

ANA RAQUEL

POSTECH

DATA ANALYTICS

DEPLOY DE APLICAÇÕES

AULA 05

SUMÁRIO

O QUE VEM POR AÍ?	3
HANDS ON	4
SAIBA MAIS.....	5
O QUE VOCÊ VIU NESTA AULA?	26
REFERÊNCIAS.....	27

EMSE

O QUE VEM POR AÍ?

Chegou o momento de colocar o modelo em produção! Nessa aula, você aprenderá a criar uma aplicação empregando o Streamlit. Você já o utilizou?

Essa ferramenta é um framework de código aberto elaborado especialmente para nós, cientistas e analistas de dados, criarmos aplicações para instanciar nossos modelos sem nenhuma dor de cabeça, pois ele não necessita de conhecimentos de front-end ou de deploy de aplicações. Vamos aprender como conceber uma aplicação utilizando essa ferramenta incrível?

EMANDA

HANDS ON

Veremos, na prática, como fazer o deploy do modelo no **Streamlit**! Nas videoaulas a seguir, utilizaremos o editor de texto **VSCode** (Visual Studio Code) como IDE (Integrated Development Environment) para construir nossa aplicação de aprovação de crédito.

Você já possui o VSCode instalado na sua máquina? Caso não, pode realizar o download no [site do VSCode](#). Para realizar os downloads necessários dos arquivos utilizados nessa aula, acesse [nosso GitHub](#).

SAIBA MAIS

Realizar o deploy do modelo de Machine Learning significa deixá-lo **disponível para ser utilizado em produção** por meio de um serviço. Podemos listar algumas maneiras para fazer o deploy, tal como:

- Localmente;
- Nuvem;
- Contêineres;
- Servidores Web;
- Dispositivos.

Nessa aula, você terá um exemplo de deploy de modelos de Machine Learning dentro de uma aplicação web no qual utilizaremos o **Streamlit**. A ideia de disponibilizar o modelo na aplicação é fazer com que usuários finais consigam interagir com o modelo por meio de uma interface amigável. Que tal dar uma olhadinha na documentação oficial da ferramenta? Acesse o [site do Streamlit](#) para mais informações.

Vamos criar nossa aplicação de aprovação de crédito?

Preparando o ambiente

Criando uma pasta com os arquivos do projeto

Precisaremos criar uma pasta no computador para armazenar alguns arquivos importantes para rodar o projeto:

- Nessa pasta, precisamos ter o **modelo** que criamos nas últimas aulas, que salvamos como “**xgb.joblib**”;
- O arquivo “**.gitignore**” (arquivo para especificar quais arquivos ou pastas devem ser ignorados pelo Git ao rastrear e versionar alterações);
- Vamos utilizar o dataframe “**df_clean.csv**” (Dataframe que normalizamos e tratamos nas últimas aulas para utilizar no modelo da aplicação);
- O arquivo “**requirements.txt**” (arquivo que contém todas as bibliotecas e versões que vamos utilizar na aplicação);

- O arquivo “**utils.py**”, com toda a pipeline que criamos nas aulas anteriores.

Vamos precisar criar a pasta e colocar um nome, como por exemplo “**credit-scoring**”, para armazenar os arquivos citados.

