Импорт библиотек

```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from IPython.display import Image
from sklearn.datasets import load_iris, load_boston, load_wine
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_repor
t
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
from sklearn.ensemble import BaggingClassifier, AdaBoostRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_err
or, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.linear_model import LinearRegression
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
from sklearn.datasets import load_iris, load_wine, load_boston
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import Image
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import KFold, RepeatedKFold, LeaveOneOut, LeavePOut, Shuffle
Split, StratifiedKFold
from sklearn.metrics import precision_score, recall_score, f1_score, classification_repor
t
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_err
or, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import learning_curve, validation_curve
import seaborn as sns
```

Загрузка данных

```
data = pd.read_csv('economic.csv')
data.head()
```

	CountryID	Country Name	WEBNAME	Region	World Rank	Region Rank	2019 Score	Property Rights	Judical Effectiveness	Government Integrity
0	1	Afghanistan	Afghanistan	Asia-Pacific	152.0	39.0	51.5	19.6	29.6	25.2
1	2	Albania	Albania	Europe	52.0	27.0	66.5	54.8	30.6	40.4
2	3	Algeria	Algeria	Middle East and North Africa	171.0	14.0	46.2	31.6	36.2	28.9
3	4	Angola	Angola	Sub-Saharan Africa	156.0	33.0	50.6	35.9	26.6	20.5
4	5	Argentina	Argentina	Americas	148.0	26.0	52.2	47.8	44.5	33.5

5 rows × 34 columns

Этот набор данных создан для суммы покупки.

Содержание Набор данных содержит несколько параметров, которые считаются важными во время применения для основных программ.

```
data.shape
```

```
(186, 34)
```

В нашем наборе данных более 500K строк и 12 столбцов. Посмотрим тип данных:

```
data.dtypes
```

```
CountryID                      int64
Country Name                  object
WEBNAME                       object
Region                        object
World Rank                   float64
Region Rank                  float64
2019 Score                   float64
Property Rights              float64
Judical Effectiveness        float64
Government Integrity         float64
Tax Burden                   float64
Gov't Spending               float64
Fiscal Health                float64
Business Freedom             float64
Labor Freedom                float64
Monetary Freedom             float64
Trade Freedom                float64
Investment Freedom           float64
Financial Freedom            float64
Tariff Rate (%)              float64
Income Tax Rate (%)          float64
Corporate Tax Rate (%)       float64
Tax Burden % of GDP          float64
Gov't Expenditure % of GDP   float64
Country                       object
Population (Millions)         object
GDP (Billions, PPP)           object
GDP Growth Rate (%)          float64
5 Year GDP Growth Rate (%)   float64
GDP per Capita (PPP)          object
Unemployment (%)              object
Inflation (%)                float64
FDI Inflow (Millions)         object
Public Debt (% of GDP)       float64
dtype: object
```

Посмотрим, есть ли пропущенные значения в данных:

```
data.isnull().sum()
```

```
CountryID                      0
Country Name                   0
WEBNAME                        0
Region                         0
World Rank                     6
Region Rank                    6
2019 Score                     6
Property Rights                1
Judical Effectiveness          1
Government Integrity           1
Tax Burden                     6
Gov't Spending                 3
Fiscal Health                  3
Business Freedom               1
Labor Freedom                  2
Monetary Freedom               2
Trade Freedom                  4
Investment Freedom             2
Financial Freedom              5
Tariff Rate (%)                4
Income Tax Rate (%)            3
Corporate Tax Rate (%)         3
Tax Burden % of GDP            7
Gov't Expenditure % of GDP     4
Country                        0
Population (Millions)          0
GDP (Billions, PPP)            1
GDP Growth Rate (%)            2
5 Year GDP Growth Rate (%)     3
GDP per Capita (PPP)           2
Unemployment (%)               5
Inflation (%)                  4
FDI Inflow (Millions)          5
Public Debt (% of GDP)         4
dtype: int64
```

```
data.columns
```

```
Index(['CountryID', 'Country Name', 'WEBNAME', 'Region', 'World Rank',
       'Region Rank', '2019 Score', 'Property Rights', 'Judical Effectivenes
s',
       'Government Integrity', 'Tax Burden', 'Gov't Spending', 'Fiscal Healt
h',
       'Business Freedom', 'Labor Freedom', 'Monetary Freedom',
       'Trade Freedom', 'Investment Freedom ', 'Financial Freedom',
       'Tariff Rate (%)', 'Income Tax Rate (%)', 'Corporate Tax Rate (%)',
       'Tax Burden % of GDP', 'Gov't Expenditure % of GDP ', 'Country',
       'Population (Millions)', 'GDP (Billions, PPP)', 'GDP Growth Rate (%)',
       '5 Year GDP Growth Rate (%)', 'GDP per Capita (PPP)',
       'Unemployment (%)', 'Inflation (%)', 'FDI Inflow (Millions)',
       'Public Debt (% of GDP)'],
      dtype='object')
```

```
data = data.dropna()
```

```
data.isnull().sum()
```

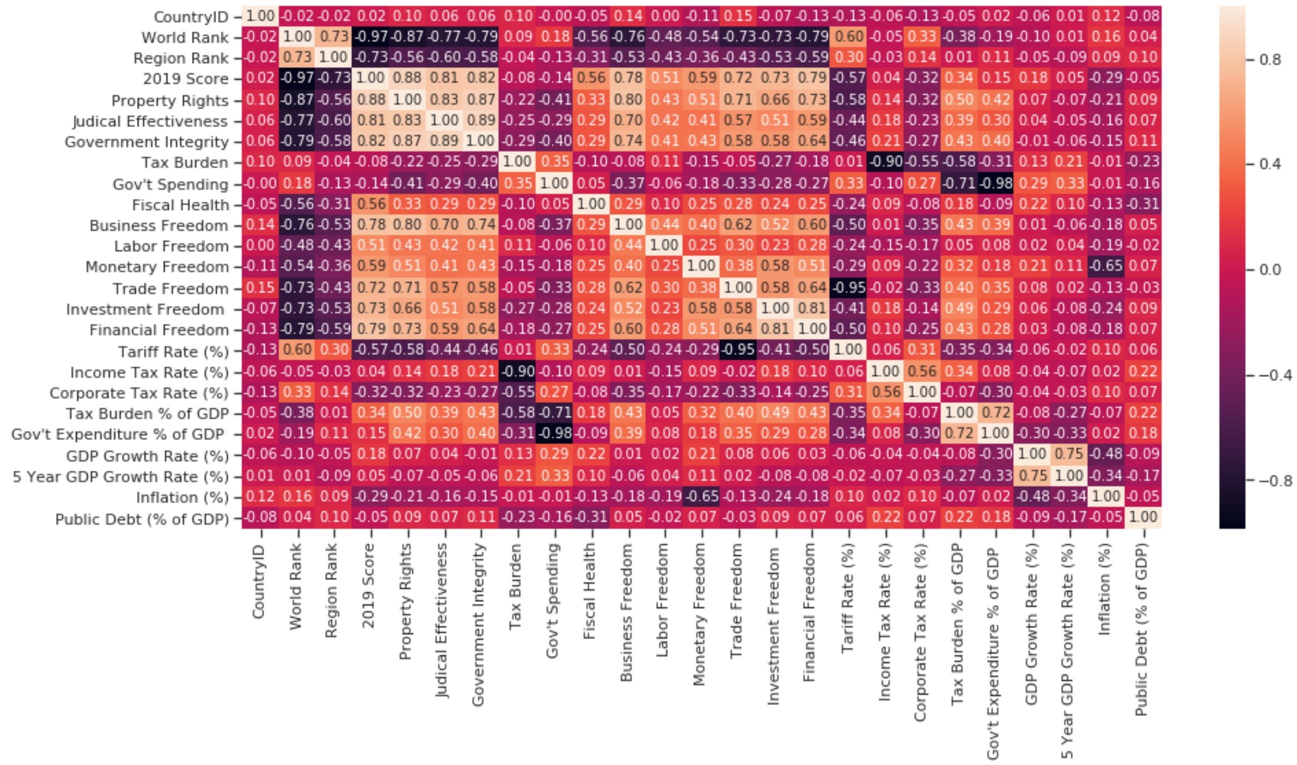
```
CountryID                      0
Country Name                   0
WEBNAME                        0
Region                         0
World Rank                     0
Region Rank                    0
2019 Score                     0
Property Rights                0
Judical Effectiveness          0
Government Integrity           0
Tax Burden                     0
Gov't Spending                 0
Fiscal Health                  0
Business Freedom               0
Labor Freedom                  0
Monetary Freedom               0
Trade Freedom                  0
Investment Freedom             0
Financial Freedom              0
Tariff Rate (%)                0
Income Tax Rate (%)            0
Corporate Tax Rate (%)         0
Tax Burden % of GDP            0
Gov't Expenditure % of GDP     0
Country                        0
Population (Millions)          0
GDP (Billions, PPP)            0
GDP Growth Rate (%)            0
5 Year GDP Growth Rate (%)     0
GDP per Capita (PPP)           0
Unemployment (%)               0
Inflation (%)                  0
FDI Inflow (Millions)          0
Public Debt (% of GDP)         0
dtype: int64
```

