

МАГИСТЕРСКАЯ ПРОГРАММА 09.04.01/12 Интеллектуальный анализ больших данных в системах поддержки принятия решений.

по лабораторной работе № 2-4

Дисциплина: Платформы промышленной аналитики

Студент	<u>ИУ6-43М</u>	<u>(Подпись, дата)</u>	<u>Ф.А. Лучкин</u>
	(Группа)		(И.О. Фамилия)
Студент	<u>ИУ6-43М</u>	<u>(Подпись, дата)</u>	<u>А.А. Павловский</u>
	(Группа)		(И.О. Фамилия)
Преподаватель		<u>(Подпись, дата)</u>	<u>М. А. Скворцова</u>
			(И.О. Фамилия)

Цель лабораторной работы №2: подготовить набор данных для построения модели машинного обучения.

Цель лабораторной работы №3: построение сложного паплайна, включающего в себя возможность проверки модели на не менее чем 3х различных методах машинного обучения, релевантных поставленной задаче.

Цель лабораторной работы №4: доработка модели машинного обучения, оценка ее качества и проверка решения задачи прогнозирования.

Из данных целей исходят следующие задания.

Задания:

1. Загрузить данные;
2. Выполнить визуальный анализ данных (boxplot);
3. Проанализировать данные и провести их;
4. Провести оценку взаимосвязи данных датасета;
5. Провести разведочный анализ данных с помощью методов PCA и TSNE, сделать вывод о структуре признаков данных;
6. Выполнить обогащение датасета;
7. Выполнить построение AutoML пайплайнов (LightAutoML, FEDOT, TPOT), предоставить схемы и сравнение эффективности работы алгоритмов;
8. Выполнить оценку качества одной из моделей (улучшить модели, если возможно), выполнить визуализацию результатов, развернуть модель в MLFlow.

Задание 1

Загрузка данных и данные показаны на рисунке 1.

Задание 2

Для начала были просмотрены типы данных колонок и количество пропусков. Из полученной информации следовало, что по каждому наблюдению может быть несколько отчётов, в колонках report количество пропущенных значений растёт с увеличением номера отчёта, данные колонки имеют строковый тип. Также, нам не интересны колонки идентификаторов, поэтому их можно удалить.

Для анализа данных были посчитаны различные статистики по фичам и таргетам, а также выполнено построение графиков boxplot для визуального анализа

распределения данных. Вывод статистик показан на рисунке 2, а графики boxplot – на рисунке 3.

