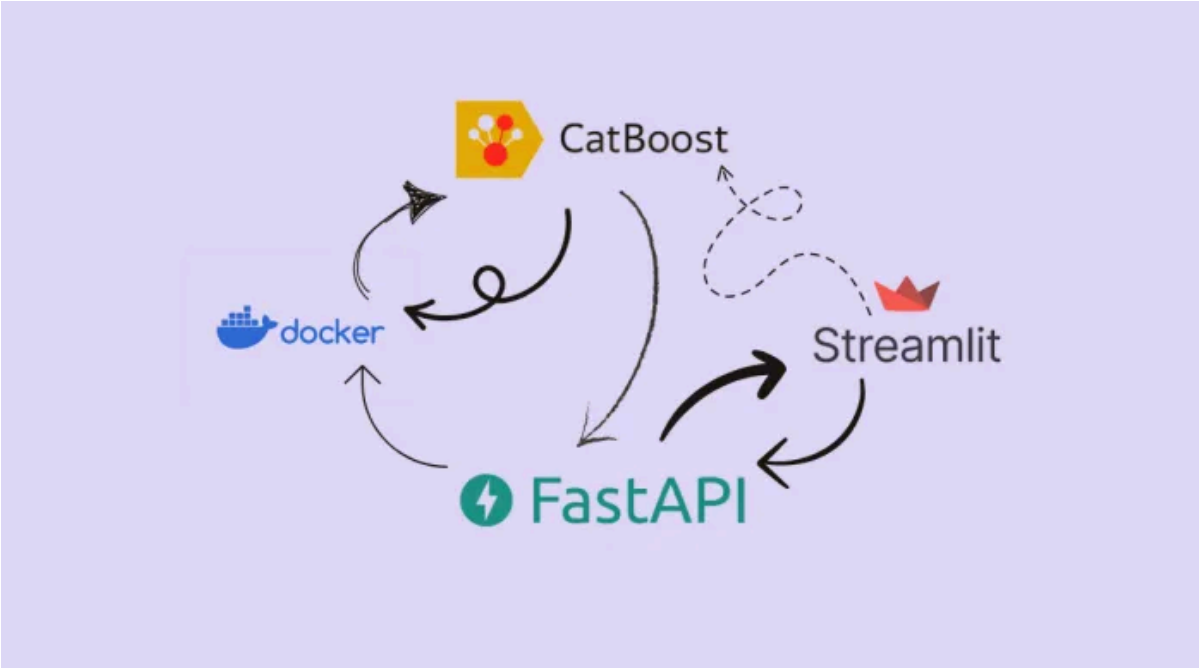
End-to-End Machine Learning Project: Churn Prediction

Ramazan Olmez · Follow
18 min read · Feb 22, 2024



The main objective of this article is to develop an end-to-end machine learning project. For a model to be truly useful, it needs to be stored and distributed. Within the scope of this article, I aim to develop a project based on a real-world scenario.

You can access the project on GitHub. You can also access the EDA and model development section on Kaggle.

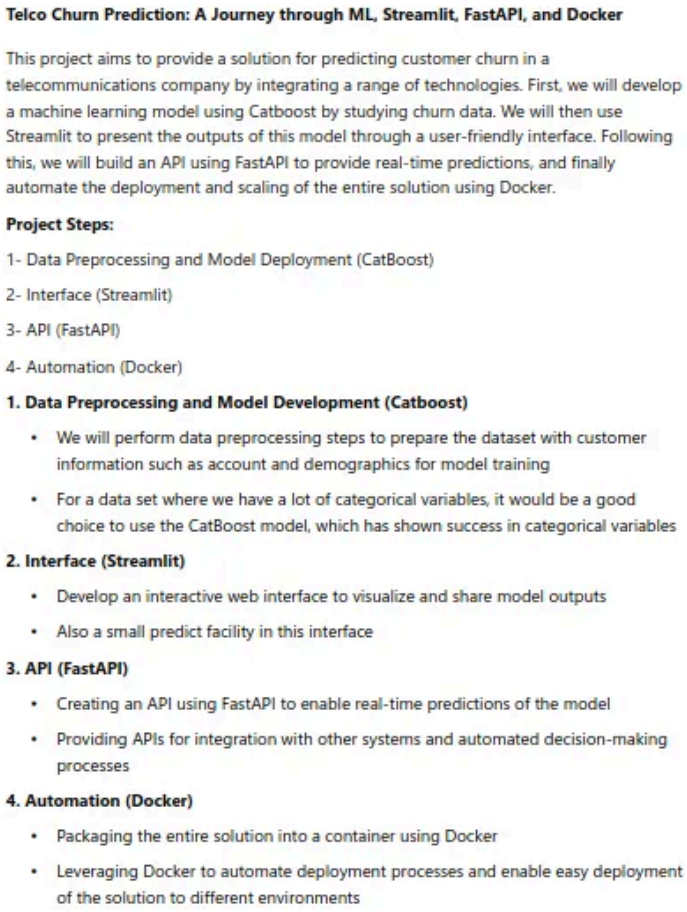
- GitHub
- EDA Kaggle
- Model Development Kaggle

The following tools will be used throughout the project.

- Catboost
- Streamlit
- FastAPI
- Docker

Table of Contents

1. Aim of the Project and Data Set



Telco Churn Prediction: A Journey through ML, Streamlit, FastAPI, and Docker

This project aims to provide a solution for predicting customer churn in a telecommunications company by integrating a range of technologies. First, we will develop a machine learning model using Catboost by studying churn data. We will then use Streamlit to present the outputs of this model through a user-friendly interface. Following this, we will build an API using FastAPI to provide real-time predictions, and finally automate the deployment and scaling of the entire solution using Docker.

Project Steps:

1- Data Preprocessing and Model Deployment (CatBoost)

2- Interface (Streamlit)

3- API (FastAPI)

4- Automation (Docker)

1. Data Preprocessing and Model Development (Catboost)

- We will perform data preprocessing steps to prepare the dataset with customer information such as account and demographics for model training
- For a data set where we have a lot of categorical variables, it would be a good choice to use the CatBoost model, which has shown success in categorical variables

2. Interface (Streamlit)

- Develop an interactive web interface to visualize and share model outputs
- Also a small predict facility in this interface

3. API (FastAPI)

- Creating an API using FastAPI to enable real-time predictions of the model
- Providing APIs for integration with other systems and automated decision-making processes

4. Automation (Docker)

- Packaging the entire solution into a container using Docker
- Leveraging Docker to automate deployment processes and enable easy deployment of the solution to different environments

Aim of the project

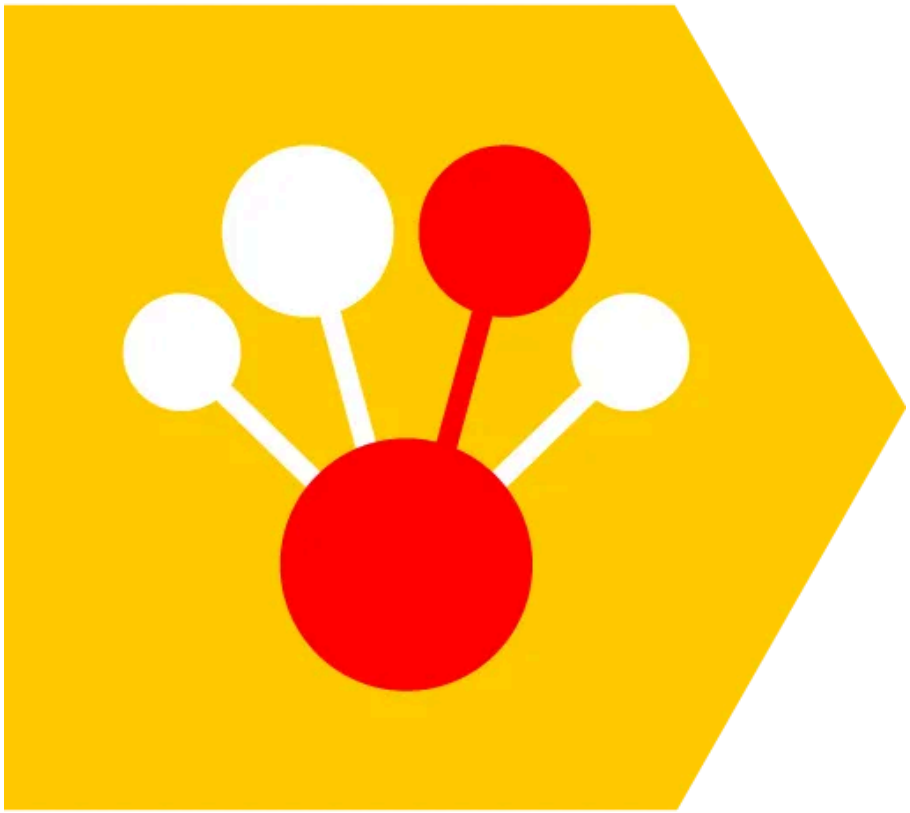
Data Set

CustomerID	A unique ID that identifies each customer
Gender	The customer's gender
Senior Citizen	Indicates if the customer is 65 or older
Partner	Indicates if the customer is married
Dependents	Indicates if the customer lives with any dependents
Tenure	Indicates the total amount of months that the customer has been with the company
Phone Service	Indicates if the customer subscribes to home phone service with the company
Multiple Lines	Indicates if the customer subscribes to multiple telephone lines with the company
Internet Service	Indicates if the customer subscribes to Internet service with the company
Online Security	Indicates if the customer subscribes to an additional online security service
Online Backup	Indicates if the customer subscribes to an additional online backup service
Device Protection Plan	Indicates if the customer subscribes to an additional device protection plan
Premium Tech Support	Indicates if the customer subscribes to an additional technical support plan
Streaming TV	Indicates if the customer uses their Internet service to stream television programing
Streaming Movies	Indicates if the customer uses their Internet service to stream movies
Contract	Indicates the customer's current contract type
Paperless Billing	Indicates if the customer has chosen paperless billing
Payment Method	Indicates how the customer pays their bill
Monthly Charge	Indicates the customer's current total monthly charge for all their services from the company.
Total Charges	Indicates the customer's total charges, calculated to the end of the quarter specified above
Churn	Directly related to Churn Value. (Target Variable)

Dataset

You can access the customer data to be used in the project from Kaggle. This dataset consists of 7043 customers with 21 columns (features). It contains customer account information, demographic information, and registered services. The target variable (Churn) provides information on whether the customer has churned.
You can also access the extended and updated dataset version from this link.

