Московский государственный технический

университет им. Н. Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Курс «Технологии машинного обучения»

Отчет по лабораторной работе №5

Линейные модели, SVM и деревья решений

Группа: РТ5-61

Студент: Савушкин Д. А.

Преподаватель: Гапанюк Ю. Е.

Москва, 2020 г.

Цель лабораторной работы: изучение линейных моделей, SVM и деревьев решений.

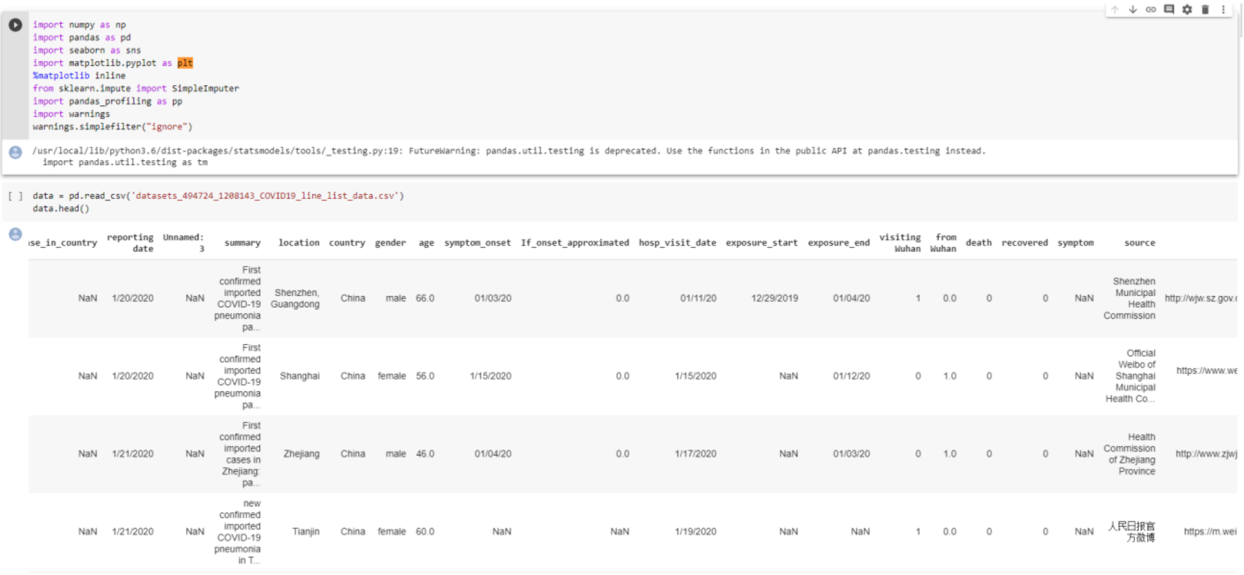
Задание:

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
4. Обучите следующие модели:
   - 4.1. одну из линейных моделей;
   - 4.2. SVM;
   - 4.3. дерево решений.
5. Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

Дополнительные задания:

1. Проведите эксперименты с важностью признаков в дереве решений.
2. Визуализируйте дерево решений.

Текст программы и экранные формы с примерами выполнения программы:

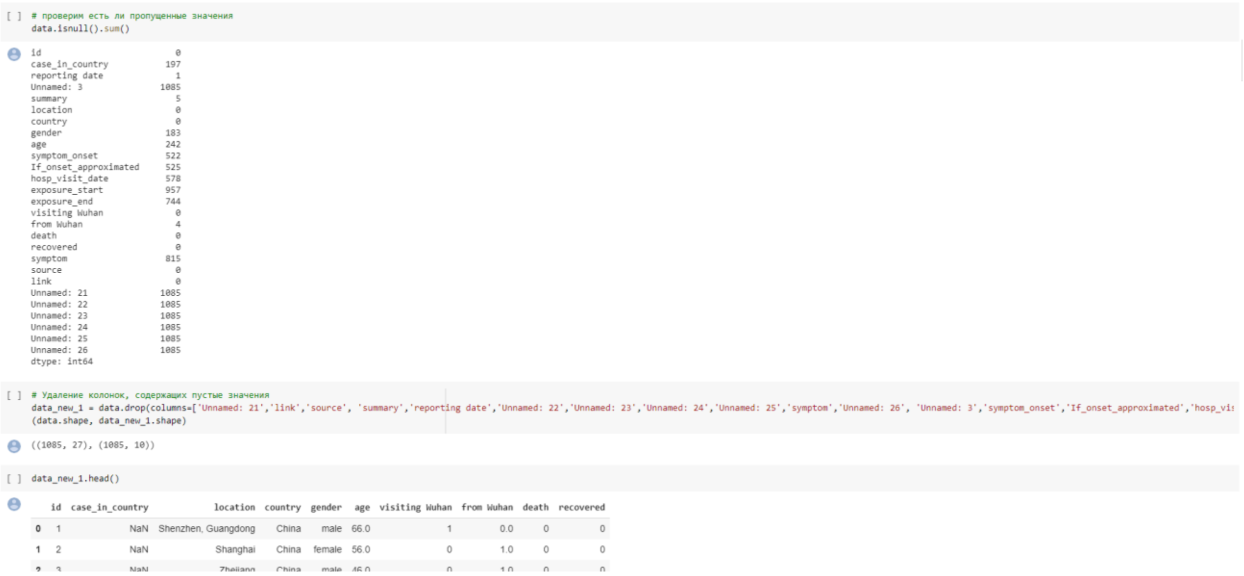


```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.impute import SimpleImputer
import pandas_profiling as pp
import warnings
warnings.simplefilter("ignore")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

```
data = pd.read_csv('datasets_494724_1208143_COVID19_line_list_data.csv')
data.head()
```

se_in_country	reporting date	Unnamed: 3	summary	location	country	gender	age	symptom_onset	If_onset_approximated	hosp_visit_date	exposure_start	exposure_end	visiting Wuhan	from Wuhan	death	recovered	symptom	source	
NaN	1/20/2020	NaN	First confirmed imported COVID-19 pneumonia pa...	Shenzhen, Guangdong	China	male	66.0	01/03/20	0.0	01/11/20	12/29/2019	01/04/20	1	0.0	0	0	NaN	Shenzhen Municipal Health Commission	http://wjw.sz.gov.c
NaN	1/20/2020	NaN	First confirmed imported COVID-19 pneumonia pa...	Shanghai	China	female	56.0	1/15/2020	0.0	1/15/2020	NaN	01/12/20	0	1.0	0	0	NaN	Official Weibo of Shanghai Municipal Health Co...	https://www.we
NaN	1/21/2020	NaN	First confirmed imported cases in Zhejiang: pa...	Zhejiang	China	male	46.0	01/04/20	0.0	1/17/2020	NaN	01/03/20	0	1.0	0	0	NaN	Health Commission of Zhejiang Province	http://www.zjwj
NaN	1/21/2020	NaN	new confirmed imported COVID-19 pneumonia in T...	Tianjin	China	female	60.0	NaN	NaN	1/19/2020	NaN	NaN	1	0.0	0	0	NaN	人民日报官方微博	https://m.wei



