**Цель проекта**

* Научиться предсказывать отток клиентов (churn) на основе исторических данных.
* Провести анализ данных (EDA) с помощью Pandas, NumPy, Matplotlib, Seaborn.
* Построить и сравнить две модели классификации: **логистическую регрессию** и **дерево решений**.
* Оценить качество моделей с помощью метрик классификации.
* Сделать вывод, какая модель лучше справляется с задачей.
* (Дополнительно) Подготовить модель для использования в API (FastAPI).

(Можно изменить при необходимости после EDA)

| **Признак** | **Тип** | **Описание** |
| --- | --- | --- |
| tenure | int | Срок пребывания клиента в месяцах |
| MonthlyCharges | float | Месячный платеж |
| TotalCharges | float | Общая сумма платежей |
| Contract | object | Тип контракта (Month-to-month, One year, Two year) |
| InternetService | object | Тип интернет-услуги (DSL, Fiber optic, No) |
| OnlineSecurity | object | Наличие онлайн-защиты (Yes, No, No internet service) |
| TechSupport | object | Наличие техподдержки (Yes, No, No internet service) |
| Churn (целевая) | object | Факт оттока (Yes = ушёл, No = остался) |

**ЭТАП 1: Анализ данных (EDA)**

**Общая загрузка**

1. Загрузить CSV-файл и вывести первые 10 строк.
2. Посмотреть .info() и .shape.
3. Проверить уникальные значения по object-признакам.
4. Преобразовать TotalCharges в числовой тип.

**Pandas-практика**

1. Средний tenure, MonthlyCharges, TotalCharges.
2. Распределение по Contract, InternetService, Churn — value\_counts().
3. Средние показатели по группам Churn.
4. Количество клиентов с tenure < 3 месяцев.
5. Доля клиентов с MonthlyCharges > 100.
6. Группировка по Contract — средний TotalCharges.

**Визуализация**

1. hist / distplot по MonthlyCharges, tenure, TotalCharges.
2. boxplot: Churn vs MonthlyCharges.
3. barplot: Churn vs Contract.
4. heatmap корреляций числовых признаков.
5. pairplot по числовым признакам + Churn.

**ЭТАП 2: Предобработка**

1. Проверить пропущенные значения и заполнить/удалить.
2. Кодировка категориальных признаков (Contract, InternetService, OnlineSecurity, TechSupport, Churn) с помощью:

* LabelEncoder или OneHotEncoder для моделей.

1. Масштабирование числовых признаков (StandardScaler).
2. Разделить данные на X и y (где y = Churn).
3. Разделить выборку на train/test (80/20).

**ЭТАП 3: Обучение моделей**

**3.1 Логистическая регрессия**

1. Обучить LogisticRegression() на обучающей выборке.
2. Сделать предсказание на тесте.
3. Посчитать метрики:

* Accuracy
* Precision, Recall
* F1-score
* Confusion Matrix (с графиком)
* ROC AUC (и построить кривую)

1. Вывести коэффициенты модели.

**3.2 Дерево решений**

1. Обучить DecisionTreeClassifier() на той же выборке.
2. Повторить шаги 24–25 для дерева.
3. Построить дерево (с помощью plot\_tree() или export\_graphviz).
4. Оценить переобучение по глубине дерева (max\_depth).

**ЭТАП 4: Сравнение моделей**

1. Сравнить Accuracy, Precision, Recall, F1-score, ROC AUC между моделями.
2. Сделать вывод: какая модель лучше и почему.
3. Подобрать параметры (грид-серч) для дерева, если оно слабее.

**(Дополнительно) ЭТАП 5: API на FastAPI**

1. Подготовить model.pkl (лучшую модель).
2. Написать FastAPI-приложение:

* Загрузка модели
* POST-запрос /predict
* Ввод: JSON с признаками
* Вывод: {"churn": true, "probability": 0.84}

1. Обработать категориальные признаки в API.
2. Запустить uvicorn main:app --reload.

**(Дополнительно) ЭТАП 6: Деплой на сервер**

1. Поднять AWS EC2 / Render / Railway.
2. Установить зависимости.
3. Перенести проект и модель.
4. Запустить сервер.
5. Протестировать POST-запрос через публичный IP или curl.