Анализ данных

Анализ данных начнем с построения матрицы корреляций:

```python
fig, ax = plt.subplots(figsize=(15,7))
sns.heatmap(data.corr(method='pearson'), ax=ax, annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4585a735f8>
```



Посмотрим более детально на следующие признаки:

- 2019 Score
- Property Rights
- Judical Effectiveness
- Government Integrity
- Business Freedom
- Monetary Freedom
- Trade Freedom
- Investment Freedom
- Financial Freedom

```python
data = data[[
    '2019 Score',
    'Property Rights',
    'Judical Effectiveness',
    'Government Integrity',
    'Business Freedom',
    'Monetary Freedom',
    'Trade Freedom',
    'Investment Freedom ',
    'Financial Freedom',
]]
data.head()
```

	2019 Score	Property Rights	Judical Effectiveness	Government Integrity	Business Freedom	Monetary Freedom	Trade Freedom	Investment Freedom	Financial Freedom
0	51.5	19.6	29.6	25.2	49.2	76.7	66.0	10.0	10.0
1	66.5	54.8	30.6	40.4	69.3	81.5	87.8	70.0	70.0
2	46.2	31.6	36.2	28.9	61.6	74.9	67.4	30.0	30.0
3	50.6	35.9	26.6	20.5	55.7	55.4	61.2	30.0	40.0
4	52.2	47.8	44.5	33.5	56.4	60.2	70.0	55.0	60.0

Теперь у нас есть только подходящие данные для анализа. Еще раз посмотрим на матрицу корреляций

```python
fig, ax = plt.subplots(figsize=(7, 5))
sns.heatmap(data.corr(method='pearson'), ax=ax, annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4585a07e48>
```

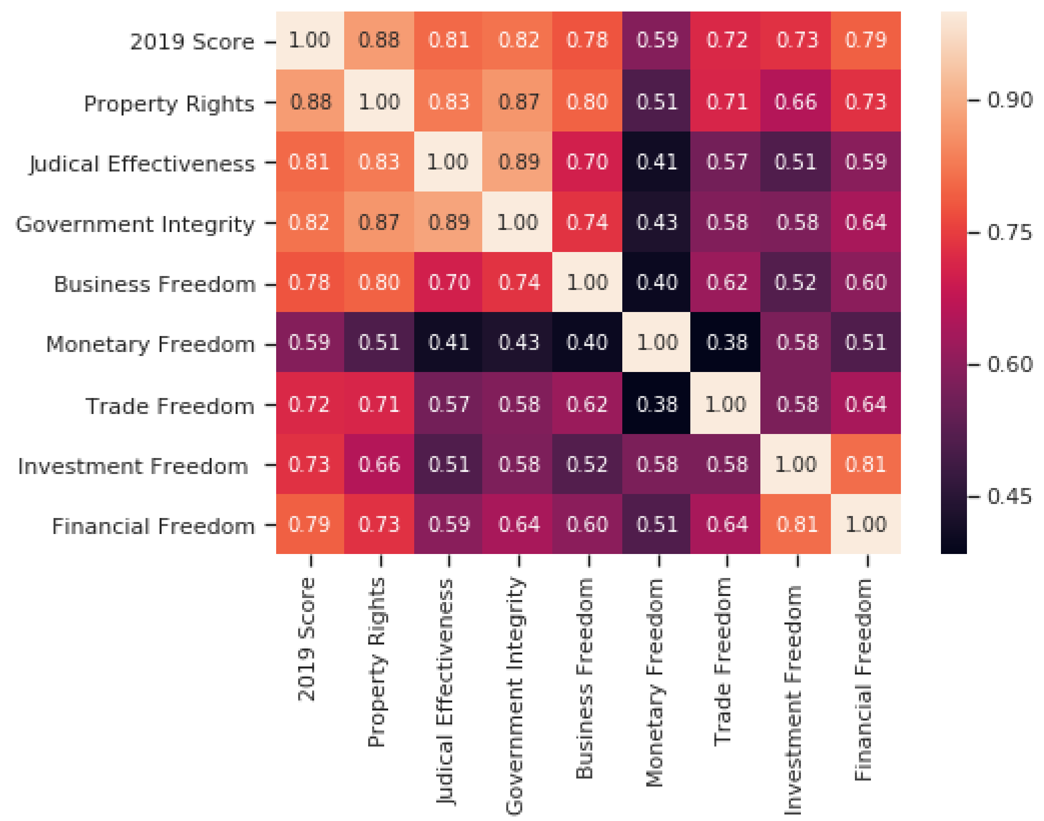
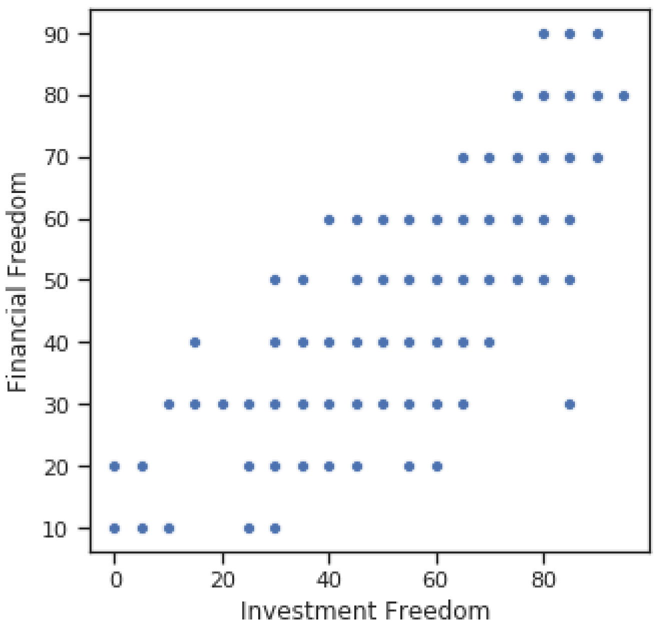