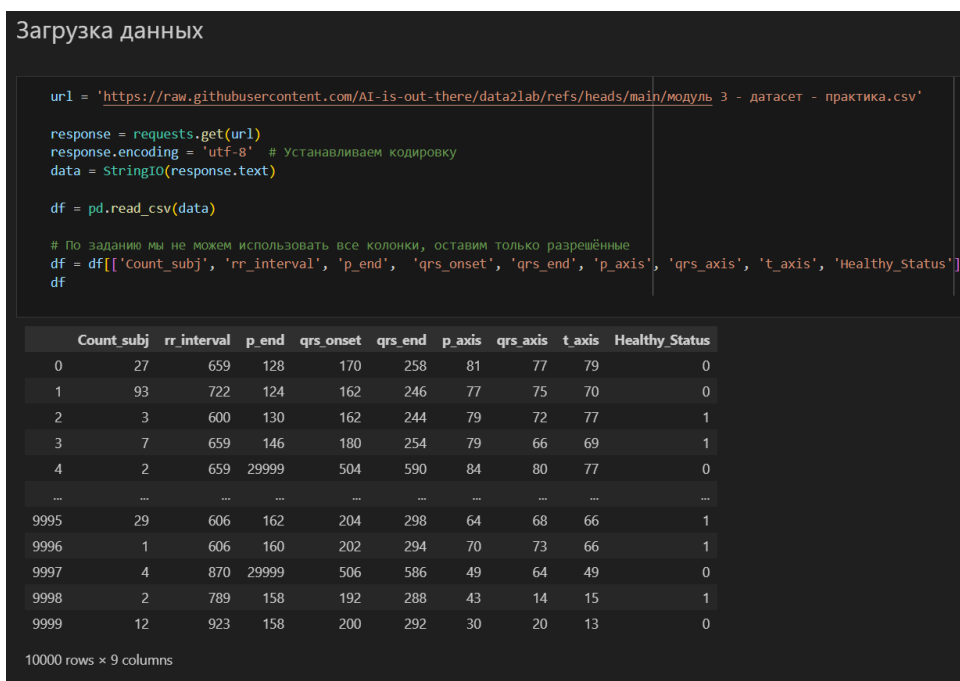


Рисунок 1 – загрузка данных

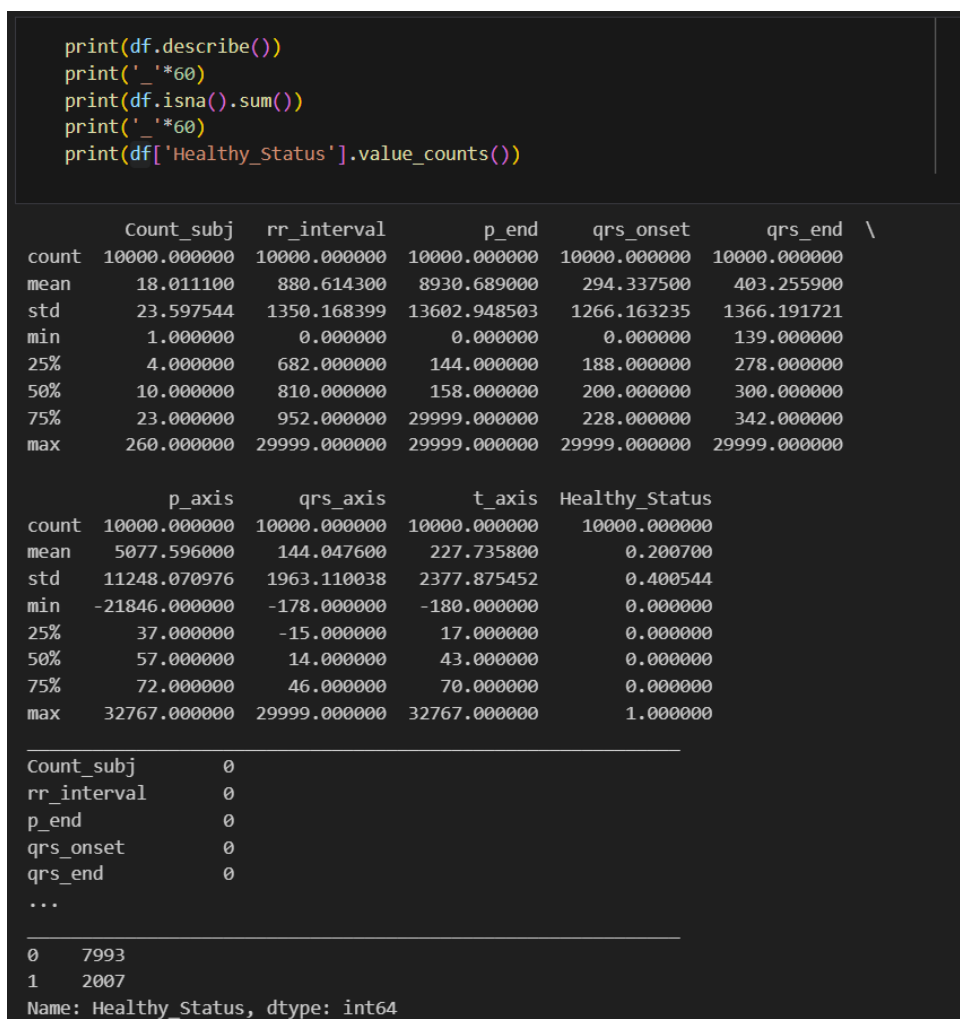


Рисунок 2 – статистический анализ данных

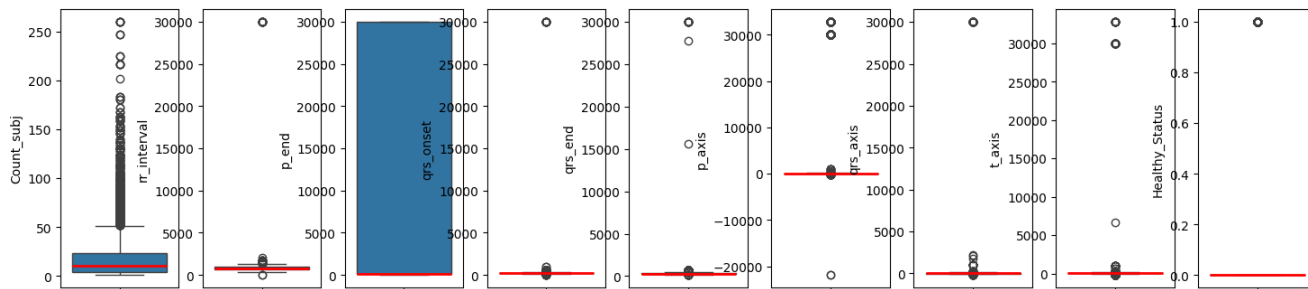


Рисунок 3 – boxplot графики

На основе полученных данных можно сделать вывод, что датасет содержит 2 класса, которые имеют сильный дисбаланс, а данные в колонках фичей имеют явные выбросы.

Задание 3

Начнём фильтрацию данных. На части данных чётко видны выбросы, избавимся от них. Также, на всякий случай отсеем строки, в которых время начала волны или комплекса больше, чем конец.

Листинг кода очистки данных от выбросов

```
# Фильтруем данные, удаляя строки с значениями больше 2000 в выбранных столбцах
filtered_df = df.loc[
    (df[num_columns] < 2000).all(axis=1)
]
# Добавляем дополнительный фильтр для проверки логичности данных в столбцах
# 'p_onset' и 'qrs_onset'
filtered_df = filtered_df.query(
    '(qrs_onset < qrs_end)'
)
filtered_df.shape[0]
```

На очищенных данных заново построим boxplot. Обновлённый график показан на рисунке 4.

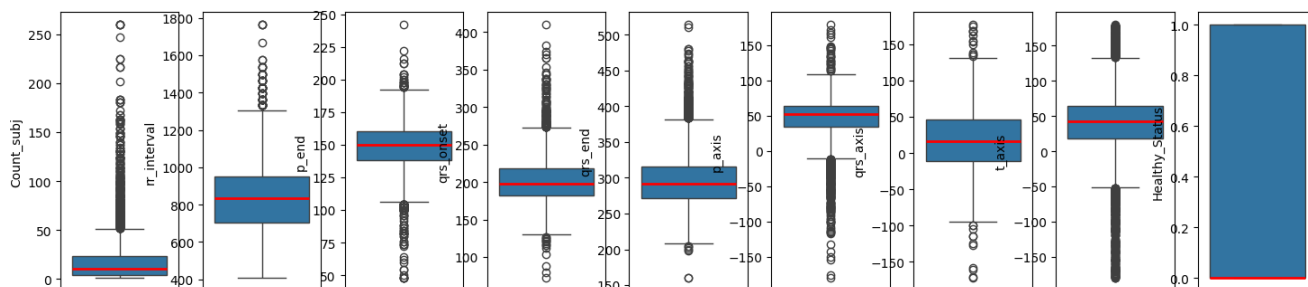


Рисунок 4 – boxplot графики на очищенных данных

Задание 4

Для оценки взаимосвязи данных были построены heatmap, scatter_matrix и pairplot графики, они приведены на рисунках 5-7.