2. Data Preprocessing and Model Development



Source

There are quite a lot of categorical variables in the dataset. By encoding these variables into numerical form, we can also use different models with different preprocessing techniques. But in this study, we will use the Catboost algorithm, which has proven its success with categorical variables.

If you want to improve performance in model development, you can try various tools that provide hyperparameter optimization such as Optuna, etc.

We can start data preprocessing and model development by creating a Python file called `train_model.py`

```
################################################## Import Libraries ############

import numpy as np
import pandas as pd
import os

from sklearn import metrics
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.metrics import (
    accuracy_score, classification_report, recall_score, confusion_matrix,
    roc_auc_score, precision_score, f1_score, roc_curve, auc
)
from sklearn.preprocessing import OrdinalEncoder

from catboost import CatBoostClassifier, Pool

################################################## Data Loading and Editing ####

data_path = "../data/WA_Fn-UseC_-Telco-Customer-Churn.csv"
df = pd.read_csv(data_path)

# Convert TotalCharges to numeric, filling NaN values
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
df['TotalCharges'].fillna(df['tenure'] * df['MonthlyCharges'], inplace=True)

# Convert SeniorCitizen to object
df['SeniorCitizen'] = df['SeniorCitizen'].astype(object)

# Replace 'No phone service' and 'No internet service' with 'No' for certain col
df['MultipleLines'] = df['MultipleLines'].replace('No phone service', 'No')
columns_to_replace = ['OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'Tec
for column in columns_to_replace:
    df[column] = df[column].replace('No internet service', 'No')

# Convert 'Churn' categorical variable to numeric
df['Churn'] = df['Churn'].replace({'No': 0, 'Yes': 1})

################################################## StratifiedShuffleSplit ######

# Create the StratifiedShuffleSplit object
strat_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=64)

train_index, test_index = next(strat_split.split(df, df["Churn"]))
```

```
# Create train and test sets
strat_train_set = df.loc[train_index]
strat_test_set = df.loc[test_index]

X_train = strat_train_set.drop("Churn", axis=1)
y_train = strat_train_set["Churn"].copy()

X_test = strat_test_set.drop("Churn", axis=1)
y_test = strat_test_set["Churn"].copy()

################################################## CATBOOST ####################

# Identify categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns.tolist()

# Initialize and fit CatBoostClassifier
cat_model = CatBoostClassifier(verbose=False, random_state=0, scale_pos_weight=3
cat_model.fit(X_train, y_train, cat_features=categorical_columns, eval_set=(X_te

# Predict on test set
y_pred = cat_model.predict(X_test)

# Calculate evaluation metrics
accuracy, recall, roc_auc, precision = [round(metric(y_test, y_pred), 4) for met

# Create a DataFrame to store results
model_names = ['CatBoost_Model']
result = pd.DataFrame({'Accuracy': accuracy, 'Recall': recall, 'Roc_Auc': roc_au

# Print results
print(result)

# Save the model in the 'model' directory
model_dir = "../model"
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

model_path = os.path.join(model_dir, "catboost_model.cbm")
cat_model.save_model(model_path)
```

Step 1: Importing Libraries

We start by importing essential libraries required for data processing, model training, and evaluation. These include libraries such as NumPy and Pandas for data manipulation, scikit-learn for machine learning functionalities, and CatBoost for building our predictive model.

Step 2: Data Loading and Editing

Next, we load the dataset from a specified path using Pandas. We perform

some initial data editing steps to ensure data consistency and readiness for modeling. This involves converting certain columns to numeric data types, handling missing values, and encoding categorical variables.

Step 3: Stratified Splitting
The unbalanced nature of the dataset was observed in the EDA study. To ensure an unbiased distribution of classes in our train-test split, we employ StratifiedShuffleSplit from scikit-learn. This method preserves the proportion of classes in both training and testing sets, critical for reliable model evaluation.

Step 4: Model Training with CatBoost
Training our predictive model using CatBoost. We initialize a CatBoostClassifier with appropriate parameters, including verbosity and random state. We fit the model to the training data, specifying categorical features to ensure proper handling of categorical variables. I would like to emphasize that the model training phase has not been elaborated and it may be good to revisit this phase in your work.

Step 5: Model Evaluation
Once the model is trained, we evaluate its performance using various metrics such as accuracy, recall, ROC AUC score, and precision. These metrics provide insights into how well our model predicts customer churn.

After developing the model with your data, we need to save the model for later use.

After analyzing the data and building the machine learning model, interactive interfaces are required to increase the usability of this model. Streamlit is a good alternative for this task.

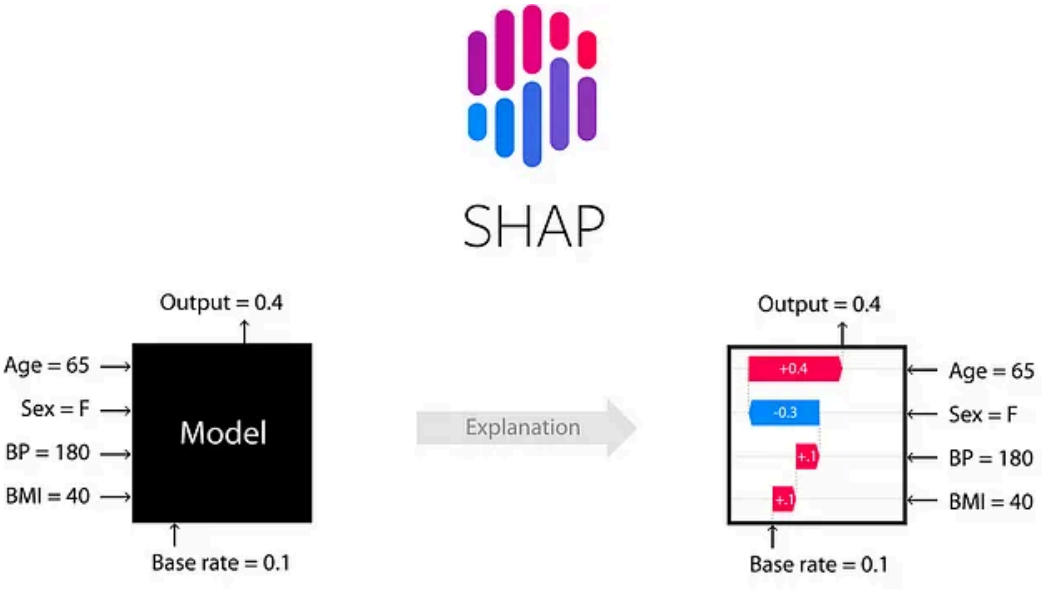
3. Interface (Streamlit)

Source

Streamlit is an open-source application framework for machine learning and data science teams. It is often used in the prototyping phase of projects as it provides a fast interface. It is easy to use and has a strong community.

We started our project by creating a python file named `streamlit-app.py` to create a fast interface. The following code allows us to create an interactive interface using Streamlit and evaluate the churn probability of customers using the CatBoost model. It also allows us to understand and visualize the model's decisions using SHAP values.

At this point in the project, I think it might be good to briefly mention **SHAP**.



Source

SHAP (SHapley Additive exPlanations) is one of the explainable artificial intelligence methods developed with a game theoretic approach. It allows us to comment on the model, making it less of a black box. It tells us to what extent each attribute used in the model affects the model outcome.