Диаграмма рассеиваний показывает зависимость двух признаков:

```
fig, ax = plt.subplots(figsize=(5,5))
sns.scatterplot(ax=ax, x='Investment Freedom ', y='Financial Freedom', data=data)
```

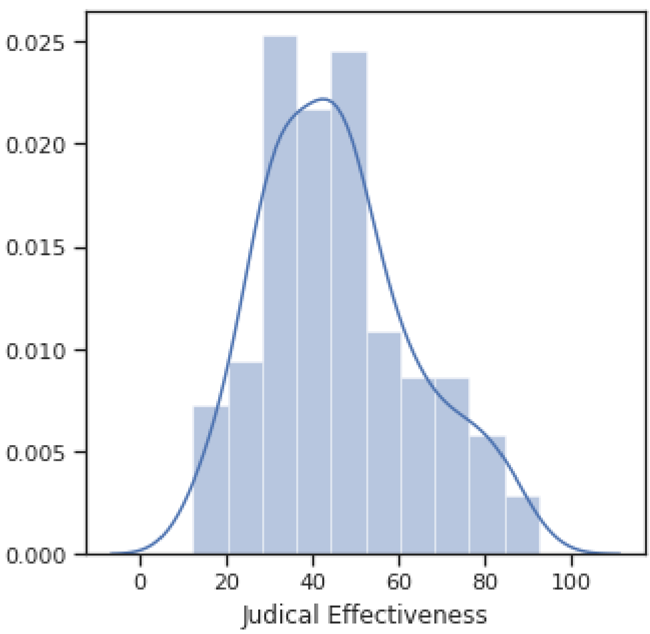
<matplotlib.axes._subplots.AxesSubplot at 0x7f4585515c50>



Видна почти линейная зависимость, обязательно попробуем линейную модель

```
#Гистограмма Позволяет оценить плотность вероятности распределения данных
fig, ax = plt.subplots(figsize=(5,5))
sns.distplot(data['Judical Effectiveness'])
```

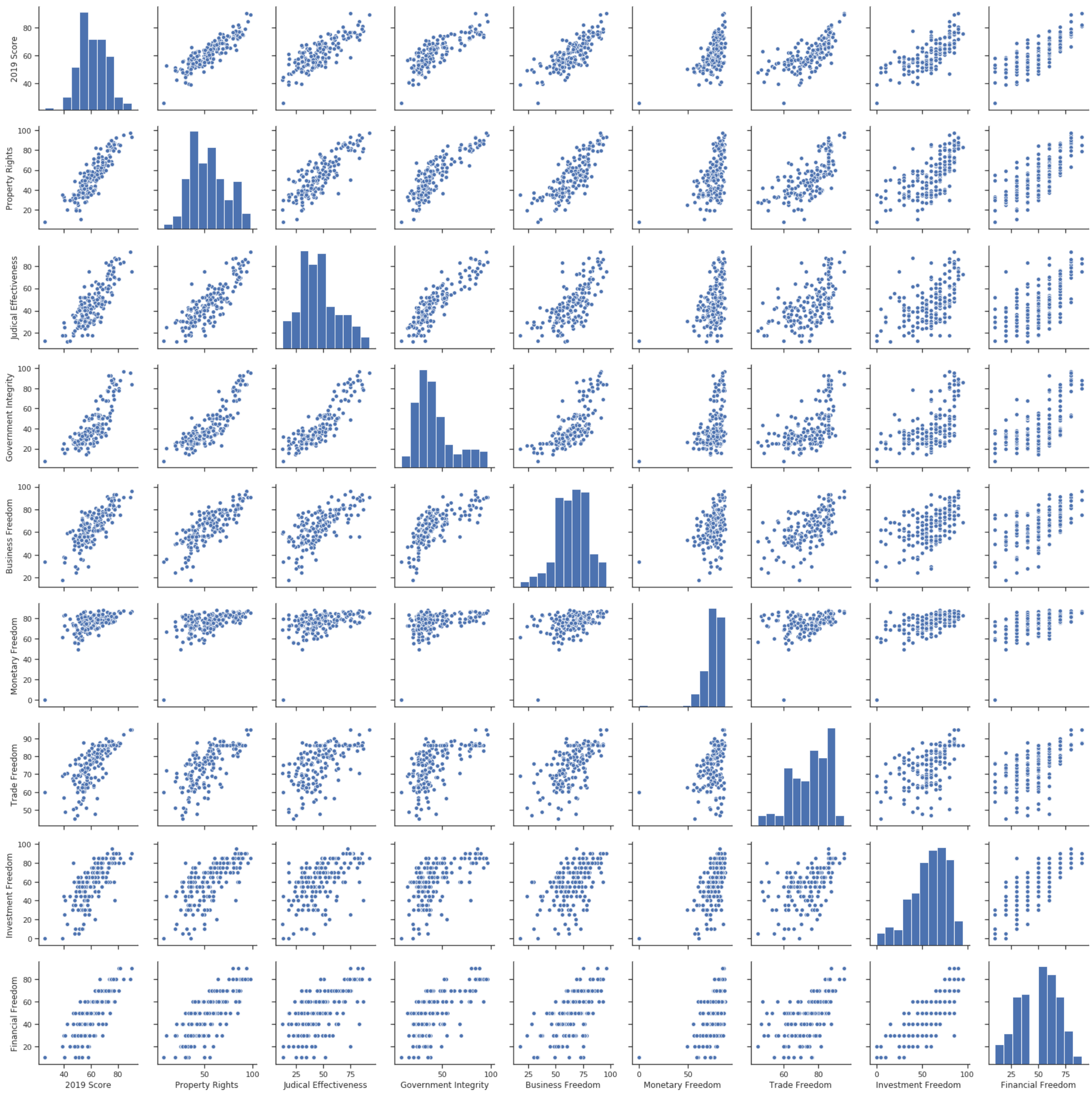
<matplotlib.axes._subplots.AxesSubplot at 0x7f45859ff320>



Можем посмотреть, как все признаки зависят между собой

```
sns.pairplot(data)
```

<seaborn.axisgrid.PairGrid at 0x7f45854bf4a8>



Опять заметно, что многие данные находятся в линейной зависимости

Разделение выборки

Для начала разделим целевой признак от остальных:

```
data_x = data[[
    '2019 Score',
    'Property Rights',
    'Judical Effectiveness',
    'Government Integrity',
    'Business Freedom',
    'Monetary Freedom',
    'Trade Freedom',
    'Investment Freedom ',
]]
data_y = data[['Financial Freedom']]
```

И теперь разделим на тренировочную выборку и тестовую, в тренировочной оставим 70% от всех данных

```
data_X_train, data_X_test, data_y_train, data_y_test = train_test_split(
    data_x, data_y, test_size=0.3, random_state=1)
data_X_train.shape, data_X_test.shape
```

```
((121, 8), (52, 8))
```

Метод ближайших соседей

Начнем с одного из самых простых методов.