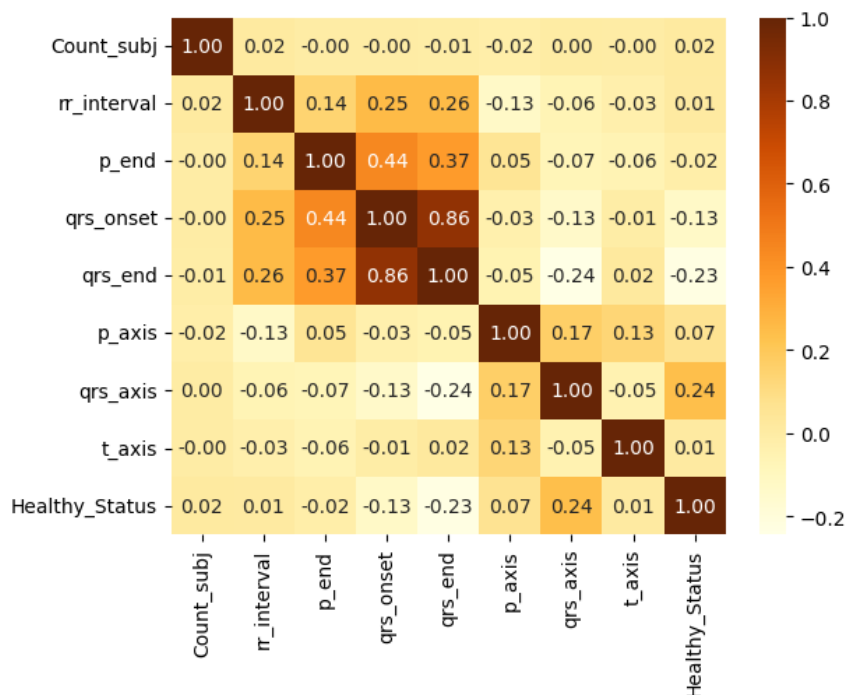


Рисунок 5 – heatmap таблица

Таблица анализа данных, коэффициент корреляции

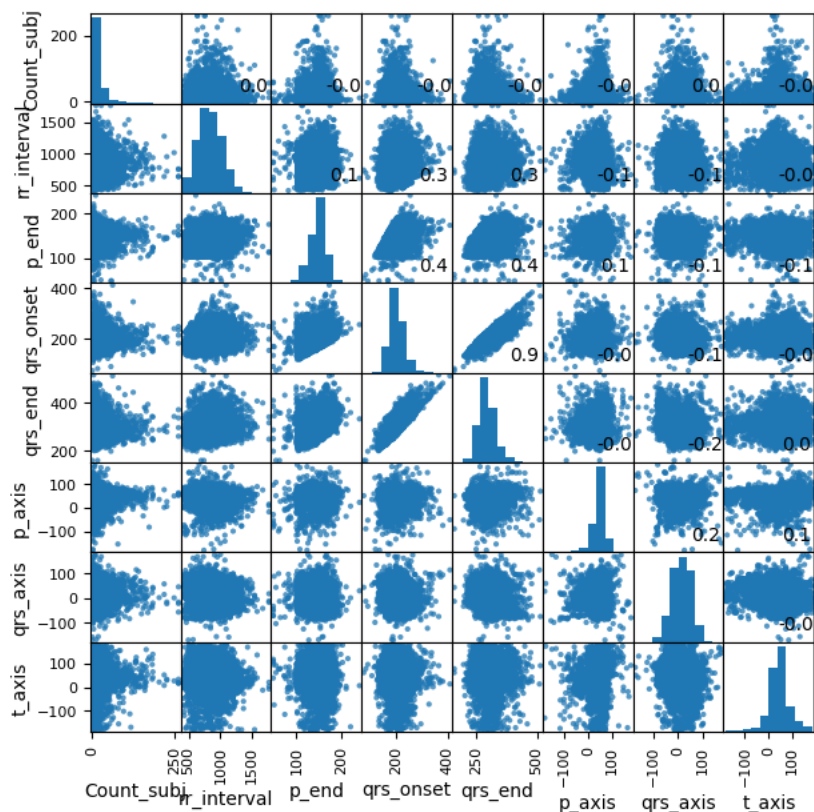


Рисунок 6 – scatter_matrix графики

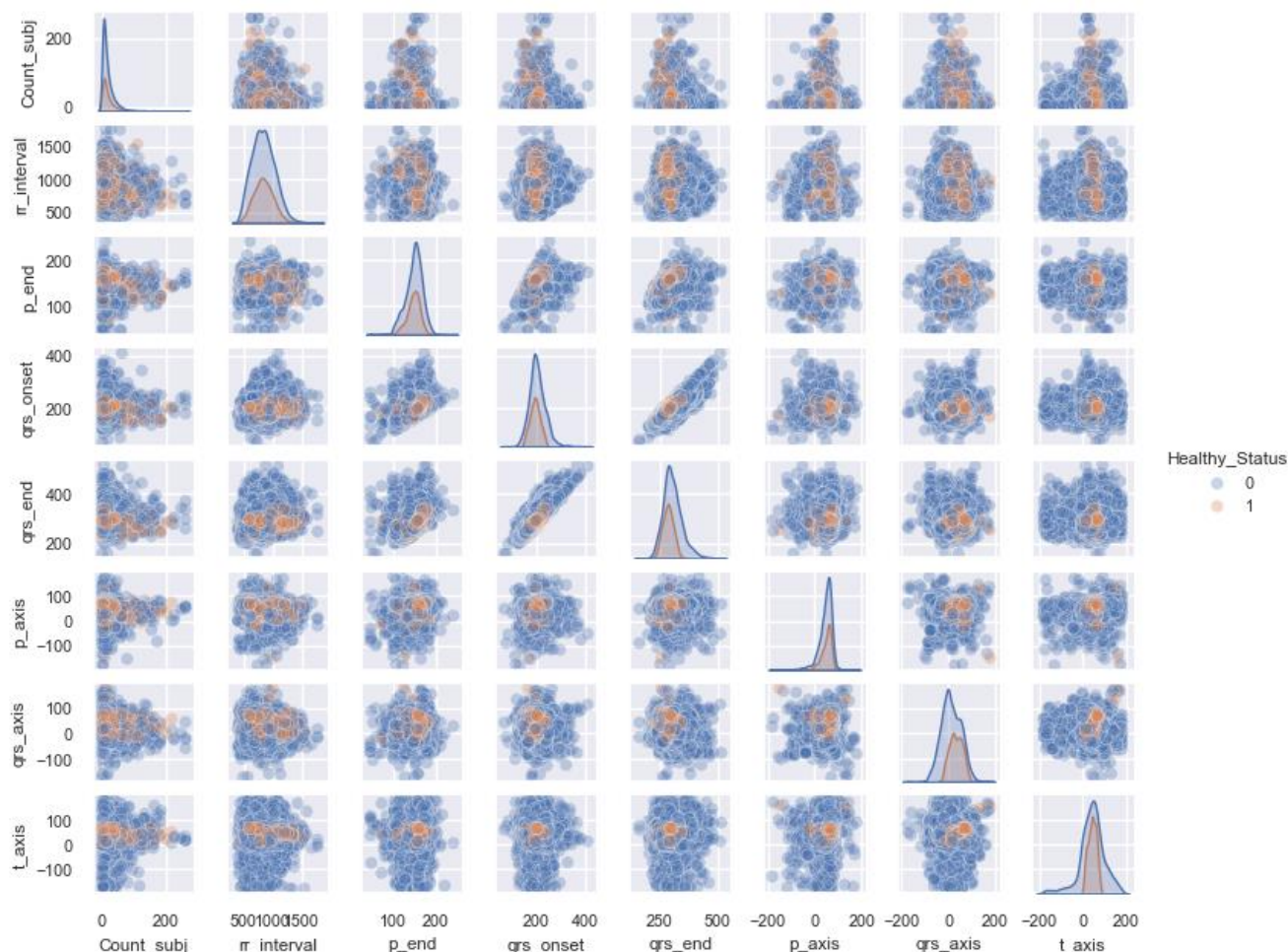


Рисунок 7 – pairplot графики

Из приведённых визуальных представлений данных можно сделать выводы, что имеется некоторая корреляция между фичами времени (в миллисекундах), однако явных корреляций, близких к 1, не обнаружено. Также, видно, что большая часть переменных имеет сильно перекрывающиеся распределения между классами, что говорит о том, что их будет сложно использовать по отдельности для хорошего разделения классов. Снова виден сильный дисбаланс классов.

