Задание: Бинарная классификация для предиктивного обслуживания оборудования

Цель задания:

Разработать модель машинного обучения, которая предсказывает, произойдет ли отказ оборудования (Target = 1) или нет (Target = 0). Результаты работы оформить в виде многостраничного Streamlit-приложения с использованием `st.navigation` и `st.Page`.

Структура проекта

Дерево файлов:

```
predictive_maintenance_project/
│
├── app.py                  # Основной файл Streamlit-приложения
(entrypoint)
├── analysis_and_model.py  # Основная страница
├── presentation.py        # Страница с презентацией
├── requirements.txt        # Файл с зависимостями для установки библиотек
├── data/                   # Папка с данными (если данные не загружаются
через интерфейс)
│   └── predictive_maintenance.csv
└── README.md               # Описание проекта
```

Создание `requirements.txt`

Задача: Создать файл `requirements.txt`, в котором перечислены все библиотеки, необходимые для запуска проекта.

Как это сделать:

1. Установите все необходимые библиотеки в виртуальном окружении:

   ```
   pip install streamlit pandas scikit-learn matplotlib seaborn
   ```

2. Сохраните список установленных библиотек в файл `requirements.txt`:

   ```
   pip freeze > requirements.txt
   ```

3. В результате файл `requirements.txt` будет выглядеть примерно так:

```
streamlit==1.41.0
pandas==1.5.3
scikit-learn==1.2.0
matplotlib==3.6.2
seaborn==0.12.1
```

Запуск приложения

1. Установите зависимости:

```
pip install -r requirements.txt
```

2. Запустите Streamlit-приложение:

```
streamlit run app.py
```

Описание датасета

Датасет **"AI4I 2020 Predictive Maintenance Dataset"** доступен по ссылке:
Predictive Maintenance Dataset

Он содержит синтетические данные, моделирующие задачу предиктивного обслуживания оборудования. Датасет состоит из **10 000 записей**, каждая из которых описывает состояние оборудования и включает **14 признаков**. Основные характеристики данных:

Переменные (Features):

Название переменной	Роль	Тип данных	Описание	Единицы измерения	Пропущенные значения
UID	ID	Integer	Уникальный идентификатор записи (от 1 до 10 000).	-	Нет
Product ID	ID	Categorical	Идентификатор продукта (L, M, H) и серийный номер.	-	Нет
Type	Feature	Categorical	Тип продукта (L, M, H).	-	Нет

Название переменной	Роль	Тип данных	Описание	Единицы измерения	Пропущенные значения
Air temperature [K]	Feature	Continuous	Температура окружающей среды.	Кельвины (K)	Нет
Process temperature [K]	Feature	Continuous	Рабочая температура процесса.	Кельвины (K)	Нет
Rotational speed [rpm]	Feature	Integer	Скорость вращения.	обороты/мин (rpm)	Нет
Torque [Nm]	Feature	Continuous	Крутящий момент.	Ньютон-метры (Nm)	Нет
Tool wear [min]	Feature	Integer	Износ инструмента.	минуты (min)	Нет

Целевые переменные (Targets):

Название переменной	Роль	Тип данных	Описание	Единицы измерения	Пропущенные значения
Machine failure	Target	Integer	Бинарная метка: 1 — отказ оборудования, 0 — отказ не произошел.	-	Нет
TWF	Target	Integer	Отказ из-за износа инструмента (Tool Wear Failure).	-	Нет
HDF	Target	Integer	Отказ из-за недостаточного теплоотвода (Heat Dissipation Failure).	-	Нет
PWF	Target	Integer	Отказ из-за недостаточной или избыточной мощности (Power Failure).	-	Нет

Название переменной	Роль	Тип данных	Описание	Единицы измерения	Пропущенные значения
OSF	Target	Integer	Отказ из-за перегрузки (Overstrain Failure).	-	Нет
RNF	Target	Integer	Случайный отказ (Random Failure).	-	Нет

Детализация данных:

1. **Product ID**:
    - Состоит из буквы (L, M, H) и серийного номера.
    - Буквы обозначают качество продукта:
        - **L (Low)**: Низкое качество (50% всех продуктов).
        - **M (Medium)**: Среднее качество (30%).
        - **H (High)**: Высокое качество (20%).
2. **Air temperature [K]**:
    - Температура окружающей среды, сгенерированная с использованием случайного блуждания.
    - Нормализована до стандартного отклонения 2 K вокруг 300 K.
3. **Process temperature [K]**:
    - Рабочая температура процесса, сгенерированная как случайное блуждание.
    - Нормализована до стандартного отклонения 1 K и добавлена к температуре окружающей среды плюс 10 K.
4. **Rotational speed [rpm]**:
    - Скорость вращения, рассчитанная на основе мощности 2860 Вт.
    - С добавлением нормально распределенного шума.
5. **Torque [Nm]**:
    - Крутящий момент, нормально распределенный вокруг 40 Нм с σ = 10 Нм.
    - Отрицательные значения отсутствуют.
6. **Tool wear [min]**:
    - Износ инструмента.
    - Для продуктов разного качества добавляется разное время износа:
        - H: +5 минут.
        - M: +3 минуты.
        - L: +2 минуты.

Типы отказов:

1. **Tool Wear Failure (TWF)**:
    - Отказ из-за износа инструмента.
    - Происходит при износе между 200 и 240 минутами.
    - В датасете: 120 случаев (69 замен, 51 отказ).
2. **Heat Dissipation Failure (HDF)**:
    - Отказ из-за недостаточного теплоотвода.
    - Происходит, если разница между температурами воздуха и процесса меньше 8.6 K, а скорость вращения меньше 1380 rpm.
    - В датасете: 115 случаев.
3. **Power Failure (PWF)**:
    - Отказ из-за недостаточной или избыточной мощности.
    - Происходит, если мощность выходит за пределы 3500–9000 Вт.
    - В датасете: 95 случаев.
4. **Overstrain Failure (OSF)**:
    - Отказ из-за перегрузки.
    - Происходит, если произведение износа инструмента и крутящего момента превышает пороговое значение:
        - L: 11,000 minNm.
        - M: 12,000 minNm.
        - H: 13,000 minNm.
    - В датасете: 98 случаев.
5. **Random Failures (RNF)**:
    - Случайные отказы.
    - Вероятность 0.1% для каждого процесса.
    - В датасете: 5 случаев.

Целевая переменная:

- **Machine failure**: Бинарная метка (0 или 1), где:
    - **0**: Отказ оборудования не произошел.
    - **1**: Отказ оборудования произошел (хотя бы один из типов отказов активирован).

Загрузка датасета:

Для загрузки датасета используйте библиотеку `ucimlrepo`. Установите её и импортируйте данные следующим образом:

1. Установите библиотеку:

```
pip install ucimlrepo
```

2. Импортируйте датасет в код:

```
from ucimlrepo import fetch_ucirepo

# Загрузка датасета
ai4i_2020_predictive_maintenance_dataset = fetch_ucirepo(id=601)

# Данные (в виде pandas DataFrame)
X = ai4i_2020_predictive_maintenance_dataset.data.features  # Признаки
y = ai4i_2020_predictive_maintenance_dataset.data.targets   # Целевые
переменные

# Метаданные
print(ai4i_2020_predictive_maintenance_dataset.metadata)

# Информация о переменных
print(ai4i_2020_predictive_maintenance_dataset.variables)
```

Ход работы

1. Загрузка и предобработка данных

Задача:

1. Загрузить данные из CSV-файла или с использованием библиотеки `ucimlrepo`.
2. Провести базовую предобработку данных:
   - Удалить ненужные столбцы.
   - Преобразовать категориальные переменные в числовые.
   - Проверить данные на наличие пропущенных значений.

