# !!! THIS IS THE OCDS P8 APP !!!

# to run locally, navigate to C:\Users\celin\DS Projets Python\OCDS-repos-all\OCDS-P8-API

# and launch app by running flask --app app.py run --debug from the command line

# web app available at https://credit-scoring-api-0p1u.onrender.com

from flask import Flask, jsonify

from joblib import load

import pandas as pd

app = Flask(\_\_name\_\_)

# Load client test data

client\_data=pd.read\_csv('test\_data\_final.csv').drop(labels=['Unnamed: 0'], axis=1)

# Load model

model = load('final\_model.joblib')

# Load custom threshold

custom\_threshold = load('optimal\_threshold.joblib')

@app.route('/', methods=['GET'])

def home():

    return '''<h1>Credit Scoring</h1>

    <p>Welcome to the HOME CREDIT Credit Scoring app.</p>

    <p>- use /predict/ID to retrieve client credit application decision</p>

    <p>where ID is the client's unique Home Credit application number (whole number between 1 and 48745)</p>'''

@app.route("/predict/<int:id>", methods=['GET'])

def predict(id):

    # Ensure client id exists in test data

    if (id-1) >= client\_data.shape[0]:

        return "Error: Client id not in application database. Enter a whole number between 1 and 48745.", 404

    if ((id-1) < 0):

        return "Error: Client id not in application database. Enter a whole number between 1 and 48745.", 404

    # Display summary client demographics

    result\_cols = ['INCOME\_TYPE', 'EMPLOYMENT\_SECTOR', 'DISPOSABLE\_INCOME\_per\_capita', 'YEAR\_BIRTH', 'CREDIT\_RATING',

                   'CLIENT\_BAD\_CREDIT\_HISTORY', 'CLIENT\_FRAUD\_FLAG', 'IS\_MALE', 'WHITE\_COLLAR', 'UPPER\_EDUCATION',

                   'IS\_MARRIED', 'LIVES\_INDEPENDENTLY']

    results = []

    row\_data = client\_data.loc[id-1, result\_cols].to\_dict()

    for k, v in row\_data.items():

        if v==0:

            row\_data[k] = 'no'

        if v==1:

            row\_data[k] = 'yes'

    row\_data['AGE'] = row\_data.pop('YEAR\_BIRTH')

    results.append(row\_data)

    # Load client data

    client\_particulars = client\_data.iloc[[id-1]]

    # Predict outcome of client credit application

    # model.predict(client\_particulars) directly returns class 0 (no default) or class 1 (default)

    prediction = model.predict\_proba(client\_particulars)

    proba = prediction[0][1] # prediction[0][0] is proba of client NOT defaulting

    if proba > custom\_threshold:

        proba\_class = 'default'

        decision = "reject loan application"

    else:

        proba\_class = 'no default'

        decision = "grant loan"

    # shap won't work with MLFlow pyfunc model => load pre-calculated Shap values for test data

    # shap\_values\_all = pd.DataFrame(load('shap\_values\_test.joblib'))

    shap\_values\_all = pd.read\_csv('shap\_values\_test\_data.csv')

    shap\_values\_client = shap\_values\_all.iloc[[id-1]]

    abs\_values = shap\_values\_client.abs()

    expected\_value = load('expected\_value.joblib')

    # identify top 5 shap values for client prediction

    top\_5\_indices = abs\_values.iloc[0].nlargest(5).index.values.tolist()

    top\_5\_columns = shap\_values\_client[top\_5\_indices].values.tolist()

    top\_5\_dict = {}

    for top\_k, top\_v in zip(top\_5\_indices, top\_5\_columns[0]):

        top\_5\_dict[top\_k] = top\_v

    sorted\_top\_5\_dict = sorted(top\_5\_dict.items(), key=lambda top\_5\_dict: top\_5\_dict[1], reverse=True)