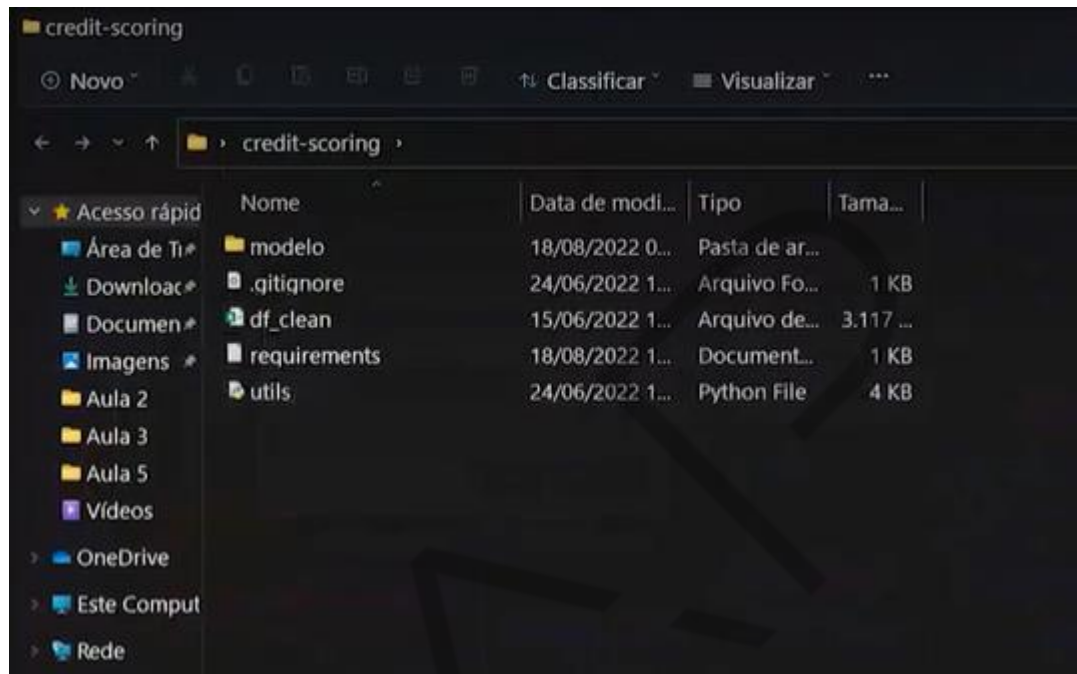


Figura 1 - Pasta com arquivos necessários do projeto
Fonte: Elaborado pela autora (2023)

Criando o ambiente no VSCode

Como próximo passo, vamos para o VSCode! Em seguida, importaremos a pasta que acabamos de criar no computador para dentro do editor de códigos já citado. Para importar a pasta, acesse “**File**”, no canto esquerdo superior do VSCode, e em seguida selecione “**Open Folder**”:

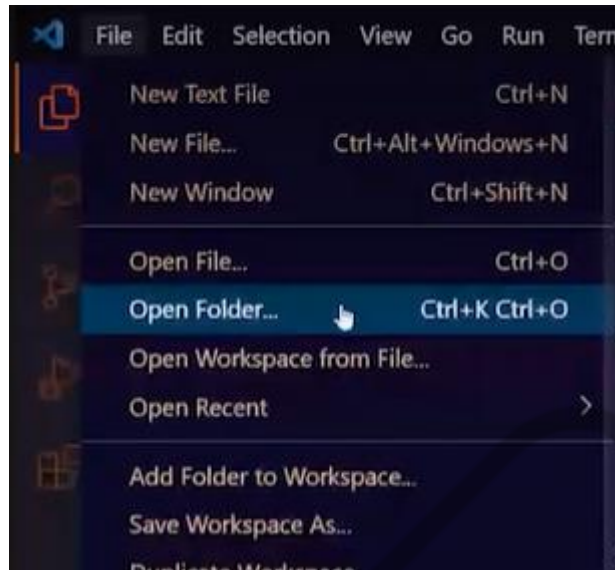


Figura 2 - Abrindo a pasta no VSCode
Fonte: Elaborado pela autora (2023)

Selecione a pasta que acabamos de criar e prontinho, temos a pasta armazenada dentro do VSCode:

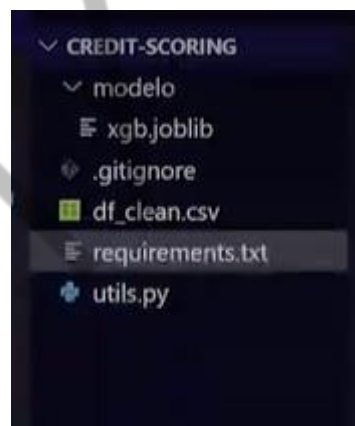


Figura 3 - Upload da pasta realizado no VSCode
Fonte: Elaborado pela autora (2023)

No próximo passo criaremos o **ambiente virtual** da nossa aplicação, que vai rodar todos os processos de forma isolada do nosso computador. Mas o que é isso? O ambiente virtual permite que seu código sempre seja executado independentemente de versões ou atualizações do seu computador. Para criá-lo, abriremos um novo **“Terminal”** (o terminal é o local em que realizaremos todos os códigos). No canto esquerdo superior, selecione **“Terminal”** e então, em seguida, **“New terminal”**.