Сначала попробуем обучать на основе двух ближайших соседей

```
KNN_1 = KNeighborsRegressor(n_neighbors=2)
KNN_1.fit(data_X_train, data_y_train)
target_KNN_1 = KNN_1.predict(data_X_test)
target_KNN_1
```

```
#средняя абсолютная ошибка при 2 сосядях
mean_absolute_error(data_y_test, target_KNN_1)
```

```
9.134615384615385
```

```
#средняя квадратичная ошибка при 2 сосядях
mean_squared_error(data_y_test, target_KNN_1)
```

```
130.28846153846155
```

```
median_absolute_error(data_y_test, target_KNN_1)
```

```
10.0
```

Теперь с помощью кросс-валидации подберем гиперпараметр:

```
n_range = np.array(range(5,55,5))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}]
```

```
clf_gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=5, scoring='neg_mean_ab
solute_error')
clf_gs.fit(data_X_train, data_y_train)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30,
                                           metric='minkowski',
                                           metric_params=None, n_jobs=None,
                                           n_neighbors=5, p=2,
                                           weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 4
0, 45, 50])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_absolute_error', verbose=0)
```

Самый лучший результат модель покажет при 5 ближайших соседях, можем в этом убедиться:

```
clf_gs.best_params_
```

```
{'n_neighbors': 5}
```

```
#10 ближайших соседей
KNN_2 = KNeighborsRegressor(n_neighbors=5)
KNN_2.fit(data_X_train, data_y_train)
target_KNN_2 = KNN_2.predict(data_X_test)
target_KNN_2
```

```
#средняя абсолютная ошибка при 5 сосядях
mean_absolute_error(data_y_test, target_KNN_2)
```

```
8.461538461538462
```

```
#средняя квадратичная ошибка при 5 сосядях
mean_squared_error(data_y_test, target_KNN_2)
```

```
110.92307692307692
```

```
median_absolute_error(data_y_test, target_KNN_2)
```

```
6.0
```

Средняя абсолютная и квадратичная ошибка стали намного меньше

Линейная модель

Многие данные находятся в линейной зависимости, поэтому попробуем линейную модель

```python
# Аналитическое вычисление коэффициентов регрессии
def analytic_regr_coef(x_array : np.ndarray,
                       y_array : np.ndarray) -> Tuple[float, float]:
    x_mean = np.mean(x_array)
    y_mean = np.mean(y_array)
    var1 = np.sum([(x-x_mean)**2 for x in x_array])
    cov1 = np.sum([(x-x_mean)*(y-x_mean) for x, y in zip(x_array, y_array)])
    b1 = cov1 / var1
    b0 = y_mean - b1*x_mean
    return b0, b1
```

Для начала найдем коэффициенты линейной зависимости и наглядно убедимся, насколько наша зависимость похожа на линейную

```python
x_array = data[['Investment Freedom ']]
y_array = data[['Financial Freedom']]
```

```python
df1 = pd.DataFrame(x_array)
df2 = pd.DataFrame(y_array)
```

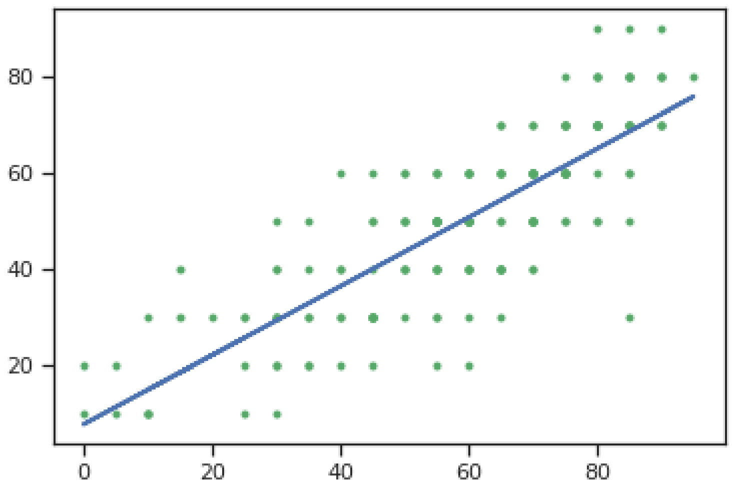
```python
b0, b1 = analytic_regr_coef(df1.values, df2.values)
b0, b1
```

```
(7.689215201011947, 0.7163064400813945)
```

```python
# Вычисление значений y на основе x для регрессии
def y_regr(x_array : np.ndarray, b0: float, b1: float) -> np.ndarray:
    res = [b1*x+b0 for x in x_array]
    return res
```

```python
y_array_regr = y_regr(df1.values, b0, b1)
```

```python
plt.plot(x_array, y_array, 'g.')
plt.plot(x_array, y_array_regr, 'b', linewidth=2.0)
plt.show()
```

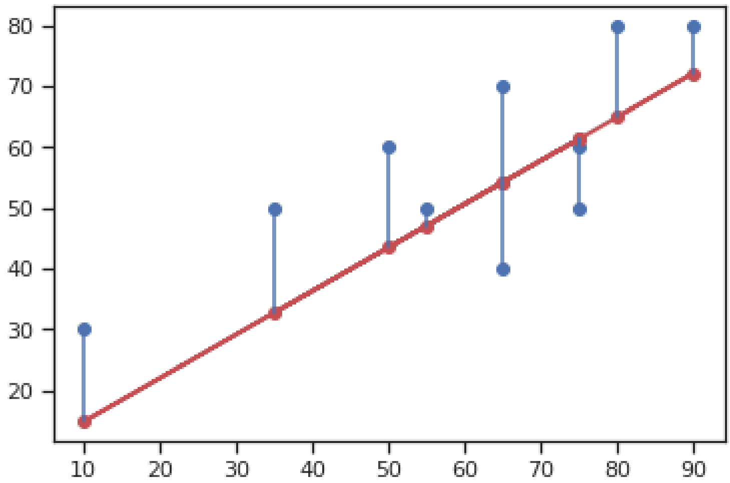


Можно посмотреть, насколько данные близко к линии. Синими отрезками показаны ошибки между истинными и предсказанными значениями

```
plt.plot(df1.values[104:114], df2.values[104:114], 'bo')
plt.plot(df1.values[104:114], y_array_regr[104:114], '-ro', linewidth=2.0)

for i in range(len(x_array[104:114])):
    x1 = df1.values[104:114][i]
    y1 = df2.values[104:114][i]
    y2 = y_array_regr[104:114][i]
    plt.plot([x1,x1],[y1,y2],'b-')

plt.show()
```



Попробуем обучить модель и предсказать значения:

```
reg1 = LinearRegression().fit(data_X_train, data_y_train)
```

```
target_LR_1 = reg1.predict(data_X_test)
```

```
mean_absolute_error(data_y_test, target_LR_1)
```

7.239789361129074

```
median_absolute_error(data_y_test, target_LR_1)
```

6.635003337887102 5

```
mean_squared_error(data_y_test, target_LR_1)
```

80.04097052577178

И до подбора гиперпараметров наша модель показывает отличный результат

```
model = LinearRegression()
parameters = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True, Fal
se]}
grid = GridSearchCV(model,parameters, cv=None)
grid.fit(data_X_train, data_y_train)
```

```
GridSearchCV(cv=None, error_score='raise-deprecating',
             estimator=LinearRegression(copy_X=True, fit_intercept=True,
                                        n_jobs=None, normalize=False),
             iid='warn', n_jobs=None,
             param_grid={'copy_X': [True, False],
                         'fit_intercept': [True, False],
                         'normalize': [True, False]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
grid.best_params_
```

```
{'copy_X': True, 'fit_intercept': False, 'normalize': True}
```

Кросс-валидация выбрала параметры

```
reg2 = LinearRegression(copy_X = True, fit_intercept = False, normalize = True).fit(data_
X_train, data_y_train)
```

```
target_LR_2 = reg2.predict(data_X_test)
```

```
mean_absolute_error(data_y_test, target_LR_2)
```

```
7.4015884967005166
```

```
median_absolute_error(data_y_test, target_LR_2)
```

```
6.673837214109131
```

```
mean_squared_error(data_y_test, target_LR_2)
```