```
[ ] # проверим есть ли пропущенные значения
    data.isnull().sum()
```

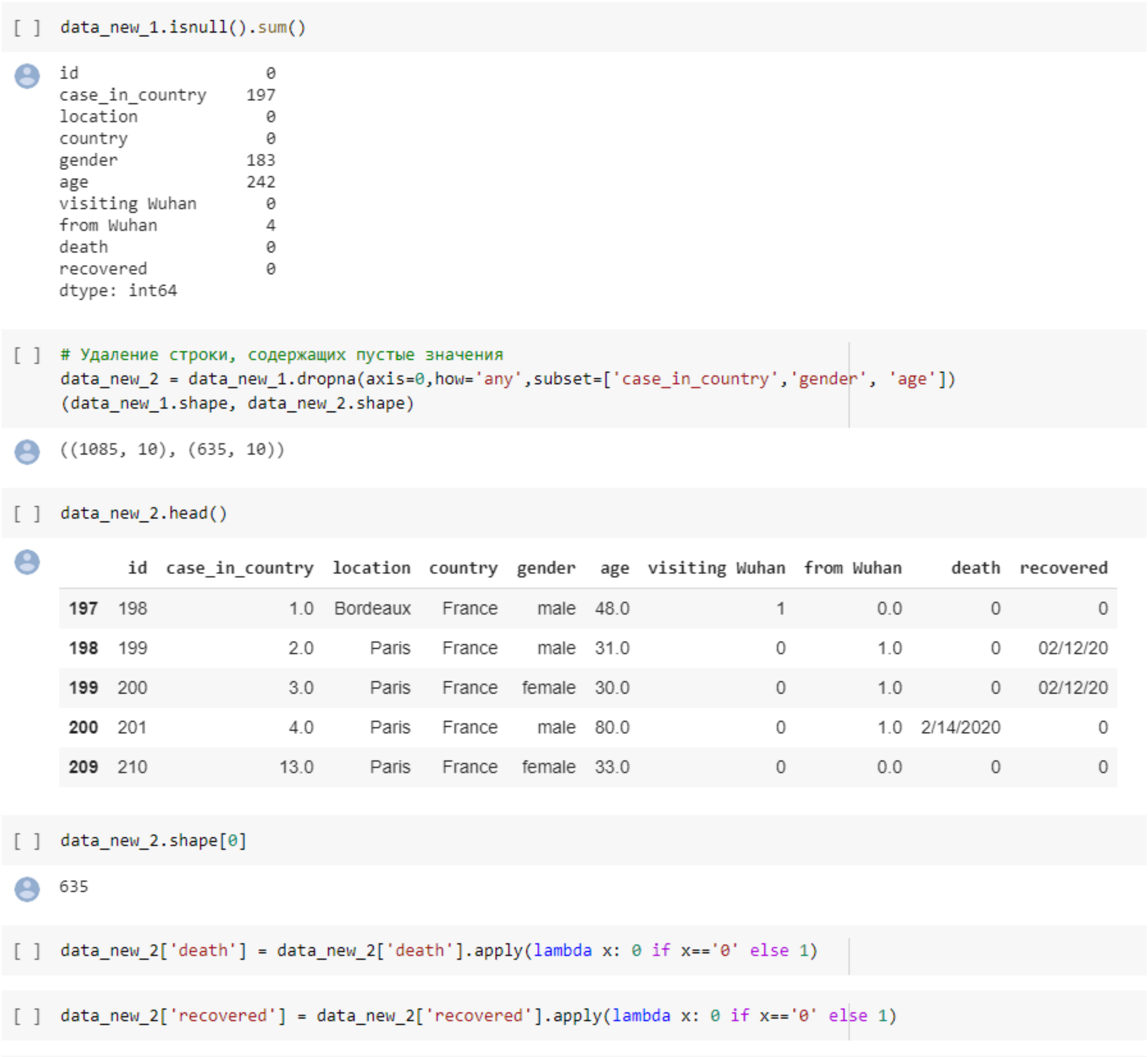
```
id                        0
case_in_country         197
reporting date            1
Unnamed: 3             1085
summary                   5
location                  0
country                   0
gender                  183
age                     242
symptom_onset           522
If_onset_approximated   525
hosp_visit_date         578
exposure_start          957
exposure_end            744
visiting Wuhan            0
from Wuhan                4
death                     0
recovered                 0
symptom                 815
source                    0
link                      0
Unnamed: 21            1085
Unnamed: 22            1085
Unnamed: 23            1085
Unnamed: 24            1085
Unnamed: 25            1085
Unnamed: 26            1085
dtype: int64
```

```
[ ] # Удаление колонок, содержащих пустые значения
    data_new_1 = data.drop(columns=['Unnamed: 21','link','source', 'summary','reporting date','Unnamed: 22','Unnamed: 23','Unnamed: 24','Unnamed: 25','symptom','Unnamed: 26', 'Unnamed: 3','symptom_onset','If_onset_approximated','hosp_vis
    (data.shape, data_new_1.shape)
```

```
((1085, 27), (1085, 10))
```

```
[ ] data_new_1.head()
```

	id	case_in_country	location	country	gender	age	visiting Wuhan	from Wuhan	death	recovered
0	1	NaN	Shenzhen, Guangdong	China	male	66.0	1	0.0	0	0
1	2	NaN	Shanghai	China	female	56.0	0	1.0	0	0
2	3	NaN	Zhejiang	China	male	46.0	0	1.0	0	0



```
[ ] data_new_1.isnull().sum()
```

```
id                  0
case_in_country   197
location            0
country             0
gender            183
age               242
visiting Wuhan      0
from Wuhan          4
death               0
recovered           0
dtype: int64
```

```
[ ] # Удаление строки, содержащих пустые значения
    data_new_2 = data_new_1.dropna(axis=0,how='any',subset=['case_in_country','gender', 'age'])
    (data_new_1.shape, data_new_2.shape)
```

```
((1085, 10), (635, 10))
```

```
[ ] data_new_2.head()
```

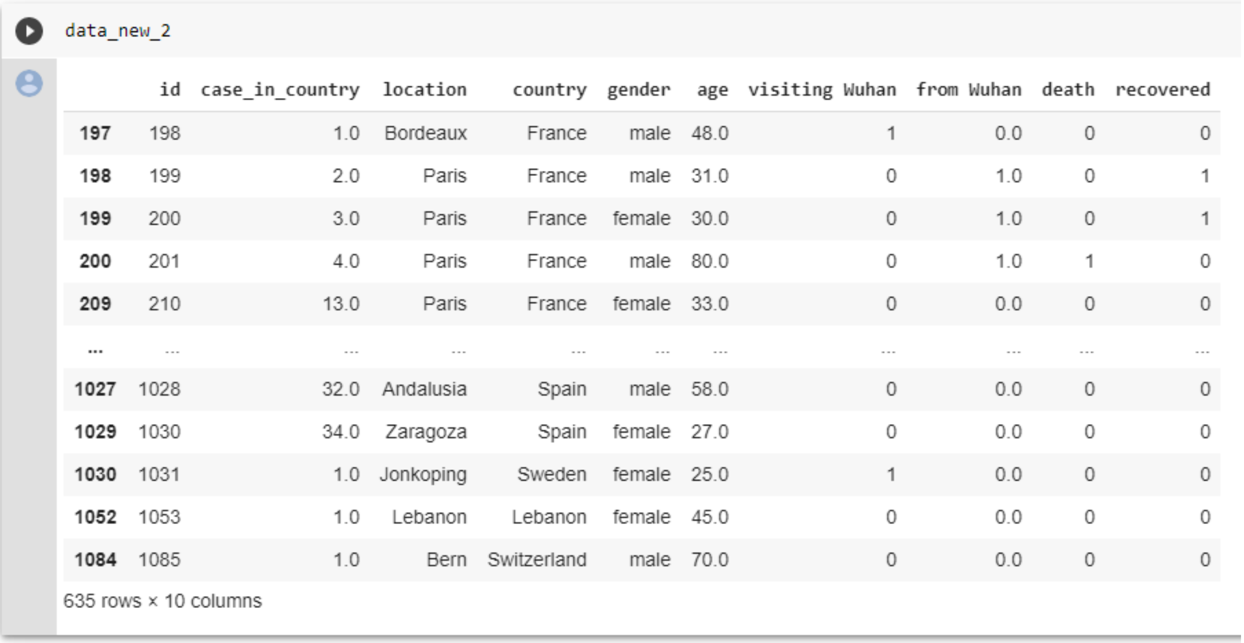
	id	case_in_country	location	country	gender	age	visiting Wuhan	from Wuhan	death	recovered
197	198	1.0	Bordeaux	France	male	48.0	1	0.0	0	0
198	199	2.0	Paris	France	male	31.0	0	1.0	0	02/12/20
199	200	3.0	Paris	France	female	30.0	0	1.0	0	02/12/20
200	201	4.0	Paris	France	male	80.0	0	1.0	2/14/2020	0
209	210	13.0	Paris	France	female	33.0	0	0.0	0	0