Задание 5

Проведём разведочный анализ данных с помощью методов PCA и TSNE. Визуальные результаты работы кода показаны на рисунках 8-9.

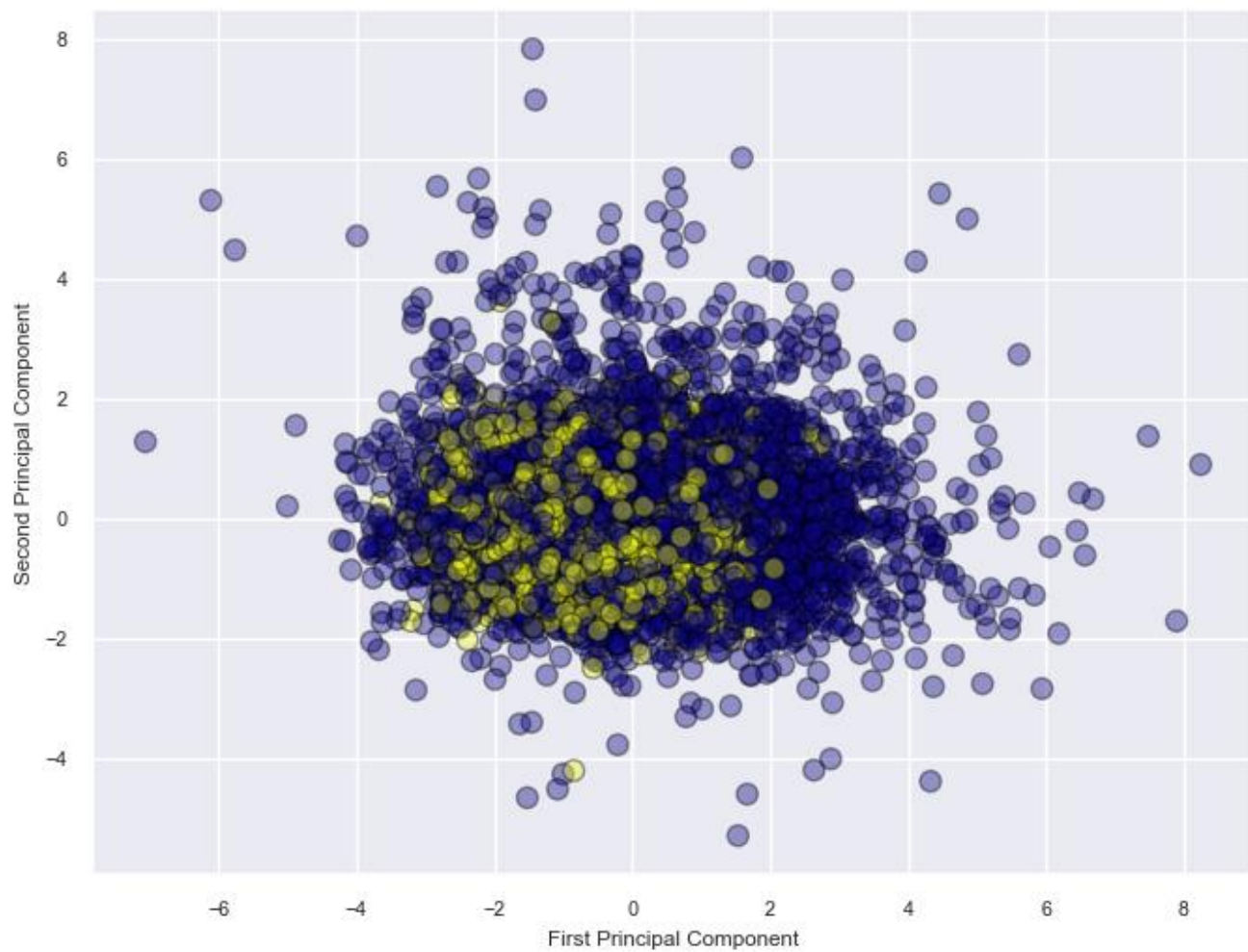


Рисунок 8 – результаты работы PCE

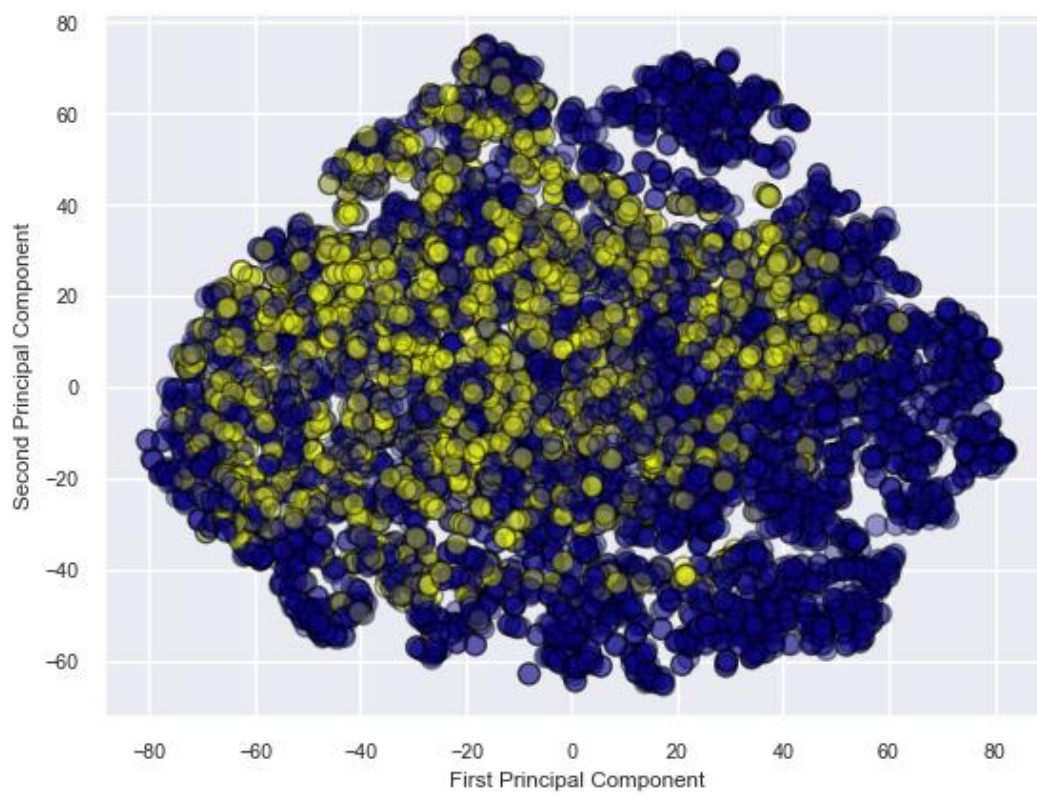


Рисунок 9 – результаты работы TSNE

Без имеющейся ранее текстовой информации, чёткого распределения кластеров не видно. Из рисунков выше, следует, что классы будет тяжело разделить.

Также, сложно не заметить, что разделение при TSNE прошло лучше, чем при PCA, что говорит, что данные имеют нелинейные зависимости, учитывание которых позволит лучше разделять классы.

Дополнительный анализ, представленный на рисунке 10, подтверждает сложность задачи разделения классов.

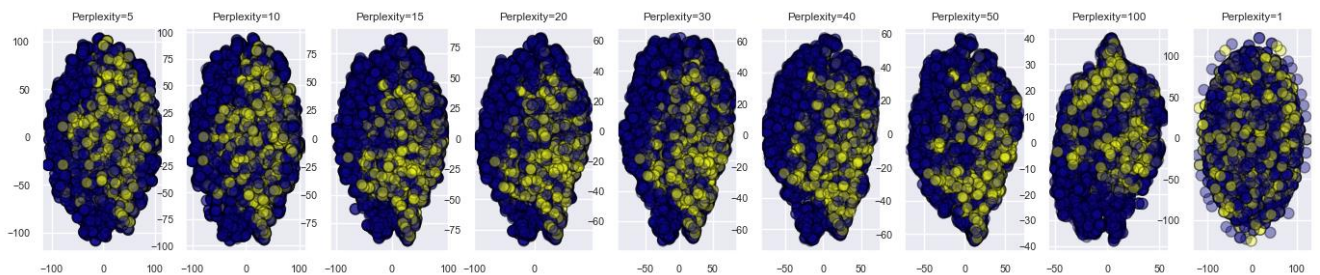


Рисунок 10 – результаты работы TSNE при разных perplexity

Задание 6

Для обогащения датасета был использован упомянутый ранее SMOTE метод, который используется для устранения дисбаланса классов. Ниже приведён код обогащения датасета

```
from imblearn.over_sampling import SMOTE

target_column = 'Healthy_Status'

smote = SMOTE(random_state=42)
X_expanded = df_expanded.drop(target_column, axis=1)
y_expanded = df_expanded[target_column]

X_resampled, y_resampled = smote.fit_resample(X_expanded, y_expanded)

resampled_df = pd.DataFrame(X_resampled, columns=X_expanded.columns)
resampled_df[target_column] = y_resampled

print(resampled_df[target_column].value_counts())
resampled_df.head(3)
```

Результат обогащения показан на рисунке 11.


```
print(resampled_df[target_column].value_counts())
resampled_df.head(3)
```

0	5039
1	5039
Name: Healthy_Status, dtype: int64	

Рисунок 11 – результат обогащения датасета

Как видно, дисбаланс классов был успешно устранён путём увеличения малого класса за счёт выпуклых комбинаций экземпляров этого класса.

Задание 7

В рамках задания были использованы LightAutoML, FEDOT и TPOT.

Код подготовки данных, обучения, использования и оценки качества модели при использовании с LightAutoML показан ниже.

```
target_column = 'Healthy_Status'

X = resampled_df.drop(target_column, axis=1)