Let's continue creating the `streamlit-app.py` file.

```
import shap
import pandas as pd
import numpy as np
import streamlit as st
from matplotlib import pyplot as plt
from pyarrow import parquet as pq
from catboost import CatBoostClassifier, Pool
import joblib

# Path of the trained model and data
MODEL_PATH = "../model/cat_model.cbm"
DATA_PATH = "../data/churn_data_regulated.parquet"

st.set_page_config(page_title="Churn Project")

@st.cache_resource
def load_data():
    data = pd.read_parquet(DATA_PATH)
    return data

def load_x_y(file_path):
    data = joblib.load(file_path)
    data.reset_index(drop=True, inplace=True)
    return data

def load_model():
    model = CatBoostClassifier()
    model.load_model(MODEL_PATH)
    return model

def calculate_shap(model, X_train, X_test):
    # Calculate SHAP values
    explainer = shap.TreeExplainer(model)
    shap_values_cat_train = explainer.shap_values(X_train)
    shap_values_cat_test = explainer.shap_values(X_test)
    return explainer, shap_values_cat_train, shap_values_cat_test

def plot_shap_values(model, explainer, shap_values_cat_train, shap_values_cat_te
    # Visualize SHAP values for a specific customer
    customer_index = X_test[X_test['customerID'] == customer_id].index[0]
    fig, ax_2 = plt.subplots(figsize=(6,6), dpi=200)
    shap.decision_plot(explainer.expected_value, shap_values_cat_test[customer_i
    st.pyplot(fig)
```

```
    plt.close()

def display_shap_summary(shap_values_cat_train, X_train):
    # Create the plot summarizing the SHAP values
    shap.summary_plot(shap_values_cat_train, X_train, plot_type="bar", plot_size
    summary_fig, _ = plt.gcf(), plt.gca()
    st.pyplot(summary_fig)
    plt.close()

def display_shap_waterfall_plot(explainer, expected_value, shap_values, feature_
    # Create SHAP waterfall drawing
    fig, ax = plt.subplots(figsize=(6, 6), dpi=150)
    shap.plots._waterfall.waterfall_legacy(expected_value, shap_values, feature_
    st.pyplot(fig)
    plt.close()

def summary(model, data, X_train, X_test):
    # Calculate SHAP values
    explainer, shap_values_cat_train, shap_values_cat_test = calculate_shap(mode

    # Summarize and visualize SHAP values
    display_shap_summary(shap_values_cat_train, X_train)

def plot_shap(model, data, customer_id, X_train, X_test):
    # Calculate SHAP values
    explainer, shap_values_cat_train, shap_values_cat_test = calculate_shap(mode

    # Visualize SHAP values
    plot_shap_values(model, explainer, shap_values_cat_train, shap_values_cat_te

    # Waterfall
    customer_index = X_test[X_test['customerID'] == customer_id].index[0]
    display_shap_waterfall_plot(explainer, explainer.expected_value, shap_values

st.title("Telco Customer Churn Project")

def main():
    model = load_model()
    data = load_data()

    X_train = load_x_y("../data/X_train.pkl")
    X_test = load_x_y("../data/X_test.pkl")
    y_train = load_x_y("../data/y_train.pkl")
    y_test = load_x_y("../data/y_test.pkl")

    max_tenure = data['tenure'].max()
    max_monthly_charges = data['MonthlyCharges'].max()
    max_total_charges = data['TotalCharges'].max()

    # Radio buttons for options
    election = st.radio("Make Your Choice:", ("Feature Importance", "User-based
    available_customer_ids = X_test['customerID'].tolist()

    # If User-based SHAP option is selected
    if election == "User-based SHAP":
        # Customer ID text input
```

```
        customer_id = st.selectbox("Choose the Customer", available_customer_ids
        customer_index = X_test[X_test['customerID'] == customer_id].index[0]
        st.write(f'Customer {customer_id}: Actual value for the Customer Churn :
        y_pred = model.predict(X_test)
        st.write(f"Customer {customer_id}: CatBoost Model's prediction for the C
        plot_shap(model, data, customer_id, X_train=X_train, X_test=X_test)

    # If Feature Importance is selected
    elif election == "Feature Importance":
        summary(model, data, X_train=X_train, X_test=X_test)

    # If Calculate CHURN Probability option is selected
    elif election == "Calculate the probability of CHURN":
        # Retrieving data from the user
        customerID = "6464-UIAEA"
        gender = st.selectbox("Gender:", ("Female", "Male"))
        senior_citizen = st.number_input("SeniorCitizen (0: No, 1: Yes)", min_va
        partner = st.selectbox("Partner:", ("No", "Yes"))
        dependents = st.selectbox("Dependents:", ("No", "Yes"))
        tenure = st.number_input("Tenure:", min_value=0, max_value=max_tenure, s
        phone_service = st.selectbox("PhoneService:", ("No", "Yes"))
        multiple_lines = st.selectbox("MultipleLines:", ("No", "Yes"))
        internet_service = st.selectbox("InternetService:", ("No", "DSL", "Fiber
        online_security = st.selectbox("OnlineSecurity:", ("No", "Yes"))
        online_backup = st.selectbox("OnlineBackup:", ("No", "Yes"))
        device_protection = st.selectbox("DeviceProtection:", ("No", "Yes"))
        tech_support = st.selectbox("TechSupport:", ("No", "Yes"))
        streaming_tv = st.selectbox("StreamingTV:", ("No", "Yes"))
        streaming_movies = st.selectbox("StreamingMovies:", ("No", "Yes"))
        contract = st.selectbox("Contract:", ("Month-to-month", "One year", "Two
        paperless_billing = st.selectbox("PaperlessBilling", ("No", "Yes"))
        payment_method = st.selectbox("PaymentMethod:", ("Electronic check", "Ma
        monthly_charges = st.number_input("Monthly Charges", min_value=0.0, max_
        total_charges = st.number_input("Total Charges", min_value=0.0, max_valu

        # Confirmation button
        confirmation_button = st.button("Confirm")

        # When the confirmation button is clicked
        if confirmation_button:
            # Convert user-entered data into a data frame
            new_customer_data = pd.DataFrame({
                "customerID": [customerID],
                "gender": [gender],
                "SeniorCitizen": [senior_citizen],
                "Partner": [partner],
                "Dependents": [dependents],
                "tenure": [tenure],
                "PhoneService": [phone_service],
                "MultipleLines": [multiple_lines],
                "InternetService": [internet_service],
                "OnlineSecurity": [online_security],
                "OnlineBackup": [online_backup],
                "DeviceProtection": [device_protection],
                "TechSupport": [tech_support],
                "StreamingTV": [streaming_tv],
```

```
                "StreamingMovies": [streaming_movies],
                "Contract": [contract],
                "PaperlessBilling": [paperless_billing],
                "PaymentMethod": [payment_method],
                "MonthlyCharges": [monthly_charges],
                "TotalCharges": [total_charges]
            })

            # Predict churn probability using the model
            churn_probability = model.predict_proba(new_customer_data)[:, 1]

            # Format churn probability
            formatted_churn_probability = "{:.2%}".format(churn_probability.item

            big_text = f"<h1>Churn Probability: {formatted_churn_probability}</h
            st.markdown(big_text, unsafe_allow_html=True)
            st.write(new_customer_data.to_dict())

if __name__ == "__main__":
    main()
```

Step 1: Setting Up

Streamlit app begins by importing necessary libraries, including Streamlit itself, Pandas for data manipulation, CatBoost for loading the trained model, and SHAP for visualizing model explanations.

Step 2: Data Loading and Model Loading

The app loads the preprocessed data and the trained CatBoost model. This ensures that we have the necessary data and model ready for inference.

Step 3: User Interface Components

The app provides a user interface with different options:

- Feature Importance: Allows users to view the summary of feature importance based on SHAP values.
- User-based SHAP: Enables users to select a specific customer and visualize the SHAP values for that customer, helping understand the model's prediction rationale.

- Calculate the probability of CHURN: Lets users input their data for a hypothetical customer and calculates the probability of churn using the trained model.

Step 4: Loading and Preparing User Data
If the user selects the “Calculate the probability of CHURN” option, the app prompts them to input various customer attributes such as gender, tenure, contract type, etc. Upon confirmation, the app formats this input data into a DataFrame and feeds it into the model to predict the probability of churn for the customer.

Step 5: Displaying Results
Finally, the app presents the results to the user. For the “Calculate the probability of CHURN” option, it displays the churn probability along with the input data in a formatted manner. For other options like “Feature Importance” and “User-based SHAP”, it generates visualizations to help users interpret the model’s behavior.

You can run this code from the terminal in the folder with the file named `streamlit-app.py` as follows

```
streamlit run streamlit-app.py
```