```
[ ] data_new_2.shape[0]
```

```
635
```

```
[ ] data_new_2['death'] = data_new_2['death'].apply(lambda x: 0 if x=='0' else 1)
```

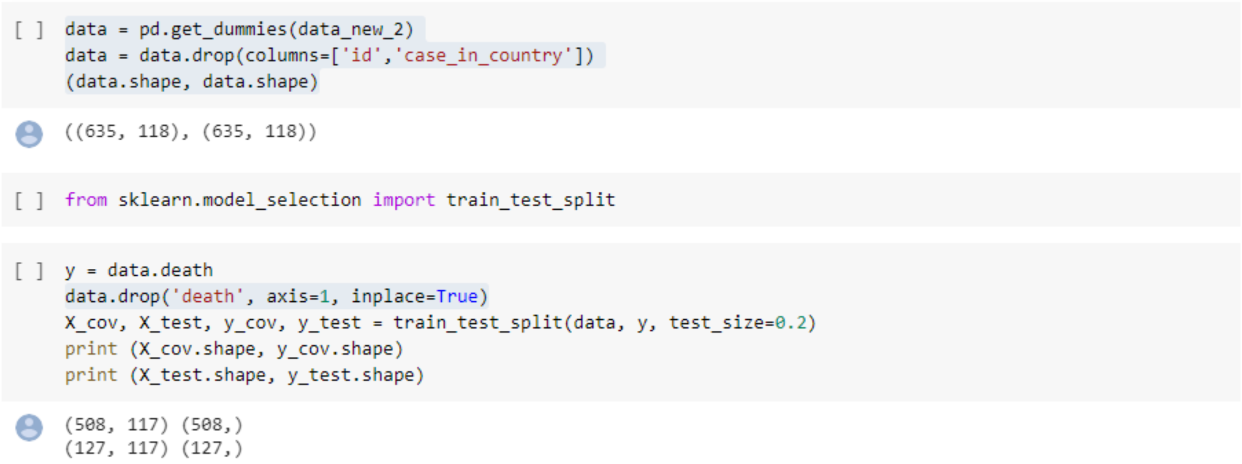
```
[ ] data_new_2['recovered'] = data_new_2['recovered'].apply(lambda x: 0 if x=='0' else 1)
```



```
data_new_2
```

	id	case_in_country	location	country	gender	age	visiting Wuhan	from Wuhan	death	recovered
197	198	1.0	Bordeaux	France	male	48.0	1	0.0	0	0
198	199	2.0	Paris	France	male	31.0	0	1.0	0	1
199	200	3.0	Paris	France	female	30.0	0	1.0	0	1
200	201	4.0	Paris	France	male	80.0	0	1.0	1	0
209	210	13.0	Paris	France	female	33.0	0	0.0	0	0
...	...	...	...	...	...	...	...	...	...	...
1027	1028	32.0	Andalusia	Spain	male	58.0	0	0.0	0	0
1029	1030	34.0	Zaragoza	Spain	female	27.0	0	0.0	0	0
1030	1031	1.0	Jonkoping	Sweden	female	25.0	1	0.0	0	0
1052	1053	1.0	Lebanon	Lebanon	female	45.0	0	0.0	0	0
1084	1085	1.0	Bern	Switzerland	male	70.0	0	0.0	0	0

635 rows × 10 columns



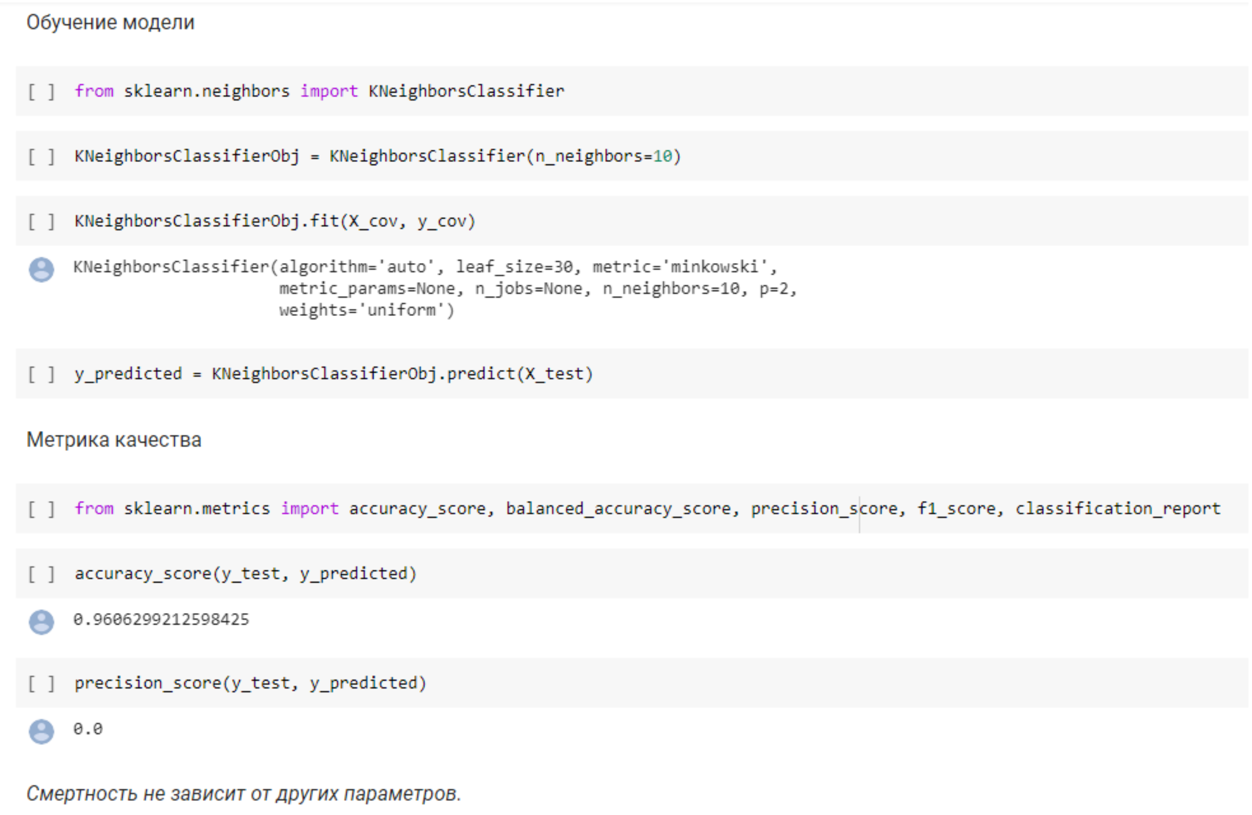
```
[ ]  data = pd.get_dummies(data_new_2)
     data = data.drop(columns=['id','case_in_country'])
     (data.shape, data.shape)
```

```
((635, 118), (635, 118))
```

```
[ ]  from sklearn.model_selection import train_test_split
```

```
[ ]  y = data.death
     data.drop('death', axis=1, inplace=True)
     X_cov, X_test, y_cov, y_test = train_test_split(data, y, test_size=0.2)
     print (X_cov.shape, y_cov.shape)
     print (X_test.shape, y_test.shape)
```

```
(508, 117) (508,)
(127, 117) (127,)
```



Обучение модели

```
[ ] from sklearn.neighbors import KNeighborsClassifier
```

```
[ ] KNeighborsClassifierObj = KNeighborsClassifier(n_neighbors=10)
```

```
[ ] KNeighborsClassifierObj.fit(X_cov, y_cov)
```

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

```
[ ] y_predicted = KNeighborsClassifierObj.predict(X_test)
```

Метрика качества

