```

```
        pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
        scoring=None, verbose=True)
```

```
knn_grid.best_params_, knn_grid.best_score_
```

```
({'knn__n_neighbors': 8}, 0.84839650145772594)
```

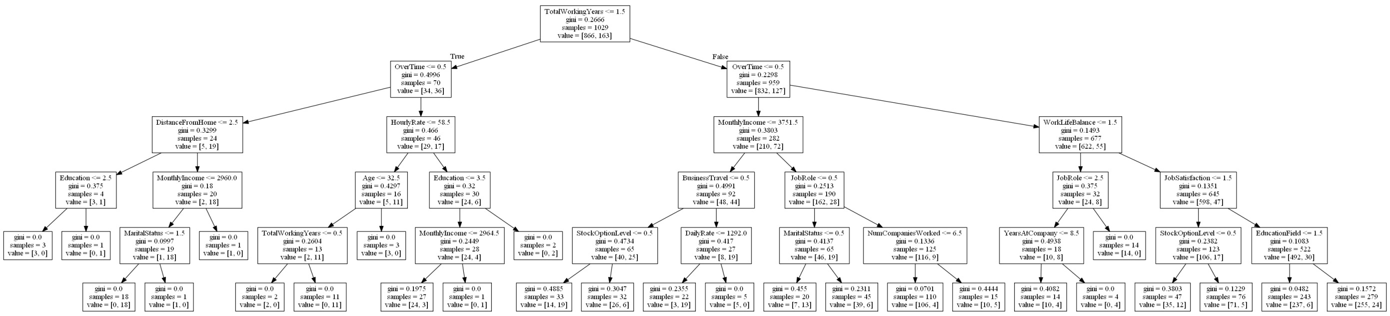
```
import numpy as np
from sklearn.tree import export_graphviz
```

```
#!pip install pydotplus
```

```
#!pip install graphviz
```

```
from sklearn import tree
from IPython.display import Image
import pydotplus
```

```
dot_data = tree.export_graphviz(tree_grid.estimator, feature_names=df.columns, out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf('df_train.pdf')
graph.write_png('df_train.png')
Image(graph.create_png())
```



Случайный лес

```
from sklearn.ensemble import RandomForestClassifier
```

```
forest = RandomForestClassifier(n_estimators=100, n_jobs=-1, random_state=17)
print(np.mean(cross_val_score(forest, X_train, y_train, cv=5))) # 0.859
```

```
0.859131853286
```

```
forest_params = {'max_depth': range(10,11),
'max_features': range(10,15)}
```

```
forest_grid = GridSearchCV(forest, forest_params,
cv=5, n_jobs=-1,
verbose=True)
```

```
%%time
forest_grid.fit(X_train, y_train) #50sec
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits


[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:   13.9s finished


Wall time: 15 s





GridSearchCV(cv=5, error_score='raise',
       estimator=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=100, n_jobs=-1, oob_score=False, random_state=17,
            verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid={'max_features': [10, 11, 12, 13, 14], 'max_depth': [10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=True)
```

```
forest_grid.best_params_, forest_grid.best_score_ #0.864
```

```
({'max_depth': 10, 'max_features': 12}, 0.86394557823129248)
```

Логистическая регрессия

```
from sklearn.linear_model import LogisticRegression
```

```
%%time
logit = LogisticRegression(n_jobs=-1, random_state=7)
logit.fit(X_train, y_train)
print(round(logit.score(X_train, y_train), 3), round(logit.score(X_holdout, y_holdout), 3))
```

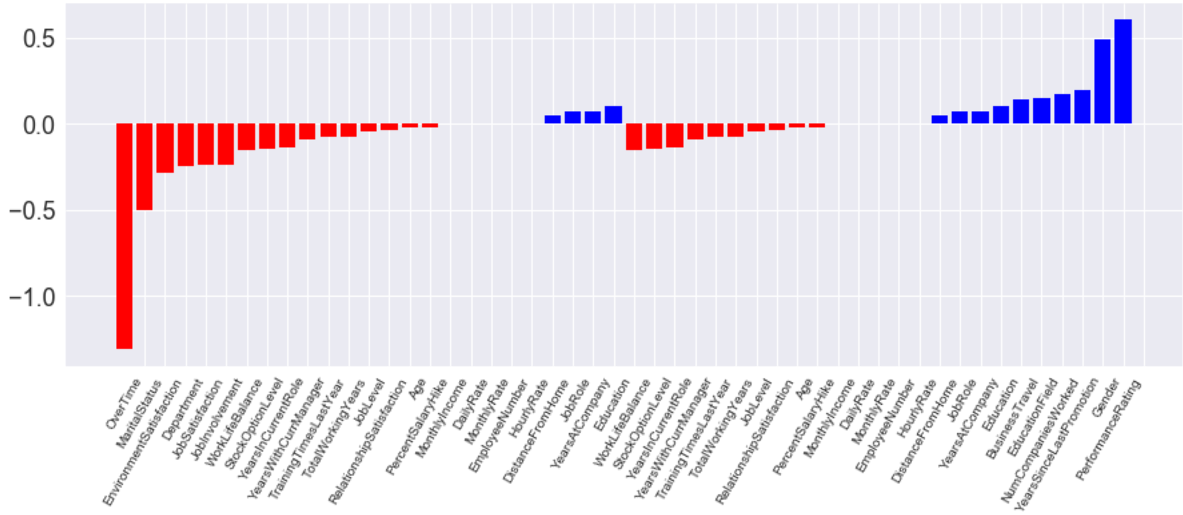
```
(0.869, 0.846)
Wall time: 47 ms
```

```
def visualize_coefficients(classifier, feature_names, n_top_features=25):
# get coefficients with large absolute values
    coef = classifier.coef_.ravel()
    positive_coefficients = np.argsort(coef)[-n_top_features:]
    negative_coefficients = np.argsort(coef)[:n_top_features]
    interesting_coefficients = np.hstack([negative_coefficients, positive_coefficients])
# plot them
    plt.figure(figsize=(15, 5))
    colors = ["red" if c < 0 else "blue" for c in coef[interesting_coefficients]]
    plt.bar(np.arange(2 * n_top_features), coef[interesting_coefficients], color=colors)
    feature_names = np.array(feature_names)
    plt.xticks(np.arange(1, 1 + 2 * n_top_features), feature_names[interesting_coefficients], rotation=60, ha="right", size=10);
```

```
#def plot_grid_scores(grid, param_name):
#    plt.plot(grid.param_grid[param_name], grid.cv_results_['mean_train_score'],
```

```
#     color='green', label='train')
#     plt.plot(grid.param_grid[param_name], grid.cv_results_['mean_test_score'],
#     color='red', label='test')
#     plt.legend();
```

```
visualize_coefficients(logit, df.columns)
```



Ссылки

http://ods.ai/
https://habr.com/company/ods/blog/322626/

Вопросы?