y = resampled_df[target_column]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

task = Task('multiclass')

automl = TabularAutoML(
    task=task,
    cpu_limit=-1,
    general_params={'use_algos': 'auto'},
    reader_params={'cv': 3, 'random_state': 42}
)

train_data = pd.concat([X_train, y_train], axis=1)
roles = {
    'target': target_column,
    'drop': []
}

oof_pred = automl.fit_predict(
    train_data=train_data,
    roles=roles,
    verbose=1
)

test_pred = automl.predict(X_test)
y_pred = test_pred.data.argmax(axis=1)

accuracy, precision, recall = (
    accuracy_score(y_test, y_pred),
```

```
precision_score(y_test, y_pred, average='weighted'),
recall_score(y_test, y_pred, average='weighted'),
)

clear_output(wait=False)
print(f'Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}')
```

В результате была получена модель, показанная на рисунке 12.

```
Тип алгоритма: LinearLBFGS
Тип модели: TorchBasedLogisticRegression

=== Параметры TorchBasedLogisticRegression ===
categorical_idx: [0, 1]
cs: [1e-05, 5e-05, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000]
data_size: 40
early_stopping: 2
embed_sizes: [11 11]
fit: <bound method TorchBasedLinearEstimator.fit of <lightautoml.ml_algo.torch_based.linear_model.TorchBasedLogisticRegression object at 0x0000025800136130>>

Функция потерь: TorchLossWrapper
max_iter: 100
metric: <lightautoml.tasks.losses.base.MetricFunc object at 0x00000258632C2700>

Архитектура нейросети:
CatMulticlass(
  (linear): Linear(in_features=38, out_features=2, bias=False)
  (final_act): SoftmaxClip(
    (smx): Softmax(dim=1)
  )
)
output_size: 2
predict: <bound method TorchBasedLogisticRegression.predict of <lightautoml.ml_algo.torch_based.linear_model.TorchBasedLogisticRegression object at 0x0000025800136130>>
tol: 1e-06
```

Рисунок 12 – лучшая модель от LightAutoML

Данная модель представляет из себя искусственную нейронную сеть и идеально выполняет задачу классификации. Accuracy: 0.8501984126984127, Precision: 0.8538389808458616, Recall: 0.8501984126984127.

Полный пайплайн работы с LightAutoML показан на рисунке 13.