    # Return bank decision on client credit application

    return jsonify({

        'Client Home Credit application number:': id,

        'Client summary information' : results,

        'Client default probability': proba,

        'Class': proba\_class,

        'Decision': decision,

        'Key Decision Factors': sorted\_top\_5\_dict,

        'Expected Shap Value' : expected\_value,

        'Shap values client' : shap\_values\_client.to\_json(orient='records'),

        'Client data' : client\_particulars.to\_json(orient='records'),

        'Threshold': custom\_threshold

    })

if \_\_name\_\_ == "\_\_main\_\_":

    app.run()

import streamlit as st

import requests

import json

import shap

import matplotlib.pyplot as plt

import numpy as np

import plotly.graph\_objects as go

import pandas as pd

import plotly.express as px

accessibility\_mode = st.session\_state.get("accessibility\_mode", False)

if accessibility\_mode:

    # Inject high contrast CSS, increase font sizes, etc.

    st.markdown(

        """

        <style>

        body {

            background-color: black !important;

            color: white !important;

            font-size: 20px !important;

        }

        /\* Customize buttons, inputs, etc. for high contrast \*/

        button, input {

            background-color: yellow !important;

            color: black !important;

            font-weight: bold !important;

        }

        </style>

        """,

        unsafe\_allow\_html=True,

    )

else:

    # Default or light theme CSS or no override

    pass

# For graphs, use color-blind-friendly palettes conditionally

import plotly.express as px

colorblind\_palette = px.colors.qualitative.Safe  # colorblind-friendly palette

default\_palette = px.colors.qualitative.Plotly

palette = colorblind\_palette if accessibility\_mode else default\_palette

# Get selected client application id

selected\_value = st.session\_state.get("selected\_value", None)

if selected\_value is None:

    st.warning("Please select a client credit application reference on the Home page first.")

else:

    st.write(f"Using selected client application: {selected\_value}")

def set\_bg\_color(color):

    st.markdown(

        f"""

        <style>

        .stApp {{

            background-color: {color};

        }}

        </style>

        """,

        unsafe\_allow\_html=True

    )

# Example usage: set the background color to lightblue

set\_bg\_color('#fbf0ef') # light pink

# set\_bg\_color('#2fbeb5') # light green

# set\_bg\_color('#f1bd5f') # sand

st.image("bandeau.png")

# Send a get request to the API using the selected client credit application reference

app\_response = requests.get(f"https://credit-scoring-api-0p1u.onrender.com/predict/{selected\_value}") # web API

# app\_response = requests.get(f"http://127.0.0.1:5000/predict/{selected\_value}") # local API

# Import elements from API response separately for graphs

app\_data = app\_response.json()

shap\_values\_client\_json = app\_data["Shap values client"]

shap\_values\_client\_dict = json.loads(shap\_values\_client\_json)[0]

shap\_values\_array = np.array(list(shap\_values\_client\_dict.values()))

feature\_names = list(shap\_values\_client\_dict.keys())

base\_value = app\_data.get("Expected Shap Value")

threshold\_value =  app\_data.get("Threshold")

client\_data\_json = app\_data["Client data"]

client\_data\_dict = json.loads(client\_data\_json)[0]

client\_data\_array = np.array(list(client\_data\_dict.values()))

# Display user's chosen basic demographics from API response

if app\_response.status\_code == 200:

    client\_info = app\_data['Client summary information'][0]

    # st.markdown("<h4 style='font-size: 28px;'>Select client demographics to display:</h4>", unsafe\_allow\_html=True)

    st.write("## ✌️ Step 2 - Select client demographics to display:")

    selected\_demographics = st.multiselect("", # Leave text empty to avoid duplicate with above

                                            options=list(client\_info.keys()),

                                            default=list(client\_info.keys())  # Show all by default

                                            )

    st.write("### You selected client demographics:")

    for demo in selected\_demographics:

        st.markdown(f"<span style='font-size:28px;'> - \*\*{demo}:\*\* {client\_info[demo]}</span>",

                  unsafe\_allow\_html=True)

import streamlit as st

import requests

import json

import shap

import matplotlib.pyplot as plt

import numpy as np

import plotly.graph\_objects as go

import pandas as pd

import plotly.express as px

accessibility\_mode = st.session\_state.get("accessibility\_mode", False)

if accessibility\_mode:

    # Inject high contrast CSS, increase font sizes, etc.

    st.markdown(

        """

        <style>

        body {

            background-color: black !important;

            color: white !important;

            font-size: 20px !important;

        }

        /\* Customize buttons, inputs, etc. for high contrast \*/

        button, input {

            background-color: yellow !important;

            color: black !important;

            font-weight: bold !important;

        }

        </style>

        """,

        unsafe\_allow\_html=True,

    )

else:

    # Default or light theme CSS or no override

    pass

# For graphs, use color-blind-friendly palettes conditionally

import plotly.express as px

colorblind\_palette = px.colors.qualitative.Safe  # colorblind-friendly palette

default\_palette = px.colors.qualitative.Plotly

palette = colorblind\_palette if accessibility\_mode else default\_palette

# Get selected client application id

selected\_value = st.session\_state.get("selected\_value", None)

if selected\_value is None:

    st.warning("Please select a client credit application reference on the Home page first.")

else:

    st.write(f"Using selected client application: {selected\_value}")

# Check if API data is loaded

    if "app\_data" not in st.session\_state:

        st.warning("API data not loaded yet. Please go back to Home page and select a client.")

    else:

        # Access stored data

        shap\_values\_client\_dict = st.session\_state.shap\_values\_client\_dict

        shap\_values\_array = st.session\_state.shap\_values\_array

        feature\_names = st.session\_state.feature\_names

        base\_value = st.session\_state.base\_value

        threshold\_value = st.session\_state.threshold\_value

        client\_data\_dict = st.session\_state.client\_data\_dict

        client\_data\_array = st.session\_state.client\_data\_array

        app\_data = st.session\_state.app\_data

        app\_response = st.session\_state.app\_response

        # st.write(f"Using client application {selected\_value}")

        # Example: show feature names and SHAP values

        # st.write("Feature names:", feature\_names)

        # st.write("SHAP values:", shap\_values\_array)

        st.write("# 💫 Client credit scoring model results:💫")

        # st.write(f"Client default probability: {app\_data['Client default probability'] \* 100:.2f}%")

        # st.write("Class :", app\_data['Class'])

        if app\_data['Class'] == 'no default':

            st.markdown(f"<span style='font-size:28px;'> \*\*Predicted behavior\*\* : client will repay loan 👍</span>",

                        unsafe\_allow\_html=True)

        else:

            st.markdown(f"<span style='font-size:28px;'> \*\*Predicted behavior\*\* : client will default on loan 👎</span>",

                        unsafe\_allow\_html=True)

        # st.write("Decision :", app\_data['Decision'])

        if app\_data['Decision'] == "grant loan":

            st.write(f"<span style='font-size: 28px;'> \*\*Decision\*\* : {app\_data['Decision']} 🥂🎈🎉</span>",

                        unsafe\_allow\_html=True)

        else:

            st.write(f"<span style='font-size: 28px;'> \*\*Decision\*\* : {app\_data['Decision']} ⛔</span>",

                        unsafe\_allow\_html=True)

        # Display client default probability on a gauge with color change above custom threshold

        bar\_color = 'red' if app\_data['Client default probability'] > threshold\_value else 'forestgreen'

        fig = go.Figure(go.Indicator(

        mode = "gauge+number+delta",

        value = app\_data['Client default probability'] \* 100,

        domain = {'x': [0, 1], 'y': [0, 1]},

        delta = {'reference': 50, 'increasing': {'color': "red"}, 'decreasing': {'color': "forestgreen"}},

        title = {'text': "<b>Client default probability %</b>",  'font': {'size': 28, 'color': 'black'}},

        gauge = {'axis': {'range': [None, 100]},

                'bar': {'color': bar\_color},

                'steps' : [{'range': [0, 50], 'color': "palegreen"},

                           {'range': [50, 100], 'color': "lightcoral"}],

                'threshold' : {'line': {'color': "red", 'width': 4}, 'thickness': 0.502, 'value': 50}

                }))

        st.plotly\_chart(fig)