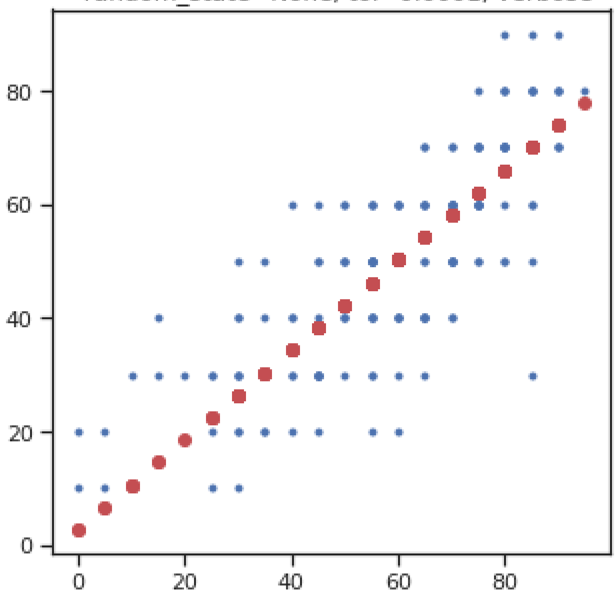
```
82.93157627713175
```

Метод опорных векторов

```
xx = df1.values
yy = df2.values
def plot_regr(clf):
    title = clf.__repr__
    clf.fit(xx.reshape(-1, 1), yy)
    y_pred = clf.predict(xx.reshape(-1, 1))
    fig, ax = plt.subplots(figsize=(5,5))
    ax.set_title(title)
    ax.plot(xx, yy, 'b.')
    ax.plot(xx, y_pred, 'ro')
    plt.show()
```

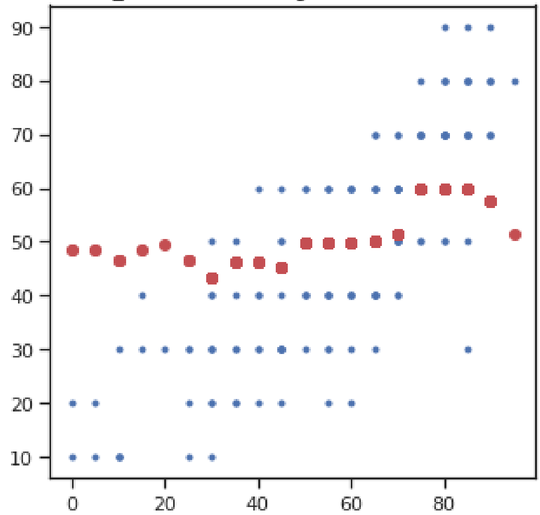
```
plot_regr(LinearSVR(C=1.0, max_iter=10000))
```

```
<bound method BaseEstimator.__repr__ of LinearSVR(C=1.0, dual=True, epsilon=0.0, fit_intercept=True,
     intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=10000,
     random_state=None, tol=0.0001, verbose=0)>
```




```
plot_regr(SVR(kernel='rbf', gamma=0.2, C=1.0))
```

```
<bound method BaseEstimator.__repr__ of SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.2,
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)>
```




```
svr_1 = SVR().fit(data_X_train, data_y_train)
```

```
target_SVR_1 = svr_1.predict(data_X_test)
```

```
mean_absolute_error(data_y_test, target_SVR_1)
```

```
14.308557310047084
```

```
median_absolute_error(data_y_test, target_SVR_1)
```

```
10.225001255894984
```

```
mean_squared_error(data_y_test, target_SVR_1)
```

```
335.1814231847919
```

```
param_grid = {'C':[1,10,100,1000],'gamma':[1,0.1,0.001,0.0001], 'kernel':['linear','rbf'
]}
grid = GridSearchCV(SVR(),param_grid,refit = True, verbose=2)
grid.fit(data_X_train, data_y_train)
```

```
grid.best_params_
```

```
{'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
```

```
svr_2 = SVR(C=1000, gamma = 1, kernel = 'linear').fit(data_X_train, data_y_train)
```

```
target_SVR_2 = svr_2.predict(data_X_test)
```

```
mean_absolute_error(data_y_test, target_SVR_2)
```

```
6.798416955338018
```

```
median_absolute_error(data_y_test, target_SVR_2)
```

```
5.702332230284814
```

```
mean_squared_error(data_y_test, target_SVR_2)
```

```
84.6924984134053
```

После подбора гиперпараметров метрики заметно улучшились. Посмотрим, как покажут себя более сложные модели

Ансамблевые модели

RandomForestRegressor

```
x_array = data[[
    '2019 Score',
    'Property Rights',
    'Judical Effectiveness',
    'Government Integrity',
    'Business Freedom',
    'Monetary Freedom',
    'Trade Freedom',
    'Investment Freedom ',
]]
y_array = data[['Financial Freedom']]

df1 = pd.DataFrame(x_array)
df2 = pd.DataFrame(y_array)
```

```python
rf_rg_1 = RandomForestRegressor(random_state=1)
rf_rg_1.fit(data_X_train, data_y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=1, verbose=
0,
                      warm_start=False)
```

```python
target_RFR_1 = rf_rg_1.predict(data_X_test)
```

```python
mean_absolute_error(data_y_test, target_RFR_1)
```

```
8.461538461538462
```

```python
median_absolute_error(data_y_test, target_RFR_1)
```

```
7.0
```

```python
mean_squared_error(data_y_test, target_RFR_1)
```

```
108.03846153846153
```

```python
tuned_parameters = {'n_estimators': [500, 700, 1000], 'max_depth': [None, 1, 2, 3]}
```

```python
CV_rfr = GridSearchCV(RandomForestRegressor(), param_grid=tuned_parameters, cv=5, n_jobs=
-1, verbose=1)
```

```python
CV_rfr.fit(data_X_train, data_y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators='warn', n_jobs=Non
e,

                                             oob_score=False, random_state=No
ne,
                                             verbose=0, warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [None, 1, 2, 3],
                         'n_estimators': [500, 700, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=1)
```

```python
CV_rfr.best_params_
```

```
{'max_depth': None, 'n_estimators': 700}
```

```
rf_rg_2 = RandomForestRegressor(random_state=1, max_depth = 3, n_estimators = 500)
rf_rg_2.fit(data_X_train, data_y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=3,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=500,
                      n_jobs=None, oob_score=False, random_state=1, verbose=
0,
                      warm_start=False)
```

```
target_RFR_2 = rf_rg_2.predict(data_X_test)
```

```
mean_absolute_error(data_y_test, target_RFR_2)
```

```
7.977509135331688
```

```
median_absolute_error(data_y_test,  target_RFR_2)
```

```
7.28687854808927
```

```
mean_squared_error(data_y_test,  target_RFR_2)
```

```
88.33672593229464
```