Вариант 1: Загрузка данных из CSV-файла

Пример кода:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Загрузка данных
data = pd.read_csv("data/predictive_maintenance.csv")

# Удаление ненужных столбцов
data = data.drop(columns=['UDI', 'Product ID', 'TWF', 'HDF', 'PWF', 'OSF',
'RNF'])

# Преобразование категориальной переменной Type в числовую
data['Type'] = LabelEncoder().fit_transform(data['Type'])

# Проверка на пропущенные значения
print(data.isnull().sum())
```

Вариант 2: Загрузка данных с использованием библиотеки `ucimlrepo`

Пример кода:

```
from ucimlrepo import fetch_ucirepo
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Загрузка датасета
dataset = fetch_ucirepo(id=601)
data = pd.concat([dataset.data.features, dataset.data.targets], axis=1)

# Удаление ненужных столбцов
data = data.drop(columns=['UDI', 'Product ID', 'TWF', 'HDF', 'PWF', 'OSF',
'RNF'])

# Преобразование категориальной переменной Type в числовую
data['Type'] = LabelEncoder().fit_transform(data['Type'])

# Проверка на пропущенные значения
print(data.isnull().sum())
```

Описание переменных:

- **Type**: Тип продукта (L, M, H). Преобразуется в числа.
- **Air temperature**: Температура окружающей среды (K).
- **Process temperature**: Рабочая температура (K).
- **Rotational speed**: Скорость вращения (rpm).
- **Torque**: Крутящий момент (Nm).
- **Tool wear**: Износ инструмента (мин).
- **Machine failure**: Целевая переменная (0 — нет отказа, 1 — отказ).

Зачем нужна предобработка?

Предобработка данных — это важный этап, который подготавливает данные для обучения модели. Она включает:

1. **Удаление лишних столбцов**: Убираем столбцы, которые не несут полезной информации (например, уникальные идентификаторы).
2. **Преобразование категориальных данных**: Модели машинного обучения работают только с числами, поэтому категориальные переменные (например, `Type`) нужно преобразовать в числовой формат.
3. **Обработка пропущенных значений**: Пропуски в данных могут нарушить работу модели, поэтому их нужно либо удалить, либо заполнить.

4. **Масштабирование данных**: Числовые признаки (например, температура, скорость вращения) часто масштабируют для улучшения сходимости модели.

Пример кода с масштабированием данных:

```
from sklearn.preprocessing import StandardScaler

# Масштабирование числовых признаков
scaler = StandardScaler()
numerical_features = ['Air temperature', 'Process temperature', 'Rotational
speed', 'Torque', 'Tool wear']
data[numerical_features] = scaler.fit_transform(data[numerical_features])

# Вывод первых строк данных после масштабирования
print(data.head())
```

Пояснение:

- Масштабирование числовых признаков (например, температуры, скорости вращения) помогает улучшить сходимость модели.
- Используется `StandardScaler`, который нормализует данные (среднее = 0, стандартное отклонение = 1).

---

2. Разделение данных на обучающую и тестовую выборки

Задача:

1. Разделить данные на признаки (`X`) и целевую переменную (`y`).
2. Разделить данные на обучающую и тестовую выборки в соотношении 80/20.

Пояснение:

- Признаки (`X`) — это данные, которые модель будет использовать для обучения.
- Целевая переменная (`y`) — это то, что модель должна предсказать (в данном случае, `Target`).
- Обучающая выборка используется для обучения модели, а тестовая — для оценки её качества.

Пример кода:

```
from sklearn.model_selection import train_test_split

# Признаки (X) и целевая переменная (y)
X = data.drop(columns=['Target'])
y = data['Target']

# Разделение данных
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

---

3. Обучение модели

Задача:

1. Обучить несколько моделей машинного обучения на обучающей выборке.
2. Сравнить их производительность и выбрать наилучшую модель.

Пояснение:

- Для бинарной классификации можно использовать различные модели, каждая из которых имеет свои преимущества и недостатки.
- Выбор модели зависит от сложности данных, интерпретируемости и требований к производительности.

Вариант 1: Logistic Regression

Преимущества:

- Простота и интерпретируемость.
- Хорошо работает на линейно разделимых данных.