        # Create SHAP waterfall plot

        if shap\_values\_array is not None:

            shap\_explanation = shap.Explanation(values=shap\_values\_array,

                                                base\_values=base\_value,

                                                feature\_names=feature\_names)

            fig, ax = plt.subplots(figsize=(10,6))

            st.title(f"Key decision factors for client {selected\_value}")

            shap.plots.waterfall(shap\_explanation, max\_display=6) # Show the top 5 features and group the remaining features

            st.pyplot(fig)

        else:

            st.error(f"Failed to fetch Shap values for client application. API status code : {app\_response.status\_code}")

import streamlit as st

import requests

import json

import shap

import matplotlib.pyplot as plt

import numpy as np

import plotly.graph\_objects as go

import pandas as pd

import plotly.express as px

accessibility\_mode = st.session\_state.get("accessibility\_mode", False)

if accessibility\_mode:

    # Inject high contrast CSS, increase font sizes, etc.

    st.markdown(

        """

        <style>

        body {

            background-color: black !important;

            color: white !important;

            font-size: 20px !important;

        }

        /\* Customize buttons, inputs, etc. for high contrast \*/

        button, input {

            background-color: yellow !important;

            color: black !important;

            font-weight: bold !important;

        }

        </style>

        """,

        unsafe\_allow\_html=True,

    )

else:

    # Default or light theme CSS or no override

    pass

# For graphs, use color-blind-friendly palettes conditionally

import plotly.express as px

colorblind\_palette = px.colors.qualitative.Safe  # colorblind-friendly palette

default\_palette = px.colors.qualitative.Plotly

palette = colorblind\_palette if accessibility\_mode else default\_palette

# Get selected client application id

selected\_value = st.session\_state.get("selected\_value", None)

if selected\_value is None:

    st.warning("Please select a client credit application reference on the Home page first.")

else:

    st.write(f"Using selected client application: {selected\_value}")

# Check if API data is loaded

    if "app\_data" not in st.session\_state:

        st.warning("API data not loaded yet. Please go back to Home page and select a client.")

    else:

        # Access stored data

        shap\_values\_client\_dict = st.session\_state.shap\_values\_client\_dict

        shap\_values\_array = st.session\_state.shap\_values\_array

        feature\_names = st.session\_state.feature\_names

        base\_value = st.session\_state.base\_value

        threshold\_value = st.session\_state.threshold\_value

        client\_data\_dict = st.session\_state.client\_data\_dict

        client\_data\_array = st.session\_state.client\_data\_array

        app\_data = st.session\_state.app\_data

        app\_response = st.session\_state.app\_response

        # st.write(f"Using client application {selected\_value}")

        # Example: show feature names and SHAP values

        # st.write("Feature names:", feature\_names)

        # st.write("SHAP values:", shap\_values\_array)

client\_data=pd.read\_csv('test\_data\_final.csv').drop(labels=['Unnamed: 0'], axis=1)

column\_names = client\_data.columns.tolist()

cat\_cols = ['NAME\_CONTRACT\_TYPE', 'INCOME\_TYPE', 'EMPLOYMENT\_SECTOR']

bool\_cols = ['IS\_MALE', 'FLAG\_OWN\_CAR', 'FLAG\_OWN\_REALTY', 'FLAG\_EMAIL', 'WHITE\_COLLAR', 'UPPER\_EDUCATION',

             'IS\_MARRIED', 'LIVES\_INDEPENDENTLY', 'PHONE\_PROVIDED', 'PHONE\_REACHABLE', 'ADDRESS\_MISMATCH']

int\_cols = ['DAYS\_ID\_PUBLISH', 'REGION\_RATING\_CLIENT\_W\_CITY', 'OBS\_60\_CNT\_SOCIAL\_CIRCLE', 'DEF\_60\_CNT\_SOCIAL\_CIRCLE',