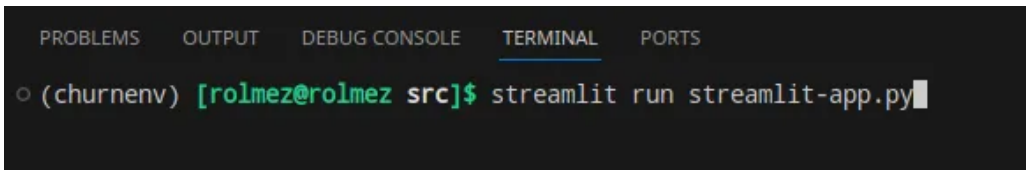
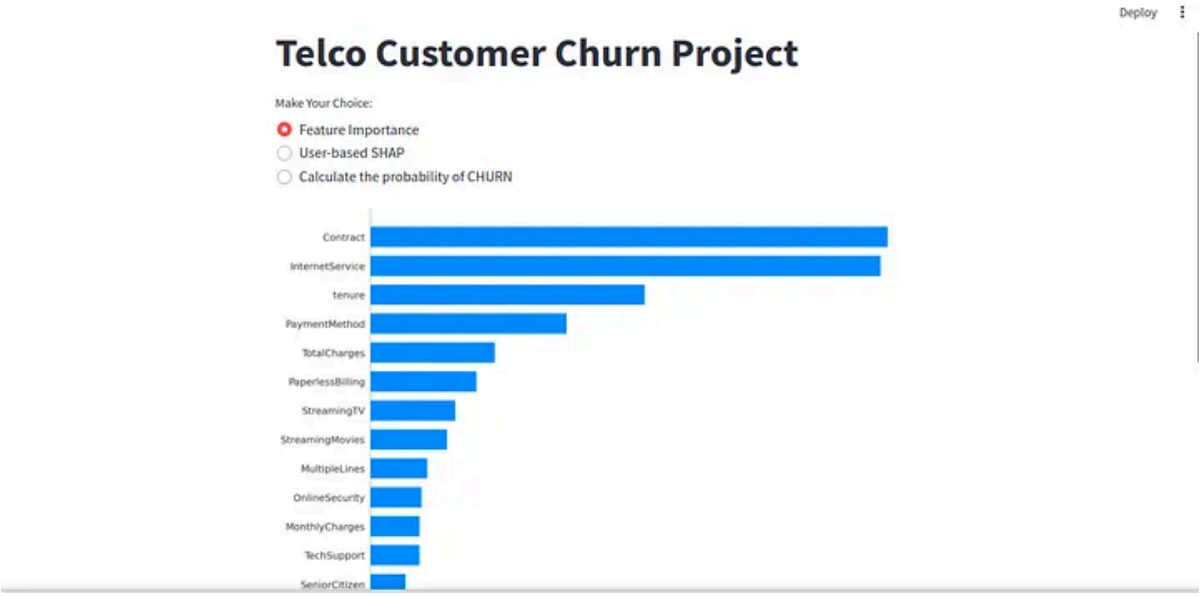


Image by author

We can see the application interface in the window that opens on localhost in the browser. This interface has different options such as the SHAP

summary, SHAP data for a specific user, and the calculation of the churn probability of a customer based on new values entered by the user.

Feature Importance: We may want to take an overview of which features are decisive in the model's decision-making process.

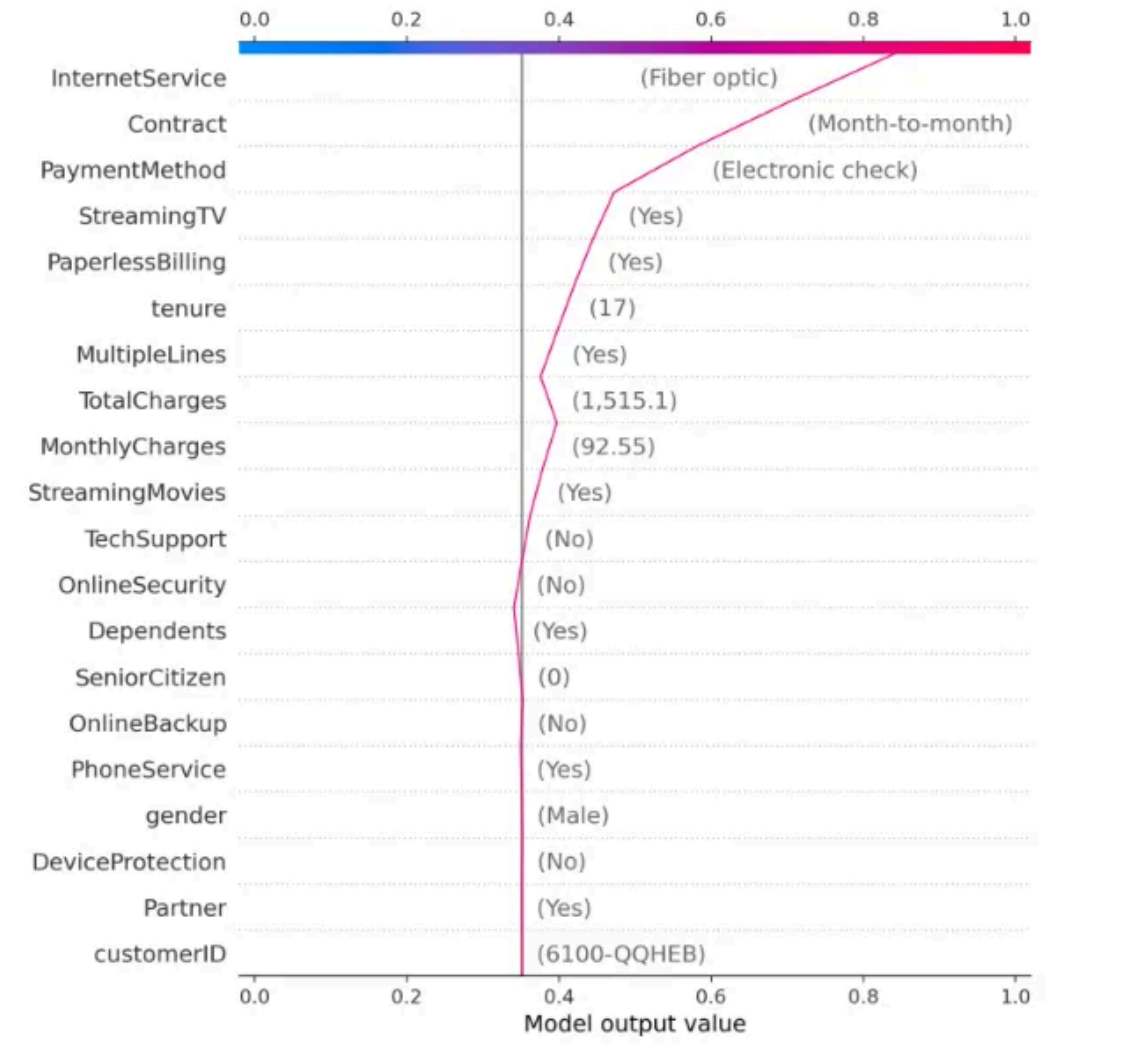


The order of importance of features for the created CatBoost model is as follows

User-based SHAP: We may want to know the criteria by which a particular customer's model result was selected. Let's take a look at the customer in our test set with customerID **6100-QQQQHEB**.

6100-QQQQQQQHEB Customer's churn values predicted by the model and actual churn values

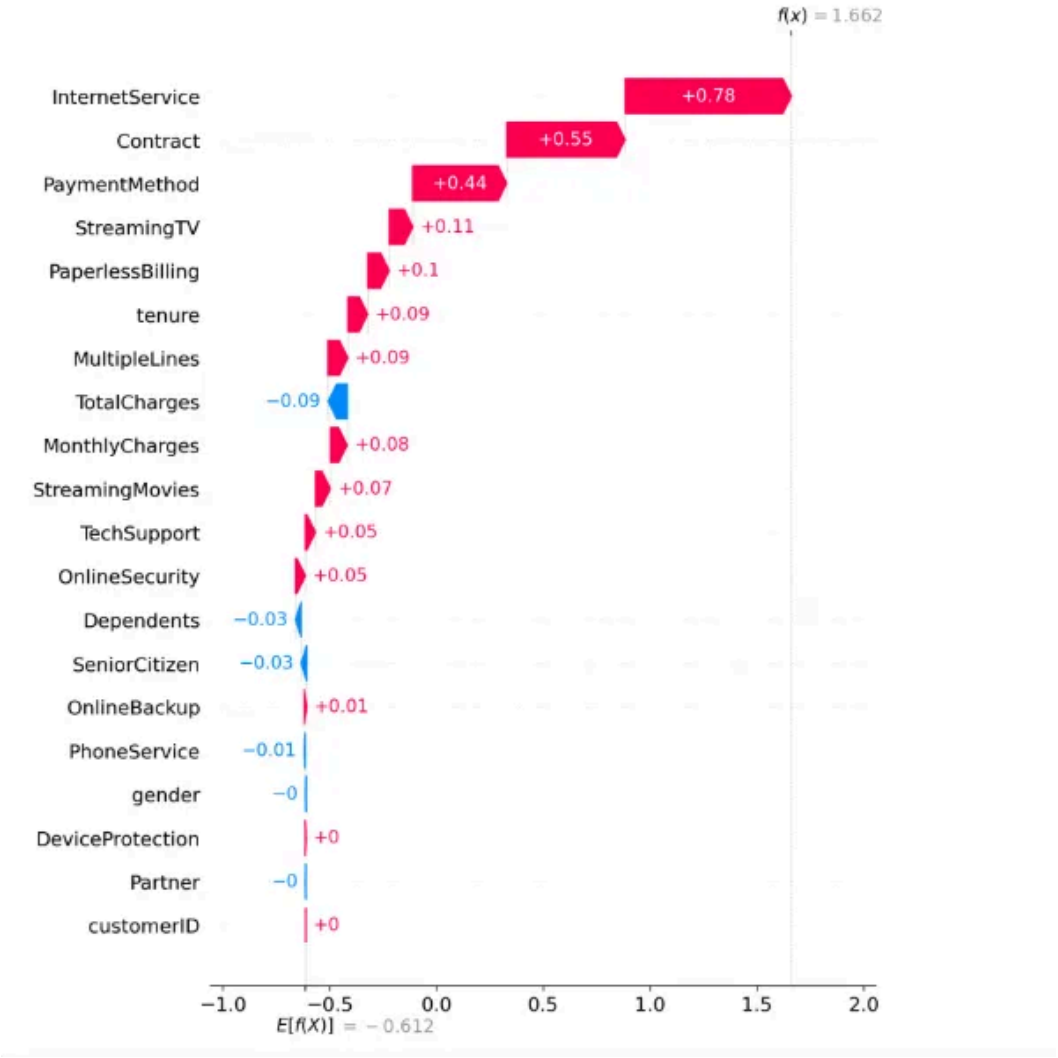
It is seen that the customer with customerID **6100-QQQHEB** is Churn according to the data set. Also, the model says that this customer will be Churn according to the training set it is trained. But this information is not enough. We may want to see based on which features it says that the customer will be Churn.