Рисунок 13 – пайплайн работы с LightAutoML

Код обучения, использования и оценки качества модели при использовании с FEDOT показан ниже.

```

# Инициализация Fedot с явным указанием доступных моделей
automl_model = Fedot(
    problem='classification',
    preset='fast_train',
    # timeout=10, # 2 минуты на подбор
    available_operations=['rf', 'logit', 'mlp', 'xgboost'],
    logging_level=logging.CRITICAL,
    with_tuning=True,
    n_jobs=-1,
    seed=42
)

try:
    pipeline = automl_model.fit(features=X_train, target=y_train)

    y_pred = automl_model.predict(features=X_test)

    accuracy, precision, recall = (
        accuracy_score(y_test, y_pred),
        precision_score(y_test, y_pred, average='weighted'),
        recall_score(y_test, y_pred, average='weighted'),
    )
    print(f'Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}')

except Exception as e:
    print(f'Произошла ошибка: {str(e)}')
    print('Попробуйте обновить Fedot или использовать другой пресет')

```

В результате была получена модель, показанная на рисунке 14.

```

Узел: rf
Параметры: {'n_jobs': 12}
Fitted_params: {
    bootstrap: True
    ccp_alpha: 0.0
    class_weight: None
    criterion: gini
    max_depth: None
    max_features: sqrt
    max_leaf_nodes: None
    max_samples: None
    min_impurity_decrease: 0.0
    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: None
    verbose: 0
    warm_start: False
}

```

Рисунок 14 – лучшая модель от FEDOT

Данная модель представляет из себя искусственную нейронную сеть, которая с высокой точностью выполняет задачу классификации. Accuracy: 0.8675595238095238, Precision: 0.8774073682133329, Recall: 0.8675595238095238.

Полный пайплайн работы с FEDOT показан на рисунке 15.



Рисунок 15 – пайплайн работы с FEDOT

FEDOT отлично работает и без преобразования таргетов.

Код подготовки данных, обучения, использования и оценки качества модели при использовании с TPOT показан ниже.

```
tpot = TPOTClassifier(generations=4, population_size=10, random_state=42, verbosity=2,
n_jobs=-1, max_time_mins=30, cv=3)
tpot.fit(X_train, y_train)

# Оценка производительности лучшей модели
y_pred = tpot.predict(X_test)
accuracy, precision, recall = (
    accuracy_score(y_test, y_pred),
    precision_score(y_test, y_pred, average='weighted'),
    recall_score(y_test, y_pred, average='weighted'),
)

print(f'Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}')
```

В результате была получена модель, показанная на рисунке 16.

```
Тип: RandomForestClassifier
Параметры:
  bootstrap: True
  ccp_alpha: 0.0
  class_weight: None
  criterion: gini
  max_depth: None
  max_features: 0.2
  max_leaf_nodes: None
  max_samples: None
  min_impurity_decrease: 0.0
  min_samples_leaf: 8
  min_samples_split: 4
  min_weight_fraction_leaf: 0.0
  n_estimators: 100
  n_jobs: None
  oob_score: False
  random_state: 42
  verbose: 0
  warm_start: False
```

Рисунок 16 – лучшая модель от ТРОТ

Данная модель представляет из себя ансамбль случайных деревьев, который с высокой точностью задачу классификации. Accuracy: 0.8467261904761905, Precision: 0.858671061344892, Recall: 0.8467261904761905.

Полный пайплайн работы с ТРОТ показан на рисунке 19.



Рисунок 19 – пайплайн работы с ТРОТ

TPOT тоже работает без преобразования таргетов, а также, имеет очень простую настройку.

Таблица 1 – Результаты работы моделей

параметр	LightAutoML	FEDOT	TPOT
Accuracy	0.8502	0.8676	0.8467
Precision	0.8538	0.8774	0.8587
Recall	0.8502	0.8676	0.8467

Учитывая фичи, поставленная задача не является простой, поэтому, как видно в таблице 1, все autoML решения не достигли таких высоких показателей метрик качества, как при работе с текстовыми фичами.

Задание 8

Для дальнейшей работы была взята модель TPOT из-за очень простой интерпретации результатов. Для нахождения оптимальных параметров значительно увеличим количество ресурсов на поиск параметров.

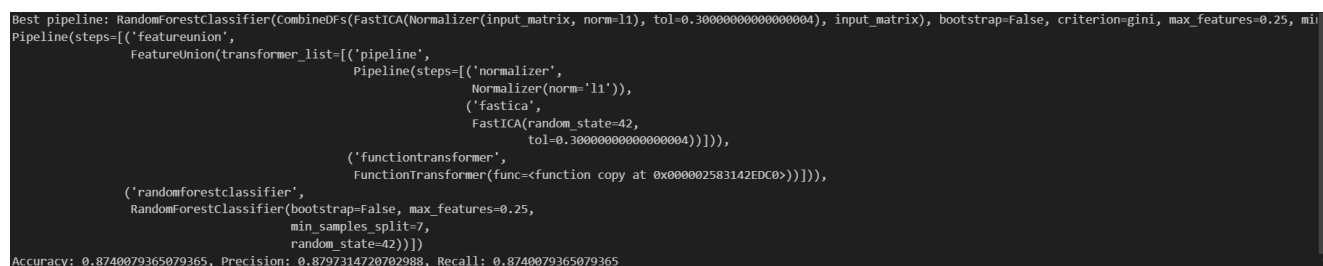
Обновлённый код обучения, использования и оценки качества модели при использовании с TPOT показан ниже.

```
tpot = TPOTClassifier(generations=20, population_size=50, random_state=42, verbosity=2,
n_jobs=-1, max_time_mins=90, cv=3)
tpot.fit(X_train, y_train)
print(tpot.fitted_pipeline_)

y_pred = tpot.predict(X_test)
accuracy, precision, recall = (
    accuracy_score(y_test, y_pred),
    precision_score(y_test, y_pred, average='weighted'),
    recall_score(y_test, y_pred, average='weighted'),
)

print(f'Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}')
```

Результат работы кода показан на рисунке 17.



```
Best pipeline: RandomForestClassifier(CombinedOfs(FastICA(Normalizer(input_matrix, norm='l1'), tol=0.30000000000000004), input_matrix), bootstrap=False, criterion=gini, max_features=0.25, min
Pipeline(steps=[('featureunion',
  FeatureUnion(transformer_list=[('pipeline',
    Pipeline(steps=[('normalizer',
      Normalizer(norm='l1')),
      ('fastica',
        FastICA(random_state=42,
          tol=0.30000000000000004))])),
    ('functiontransformer',
      FunctionTransformer(func=<function copy at 0x00002583142EDC0>))])),
  ('randomforestclassifier',
    RandomForestClassifier(bootstrap=False, max_features=0.25,
      min_samples_split=7,
      random_state=42))]))
Accuracy: 0.8740079365079365, Precision: 0.8797314720702988, Recall: 0.8740079365079365
```