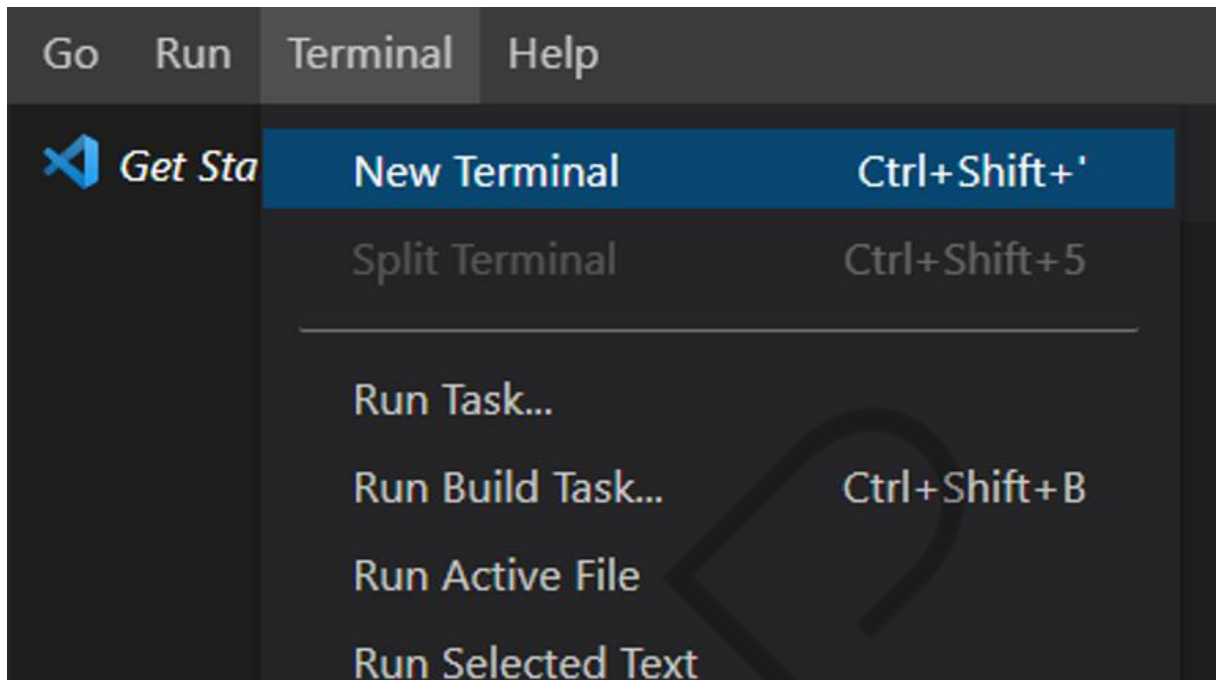


Figura 4 - Abrindo um novo terminal no VSCode
Fonte: Elaborado pela autora (2023)

Uma nova janela será aberta na tela:

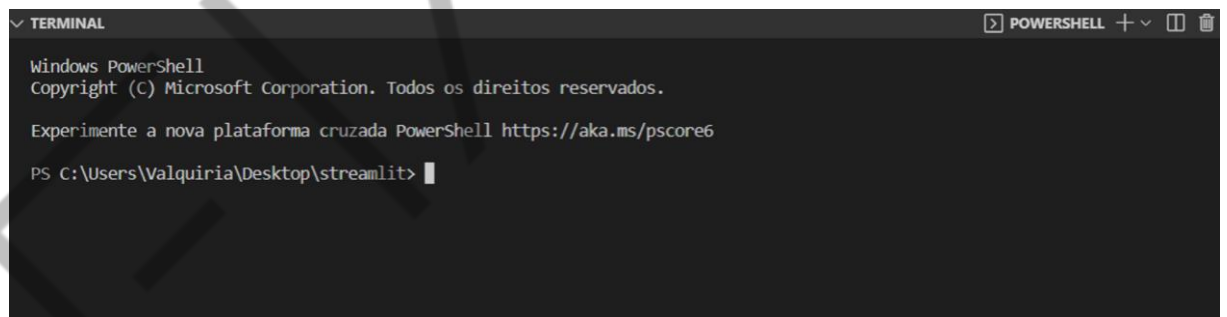


Figura 5 -Terminal no VSCode
Fonte: Elaborado pela autora (2023)