AdaBoost

```
ab_1 = AdaBoostRegressor(random_state=1, base_estimator = RandomForestRegressor(random_st
ate=1, max_depth = 3, n_estimators = 500))
ab_1.fit(data_X_train, data_y_train)
```

```
AdaBoostRegressor(base_estimator=RandomForestRegressor(bootstrap=True,
                                                       criterion='mse',
                                                       max_depth=3,
                                                       max_features='auto',
                                                       max_leaf_nodes=None,
                                                       min_impurity_decrease=
0.0,
                                                       min_impurity_split=Non
e,
                                                       min_samples_leaf=1,
                                                       min_samples_split=2,
                                                       min_weight_fraction_le
af=0.0,
                                                       n_estimators=500,
                                                       n_jobs=None,
                                                       oob_score=False,
                                                       random_state=1,
                                                       verbose=0,
                                                       warm_start=False),
                  learning_rate=1.0, loss='linear', n_estimators=50,
                  random_state=1)
```

```
target_AB_1 = ab_1.predict(data_X_test)
```

```
mean_absolute_error(data_y_test, target_AB_1)
```

```
8.493196077288827
```

```
median_absolute_error(data_y_test, target_AB_1)
```

```
7.588438957313464
```

```
mean_squared_error(data_y_test, target_AB_1)
```

```
105.80526124111854
```

```
parameters = {'n_estimators': (1, 2), 'base_estimator__max_depth': (1, 2)}
```

```
CV_ab = GridSearchCV(ab_1, parameters)
```

```
CV_ab.fit(data_X_train, data_y_train)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=AdaBoostRegressor(base_estimator=RandomForestRegressor
(bootstrap=True,

criterion='mse',

max_depth=3,

max_features='auto',

max_leaf_nodes=None,

min_impurity_decrease=0.0,

min_impurity_split=None,

min_samples_leaf=1,

min_samples_split=2,

min_weight_fraction_leaf=0.0,

n_estimators=500,

n_jobs=None,

oob_score=False,

random_state=1,

verbose=0,

warm_start=False),
                                         learning_rate=1.0, loss='linear',
                                         n_estimators=50, random_state=1),
             iid='warn', n_jobs=None,
             param_grid={'base_estimator__max_depth': (1, 2),
                         'n_estimators': (1, 2)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
CV_ab.best_params_
```

```
{'base_estimator__max_depth': 2, 'n_estimators': 1}
```

```
ab_2 = AdaBoostRegressor(random_state=1, base_estimator = rf_rg_2, n_estimators = 1)
ab_2.fit(data_X_train, data_y_train)
```

```
AdaBoostRegressor(base_estimator=RandomForestRegressor(bootstrap=True,
                                                       criterion='mse',
                                                       max_depth=3,
                                                       max_features='auto',
                                                       max_leaf_nodes=None,
                                                       min_impurity_decrease=
0.0,
                                                       min_impurity_split=Non
e,
                                                       min_samples_leaf=1,
                                                       min_samples_split=2,
                                                       min_weight_fraction_le
af=0.0,
                                                       n_estimators=500,
                                                       n_jobs=None,
                                                       oob_score=False,
                                                       random_state=1,
                                                       verbose=0,
                                                       warm_start=False),
                  learning_rate=1.0, loss='linear', n_estimators=1,
                  random_state=1)
```

```
target_AB_2 = ab_2.predict(data_X_test)
```

```
mean_absolute_error(data_y_test, target_AB_2)
```

8.27117863166269

```
median_absolute_error(data_y_test, target_AB_2)
```

7.103907753721927

```
mean_squared_error(data_y_test, target_AB_2)
```

107.7196807155559

Метод группового учета аргументов

```
from gmdhpy import gmdh
```

```
from gmdhpy.gmdh import MultilayerGMDH
gmdh_1 = MultilayerGMDH()
```

```
gmdh_1.fit(data_X_train, data_y_train)
target_GMDH_1 = gmdh_1.predict(data_X_test)
```

```
train layer0 in 0.03 sec
train layer1 in 0.09 sec
train layer2 in 0.08 sec
train layer3 in 0.10 sec
train layer4 in 0.08 sec
train layer5 in 0.08 sec
train layer6 in 0.09 sec
train layer7 in 0.08 sec
```

```
mean_absolute_error(data_y_test, target_GMDH_1)
```

```
6.623325613949651
```

```
median_absolute_error(data_y_test, target_GMDH_1)
```

```
6.152133536715617
```

```
mean_squared_error(data_y_test, target_GMDH_1)
```

```
68.52805863320974
```

```
gmdh_2 = MultilayerGMDH(ref_functions=('linear_cov', 'quadratic', 'cubic', 'linear'))
gmdh_2.fit(data_X_train, data_y_train)
target_GMDH_2 = gmdh_2.predict(data_X_test)
```

```
train layer0 in 0.12 sec
train layer1 in 0.41 sec
train layer2 in 0.37 sec
train layer3 in 0.40 sec
train layer4 in 0.38 sec
train layer5 in 0.47 sec
train layer6 in 0.37 sec
train layer7 in 0.37 sec
train layer8 in 0.39 sec
train layer9 in 0.39 sec
train layer10 in 0.38 sec
train layer11 in 0.40 sec
```

```
mean_absolute_error(data_y_test, target_GMDH_2)
```

```
7.3591166939950154
```

```
median_absolute_error(data_y_test, target_GMDH_2)
```

```
6.445849941624356
```

```
mean_squared_error(data_y_test, target_GMDH_2)
```

```
80.67467130677497
```

Анализ

```python
#1 - KNN
#2 - Линейная
#3 - Опорные векторы
#4 - Случайный лес
#5 - AdaBoost
#6 - Метод группового учета аргументов
```

```python
d2 = [{"model_№": 1, "model": "KNN", "mean_absolute_error" : mean_absolute_error(data_y_test, target_KNN_2), "median_absolute_error": median_absolute_error(data_y_test, target_KNN_2),
     "mean_squared_error": mean_squared_error(data_y_test, target_KNN_2)}, {"model_№": 2, "model": "LR","mean_absolute_error" : mean_absolute_error(data_y_test, target_LR_2), "median_absolute_error": median_absolute_error(data_y_test, target_LR_2),
     "mean_squared_error": mean_squared_error(data_y_test, target_LR_2)}, {"model_№": 3, "model": "SVR", "mean_absolute_error" : mean_absolute_error(data_y_test, target_SVR_2), "median_absolute_error": median_absolute_error(data_y_test, target_SVR_2),
     "mean_squared_error": mean_squared_error(data_y_test, target_SVR_2)}, {"model_№": 4, "model": "RFR", "mean_absolute_error" : mean_absolute_error(data_y_test, target_RFR_2), "median_absolute_error": median_absolute_error(data_y_test, target_RFR_2),
     "mean_squared_error": mean_squared_error(data_y_test, target_RFR_2)}, {"model_№": 5, "model": "AB", "mean_absolute_error" : mean_absolute_error(data_y_test, target_AB_2), "median_absolute_error": median_absolute_error(data_y_test, target_AB_2),
     "mean_squared_error": mean_squared_error(data_y_test, target_AB_2)}, {"model_№": 6, "model": "GMDH", "mean_absolute_error" : mean_absolute_error(data_y_test, target_GMDH_2), "median_absolute_error": median_absolute_error(data_y_test, target_GMDH_2),
     "mean_squared_error": mean_squared_error(data_y_test, target_GMDH_2)}  ]
```

```python
dd2 = pd.DataFrame(d2)
```