Рисунок 17 – лучшая модель от TPOT

Как видно, метрики качества выросли. Accuracy: 0.8740079365079365, Precision: 0.8797314720702988, Recall: 0.8740079365079365. Реализуем полученный пайплайн:

```
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import Normalizer, FunctionTransformer
from sklearn.decomposition import FastICA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score
from copy import copy # Import copy

# TPOT pipeline
pipeline = Pipeline([
    ('featureunion', FeatureUnion(transformer_list=[
        ('pipeline', Pipeline([
            ('normalizer', Normalizer(norm='l1')),
            ('fastica', FastICA(random_state=42, tol=0.30000000000000004))
        ])),
    ('functiontransformer', FunctionTransformer(func=copy)) # Use copy function
    ])),
    ('randomforestclassifier', RandomForestClassifier(
        bootstrap=False, criterion='gini', max_features=0.25,
        min_samples_leaf=1, min_samples_split=7, n_estimators=100, random_state=42
    ))
])

# Fit the pipeline
pipeline.fit(X_train, y_train)

# Make predictions
y_pred = pipeline.predict(X_test)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# Print metrics
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
```

Accuracy: 0.8740079365079365, Precision: 0.8797314720702988, Recall: 0.8740079365079365. Дальнейшие эксперименты с параметрами показали, что самый большой эффект даёт работа с параметрами RandomForestClassifier, а остальные параметры либо не дают эффекта, либо мешают. Поэтому займёмся настройкой конкретно RandomForestClassifier.

```

from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import GridSearchCV

pipeline = Pipeline([
    ('randomforestclassifier', RandomForestClassifier(
        bootstrap=False, criterion='gini', random_state=42
    ))
])

param_grid = {
    'randomforestclassifier__n_estimators': [80, 100, 120, 140],
    'randomforestclassifier__max_features': [0.2, 0.25, 0.3, 0.35],
    'randomforestclassifier__min_samples_split': [2, 4, 6, 8],
    'randomforestclassifier__min_samples_leaf': [1, 2, 3, 4]
}

grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='f1', n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)

print("Best parameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)

y_pred = grid_search.predict(X_test)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# Print metrics
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')

```

В итоге была получена модель более высокого качества:

```

Best parameters: {
    'randomforestclassifier__max_features': 0.25,
    'randomforestclassifier__min_samples_leaf': 1,
    'randomforestclassifier__min_samples_split': 6,
    'randomforestclassifier__n_estimators': 120
}
Accuracy: 0.8745039682539683
Precision: 0.8826424647707535
Recall: 0.8745039682539683

```

Далее выполним реализацию итоговой модели:

```

final_model = RandomForestClassifier(
    bootstrap=False, # Параметр из ТРОТ
    criterion='gini', # Параметр из ТРОТ
    max_features=0.25, # Оптимизированный параметр
    min_samples_leaf=1, # Оптимизированный параметр
    min_samples_split=6, # Оптимизированный параметр

```

```

n_estimators=120, # Оптимизированный параметр
random_state=42 # Для воспроизводимости
)

final_model.fit(X_train, y_train)

y_pred = final_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

dump(final_model, f'{directory}/final_tpot_model.joblib')

```

В итоге нам удалось существенно нарастить метрики:

- Accuracy: 0.8467261904761905 --> 0.8745039682539683
- Precision: 0.858671061344892 --> 0.8826424647707535
- Recall: 0.8467261904761905 --> 0.8745039682539683

Результаты работы модели на тестовой выборке показаны на рисунке 18.

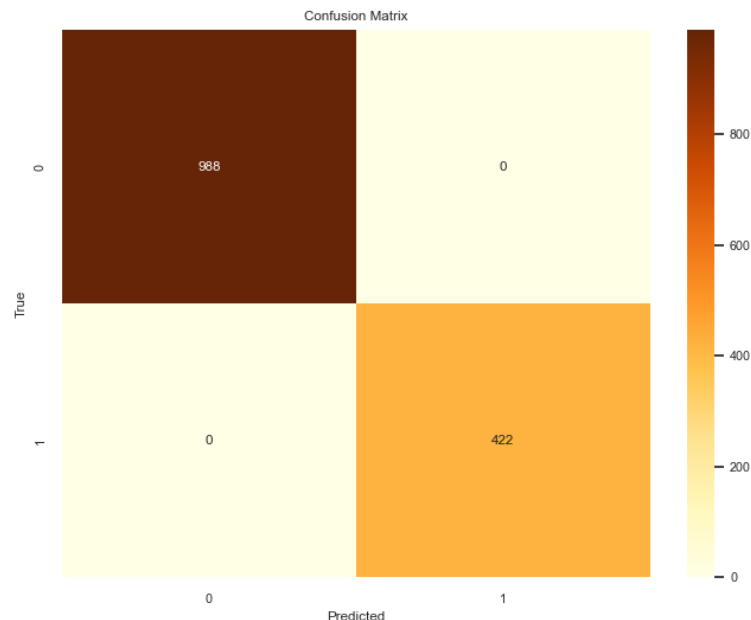


Рисунок 18 – визуализация работы модели на тестовой выборке

Как видно, модель действительно более точные прогнозы.

Запуск сервера MLFlow производится при помощи команды:

```
mlflow server --host 127.0.0.1 --port 8080
```

Интерфейс можно открыть в браузере по адресу: <http://127.0.0.1:8080>

Код работы с MLFlow на примере ТРОТ приведён ниже.

```
mlflow.set_tracking_uri("http://127.0.0.1:8080")
mlflow.set_experiment("RandomForest_ECG_Classification")

params = {
    'bootstrap': False,
    'criterion': 'gini',
    'max_features': 0.25,
    'min_samples_leaf': 1,
    'min_samples_split': 6,
    'n_estimators': 120,
    'random_state': 42
}

with mlflow.start_run():
    mlflow.log_params(params)

    final_model = RandomForestClassifier(**params)

    final_model.fit(X_train, y_train)

    y_pred = final_model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')

    mlflow.log_metrics({
        "accuracy": accuracy,
        "precision": precision,
        "recall": recall
    })

    print(f'Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}')

    mlflow.sklearn.log_model(final_model, "randomforest_model")

    model_path = f"{directory}/randomforest_model.joblib"
    dump(final_model, model_path)
    mlflow.log_artifact(model_path)

print("Finished training and logging model to MLflow.")
```

Результаты работы отображены на рисунках 19-21.