No próximo passo, realizaremos os códigos necessários para criar o ambiente virtual. Dentro do terminal, digite o código a seguir:

```
python -m venv venv
```

A palavra “**venv**” quer dizer virtual environment. Após o comando “venv” você pode colocar o nome que desejar para a aplicação. Nesse caso, estamos deixando “venv” mesmo. Após digitar o código e pressionar o botão “**Enter**”, o ambiente virtual será criado!

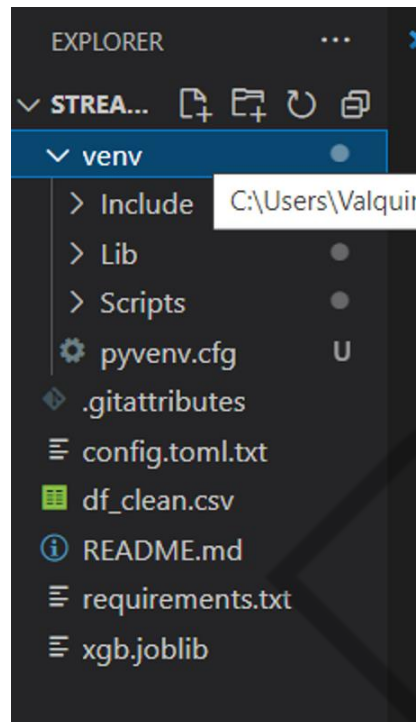


Figura 6 - Ambiente virtual criado no VSCode
Fonte: Elaborado pela autora (2023)

O próximo passo será ativar o ambiente virtual. Podemos realizar essa ativação com o seguinte comando:

`venv\Scripts\activate`

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

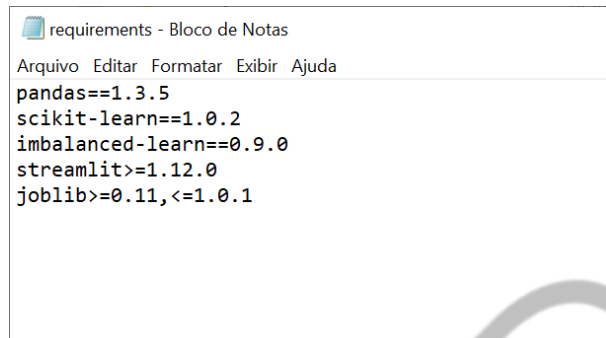
Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\Users\Valquiria\Desktop\streamlit> python -m venv venv
PS C:\Users\Valquiria\Desktop\streamlit> venv\Scripts\activate
(venv) PS C:\Users\Valquiria\Desktop\streamlit> 
```

Figura 7 - Terminal no VSCode com ambiente virtual criado
Fonte: Elaborado pela autora (2023)

Que tal dar uma olhadinha [nesse artigo](#) para aprender mais sobre ambientes virtuais?

Após criar o ambiente, está na hora de instalar todas as bibliotecas necessárias para a nossa aplicação que está no arquivo **“requirements.txt”**.



```
requirements - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
pandas==1.3.5
scikit-learn==1.0.2
imbalanced-learn==0.9.0
streamlit>=1.12.0
joblib>=0.11,<=1.0.1
```

Figura 8 - Arquivo requirements com as bibliotecas
Fonte: Elaborado pela autora (2023)

Perceba que as bibliotecas estão configuradas com as respectivas versões utilizadas no projeto. O legal de ter um ambiente virtual é que elas estão instaladas no nosso ambiente virtual, evitando assim que ocorra alguma divergência com as bibliotecas instaladas em nossa máquina local, o que diminui possíveis erros de versionamento. Para instalar, execute o comando a seguir no terminal:

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

Após todas as bibliotecas serem instaladas, chegou o momento de testarmos se o Streamlit está funcionando. No terminal, digite este código:

```
streamlit hello
```

Caso tenha dado tudo certo, a página do Streamlit será aberta:

Welcome to Streamlit! 🙌

Streamlit is an open-source app framework built specifically for Machine Learning and Data Science projects. 🖱️ **Select a demo from the sidebar** to see some examples of what Streamlit can do!

Want to learn more?

- Check out streamlit.io
- Jump into our [documentation](#)
- Ask a question in our [community forums](#)

See more complex demos

- Use a neural net to [analyze the Udacity Self-driving Car Image Dataset](#)
- Explore a [New York City rideshare dataset](#)

Figura 9 - Streamlit
Fonte: Elaborado pela autora (2023)

Criando a aplicação no Streamlit

Nossa aplicação será um **formulário de concessão de crédito**, composto por todas as perguntas que encontramos nas colunas do nosso modelo (Tal como “Qual é o seu grau de escolaridade?”, “Qual é o seu rendimento anual?”, entre outras) para prever se a pessoa que está preenchendo o formulário poderá ter crédito concebido ou não.

O modelo que criamos anteriormente e toda a pipeline serão utilizados dentro da nossa aplicação de concessão de crédito. Mas como vamos colocar tudo isso no Streamlit? Bem, o primeiro passo será criar um novo arquivo “**.py**” no VSCode para receber todos os códigos necessários para criarmos a aplicação. Na pastinha que criamos para a aplicação, existe um ícone com o símbolo de um arquivo e o sinal de mais (+), que está nomeado como “**New File**”. Você pode clicar nesse ícone e nomear um arquivo .py.

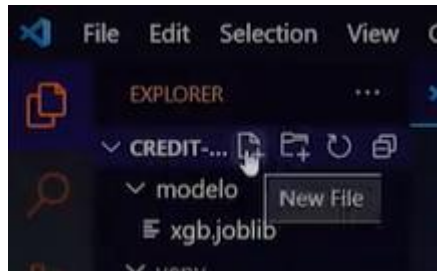


Figura 10 - New File para criar o arquivo .py
 Fonte: Elaborado pela autora (2023)

Vamos criar um arquivo nomeado como “**app.py**.” para iniciar a construção da aplicação.

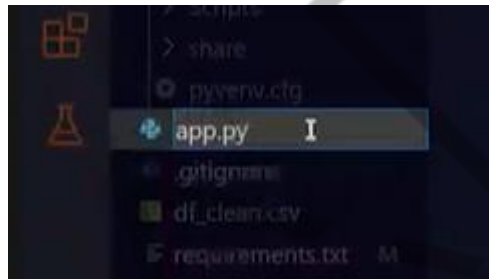


Figura 11 – Criando um arquivo na extensão python
 Fonte: Elaborado pela autora (2023)

Como utilizaremos o Streamlit, dentro do arquivo “app.py.”, você vai digitar o comando a seguir, que possibilita a importação da biblioteca do Streamlit ser utilizado dentro do Python:

`import streamlit as st`

Após importar o Streamlit, começaremos a criar o formulário. O legal dessa biblioteca é que é possível visualizar em tempo real todas as modificações realizadas. Para acompanhar as modificações, utilize o seguinte comando:

`streamlit run app.py`

Agora, vamos começar a criar as perguntinhas da aplicação que será utilizada no formulário. Para demonstrar um exemplo, vou apresentar um modelo utilizado nas aulas, mas **recomendo que você assista todas as videosaulas da disciplina para obter mais detalhes e aprender a criar toda a aplicação end-to-end.**

Começamos importando o **Pandas** e em seguida a **base de dados limpa**, “**df_clean.csv**”.