Пример кода:

```
from sklearn.linear_model import LogisticRegression

# Создание и обучение модели
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
```

Вариант 2: Random Forest

Преимущества:

- Устойчивость к переобучению.
- Хорошо работает на нелинейных данных.

Пример кода:

```
from sklearn.ensemble import RandomForestClassifier

# Создание и обучение модели
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

Вариант 3: Gradient Boosting (XGBoost)

Преимущества:

- Высокая точность на сложных данных.
- Поддержка регуляризации для предотвращения переобучения.

Пример кода:

```
from xgboost import XGBClassifier

# Создание и обучение модели
xgb = XGBClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
xgb.fit(X_train, y_train)
```

Вариант 4: Support Vector Machine (SVM)

Преимущества:

- Эффективен на данных с высокой размерностью.
- Подходит для сложных границ решений.

Пример кода:

```
from sklearn.svm import SVC

# Создание и обучение модели
svm = SVC(kernel='linear', random_state=42, probability=True)  #
probability=True для ROC-AUC
svm.fit(X_train, y_train)
```

---

4. Оценка модели

Задача:

1. Сделать предсказания на тестовой выборке для каждой модели.
2. Оценить качество моделей с помощью метрик:
    - **Accuracy** — доля правильных предсказаний.

- **Confusion Matrix** — матрица ошибок, показывающая, сколько примеров каждого класса было предсказано правильно и неправильно.
- **Classification Report** — отчет с метриками precision, recall, F1-score.
- **ROC-AUC** — площадь под ROC-кривой, которая показывает способность модели разделять классы.

Пример кода для оценки моделей:

```python
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# Функция для оценки модели
def evaluate_model(model, X_test, y_test):
    # Предсказания
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]  # Вероятности для
ROC-AUC

    # Метрики
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred_proba)

    # Вывод результатов
    print("Accuracy:", accuracy)
    print("Confusion Matrix:\n", conf_matrix)
    print("Classification Report:\n", class_report)
    print("ROC-AUC:", roc_auc)

    # Построение ROC-кривой
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    plt.plot(fpr, tpr, label=f"{model.__class__.__name__} (AUC =
{roc_auc:.2f})")

# Оценка Logistic Regression
print("Logistic Regression:")
evaluate_model(log_reg, X_test, y_test)

# Оценка Random Forest
print("Random Forest:")
evaluate_model(rf, X_test, y_test)

# Оценка XGBoost
print("XGBoost:")
evaluate_model(xgb, X_test, y_test)

# Оценка SVM
print("SVM:")
evaluate_model(svm, X_test, y_test)

# Визуализация ROC-кривых
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random
Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-кривые')
plt.legend()
plt.show()
```

6. Написание приложения и презентации

Задача:

1. Создать Streamlit-приложение с двумя страницами:
    - Основная страница: загрузка данных, обучение модели, визуализация результатов, предсказания.
    - Страница с презентацией: описание проекта и процесса работы.
2. Использовать `st.navigation` и `st.Page` для создания многостраничного приложения.
3. Использовать `streamlit-reveal-slides` для создания презентации (установка `pip install streamlit-reveal-slides`).

---

1. Основная страница: Анализ и модель

Задача: Создать страницу, на которой:

1. Пользователь может загрузить данные.
2. Обучить модель и вывести метрики.
3. Визуализировать результаты.
4. Ввести новые данные для предсказания.

Пример кода для `analysis_and_model.py`:

```
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report, roc_curve, roc_auc_score
from sklearn.preprocessing import LabelEncoder

def analysis_and_model_page():
    st.title("Анализ данных и модель")

    # Загрузка данных
    uploaded_file = st.file_uploader("Загрузите датасет (CSV)", type="csv")

    if uploaded_file is not None:
        data = pd.read_csv(uploaded_file)

        # Предобработка данных (дописать)
        # Удалить ненужные столбцы, преобразовать категориальные
переменные, проверить на пропуски

        # Разделение данных (дописать)
        # X = данные без целевой переменной
```

```
        # y = целевая переменная
        # X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