            'DAYS\_LAST\_PHONE\_CHANGE', 'REQUESTS\_ABOUT\_CLIENT\_1Y', 'pcb\_CNT\_INSTALMENT\_FUTURE\_max',

            'pa\_NFLAG\_INSURED\_ON\_APPROVAL\_sum', 'pa\_CREDIT\_SECURITY\_sum', 'TOTAL\_APPROVED\_CREDITS', 'TOTAL\_ACTIVE\_CAR\_LOANS',

            'TOTAL\_ACTIVE\_OTHER\_LOANS', 'TOTAL\_ACTIVE\_CONSUMER\_LOANS', 'TOTAL\_ACTIVE\_MICRO\_LOANS', 'TOTAL\_ACTIVE\_MORTGAGES',

            'TOTAL\_ACTIVE\_CREDIT\_CARDS', 'DEBT\_RENEGOCIATIONS', 'CLIENT\_BAD\_CREDIT\_HISTORY', 'CLIENT\_FRAUD\_FLAG',

            'CLIENT\_WITHDRAWN\_APPLICATIONS', 'YEAR\_BIRTH']

float\_cols = ['AMT\_CREDIT', 'DISPOSABLE\_INCOME', 'DISPOSABLE\_INCOME\_per\_capita', 'YEARS\_EMPLOYED\_AS\_ADULT\_%',

              'CREDIT\_RATING', 'NB\_APPLICATION\_DOCUMENTS\_%', 'b\_AMT\_CREDIT\_MAX\_OVERDUE\_max', 'b\_AMT\_CREDIT\_SUM\_OVERDUE\_sum',

              'ccb\_AMT\_CREDIT\_LIMIT\_ACTUAL\_mean', 'ccb\_CNT\_DRAWINGS\_TOTAL\_mean', 'ccb\_CARD\_OVERDRAWN\_%\_mean',

              'ip\_EARLY\_PMT\_mean', 'ip\_AMT\_OVERPAID\_%\_mean', 'ip\_AMT\_UNDERPAID\_%\_mean', 'pa\_AMT\_APPLICATION\_mean',

              'pa\_RATE\_DOWN\_PAYMENT\_mean', 'pa\_REMAINING\_CREDIT\_DURATION\_Y\_mean', 'TOTAL\_PAYMENT\_DELAYS\_DAYS',

              'DOWN\_PAYMENT\_CURR\_%', 'DEBT\_RATE\_INC\_CURR\_%', 'b\_DAYS\_CREDIT\_CARD\_max']

st.write("## 👌 Step 3 - Choose field for client univariate analysis display:")

    # Create boxplot

st.write("### Choose a column for the box plot:")

box\_column = st.selectbox("", float\_cols)

fig2 = go.Figure()

fig2.add\_trace(go.Violin(y=client\_data[box\_column],

                  name='Clients Distribution',

                  marker=dict(color='#F1BD5F'),

                  opacity=0.8))

if selected\_value in client\_data.index:

    selected\_client\_value = client\_data.loc[selected\_value, box\_column]

    # add a second axis that overlays the existing one

    fig2.layout.xaxis2 = go.layout.XAxis(overlaying='x', range=[0, 2], showticklabels=False)

    fig2.add\_scatter(x = [0, 2],

                    # y = [20, 20],

                    y=[selected\_client\_value,selected\_client\_value], # see https://stackoverflow.com/questions/58679441/python-plotly-add-horizontal-line-to-box-plot

                    mode='lines',

                    xaxis='x2',

                    showlegend=True,

                    name = f'Client ID: {selected\_value}',

                    line=dict(dash='dot', color = "#f05876", width = 2))

    average\_value = client\_data[box\_column].mean()

    # add a third axis that overlays the existing ones

    fig2.layout.xaxis3 = go.layout.XAxis(overlaying='x', range=[0, 2], showticklabels=False)

    fig2.add\_scatter(x = [0, 2],

                # y = [20, 20],

                y=[average\_value,average\_value],

                mode='lines',

                xaxis='x3',

                showlegend=True,

                name = "Average - All clients",

                line=dict(dash='solid', color = "#242164", width = 2))

else:

    st.warning(f"Client ID {selected\_value} not found in the dataset for the box plot.")