```
# importando o pandas:
```

```
import pandas as pd
```

```
#carregando os dados:
```

```
#carregando os dados
```

```
dados =
```

```
pd.read_csv('https://raw.githubusercontent.com/FIAP/Pos_Tech_DTAT/Deploy-de-Aplica%C3%A7%C3%B5es-Machine-Learning/df_clean.csv')
```

Como vamos criar perguntas para o formulário, entendemos que essas perguntas vão receber input de dados, certo? Quando a pessoa preencher o formulário, será necessário enviar os dados por meio de algum botão. O legal do Streamlit são as diversas opções do que chamamos de “**widgets**”, que nos possibilita criar de forma personalizada vários tipos de interações no aplicativo com botões, controles deslizantes, entradas de texto e muito mais! Que tal dar uma olhadinha nessa parte da [documentação](#) para aprender mais?

Para exemplificar um tipo de widget do Streamlit, criaremos um **slider**. Vamos adicionar um título com **st.write** (o “st” é o apelido do Streamlit, que nomeamos ao importar a biblioteca e o **write** é o comando para escrever texto ou títulos na página) e criar a variável **input_idade** do tipo float:

```
# Idade
```

```
st.write("""### Idade""")
```

```
input_idade = float(st.slider('Selecione a sua idade', 18, 100))
```

Viu como é simples? Se clicarmos no “ctrl s” para salvar nossas alterações e visualizarmos na página do Streamlit, observe que a aplicação possui um tipo de interação do tipo slider para a pessoa configurar a idade.

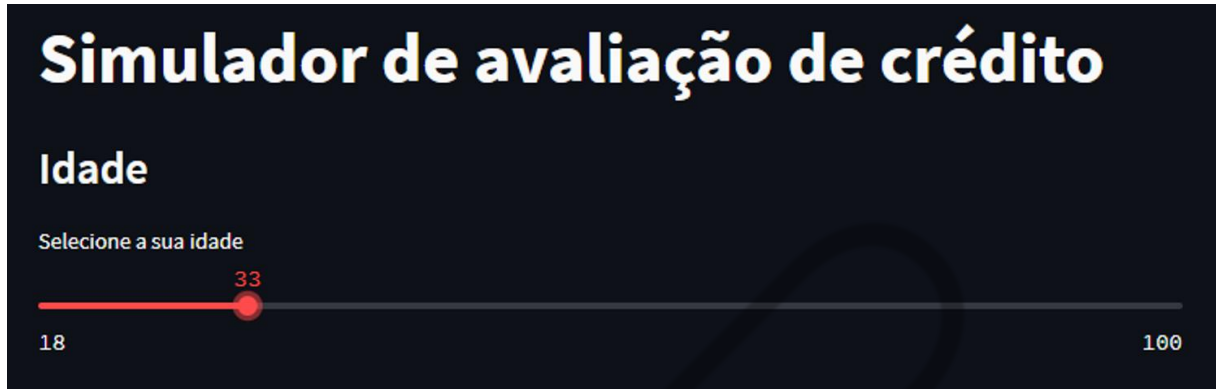


Figura 12 - Criando um slider no Streamlit
Fonte: Elaborado pela autora (2023)

Para aprender mais detalhes e como realizar a criação de todas as perguntas do modelo, acompanhe as videoaulas! Lá explicamos tudo direitinho para você.

Depois de criar todas as perguntas, o próximo passo é criar uma lista com todas as perguntas feitas na aplicação para realizar a predição no nosso modelo de concessão de crédito. Para criar a lista, elaboramos uma no Python que será nomeada como **novo_cliente** e vamos adicionar as variáveis criadas ao longo da aplicação.

Importante: as variáveis devem estar na mesma ordem do dataset “df_clean”. No final, deixamos um 0 para representar a variável Mau:

Lista de todas as variáveis:

```
novo_cliente = [0, # ID_Cliente
```

```
    input_carro_proprio, # Tem_carro
```

```
    input_casa_propria, # Tem_Casa_Propria
```

```
    telefone_trabalho, # Tem_telefone_trabalho
```

```
    telefone, # Tem_telefone_fixo
```

```
    email, # Tem_email
```

```

membros_familia, # Tamanho_Familia

input_rendimentos, # Rendimento_anual

input_idade, # Idade

input_tempo_experiencia, # Anos_empregado

input_categoria_renda, # Categoria_de_renda

input_grau_escolaridade, # Grau_Escolaridade

input_estado_civil, # Estado_Civil

input_tipo_moradia, # Moradia

input_ocupacao, # Ocupacao

0 # target (Mau)