Features that influenced the model to make this decision

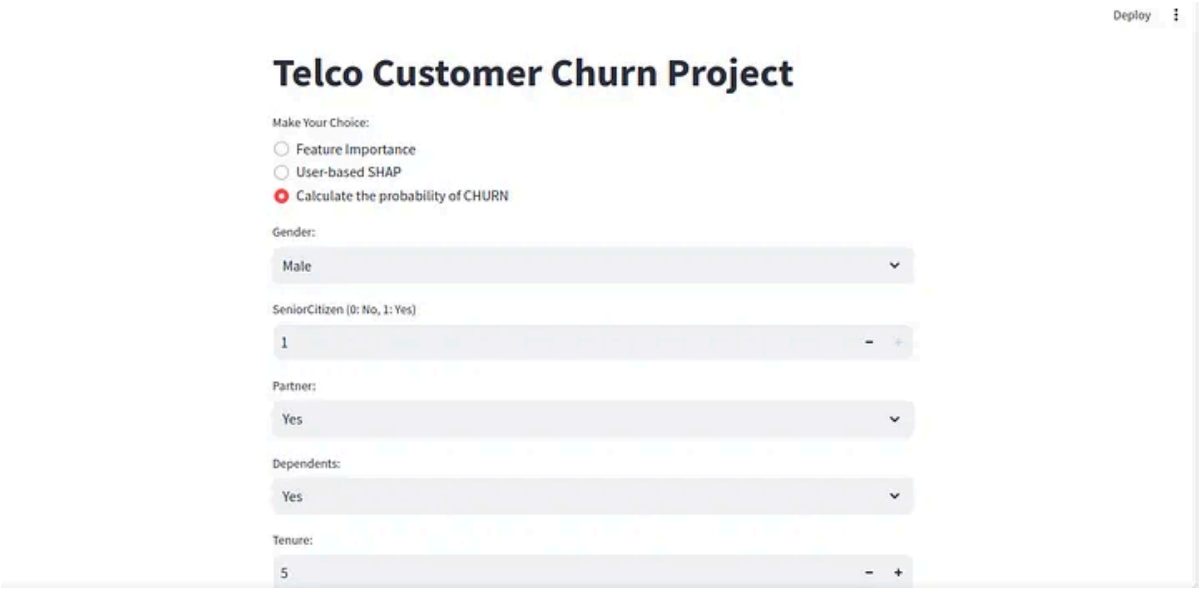
The SHAP chart in the photo shows that the customer with ID **6100-QQQQHEB** has a Churn probability of over **80%**. The model makes this decision because the user's InternetService feature is Fiber Optic, Contract feature is Month-to-month, and PaymentMethod feature is Electronic check. Other features also confirm this decision but are negligible.

Similar inferences can be seen in another SHAP graph.



Features that influenced the model to make this decision

Calculate the probability of CHURN: Through the model and streamlit interface we have created, we can ask users to give new customer values and calculate the churn probability in line with these values.



Calculate the probability of CHURN 1

Calculate the probability of CHURN 2

According to the values entered, the churn probability of the new customer created was **71.33%**.

Besides an interactive interface, it is vital to quickly and reliably convert the model into an API. API enables the model to be used by a wider audience, integrated with other applications and made easier to use. FastAPI is a good example of all these processes.

4. API (FastAPI)

Source

FastAPI is a Python web framework for developing fast web applications and APIs. It is especially focused on performance, speed, and security, and is very useful for API development thanks to its asynchronous programming support and Python typing support.

Within the scope of the project, uvicorn was used to launch the FastAPI application. Uvicorn is a server application for Python web servers. It

provides a fast installation like FastAPI. It is also very easy to use. Alternatives such as Gunicorn, Hypercorn, and Daphne can be used instead of Uvicorn. Which server to use depends on the requirements and performance expectations of the project.

To provide a simple API to the project, we created a Python file named `fast-api.py` in the project folder. The following code allows to create a fast API using uvicorn and FastAPI.

The API layer is constructed as follows:

```
import uvicorn
import pandas as pd
from fastapi import FastAPI
from catboost import CatBoostClassifier

# Path to the model
MODEL_PATH = "../model/cat_model.cbm"

# Function to load the trained model
def load_model():
    model = CatBoostClassifier()
    model.load_model(MODEL_PATH)
    return model

# Function to predict churn probability from data in DataFrame format
def get_churn_probability(data, model):
    # Convert incoming data into a DataFrame
    dataframe = pd.DataFrame.from_dict(data, orient='index').T
    # Make the prediction
    churn_probability = model.predict_proba(dataframe)[0][1]
    return churn_probability

# Load the model
model = load_model()

# Create the FastAPI application
app = FastAPI(title="Churn Prediction API", version="1.0")

@app.get('/')
def index():
    return {'message': 'CHURN Prediction API'}

# Define the API endpoint
@app.post('/predict/')
def predict_churn(data: dict):
    # Get the prediction
```

```
        churn_probability = get_churn_probability(data, model)
        # Return the prediction
        return {'Churn Probability': churn_probability}

    # Run the application
    if __name__ == '__main__':
        uvicorn.run("fast-api:app", host='127.0.0.1', port=5000)
```

Step 1: Setting Up

FastAPI script begins by importing necessary libraries, including FastAPI itself, Pandas for data manipulation, and CatBoost for loading the trained model.

Step 2: Loading the Model

The script defines a function `load_model()` to load the pre-trained CatBoost model from the specified path.

Step 3: Predicting Churn Probability

Another function `get_churn_probability()` is defined to predict churn probability from the input data in DataFrame format. This function converts the incoming data dictionary into a DataFrame and then uses the loaded model to predict the churn probability.

Step 4: Creating FastAPI Application

The FastAPI application is created with the title “Churn Prediction API” and version “1.0”.

Step 5: Defining API Endpoints

- The `/` endpoint is defined to provide a simple message indicating the availability of the CHURN Prediction API.
- The `/predict/` endpoint is defined to accept POST requests with data in JSON format. It calls the `predict_churn()` function, which in turn uses the loaded model to predict churn probability for the provided data.

Step 6: Running the Application

Finally, the application is run using the `uvicorn.run()` function. It specifies

the script file (`fast-api`) and the FastAPI instance (`app`), along with host and port configurations.

You can start the API inside the folder containing the `fast-api.py` file as follows:

```
python fast-api.py
```

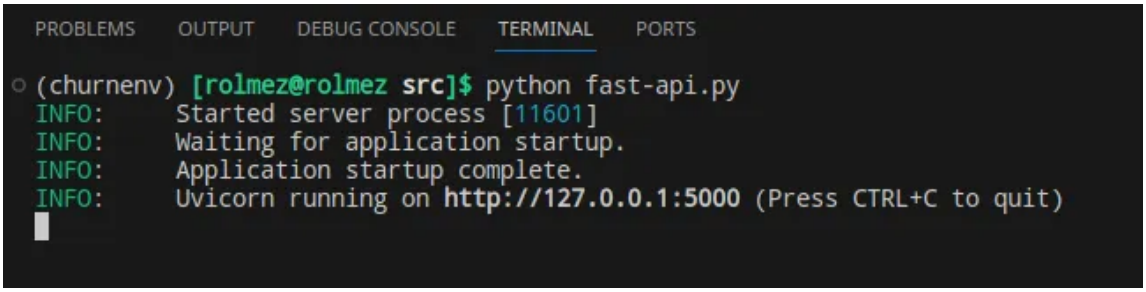
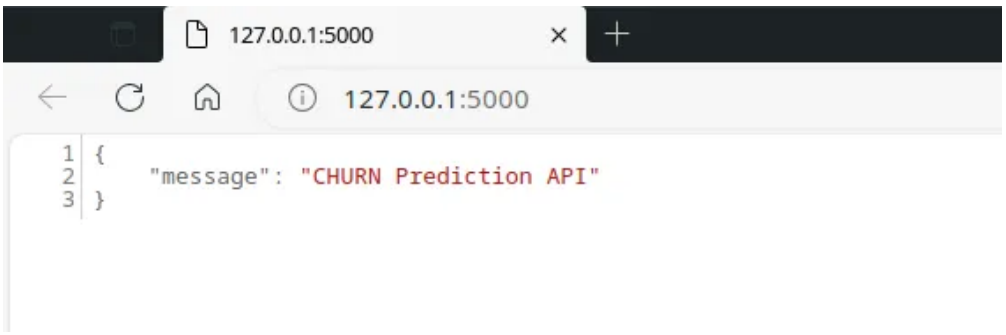