    # fig2.update\_layout(title=f"Box plot of {box\_column} with Selected Client",

    #                    yaxis\_title=box\_column)

st.plotly\_chart(fig2)

import streamlit as st

import requests

import json

import shap

import matplotlib.pyplot as plt

import numpy as np

import plotly.graph\_objects as go

import pandas as pd

import plotly.express as px

accessibility\_mode = st.session\_state.get("accessibility\_mode", False)

if accessibility\_mode:

    # Inject high contrast CSS, increase font sizes, etc.

    st.markdown(

        """

        <style>

        body {

            background-color: black !important;

            color: white !important;

            font-size: 20px !important;

        }

        /\* Customize buttons, inputs, etc. for high contrast \*/

        button, input {

            background-color: yellow !important;

            color: black !important;

            font-weight: bold !important;

        }

        </style>

        """,

        unsafe\_allow\_html=True,

    )

else:

    # Default or light theme CSS or no override

    pass

# For graphs, use color-blind-friendly palettes conditionally

import plotly.express as px

colorblind\_palette = px.colors.qualitative.Safe  # colorblind-friendly palette

default\_palette = px.colors.qualitative.Plotly

palette = colorblind\_palette if accessibility\_mode else default\_palette

# Get selected client application id

selected\_value = st.session\_state.get("selected\_value", None)

if selected\_value is None:

    st.warning("Please select a client credit application reference on the Home page first.")

else:

    st.write(f"Using selected client application: {selected\_value}")

# Check if API data is loaded

    if "app\_data" not in st.session\_state:

        st.warning("API data not loaded yet. Please go back to Home page and select a client.")

    else:

       # Access stored data

        shap\_values\_client\_dict = st.session\_state.shap\_values\_client\_dict

        shap\_values\_array = st.session\_state.shap\_values\_array

        feature\_names = st.session\_state.feature\_names

        base\_value = st.session\_state.base\_value

        threshold\_value = st.session\_state.threshold\_value

        client\_data\_dict = st.session\_state.client\_data\_dict

        client\_data\_array = st.session\_state.client\_data\_array

        app\_data = st.session\_state.app\_data

        app\_response = st.session\_state.app\_response

        # st.write(f"Using client application {selected\_value}")

        # Example: show feature names and SHAP values

        # st.write("Feature names:", feature\_names)

        # st.write("SHAP values:", shap\_values\_array)

st.write("## 🖖 Step 4 - Choose fields for client bivariate analysis display:")

# Create scatter plot

client\_data=pd.read\_csv('test\_data\_final.csv').drop(labels=['Unnamed: 0'], axis=1)

column\_names = client\_data.columns.tolist()

cat\_cols = ['NAME\_CONTRACT\_TYPE', 'INCOME\_TYPE', 'EMPLOYMENT\_SECTOR']

bool\_cols = ['IS\_MALE', 'FLAG\_OWN\_CAR', 'FLAG\_OWN\_REALTY', 'FLAG\_EMAIL', 'WHITE\_COLLAR', 'UPPER\_EDUCATION',

             'IS\_MARRIED', 'LIVES\_INDEPENDENTLY', 'PHONE\_PROVIDED', 'PHONE\_REACHABLE', 'ADDRESS\_MISMATCH']

int\_cols = ['DAYS\_ID\_PUBLISH', 'REGION\_RATING\_CLIENT\_W\_CITY', 'OBS\_60\_CNT\_SOCIAL\_CIRCLE', 'DEF\_60\_CNT\_SOCIAL\_CIRCLE',

            'DAYS\_LAST\_PHONE\_CHANGE', 'REQUESTS\_ABOUT\_CLIENT\_1Y', 'pcb\_CNT\_INSTALMENT\_FUTURE\_max',

            'pa\_NFLAG\_INSURED\_ON\_APPROVAL\_sum', 'pa\_CREDIT\_SECURITY\_sum', 'TOTAL\_APPROVED\_CREDITS', 'TOTAL\_ACTIVE\_CAR\_LOANS',