```
[ ] from sklearn.metrics import accuracy_score, balanced_accuracy_score, precision_score, f1_score, classification_report
```

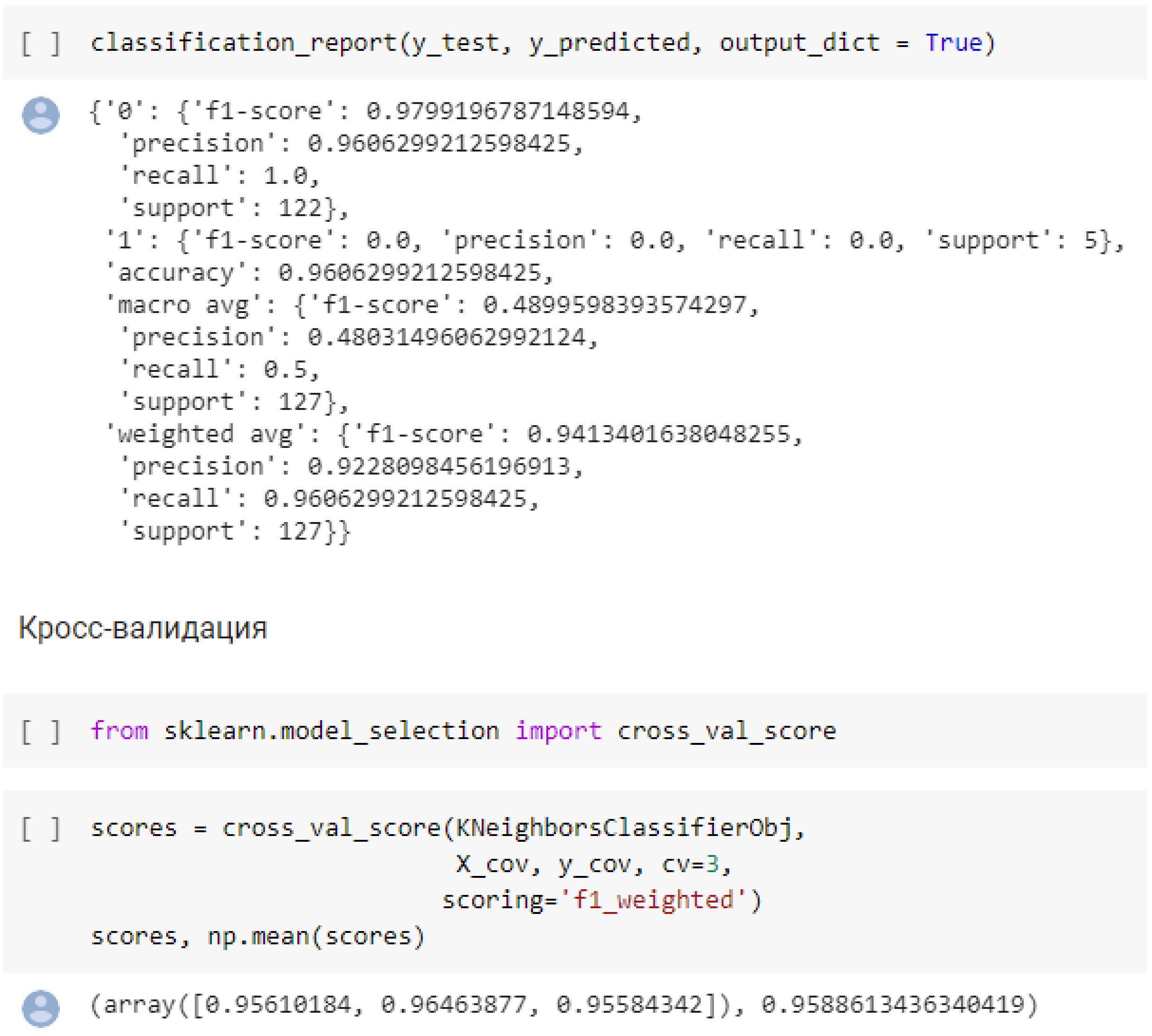
```
[ ] accuracy_score(y_test, y_predicted)
```

```
0.9606299212598425
```

```
[ ] precision_score(y_test, y_predicted)
```

```
0.0
```

Смертность не зависит от других параметров.



```
[ ] classification_report(y_test, y_predicted, output_dict = True)
```

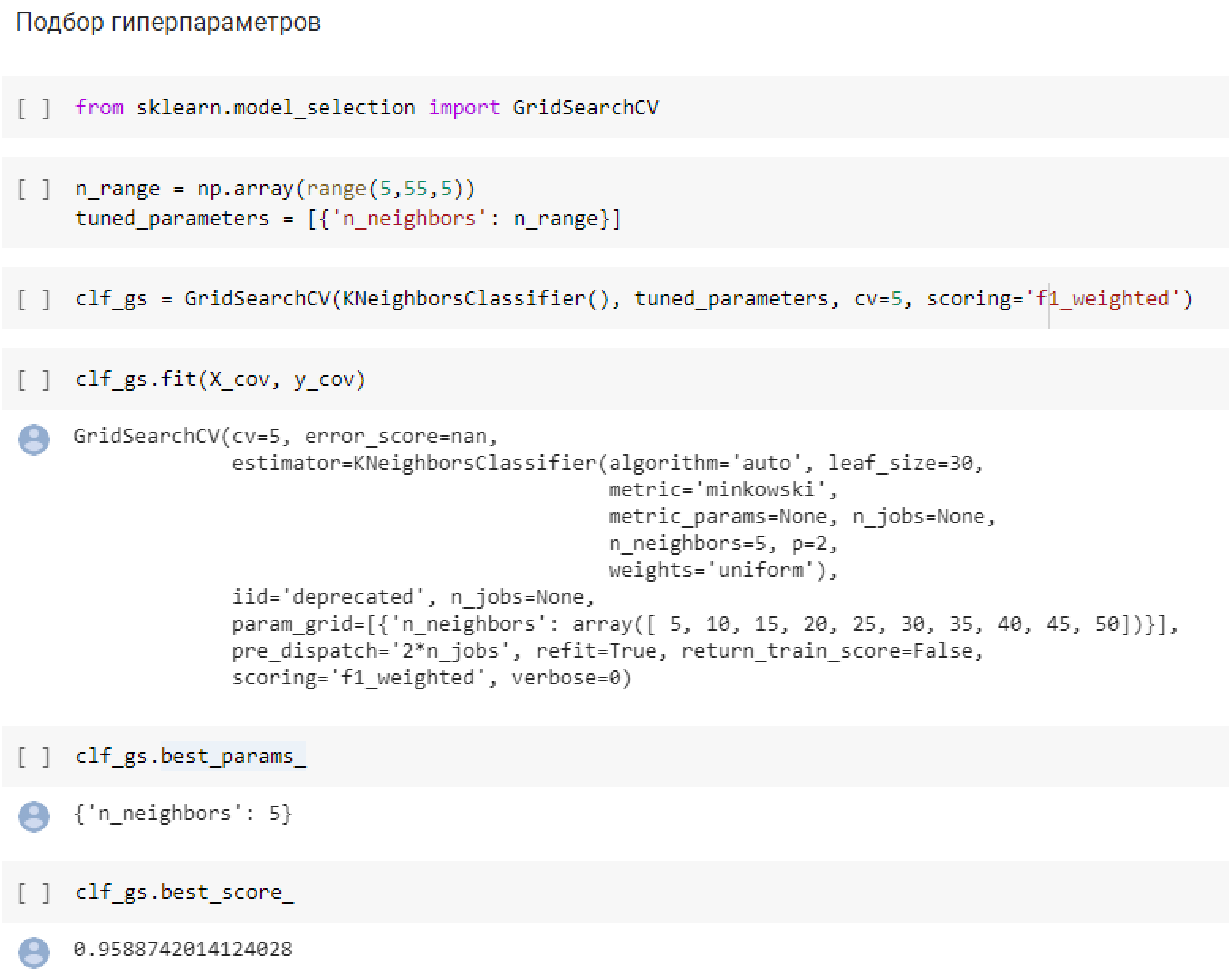
```
{'0': {'f1-score': 0.9799196787148594,
  'precision': 0.9606299212598425,
  'recall': 1.0,
  'support': 122},
 '1': {'f1-score': 0.0, 'precision': 0.0, 'recall': 0.0, 'support': 5},
 'accuracy': 0.9606299212598425,
 'macro avg': {'f1-score': 0.4899598393574297,
  'precision': 0.48031496062992124,
  'recall': 0.5,
  'support': 127},
 'weighted avg': {'f1-score': 0.9413401638048255,
  'precision': 0.9228098456196913,
  'recall': 0.9606299212598425,
  'support': 127}}
```

Кросс-валидация

```
[ ] from sklearn.model_selection import cross_val_score
```

```
[ ] scores = cross_val_score(KNeighborsClassifierObj,
                             X_cov, y_cov, cv=3,
                            scoring='f1_weighted')
    scores, np.mean(scores)
```

```
(array([0.95610184, 0.96463877, 0.95584342]), 0.9588613436340419)
```



Подбор гиперпараметров

```
[ ] from sklearn.model_selection import GridSearchCV
```

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

```
[ ] clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='f1_weighted')
```