        # Обучение модели (дописать)
        # model = LogisticRegression()
        # model.fit(X_train, y_train)

        # Оценка модели (дописать)
        # y_pred = model.predict(X_test)
        # accuracy = accuracy_score(y_test, y_pred)
        # conf_matrix = confusion_matrix(y_test, y_pred)
        # classification_rep = classification_report(y_test, y_pred)

        # Визуализация результатов
        st.header("Результаты обучения модели")
        st.write(f"Accuracy: {accuracy:.2f}")

        st.subheader("Confusion Matrix")
        fig, ax = plt.subplots()
        sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', ax=ax)
        st.pyplot(fig)

        st.subheader("Classification Report")
        st.text(classification_rep)

        # Интерфейс для предсказания
        st.header("Предсказание по новым данным")
        with st.form("prediction_form"):
            st.write("Введите значения признаков для предсказания:")
            productID = st.selectbox("productID", ["L", "M", "H"])
            air_temp = st.number_input("air temperature [K]")
            process_temp = st.number_input("process temperature [K]")
            rotational_speed = st.number_input("rotational speed [rpm]")
            torque = st.number_input("torque [Nm]")
            tool_wear = st.number_input("tool wear [min]")

            submit_button = st.form_submit_button("Предсказать")

            if submit_button:
                # Преобразование введенных данных (дописать)
                # input_data = pd.DataFrame({...})

                # Предсказание (дописать)
                # prediction = model.predict(input_data)
                # prediction_proba = model.predict_proba(input_data)[:, 1]

                st.write(f"Предсказание: {prediction[0]}")
                st.write(f"Вероятность отказа: {prediction_proba[0]:.2f}")
```

---

2. Страница с презентацией

Задача: Создать страницу, на которой будет отображаться презентация проекта с использованием `streamlit-reveal-slides`.

Пример кода для `presentation.py`:

```python
import streamlit as st
import reveal_slides as rs

def presentation_page():
    st.title("Презентация проекта")

    # Содержание презентации в формате Markdown
    presentation_markdown = """
    # Прогнозирование отказов оборудования
    ---
    ## Введение
    - Описание задачи и датасета.
    - Цель: предсказать отказ оборудования (Target = 1) или его отсутствие
(Target = 0).
    ---
    ## Этапы работы
    1. Загрузка данных.
    2. Предобработка данных.
    3. Обучение модели.
    4. Оценка модели.
    5. Визуализация результатов.
    ---
    ## Streamlit-приложение
    - Основная страница: анализ данных и предсказания.
    - Страница с презентацией: описание проекта.
    ---
    ## Заключение
    - Итоги и возможные улучшения.
    """

    # Настройки презентации
    with st.sidebar:
        st.header("Настройки презентации")
        theme = st.selectbox("Тема", ["black", "white", "league", "beige",
"sky", "night", "serif", "simple", "solarized"])
        height = st.number_input("Высота слайдов", value=500)
        transition = st.selectbox("Переход", ["slide", "convex", "concave",
"zoom", "none"])
        plugins = st.multiselect("Плагины", ["highlight", "katex",
"mathjax2", "mathjax3", "notes", "search", "zoom"], [])

    # Отображение презентации
    rs.slides(
        presentation_markdown,
        height=height,
        theme=theme,
        config={
            "transition": transition,
            "plugins": plugins,
        },
        markdown_props={"data-separator-vertical": "^--$"},
    )
```

3. Основной файл приложения (`app.py`)

Задача: Создать entrypoint-файл, который управляет навигацией между страницами.

Пример кода для `app.py`:

```
import streamlit as st

# Настройка навигации
pages = {
    "Анализ и модель": st.Page("analysis_and_model.py", title="Анализ и модель"),
    "Презентация": st.Page("presentation.py", title="Презентация"),
}