            'TOTAL\_ACTIVE\_OTHER\_LOANS', 'TOTAL\_ACTIVE\_CONSUMER\_LOANS', 'TOTAL\_ACTIVE\_MICRO\_LOANS', 'TOTAL\_ACTIVE\_MORTGAGES',

            'TOTAL\_ACTIVE\_CREDIT\_CARDS', 'DEBT\_RENEGOCIATIONS', 'CLIENT\_BAD\_CREDIT\_HISTORY', 'CLIENT\_FRAUD\_FLAG',

            'CLIENT\_WITHDRAWN\_APPLICATIONS', 'YEAR\_BIRTH']

float\_cols = ['AMT\_CREDIT', 'DISPOSABLE\_INCOME', 'DISPOSABLE\_INCOME\_per\_capita', 'YEARS\_EMPLOYED\_AS\_ADULT\_%',

              'CREDIT\_RATING', 'NB\_APPLICATION\_DOCUMENTS\_%', 'b\_AMT\_CREDIT\_MAX\_OVERDUE\_max', 'b\_AMT\_CREDIT\_SUM\_OVERDUE\_sum',

              'ccb\_AMT\_CREDIT\_LIMIT\_ACTUAL\_mean', 'ccb\_CNT\_DRAWINGS\_TOTAL\_mean', 'ccb\_CARD\_OVERDRAWN\_%\_mean',

              'ip\_EARLY\_PMT\_mean', 'ip\_AMT\_OVERPAID\_%\_mean', 'ip\_AMT\_UNDERPAID\_%\_mean', 'pa\_AMT\_APPLICATION\_mean',

              'pa\_RATE\_DOWN\_PAYMENT\_mean', 'pa\_REMAINING\_CREDIT\_DURATION\_Y\_mean', 'TOTAL\_PAYMENT\_DELAYS\_DAYS',

              'DOWN\_PAYMENT\_CURR\_%', 'DEBT\_RATE\_INC\_CURR\_%', 'b\_DAYS\_CREDIT\_CARD\_max']

# Column selection

st.write("### Choose the horizontal axis for the scatter plot:")

x\_column = st.selectbox(" ", float\_cols)

st.write("### Choose the vertical axis for the scatter plot:")

y\_column = st.selectbox("  ", float\_cols)

# facet\_x = st.selectbox("Choose horizontal segmentation category:", cat\_cols)

# facet\_x = 'IS\_MALE'

# facet\_y = st.selectbox("Choose vertical segmentation category:", bool\_cols)

# Fetch client data

# client\_data\_response = requests.get(f"http://127.0.0.1:5000/client\_data/{selected\_value}")

if app\_response.status\_code == 200:

    # client\_data['IS\_MALE'] = client\_data['IS\_MALE'].astype(str)

    # color\_map = {'0': 'hotpink', '1': 'cornflowerblue'}

    # client\_data['IS\_MALE\_COLOR'] = client\_data['IS\_MALE'].map(color\_map)

    fig = px.scatter(client\_data, x=x\_column, y=y\_column,

                     color\_discrete\_sequence=['#F1BD5F'],

                     # color='IS\_MALE',

                     # color\_discrete\_map = {'0':'hotpink', '1':'cornflowerblue'},

                     trendline="ols",

                     opacity=0.01,

                     # facet\_col=facet\_y,

                     # facet\_row=facet\_x,

                     # labels={x\_column: x\_column, y\_column: y\_column}

                     )

    # Highlight trendline

    # fig.data[1].line.color = "fuchsia"

    # Highlight selected client on scatterplot

    if selected\_value in client\_data.index:

        selected\_client = client\_data.loc[[selected\_value]]

        fig.add\_trace(go.Scatter(x=selected\_client[x\_column],

                                 y=selected\_client[y\_column],

                                 mode='markers',

                                 marker=dict(color='#242164', symbol='star', size=15),

                                 name=f'Client ID: {selected\_value}'

                                 ))

    else:

        st.warning(f"Client ID {selected\_value} not found in the dataset for the scatter plot.")

    st.plotly\_chart(fig)

    st.write("")

    st.write("")