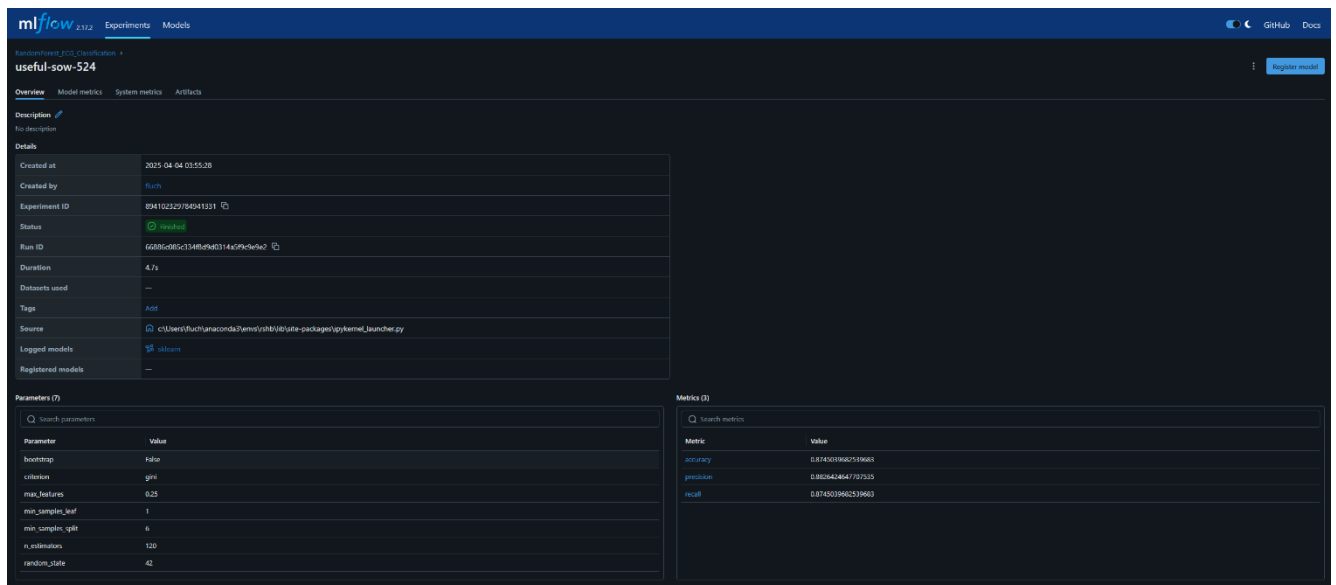


Рисунок 19 – страница рана модели

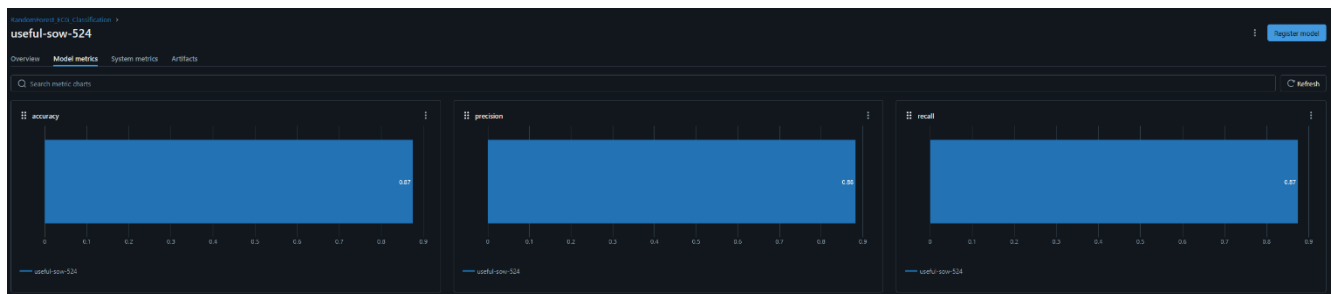


Рисунок 20 – страница метрик качества работы модели

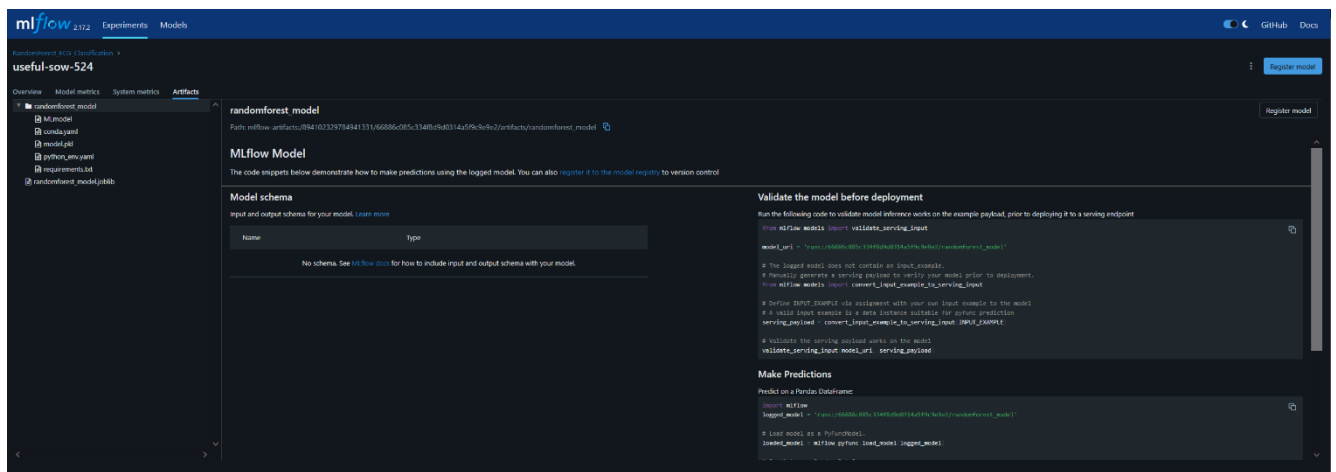


Рисунок 21 – страница артефактов, полученных при обучении модели

Как показано на рисунках выше, взаимодействие с MLFlow прошло успешно.

Вывод

В процессе выполнения лабораторной работы был собран, проанализирован (в том числе, при помощи методов PCA и TSNE), обогащён и сбалансирован датасет, а затем выполнено построение моделей машинного обучения при помощи

autoML решений. Результирующая модель и процесс её обучения и инференса были загружены на MLFlow.