# Отображение навигации
current_page = st.navigation(pages, position="sidebar", expanded=True)
current_page.run()
```

---

4. Запуск приложения

1. Установите зависимости:

   ```
   pip install -r requirements.txt
   ```

2. Запустите Streamlit-приложение:

   ```
   streamlit run app.py
   ```

Формат сдачи проекта

Для сдачи проекта необходимо подготовить Git-репозиторий, который будет содержать все необходимые файлы и инструкции для запуска приложения. Репозиторий должен быть структурирован следующим образом:

Структура репозитория

```
predictive_maintenance_project/
│
├── app.py                  # Основной файл Streamlit-приложения
(entrypoint)
├── analysis_and_model.py   # Основная страница с анализом данных и моделью
├── presentation.py         # Страница с презентацией проекта
├── requirements.txt        # Файл с зависимостями для установки библиотек
├── data/                   # Папка с данными (если данные не загружаются
через интерфейс)
│   └── predictive_maintenance.csv
├── README.md               # Подробное описание проекта
└── video/                  # Папка с видео-демонстрацией (опционально)
    └── demo.mp4
```

Требования к репозиторию

1. **README.md**:
    - Должен содержать подробное описание проекта, включая:
        - Цель проекта.
        - Описание датасета.
        - Инструкции по установке и запуску приложения.
        - Описание структуры репозитория.
        - Ссылку на видео-демонстрацию (или встроенное видео).
    - Пример структуры README.md:

```
# Проект: Бинарная классификация для предиктивного обслуживания
оборудования

## Описание проекта
Цель проекта — разработать модель машинного обучения, которая
предсказывает, произойдет ли отказ оборудования (Target = 1) или нет
(Target = 0). Результаты работы оформлены в виде Streamlit-приложени

## Датасет
Используется датасет **"AI4I 2020 Predictive Maintenance Dataset"**,
содержащий 10 000 записей с 14 признаками. Подробное описание датасе
можно найти в [документации]
(https://archive.ics.uci.edu/dataset/601/predictive+maintenance+data

## Установка и запуск
1. Клонируйте репозиторий:

   git clone <ссылка на репозиторий>

2. Установите зависимости:

   pip install -r requirements.txt

3. Запустите приложение:

   streamlit run app.py


## Структура репозитория
- `app.py`: Основной файл приложения.
- `analysis_and_model.py`: Страница с анализом данных и моделью.
- `presentation.py`: Страница с презентацией проекта.
- `requirements.txt`: Файл с зависимостями.
- `data/`: Папка с данными.
- `README.md`: Описание проекта.

## Видео-демонстрация
[Ссылка на видео](video/demo.mp4) или встроенное видео ниже:
<video src="video/demo.mp4" controls width="100%"></video>
```

2. **requirements.txt**:
   - Должен содержать все необходимые библиотеки для запуска проекта.
   - Пример содержимого:

```
streamlit==1.41.0
pandas==1.5.3
scikit-learn==1.2.0
matplotlib==3.6.2
seaborn==0.12.1
ucimlrepo==0.0.3
xgboost==1.7.6
streamlit-reveal-slides==0.1.0
```

3. **Видео-демонстрация**:
    - В репозитории должен быть файл с видео-демонстрацией (например, `video/demo.mp4`), где студенты показывают:
        - Клонирование репозитория.
        - Установку зависимостей.
        - Запуск приложения.
        - Демонстрацию функционала приложения (загрузка данных, обучение модели, предсказания).
    - Видео должно быть записано с нуля (начиная с `git clone`).

Инструкция по созданию видео

1. Запишите видео, начиная с клонирования репозитория:

```
git clone <ссылка на репозиторий>
cd predictive_maintenance_project
```

2. Продемонстрируйте функционал приложения:
    - Загрузку данных.
    - Обучение модели.
    - Визуализацию результатов.
    - Предсказания на новых данных.
3. Сохраните видео в папку `video/` и добавьте ссылку на него в `README.md`.

Проверка перед сдачей

Перед сдачей убедитесь, что:

1. Репозиторий содержит все необходимые файлы.
2. `README.md` содержит подробное описание проекта и инструкции по запуску.
3. Видео-демонстрация корректно загружена и доступна для просмотра.
4. Приложение запускается без ошибок на чистой среде (после `git clone` и `pip install -r requirements.txt`).

Дополнительные рекомендации

1. **Коммиты**:
    - Делайте частые и осмысленные коммиты. Каждый коммит должен отражать выполнение конкретной задачи (например, "Добавлен файл requirements.txt", "Реализована загрузка данных", "Добавлена страница с презентацией").
2. **Комментарии в коде**:

- Добавляйте комментарии в код, чтобы объяснить ключевые моменты (например, предобработку данных, обучение модели, оценку результатов).

3. **Тестирование**:
   - Убедитесь, что приложение работает корректно на разных платформах (Windows, macOS, Linux).

Рекомендуемые материалы:

1. **Predictive Maintenance Dataset**
   Официальная страница набора данных на UCI Machine Learning Repository.
   https://archive.ics.uci.edu/dataset/601/predictive+maintenance+dataset
   Описание: Датасет для задачи предиктивного обслуживания оборудования. Содержит 10 000 записей с 14 признаками.

2. **Streamlit Reveal Slides**
   Репозиторий с примером использования Streamlit для создания презентаций.
   https://github.com/bouzidanas/streamlit-reveal-slides
   Описание: Библиотека для создания интерактивных презентаций в Streamlit.

3. **Scikit-learn Documentation**
   Официальная документация библиотеки Scikit-learn.
   https://scikit-learn.org/stable/
   Описание: Руководства, примеры и API для работы с моделями машинного обучения.

4. **Streamlit Documentation**
   Официальная документация библиотеки Streamlit.
   https://docs.streamlit.io/
   Описание: Руководства по созданию веб-приложений с использованием Streamlit.

5. **XGBoost Documentation**
   Официальная документация библиотеки XGBoost.
   https://xgboost.readthedocs.io/
   Описание: Руководство по использованию XGBoost для задач классификации и регрессии.

6. **Pandas Documentation**
   Официальная документация библиотеки Pandas.
   https://pandas.pydata.org/docs/
   Описание: Руководство по работе с табличными данными, включая загрузку, обработку и анализ.

7. **Matplotlib Documentation**
   Официальная документация библиотеки Matplotlib.
   https://matplotlib.org/stable/contents.html
   Описание: Руководство по созданию графиков и визуализации данных.

8. **Seaborn Documentation**
   Официальная документация библиотеки Seaborn.

https://seaborn.pydata.org/
Описание: Руководство по созданию статистических графиков.

9. **Git Documentation**
   Официальная документация Git.
   https://git-scm.com/doc
   Описание: Руководство по использованию Git для управления версиями кода.

10. **Markdown Guide**
    Руководство по синтаксису Markdown.
    https://www.markdownguide.org/
    Описание: Полезный ресурс для оформления README.md и других текстовых файлов.

11. **Python Virtual Environments**
    Официальное руководство по созданию виртуальных окружений в Python.
    https://docs.python.org/3/tutorial/venv.html
    Описание: Как создать и использовать виртуальные окружения для изоляции зависимостей.

12. **ROC-AUC Explanation**
    Статья с объяснением метрики ROC-AUC.
    https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
    Описание: Подробное объяснение ROC-кривой и площади под ней (AUC).

13. **Confusion Matrix Explanation**
    Статья с объяснением матрицы ошибок (Confusion Matrix).
    https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/
    Описание: Как интерпретировать матрицу ошибок и использовать её для оценки моделей.

14. **Streamlit Components Gallery**
    Коллекция компонентов для Streamlit.
    https://streamlit.io/gallery
    Описание: Примеры использования Streamlit для создания интерактивных приложений.

15. **GitHub Guides**
    Официальные руководства по работе с GitHub.
    https://guides.github.com/