```
[ ] clf_gs.fit(X_cov, y_cov)
```

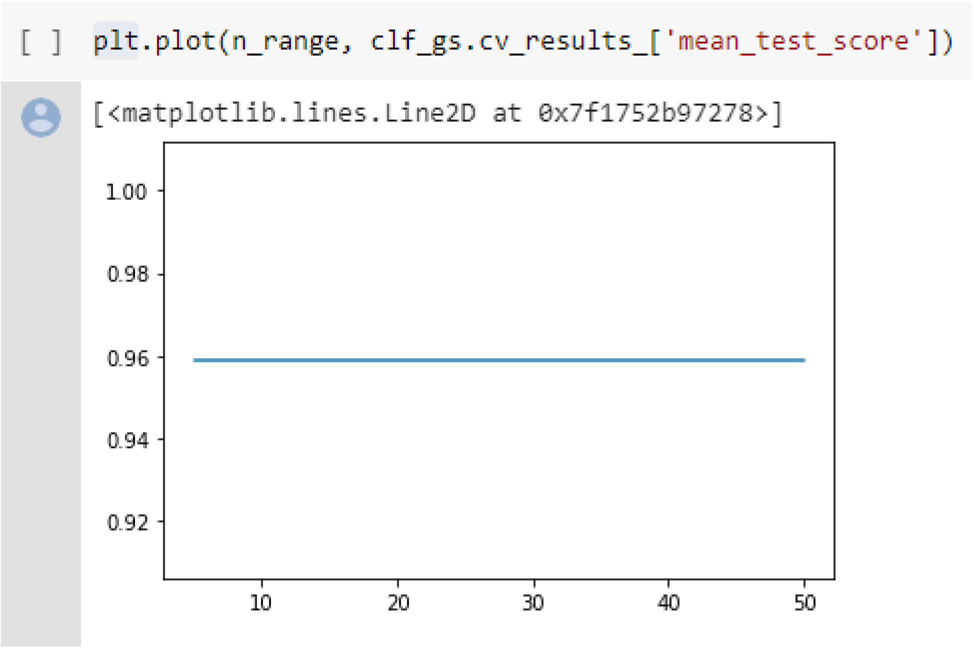
```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='f1_weighted', verbose=0)
```

```
[ ] clf_gs.best_params_
```

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

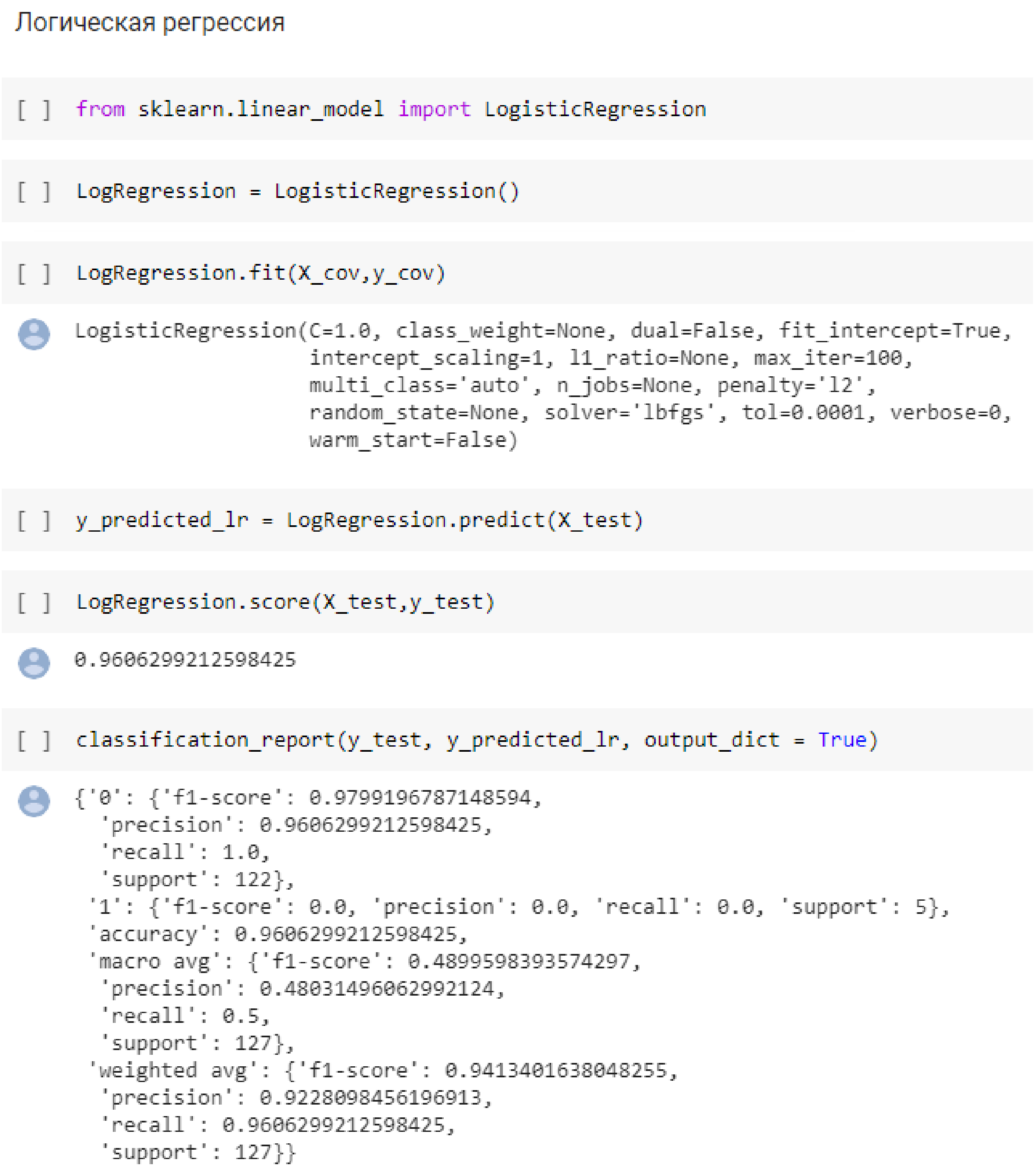
```
[ ] clf_gs.best_score_
```

```
0.9588742014124028
```



```
[ ] plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7f1752b97278>]
```



Логическая регрессия

```
[ ]  from sklearn.linear_model import LogisticRegression
```

```
[ ]  LogRegression = LogisticRegression()
```

```
[ ]  LogRegression.fit(X_cov,y_cov)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