]

```

Agora vamos criar uma função para realizar a separação do dataframe em treino e teste:

Separando os dados em treino e teste

```
def data_split(df, test_size):
```

```
    SEED = 1561651
```

```
    treino_df, teste_df = train_test_split(df, test_size=test_size, random_state=SEED)
```

```
    return treino_df.reset_index(drop=True), teste_df.reset_index(drop=True)
```

```
treino_df, teste_df = data_split(dados, 0.2)
```

Em seguida, vamos criar um dataframe do novo cliente e concatená-lo ao test_df:

```
#Criando novo cliente
```

```
cliente_predict_df = pd.DataFrame([novo_cliente],columns=teste_df.columns)
```

```
#Concatenando novo cliente ao dataframe dos dados de teste
```

```
teste_novo_cliente = pd.concat([teste_df,cliente_predict_df],ignore_index=True)
```

Temos nosso novo cliente adicionado na última linha do nosso dataframe.

Agora a variável **teste_novo_cliente** terá que passar pela pipeline para a transformação das variáveis! Usaremos o pipeline da aula, mas faremos uma modificação: não será necessário fazer o oversampling agora. Então, temos uma função pipeline_teste somente com as classes necessárias:

```
#Pipeline
```

```
def pipeline_teste(df):
```

```
    pipeline = Pipeline([
```

```
        ('feature_dropper', DropFeatures()),
```

```
        ('OneHotEncoding', OneHotEncodingNames()),
```

```
        ('ordinal_feature', OrdinalFeature()),
```

```
        ('min_max_scaler', MinMaxWithFeatNames()),
```

```
    ])
```

```
    df_pipeline = pipeline.fit_transform(df)
```

```
    return df_pipeline
```

Vamos colocar o novo dataframe na pipeline:

#Aplicando a pipeline

```
teste_novo_cliente = pipeline_teste(teste_novo_cliente)
```

A coluna Mau é retirada:

#retirando a coluna target

```
cliente_pred = teste_novo_cliente.drop(['Mau'], axis=1)
```

E agora vamos fazer nossa belíssima predição! Para colocá-la dentro do nosso formulário, criaremos uma lógica para um botão “**Enviar**” da seguinte maneira:

1. Criamos dentro de um **if** um botão com a palavra **Enviar**, para que a pessoa que preencheu o formulário possa clicar e saber o resultado;
2. O **modelo xgb** é carregado com **joblib_load**;
3. Fazemos a predição com **model.predict** e passamos **cliente_pred**;
4. Criamos um **if e else** onde, se a predição do último elemento for igual a 0, vamos ter como retorno “Parabéns! Você teve o cartão de crédito aprovado.”. Para animar o resultado, vamos adicionar **st.ballons** para subirem balões quando o resultado sair. Caso contrário, surgirá a mensagem “Infelizmente, não podemos liberar crédito para você agora!”

#Predições

```
if st.button('Enviar'):
```

```
    model = joblib.load('modelo/xgb.joblib')
```

```
    final_pred = model.predict(cliente_pred)
```

```
    if final_pred[-1] == 0:
```

```
        st.success('### Parabéns! Você teve o cartão de crédito aprovado')
```

```
        st.balloons()
```

else:

```
st.error('### Infelizmente, não podemos liberar crédito para você agora!')
```

Para o pipeline funcionar, vamos criar um arquivo chamado **utils.py** que vai conter as classes da pipeline e as bibliotecas e métodos necessários para rodá-las. Aqui está toda a pipeline completinha, caso queira copiar e colar!

```
import pandas as pd
```

```
from sklearn.base import BaseEstimator, TransformerMixin
```

```
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler, OrdinalEncoder
```

```
# Classes para pipeline
```

```
class DropFeatures(BaseEstimator,TransformerMixin):
```

```
    def __init__(self,feature_to_drop = ['ID_Cliente']):
```

```
        self.feature_to_drop = feature_to_drop
```

```
    def fit(self,df):
```

```
        return self
```

```
    def transform(self,df):
```

```
        if (set(self.feature_to_drop).issubset(df.columns)):
```

```
            df.drop(self.feature_to_drop,axis=1,inplace=True)
```

```
        return df
```

```
else:
```

```
    print('Uma ou mais features não estão no DataFrame')
```

```
    return df
```

```
class OneHotEncodingNames(BaseEstimator,TransformerMixin):
```

```
    def __init__(self,OneHotEncoding = ['Estado_civil', 'Moradia', 'Categoria_de_renda',
                                         'Ocupacao']):
```