    Описание: Как создавать репозитории, работать с ветками и управлять проектами на GitHub.

Требования к отчету в формате DOCX

Отчет в формате DOCX должен содержать описание всех этапов работы над проектом, с акцентом на обоснование принятых решений и анализ результатов. Код должен быть либо вынесен в приложение, либо заменен ссылкой на репозиторий. Ниже приведена структура отчета.

Структура отчета

1. **Титульный лист**:
    - Название проекта: "Бинарная классификация для предиктивного обслуживания оборудования".
    - ФИО студента(ов).
    - Группа.
2. **Оглавление**:
    - Укажите номера страниц для каждого раздела.
3. **Введение**:
    - Краткое описание задачи и цели проекта.
    - Обоснование актуальности задачи (например, важность предиктивного обслуживания в промышленности).
4. **Описание датасета**:
    - Источник данных (ссылка на датасет).
    - Описание признаков и целевой переменной.
    - Примеры данных (можно вставить таблицу с несколькими строками датасета).
5. **Предобработка данных**:
    - Описание выполненных шагов и их обоснование:
        - Удаление ненужных столбцов (например, уникальные идентификаторы не несут полезной информации для модели).
        - Преобразование категориальных переменных (например, `Type` был преобразован в числовой формат, так как модели машинного обучения работают только с числами).
        - Проверка на пропущенные значения (пропусков не обнаружено, поэтому дополнительная обработка не потребовалась).
        - Масштабирование данных (например, числовые признаки были масштабированы для улучшения сходимости модели).
6. **Разделение данных**:
    - Обоснование выбора соотношения 80/20 для обучающей и тестовой выборок (стандартное соотношение, которое обеспечивает достаточный объем данных для обучения и проверки модели).
7. **Обучение модели**:
    - Описание выбранных моделей и их обоснование:
        - **Logistic Regression**: Простая и интерпретируемая модель, подходящая для бинарной классификации.
        - **Random Forest**: Устойчивая к переобучению модель, способная работать с нелинейными данными.
        - **XGBoost**: Мощная модель, которая часто показывает высокую точность на сложных данных.

- Краткое описание процесса обучения (например, модели были обучены на обучающей выборке с использованием стандартных параметров).

8. **Оценка модели**:
   - Описание метрик, используемых для оценки (Accuracy, Confusion Matrix, ROC-AUC).
   - Результаты оценки для каждой модели (можно вставить таблицу или графики).
   - Сравнение моделей и выбор наилучшей (например, Random Forest показал наилучшие результаты с Accuracy = 0.95 и ROC-AUC = 0.98).
9. **Streamlit-приложение**:
   - Описание функционала приложения:
     - Основная страница: загрузка данных, обучение модели, визуализация результатов, предсказания.
     - Страница с презентацией: описание проекта.
   - Скриншоты интерфейса приложения.
   - Обоснование выбора Streamlit (простота использования, возможность быстрого создания интерактивных веб-приложений).
10. **Заключение**:
    - Итоги работы: что удалось сделать, какие результаты получены.
    - Возможные улучшения (например, использование других моделей, более глубокая предобработка данных).
11. **Приложения**:
    - Ссылка на репозиторий с полным кодом проекта.
    - Скриншоты интерфейса приложения.
    - Графики и таблицы с результатами.

Рекомендации по оформлению

1. **Форматирование**:
   - Используйте заголовки и подзаголовки для структурирования текста.
   - Выделяйте ключевые моменты (например, названия моделей, метрики) жирным шрифтом или курсивом.
   - Добавьте нумерованные или маркированные списки для перечисления шагов.
2. **Графики и таблицы**:
   - Вставляйте графики (например, ROC-кривые, Confusion Matrix) и таблицы с результатами.
   - Под каждым графиком или таблицей добавьте пояснение (например, "Рисунок 1: ROC-кривая для модели Logistic Regression").
3. **Скриншоты**:

- Добавьте скриншоты интерфейса Streamlit-приложения.
- Под каждым скриншотом добавьте пояснение (например, "Рисунок 2: Интерфейс загрузки данных").

Заключение

Следуя этим инструкциям, вы сможете правильно оформить и сдать проект. Убедитесь, что все требования выполнены, и репозиторий содержит всю необходимую информацию для запуска и проверки вашего приложения. Удачи!