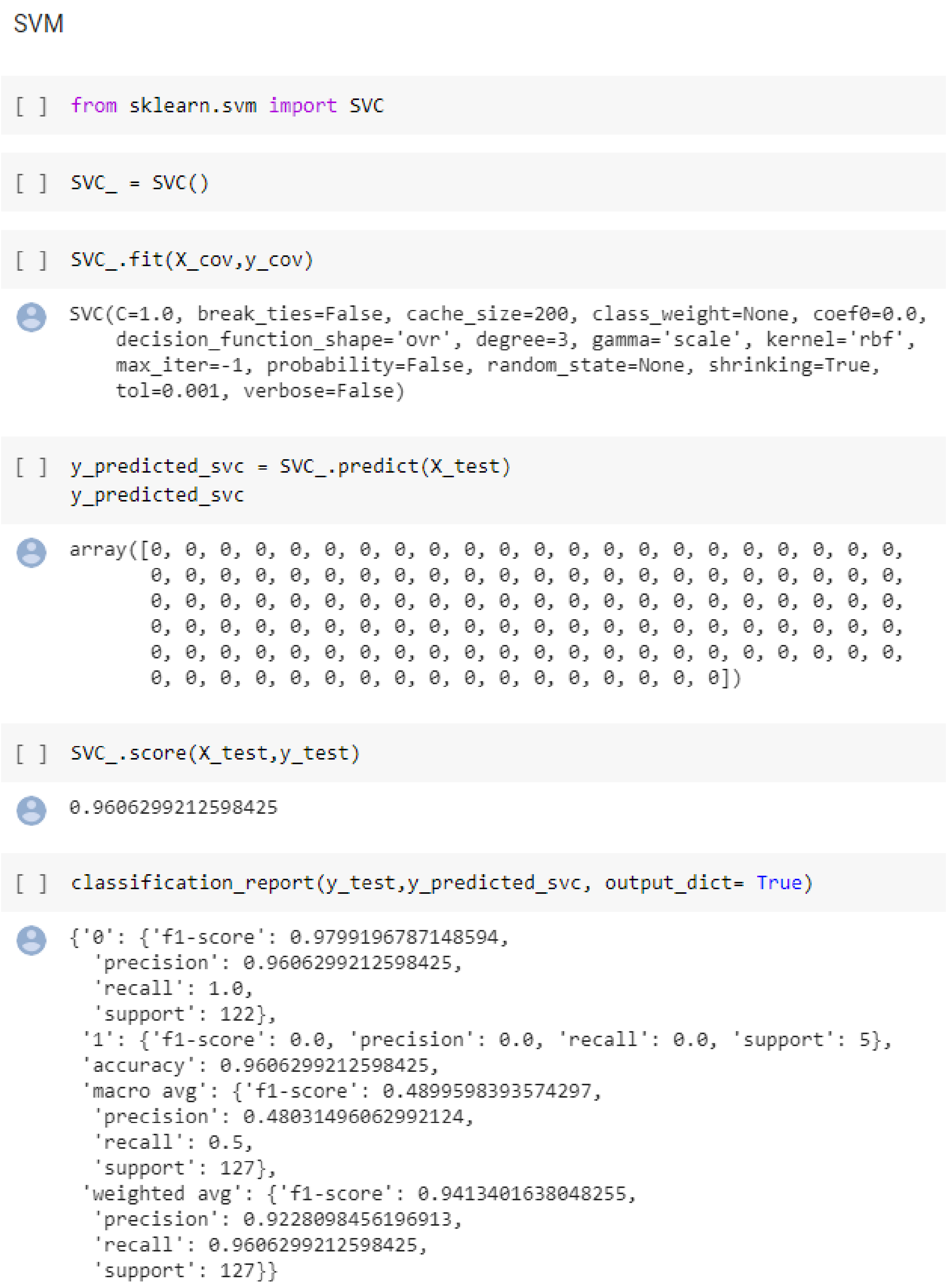
```
[ ]  y_predicted_lr = LogRegression.predict(X_test)
```

```
[ ]  LogRegression.score(X_test,y_test)
```

```
0.9606299212598425
```

```
[ ]  classification_report(y_test, y_predicted_lr, output_dict = True)
```

```
{'0': {'f1-score': 0.9799196787148594,
  'precision': 0.9606299212598425,
  'recall': 1.0,
  'support': 122},
 '1': {'f1-score': 0.0, 'precision': 0.0, 'recall': 0.0, 'support': 5},
 'accuracy': 0.9606299212598425,
 'macro avg': {'f1-score': 0.4899598393574297,
  'precision': 0.48031496062992124,
  'recall': 0.5,
  'support': 127},
 'weighted avg': {'f1-score': 0.9413401638048255,
  'precision': 0.9228098456196913,
  'recall': 0.9606299212598425,
  'support': 127}}
```



SVM

```
[ ] from sklearn.svm import SVC
```

```
[ ] SVC_ = SVC()
```

```
[ ] SVC_.fit(X_cov,y_cov)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
[ ] y_predicted_svc = SVC_.predict(X_test)
    y_predicted_svc
```

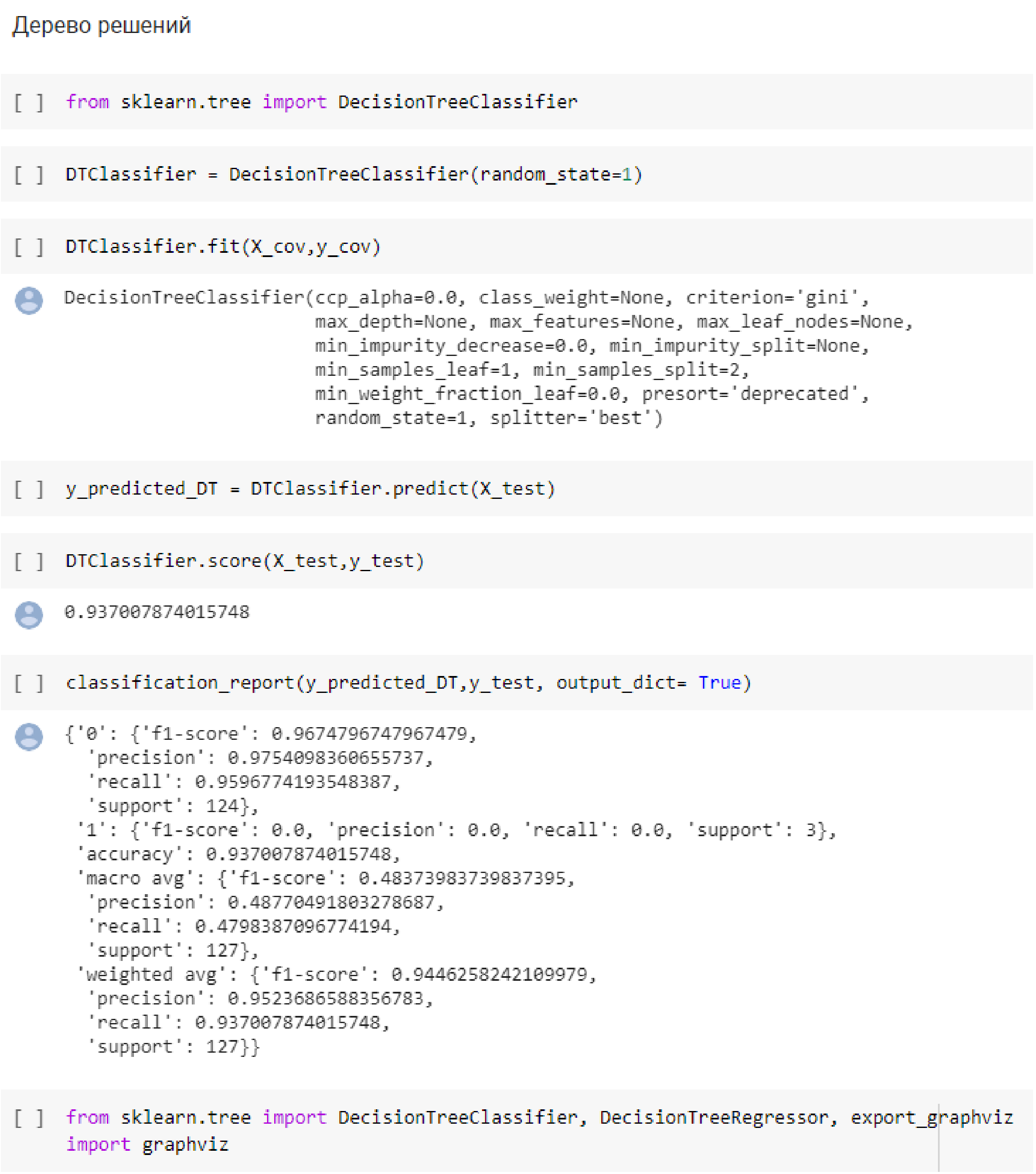
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
[ ] SVC_.score(X_test,y_test)
```

```
0.9606299212598425
```

```
[ ] classification_report(y_test,y_predicted_svc, output_dict= True)
```

```
{'0': {'f1-score': 0.9799196787148594,
  'precision': 0.9606299212598425,
  'recall': 1.0,
  'support': 122},
 '1': {'f1-score': 0.0, 'precision': 0.0, 'recall': 0.0, 'support': 5},
 'accuracy': 0.9606299212598425,
 'macro avg': {'f1-score': 0.4899598393574297,
  'precision': 0.48031496062992124,
  'recall': 0.5,
  'support': 127},
 'weighted avg': {'f1-score': 0.9413401638048255,
  'precision': 0.9228098456196913,
  'recall': 0.9606299212598425,
  'support': 127}}
```



Дерево решений

```
from sklearn.tree import DecisionTreeClassifier
```

```
DTClassifier = DecisionTreeClassifier(random_state=1)
```

```
DTClassifier.fit(X_cov,y_cov)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, 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, presort='deprecated',
                       random_state=1, splitter='best')
```