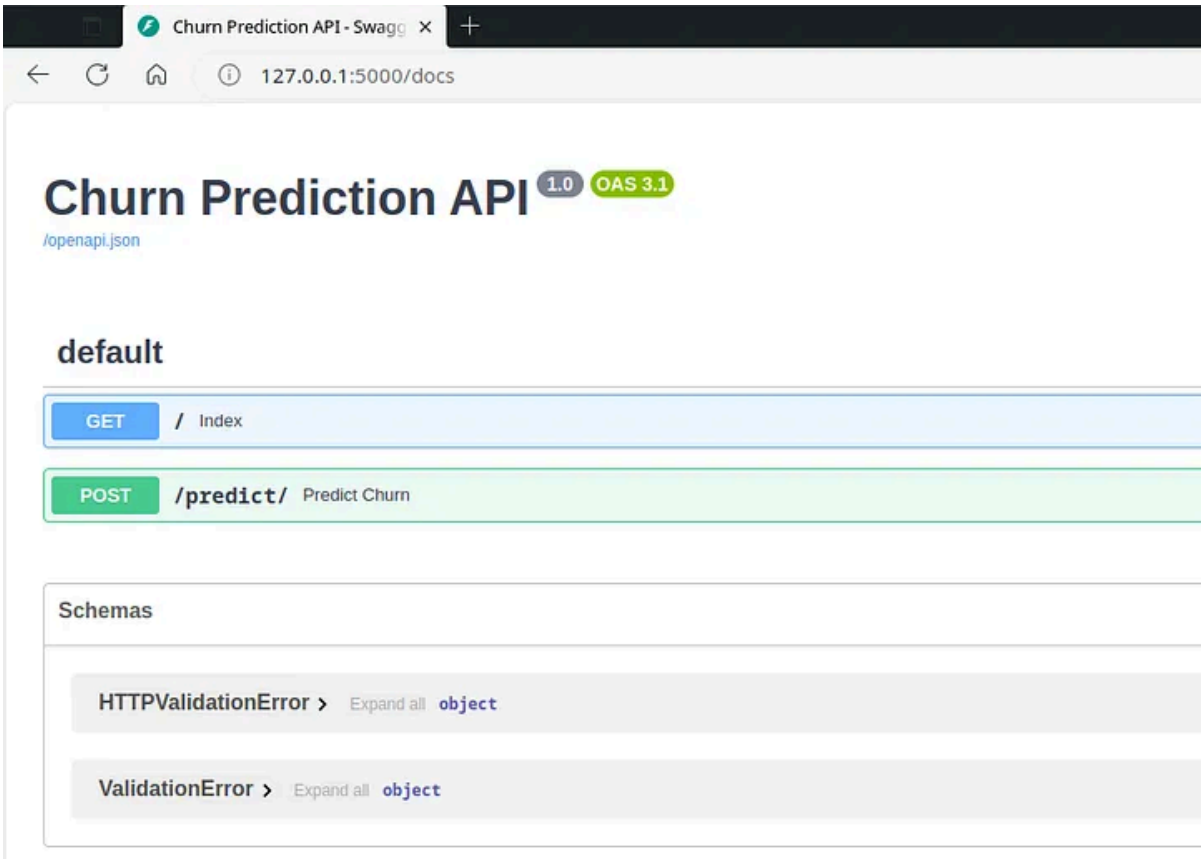
Image by author

We can see the API by going to the Host given to Uvicorn.



localhost

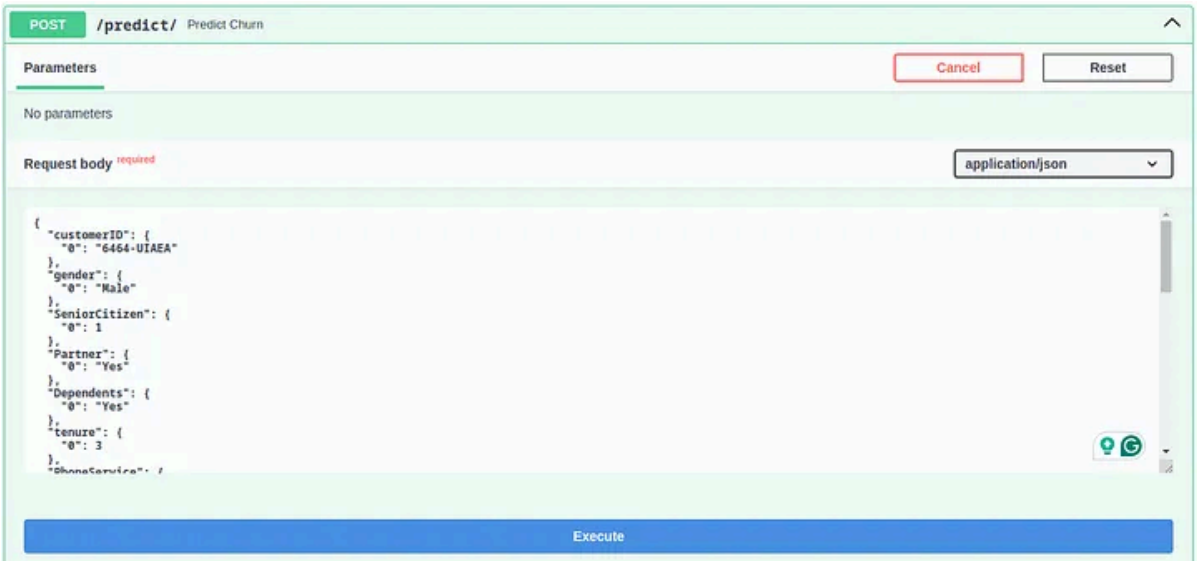
But to access the API we created, we need to add `/docs` or `/redoc` at the end of the extension `(http://127.0.0.1:5000/docs)`.



API in localhost

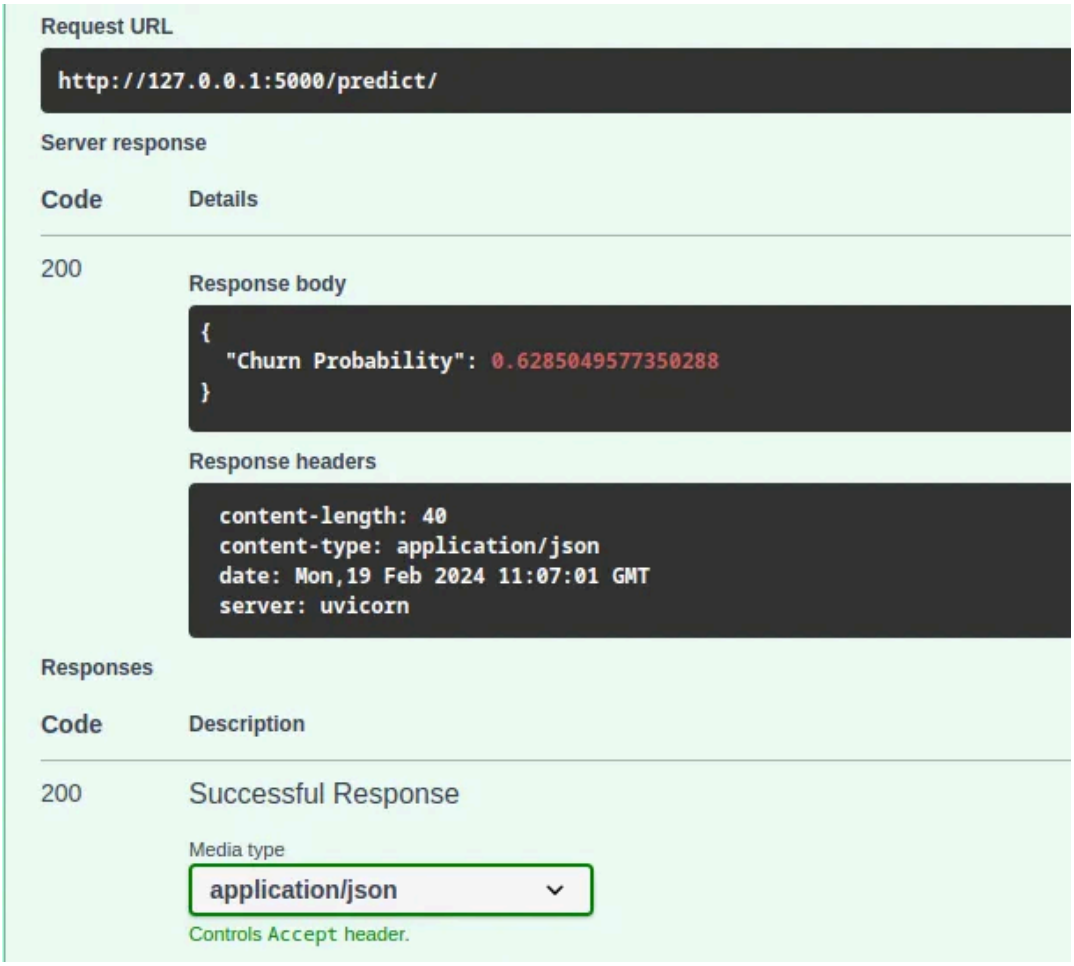
The API created allows us to interactively see the churn probability of the customer we will create.

We can enter the generated customer data by clicking on *Try it out* under *POST -> /predict/*.



Execute

By pressing *Execute*, we evaluate the churn probability of the customer. If the API returns a successful response, we should see a result like below.



API answer

It is seen that the customer's Churn probability is **62.8504%**. Using the same customer data, we could also obtain the Churn probability via Streamlit.