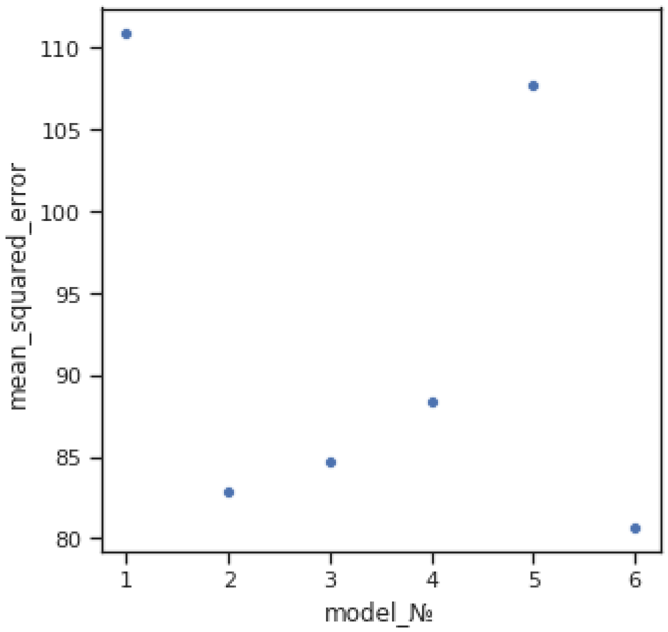
```python
print(dd2)
```

```
   mean_absolute_error  mean_squared_error  median_absolute_error model  \
0             8.461538          110.923077               6.000000   KNN   
1             7.401588           82.931576               6.673837    LR   
2             6.798417           84.692498               5.702332   SVR   
3             7.977509           88.336726               7.286879   RFR   
4             8.271179          107.719681               7.103908    AB   
5             7.359117           80.674671               6.445850  GMDH   

   model_№  
0        1  
1        2  
2        3  
3        4  
4        5  
5        6  
```

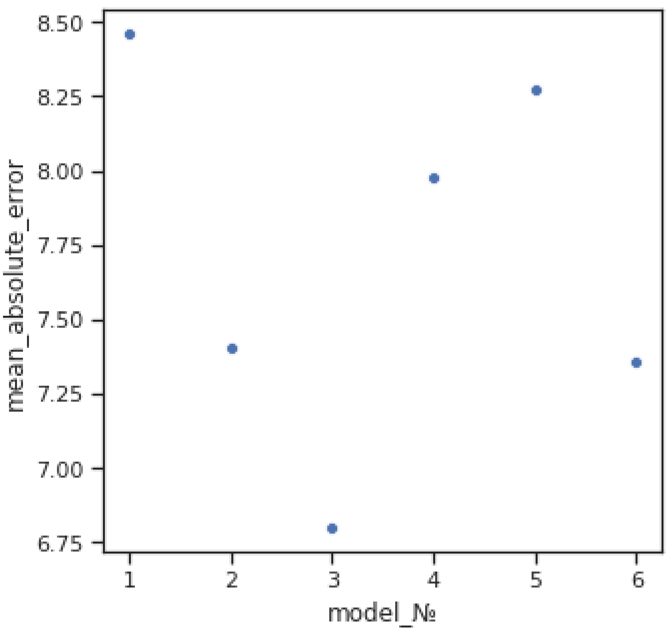
```
fig, ax = plt.subplots(figsize=(5,5))
sns.scatterplot(ax=ax, x='model_№', y='mean_squared_error', data=dd2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f45c01356d8>
```



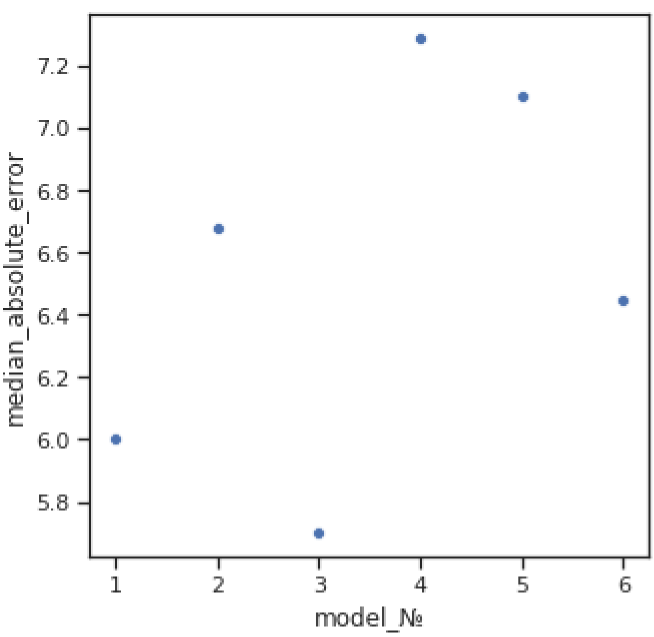

```
fig, ax = plt.subplots(figsize=(5,5))
sns.scatterplot(ax=ax, x='model_№', y='mean_absolute_error', data=dd2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4582708390>
```



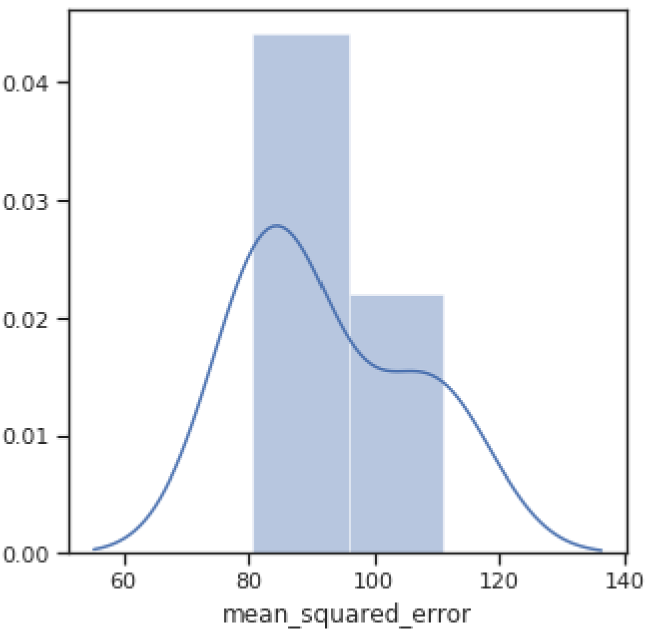
```python
fig, ax = plt.subplots(figsize=(5,5))
sns.scatterplot(ax=ax, x='model_№', y='median_absolute_error', data=dd2)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f45c0050780>



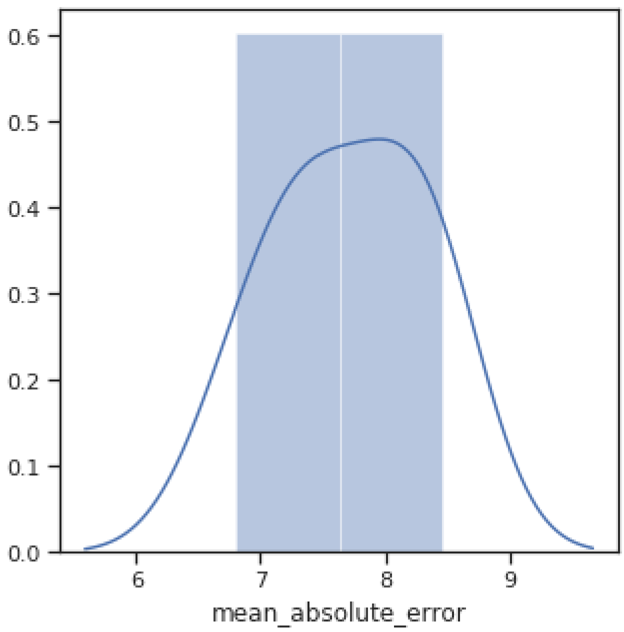

```python
#Гистограмма Позволяет оценить плотность вероятности распределения данных
fig, ax = plt.subplots(figsize=(5,5))
sns.distplot(dd2['mean_squared_error'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f458271b898>