```
y_predicted_DT = DTClassifier.predict(X_test)
```

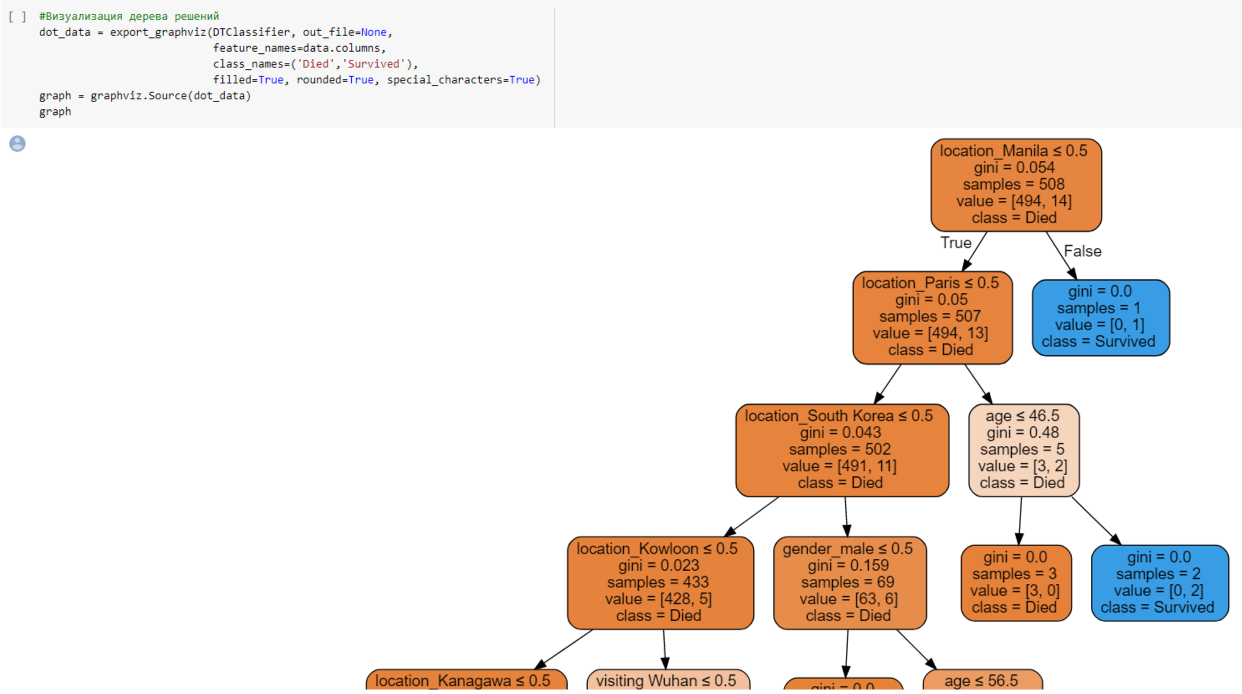
```
DTClassifier.score(X_test,y_test)
```

```
0.937007874015748
```

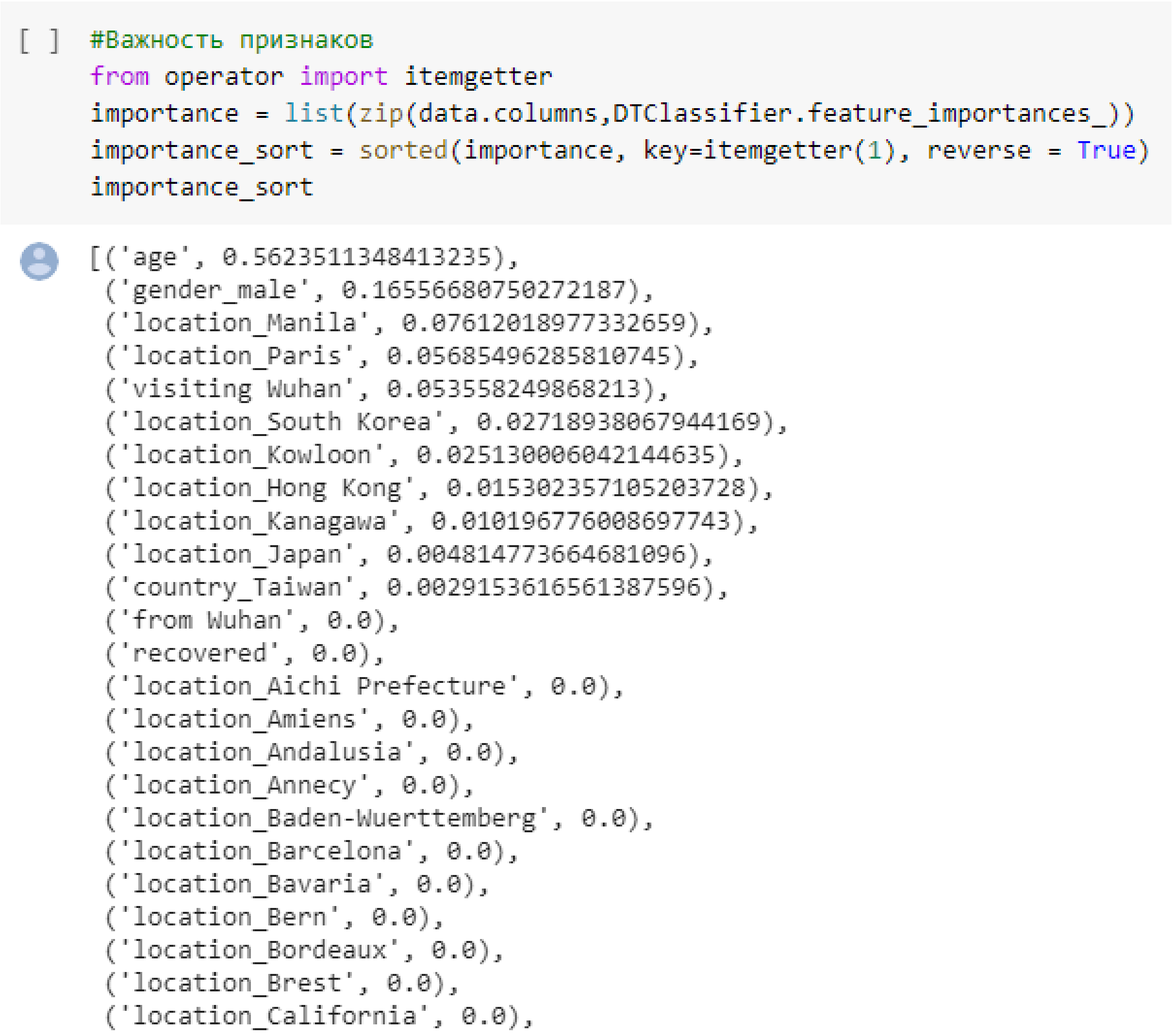
```
classification_report(y_predicted_DT,y_test, output_dict= True)
```

```
{'0': {'f1-score': 0.9674796747967479,
  'precision': 0.9754098360655737,
  'recall': 0.9596774193548387,
  'support': 124},
 '1': {'f1-score': 0.0, 'precision': 0.0, 'recall': 0.0, 'support': 3},
 'accuracy': 0.937007874015748,
 'macro avg': {'f1-score': 0.48373983739837395,
  'precision': 0.48770491803278687,
  'recall': 0.4798387096774194,
  'support': 127},
 'weighted avg': {'f1-score': 0.9446258242109979,
  'precision': 0.9523686588356783,
  'recall': 0.937007874015748,
  'support': 127}}
```

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
import graphviz
```