Now that we have successfully built the Streamlit interface and API, it's time to move the model to the product environment. In this process, the necessary work will be done to ensure that the model can be used comfortably on every platform and device.

5. Automation (Docker)

Source

Docker is referred to as a containerization technology. Containers are used to run applications and their dependencies in an independent and isolated

environment. A Docker container contains all the application code, libraries, and configuration files.

From this point on, we need to automate the model file that we know works in our system. So why do we need such a solution? A model that works properly on our system may not work on other systems due to issues such as compatibility. We need to standardize the model and make it work properly in the system components we build. Docker offers a good solution to avoid facing problems such as who has which packages installed on whose system, and who received an error at which stage. Besides all these, it provides portability, dependency management, isolation, and easier deployment.

In the first step, we created `train_model.py` file where the data preprocessing steps and the model are developed. Instead of working with this file, we will create predict.py, which contains the same process, but with an extra prediction section.

The model created in the train_model.py script needs to be saved to be used here. Edit the file path in the script below according to the file path where you saved the model.

`predict.py` file:

```
import pandas as pd
from catboost import CatBoostClassifier

# Load the trained model
MODEL_PATH = "model/cat_model.cbm"
model = CatBoostClassifier()
model.load_model(MODEL_PATH)

def predict_churn(user_input):
    try:
        # Prepare data for prediction
        user_data = pd.DataFrame([user_input])

        # Make prediction
        prediction = model.predict_proba(user_data)[:, 1][0]
```

```
            return {"Churn Probability": float(prediction)}
        except Exception as e:
            return {"error": str(e)}

    if __name__ == "__main__":
        # Ask user for input
        customerID = "6464-UIAEA"
        print("Please enter the following information:")
        gender = input("Gender (Male/Female): ").strip()
        senior_citizen = int(input("Senior Citizen (0/1): ").strip())
        partner = input("Partner (Yes/No): ").strip()
        dependents = input("Dependents (Yes/No): ").strip()
        tenure = int(input("Tenure (months): ").strip())
        phone_service = input("Phone Service (Yes/No): ").strip()
        multiple_lines = input("Multiple Lines (Yes/No): ").strip()
        internet_service = input("Internet Service (DSL/Fiber optic/No): ").strip()
        online_security = input("Online Security (Yes/No): ").strip()
        online_backup = input("Online Backup (Yes/No): ").strip()
        device_protection = input("Device Protection (Yes/No): ").strip()
        tech_support = input("Tech Support (Yes/No): ").strip()
        streaming_tv = input("Streaming TV (Yes/No): ").strip()
        streaming_movies = input("Streaming Movies (Yes/No): ").strip()
        contract = input("Contract (Month-to-month/One year/Two year): ").strip()
        paperless_billing = input("Paperless Billing (Yes/No): ").strip()
        payment_method = input("Payment Method (Electronic check/Mailed check/Bank t
        monthly_charges = float(input("Monthly Charges ($): ").strip())
        total_charges = float(input("Total Charges ($): ").strip())

        new_customer_data = pd.DataFrame({
                    "customerID": [customerID],
                    "gender": [gender],
                    "SeniorCitizen": [senior_citizen],
                    "Partner": [partner],
                    "Dependents": [dependents],
                    "tenure": [tenure],
                    "PhoneService": [phone_service],
                    "MultipleLines": [multiple_lines],
                    "InternetService": [internet_service],
                    "OnlineSecurity": [online_security],
                    "OnlineBackup": [online_backup],
                    "DeviceProtection": [device_protection],
                    "TechSupport": [tech_support],
                    "StreamingTV": [streaming_tv],
                    "StreamingMovies": [streaming_movies],
                    "Contract": [contract],
                    "PaperlessBilling": [paperless_billing],
                    "PaymentMethod": [payment_method],
                    "MonthlyCharges": [monthly_charges],
                    "TotalCharges": [total_charges]
                })

        # Predict churn probability using the model
        churn_probability = model.predict_proba(new_customer_data)[:, 1]

        # Format churn probability
        formatted_churn_probability = "{:.2%}".format(churn_probability.item())
```

```
    print(f"Churn Probability: {formatted_churn_probability}")
```

This Python script allows the user to predict churn using the pre-trained model. It converts the user's input into a DataFrame and gives it to the model to make a prediction. The script ends by printing the result to the screen.

Requirements should be well specified in the Dockerfile created for the deployment of the model. At this stage, we continue by creating a **requirements.txt** file. There are several ways to create this file. Specific libraries used when creating the model can be specified manually. Or if you used virtual environments such as virtualenv, pip freeze, pipreqs or piqar can be used as a good option. However, it should be noted that pip freeze will save all the packages in the virtual environment, not the environment variables used in your work. You can review here for the use of pip freeze.

For this work, I used **pip freeze** and created the `requirements.txt` file in the project folder:

```
altair==5.2.0
annotated-types==0.6.0
anyio==4.2.0
asttokens==2.4.1
attrs==23.2.0
blinker==1.7.0
cachetools==5.3.2
catboost==1.2.2
certifi==2024.2.2
charset-normalizer==3.3.2
click==8.1.7
cloudpickle==3.0.0
contourpy==1.2.0
cycler==0.12.1
decorator==5.1.1
executing==2.0.1
fastapi==0.109.2
fonttools==4.48.1
gitdb==4.0.11
GitPython==3.1.41
graphviz==0.20.1
```

```
h11==0.14.0
idna==3.6
importlib-metadata==7.0.1
ipython
jedi==0.19.1
Jinja2==3.1.3
joblib==1.3.2
jsonschema==4.21.1
jsonschema-specifications==2023.12.1
kiwisolver==1.4.5
llvmlite==0.42.0
markdown-it-py==3.0.0
MarkupSafe==2.1.5
matplotlib==3.8.3
matplotlib-inline==0.1.6
mdurl==0.1.2
numba==0.59.0
numpy==1.26.4
packaging==23.2
pandas==2.2.0
parso==0.8.3
pexpect==4.9.0
pillow==10.2.0
plotly==5.18.0
prompt-toolkit==3.0.43
protobuf==4.25.2
ptyprocess==0.7.0
pure-eval==0.2.2
pyarrow==15.0.0
pydantic==2.6.1
pydantic_core==2.16.2
pydeck==0.8.1b0
Pygments==2.17.2
pyparsing==3.1.1
python-dateutil==2.8.2
pytz==2024.1
referencing==0.33.0
requests==2.31.0
rich==13.7.0
rpds-py==0.18.0
scikit-learn==1.4.0
scipy==1.12.0
shap==0.44.1
six==1.16.0
slicer==0.0.7
smmap==5.0.1
sniffio==1.3.0
stack-data==0.6.3
starlette==0.36.3
streamlit==1.31.1
tenacity==8.2.3
threadpoolctl==3.3.0
toml==0.10.2
toolz==0.12.1
tornado==6.4
tqdm==4.66.2
```

```
traitlets==5.14.1
typing_extensions==4.9.0
tzdata==2024.1
tzlocal==5.2
urllib3==2.2.0
uvicorn==0.27.1
validators==0.22.0
watchdog==4.0.0
wcwidth==0.2.13
zipp==3.17.0
```

Dockerfile is a configuration file that determines the structure of Docker containers. It contains instructions such as how to run the application and which dependencies should be installed. Docker creates a container using this Dockerfile and converts this container into a file called an “image”. This “image” then becomes executable on any system.

After creating the `requirements.txt` file, we can create the Dockerfile:

```
# Use a Python-based image
FROM python:3.9-slim

# Copy requirements.txt and install dependencies
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Specify working directory
WORKDIR /app

# Copy application code
COPY . .

# Run the application
CMD ["python", "src/predict.py"]
```

Step 1: Base Image Selection

We start by specifying the base image that provides the minimum Python environment in which the container will be created (python:3.9-slim).

Step 2: Installing Dependencies

Copy the `requirements.txt` file from the host machine to the container and install the dependencies listed in it using pip.

Step 3: Setting the Working Directory and Copying Application Code

Set the working directory to `/app`. Copy all files and directories from the host machine to the `/app` directory within the container.

Step 4: Running the Application

Execute the `predict.py` script using the Python interpreter as the default command when the container starts.