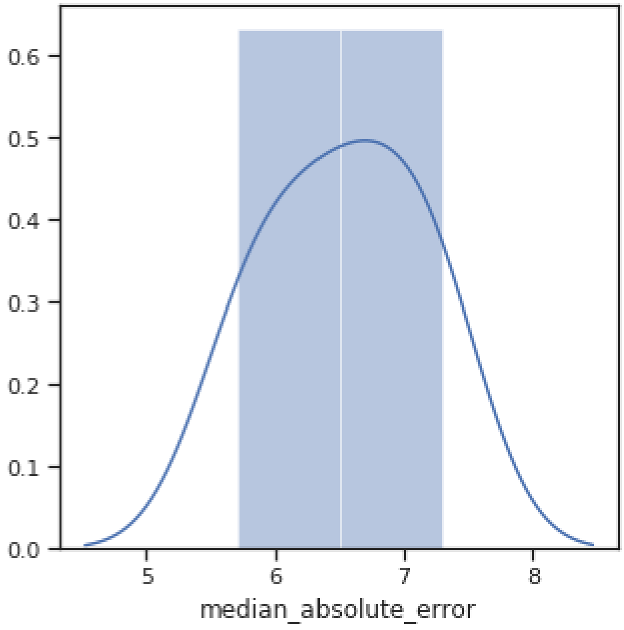
```
fig, ax = plt.subplots(figsize=(5,5))
sns.distplot(dd2['mean_absolute_error'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f45bf784e10>



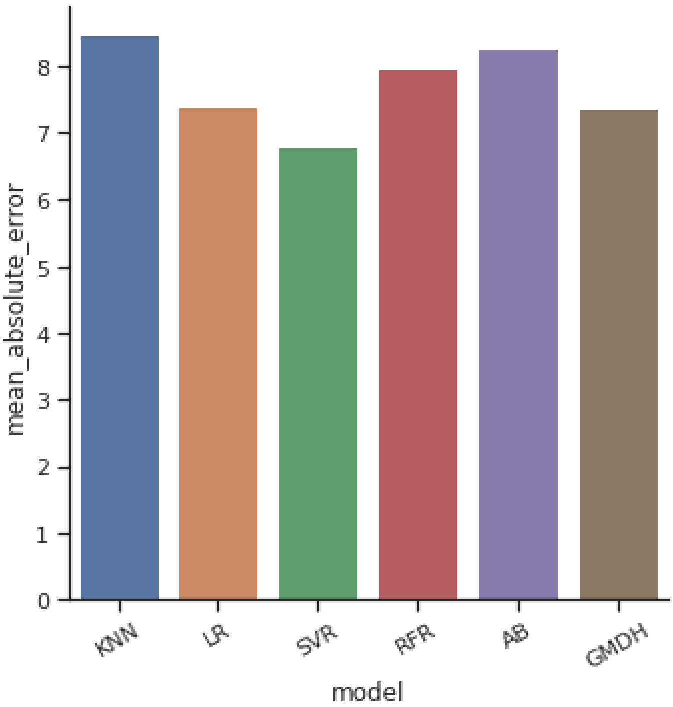

```
fig, ax = plt.subplots(figsize=(5,5))
sns.distplot(dd2['median_absolute_error'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f45bf757f28>



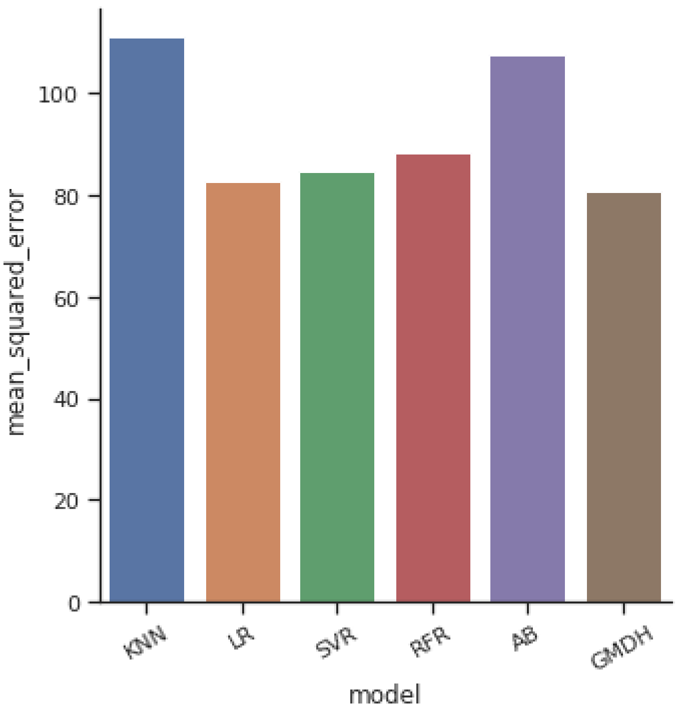
```
g = sns.factorplot(x='model'
                   ,y= 'mean_absolute_error'
                   ,data=dd2
                   ,kind='bar'
                   )
g.set_xticklabels(rotation=30)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f45bf72f748>
```




```
g = sns.factorplot(x='model'
                   ,y= 'mean_squared_error'
                   ,data=dd2
                   ,kind='bar'
                   )
g.set_xticklabels(rotation=30)
```

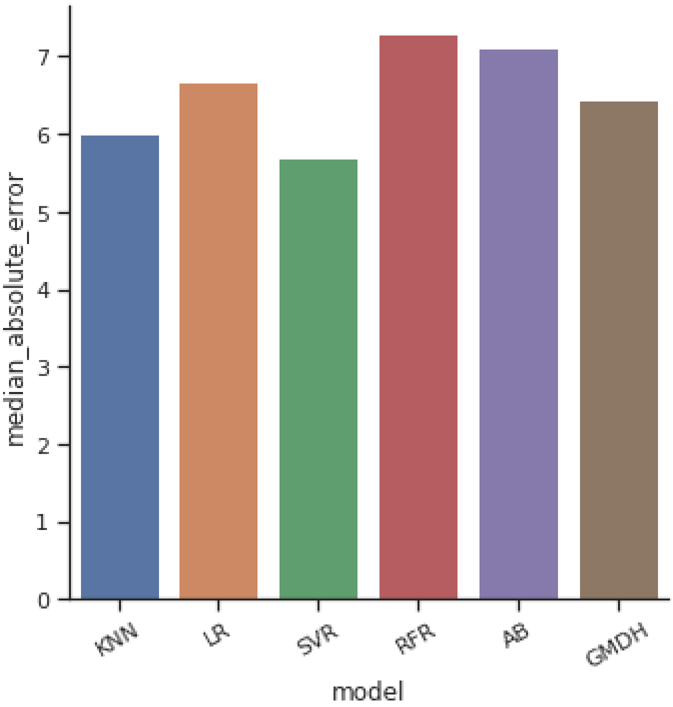
```
<seaborn.axisgrid.FacetGrid at 0x7f45bf610a58>
```



```python
g = sns.factorplot(x='model'
                   ,y= 'median_absolute_error'
                   ,data=dd2
                   ,kind='bar'

                   )
g.set_xticklabels(rotation=30)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f45bf56f6d8>
```




```python
print(dd2['mean_absolute_error'].describe())
```

```
count    6.000000
mean     7.711558
std      0.632534
min      6.798417
25%      7.369735
50%      7.689549
75%      8.197761
max      8.461538
Name: mean_absolute_error, dtype: float64
```

```python
print(dd2['median_absolute_error'].describe())
```

```
count    6.000000
mean     6.535468
std      0.615753
min      5.702332
25%      6.111462
50%      6.559844
75%      6.996390
max      7.286879
Name: median_absolute_error, dtype: float64
```

```python
print(dd2['mean_squared_error'].describe())
```

```
count      6.000000
mean      92.546372
std       13.271915
min       80.674671
25%       83.371807
50%       86.514612
75%      102.873942
max      110.923077
Name: mean_squared_error, dtype: float64
```