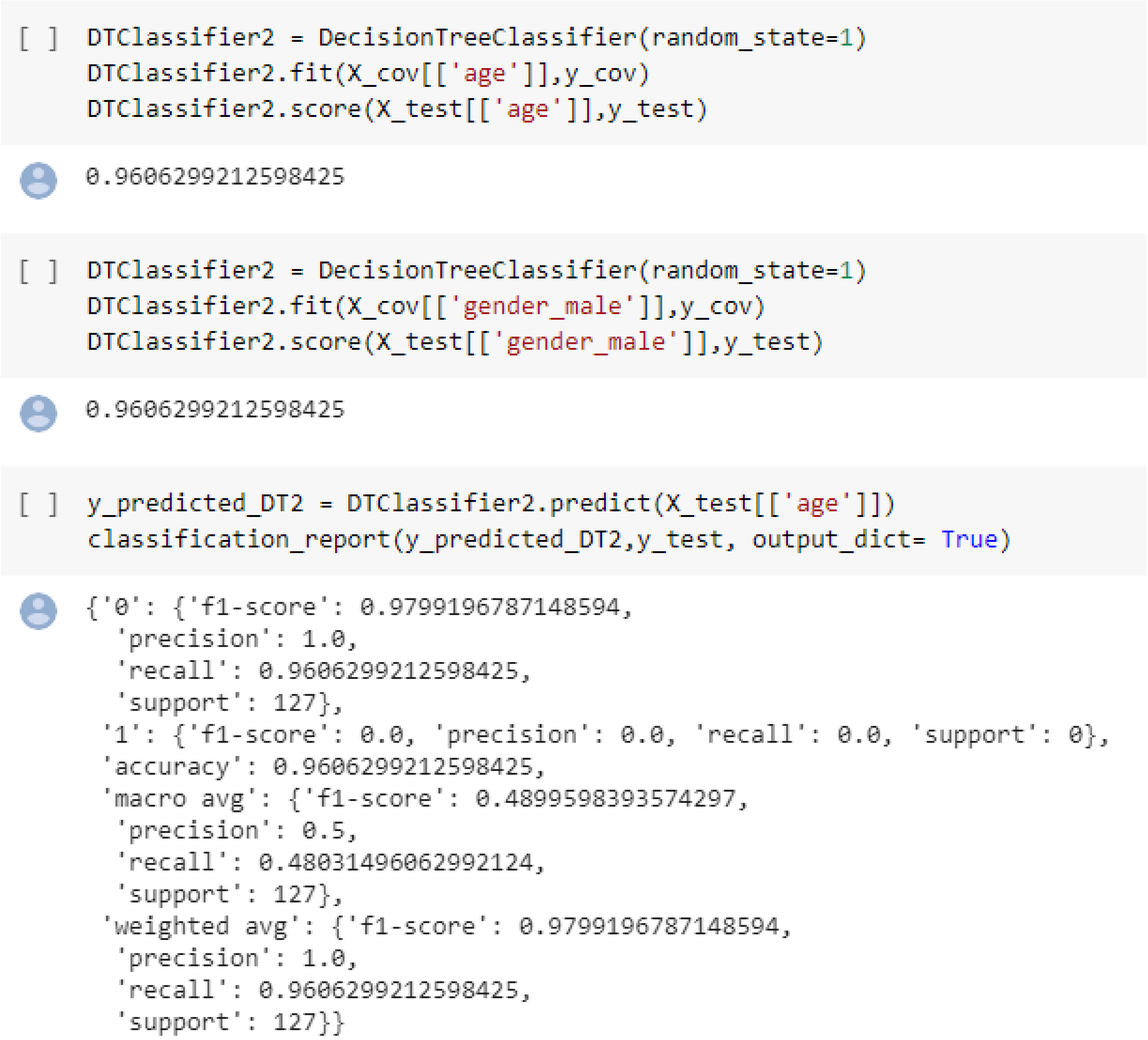


```
[ ] #Визуализация дерева решений
    dot_data = export_graphviz(DTClassifier, out_file=None,
                               feature_names=data.columns,
                               class_names=('Died','Survived'),
                               filled=True, rounded=True, special_characters=True)
    graph = graphviz.Source(dot_data)
    graph
```



```
[ ] #Важность признаков
    from operator import itemgetter
    importance = list(zip(data.columns,DTClassifier.feature_importances_))
    importance_sort = sorted(importance, key=itemgetter(1), reverse = True)
    importance_sort
```

```
[('age', 0.5623511348413235),
 ('gender_male', 0.16556680750272187),
 ('location_Manila', 0.07612018977332659),
 ('location_Paris', 0.056854962858107745),
 ('visiting Wuhan', 0.053558249868213),
 ('location_South Korea', 0.02718938067944169),
 ('location_Kowloon', 0.025130006042144635),
 ('location_Hong Kong', 0.015302357105203728),
 ('location_Kanagawa', 0.010196776008697743),
 ('location_Japan', 0.004814773664681096),
 ('country_Taiwan', 0.0029153616561387596),
 ('from Wuhan', 0.0),
 ('recovered', 0.0),
 ('location_Aichi Prefecture', 0.0),
 ('location_Amiens', 0.0),
 ('location_Andalusia', 0.0),
 ('location_Annecy', 0.0),
 ('location_Baden-Wuerttemberg', 0.0),
 ('location_Barcelona', 0.0),
 ('location_Bavaria', 0.0),
 ('location_Bern', 0.0),
 ('location_Bordeaux', 0.0),
 ('location_Brest', 0.0),
 ('location_California', 0.0),
```



```
DTClassifier2 = DecisionTreeClassifier(random_state=1)
DTClassifier2.fit(X_cov[['age']],y_cov)
DTClassifier2.score(X_test[['age']],y_test)
```

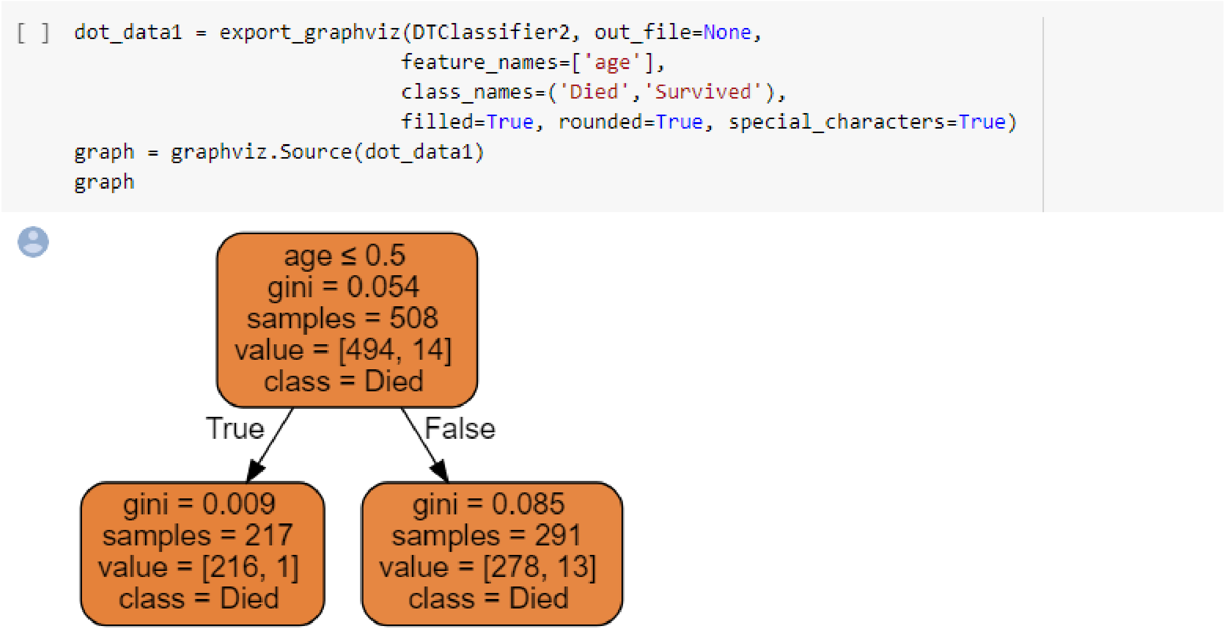
```
0.9606299212598425
```

```
DTClassifier2 = DecisionTreeClassifier(random_state=1)
DTClassifier2.fit(X_cov[['gender_male']],y_cov)
DTClassifier2.score(X_test[['gender_male']],y_test)
```

```
0.9606299212598425
```

```
y_predicted_DT2 = DTClassifier2.predict(X_test[['age']])
classification_report(y_predicted_DT2,y_test, output_dict= True)
```

```
{'0': {'f1-score': 0.9799196787148594,
  'precision': 1.0,
  'recall': 0.9606299212598425,
  'support': 127},
 '1': {'f1-score': 0.0, 'precision': 0.0, 'recall': 0.0, 'support': 0},
 'accuracy': 0.9606299212598425,
 'macro avg': {'f1-score': 0.4899598393574297,
  'precision': 0.5,
  'recall': 0.48031496062992124,
  'support': 127},
 'weighted avg': {'f1-score': 0.9799196787148594,
  'precision': 1.0,
  'recall': 0.9606299212598425,
  'support': 127}}
```



```
dot_data1 = export_graphviz(DTClassifier2, out_file=None,
                            feature_names=['age'],
                            class_names=('Died','Survived'),
                            filled=True, rounded=True, special_characters=True)
graph = graphviz.Source(dot_data1)
graph
```

Лучшеми моделями являются логическая регрессия и SVM. Показатели их точности одинаковы.