To continue at this stage of the project, after installing Docker and creating the necessary files, we can go to the directory where the Dockerfile is located and build it as follows.

```
docker build -t churn-prediction-app .
```

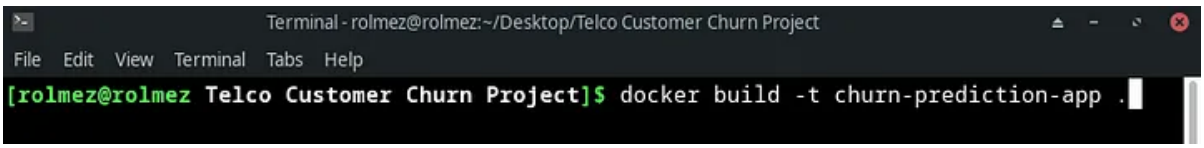


Image by author

```
Step 4/6 : WORKDIR /app
 ---> Running in aa3ca040fe14
Removing intermediate container aa3ca040fe14
 ---> 1242e7f27c2f
Step 5/6 : COPY . .
 ---> 3573cdfd7095
Step 6/6 : CMD ["python", "predict.py"]
 ---> Running in cb84aeec7003
Removing intermediate container cb84aeec7003
 ---> ff730dad35d9
Successfully built ff730dad35d9
Successfully tagged churn-prediction-app:latest
[rolmez@rolmez Telco Customer Churn Project]$
```

Image by author

You can see if the Dockerfile has been successfully built with the command below:

```
[rolmez@rolmez ~]$ docker images
REPOSITORY              TAG         IMAGE ID       CREATED             SIZE
churn-prediction-app    latest      1a4a57d140b6   About an hour ago   1.34GB
python                  3.9-slim    9c14a9ca1040   2 weeks ago         126MB
[rolmez@rolmez ~]$
```

Images list

We have done the build, we can run the image we created in the terminal.

```
docker run -it churn-prediction-app
```

We can see the result of the model by entering the customer information we want to see the churn probability in order.

```
[rolmez@rolmez ~]$ docker run -it churn-prediction-app
Please enter the following information:
Gender (Male/Female): Male
Senior Citizen (0/1): 1
Partner (Yes/No): Yes
Dependents (Yes/No): Yes
Tenure (months): 3
Phone Service (Yes/No): Yes
Multiple Lines (Yes/No): No
Internet Service (DSL/Fiber optic/No): DSL
Online Security (Yes/No): Yes
Online Backup (Yes/No): No
Device Protection (Yes/No): No
Tech Support (Yes/No): No
Streaming TV (Yes/No): Yes
Streaming Movies (Yes/No): Yes
Contract (Month-to-month/One year/Two year): Month-to-month
Paperless Billing (Yes/No): No
Payment Method (Electronic check/Mailed check/Bank transfer (automatic)/Credit card (automatic)): Bank transfer (automatic)
Monthly Charges ($): 35
Total Charges ($): 179
Churn Probability: 62.85%
[rolmez@rolmez ~]$
```

Predict

Through Docker, we can see the Churn probability of the customer we created for prediction in Streamlit and FastAPI. Users who want to predict the model we have created can access this convenience by building and running the Dockerfile.

6. Conclusion

During the development of the project, a machine learning model was developed using CatBoost with customer data provided by the telecom company. To examine the model interactively, an interface was designed using Streamlit. In addition, an API was created with FastAPI to quickly access and integrate the model. Finally, optimization was achieved using Docker to make the project more scalable and deployable.

> *I would like to express my sincere thanks to Hasan Avcı for his support during this journey and to inzva for the opportunities they provided.*

Thank you for reading and please do not hesitate to share your comments and suggestions.

LinkedIn: https://www.linkedin.com/in/ramazanolmez/
Mail: ramazanolmeez@gmail.com

Classification Catboost Fastapi Docker Streamlit

Written by Ramazan Olmez

278 Followers

Follow

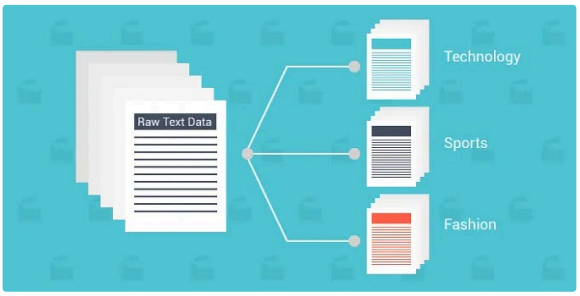
More from Ramazan Olmez

Ramazan Olmez

Handling Missing Values in Data Science / Machine Learning...

Failure to handle missing data correctly can skew the results of machine learning models...

22 min read · Feb 8, 2024



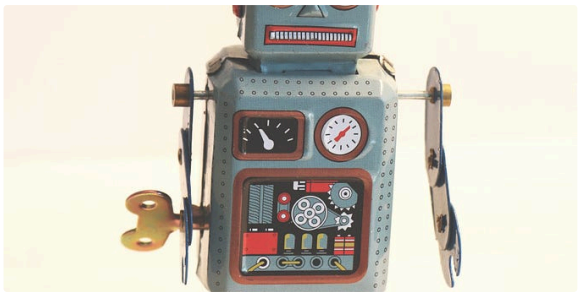
Ramazan Olmez

Metin Sınıflandırma: Başlangıçtan BERT'e

Günün sonunda mailimize gelen bir e-postanın spam olup olmadığını incelemeden...

15 min read · Jan 18, 2024

7



Ramazan Olmez

Büyük Dil Modellerinde Halüsinasyonlar

Bu yazıda, büyük dil modellerinde halüsinasyonun ne olduğunu, türlerini,...

9 min read · Jan 8, 2024

15

2

Ramazan Olmez

T5 modelini Kullanarak Geçmişten Günümüze Metin Özetleme

Bu yazının bir kısmı metin özetlemenin geçmişten günümüze tarihini anlatmakta ve...

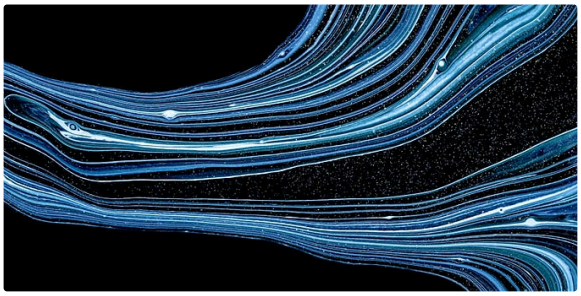
10 min read · Jan 5, 2024

8

See all from Ramazan Olmez

Recommended from Medium

Nathan Rosidi



Oleh Dubetcky

Practical MLOps—MLflow

Top 10 Machine Learning Algorithms for Beginner Data...

Let's explore the machine learning algorithms perfect for beginners in data science. We'll...

19 min read · Apr 25, 2024

794 13

MLflow is a popular open-source tool that plays a key role in implementing MLOps,...

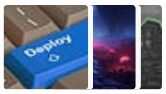
8 min read · Mar 30, 2024

72

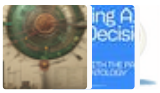
Lists

Coding & Development
11 stories · 645 saves

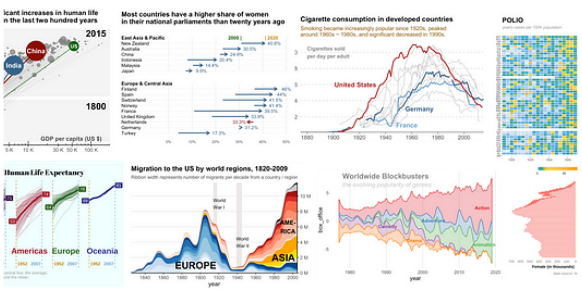
Practical Guides to Machine Learning
10 stories · 1527 saves



Predictive Modeling w/ Python
20 stories · 1268 saves



data science and AI
40 stories · 177 saves



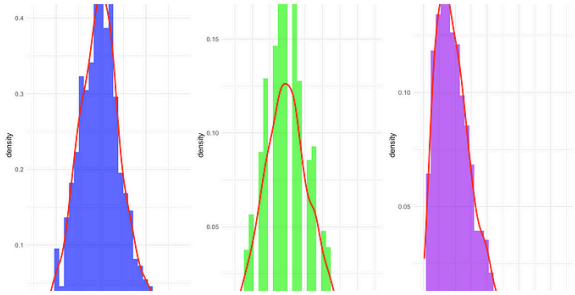
Bo Yuan, Ph.D.

Awesome Strategies to Visualize Change with Time

This article summarizes effective strategies to visualize temporal changes, illustrated with...

5 min read · May 7, 2024

1K 11



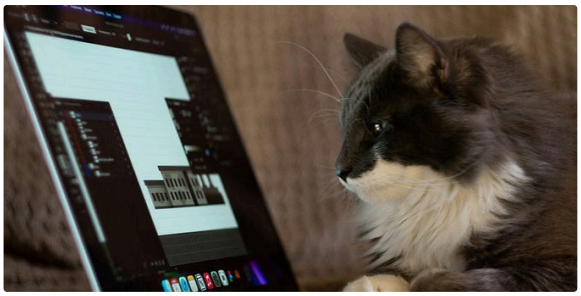
René F. Najera, MPH, DrPH

Not Everything Is Normal: Three Statistical Distributions and Whe...

Be mindful of linear regressions. They're not always the indicated statistical analysis.

· 9 min read · May 27, 2024

394 6



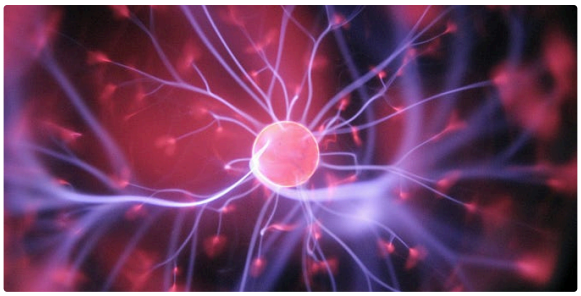
MargaretEfron in Learning Data

I Reviewed 100 LinkedIn Data Analyst Profiles. These were the...

Self-branding is important, and your LinkedIn is integral to your online presence. When yo...

7 min read · May 28, 2024

361 4



Rosaria Silipo in Low Code for Data Science

Is Data Science dead?

In the last six months I have heard this question thousands of time: “Is data science...

6 min read · Mar 11, 2024

2.5K 57

See more recommendations