```
        self.OneHotEncoding = OneHotEncoding
```

```
    def fit(self,df):
```

```
        return self
```

```
    def transform(self,df):
```

```
        if (set(self.OneHotEncoding).issubset(df.columns)):
```

```
            # função para one-hot-encoding das features
```

```
            def one_hot_enc(df,OneHotEncoding):
```

```
                one_hot_enc = OneHotEncoder()
```

```
                one_hot_enc.fit(df[OneHotEncoding])
```

```
                # obtendo o resultado dos nomes das colunas
```

```
                feature_names = one_hot_enc.get_feature_names_out(OneHotEncoding)
```

mudando o array do one hot encoding para um dataframe com os nomes das colunas

```
df = pd.DataFrame(one_hot_enc.transform(df[self.OneHotEncoding]).toarray(),
                  columns=feature_names,index=df.index)

return df
```

função para concatenar as features com aquelas que não passaram pelo one-hot-encoding

```
def concat_with_rest(df,one_hot_enc_df,OneHotEncoding):

    outras_features = [feature for feature in df.columns if feature not in
                        OneHotEncoding]

    # concatenar o restante das features com as features que passaram pelo
    one-hot-encoding

    df_concat = pd.concat([one_hot_enc_df, df[outras_features]],axis=1)

    return df_concat
```

one hot encoded dataframe

```
df_OneHotEncoding = one_hot_enc(df,self.OneHotEncoding)
```

retorna o dataframe concatenado

```
df_full = concat_with_rest(df, df_OneHotEncoding,self.OneHotEncoding)
```

```
return df_full
```

```
class OrdinalFeature(BaseEstimator,TransformerMixin):
```

```
    def __init__(self,ordinal_feature = ['Grau_escolaridade']):
```

```
        self.ordinal_feature = ordinal_feature
```

```
    def fit(self,df):
```

```
        return self
```

```
    def transform(self,df):
```

```
        if 'Grau_escolaridade' in df.columns:
```

```
            ordinal_encoder = OrdinalEncoder()
```

```
            df[self.ordinal_feature]
```

```
            ordinal_encoder.fit_transform(df[self.ordinal_feature])
```

```
            return df
```

```
        else:
```

```
            print('Grau_escolaridade não está no DataFrame')
```

```
            return df
```

```
class MinMaxWithFeatNames(BaseEstimator,TransformerMixin):
```

```
    def __init__(self,min_max_scaler_ft = ['Idade', 'Rendimento_anual',
'Tamanho_familia', 'Anos_empregado']):
```

```
        self.min_max_scaler_ft = min_max_scaler_ft
```

```
    def fit(self,df):
```

```

    return self

    def transform(self,df):

        if (set(self.min_max_scaler_ft).issubset(df.columns)):

            min_max_enc = MinMaxScaler()

            df[self.min_max_scaler_ft] =
min_max_enc.fit_transform(df[self.min_max_scaler_ft])

            return df

        else:

            print('Uma ou mais features não estão no DataFrame')

            return df

```

Não se esqueça que, para toda nossa pipeline funcionar, no arquivo app.py adicione no começo do arquivo, na parte de importação de bibliotecas, as demais bibliotecas necessárias para a execução do pré-processamento de dados e da pipeline.

```

from sklearn.model_selection import train_test_split

from utils import DropFeatures, OneHotEncodingNames, OrdinalFeature,
MinMaxWithFeatNames

from sklearn.pipeline import Pipeline

import joblib

from joblib import load

```

Colocando o modelo em produção

Com todas as perguntas elaboradas com seus respectivos widgets, a pipeline foi criada, perfeito! Agora temos a aplicação pronta! Para colocar o modelo em produção, que seria o famoso deploy, vamos na página do Streamlit e com uma conta criada, no canto direito superior, clique em **“New App”**:

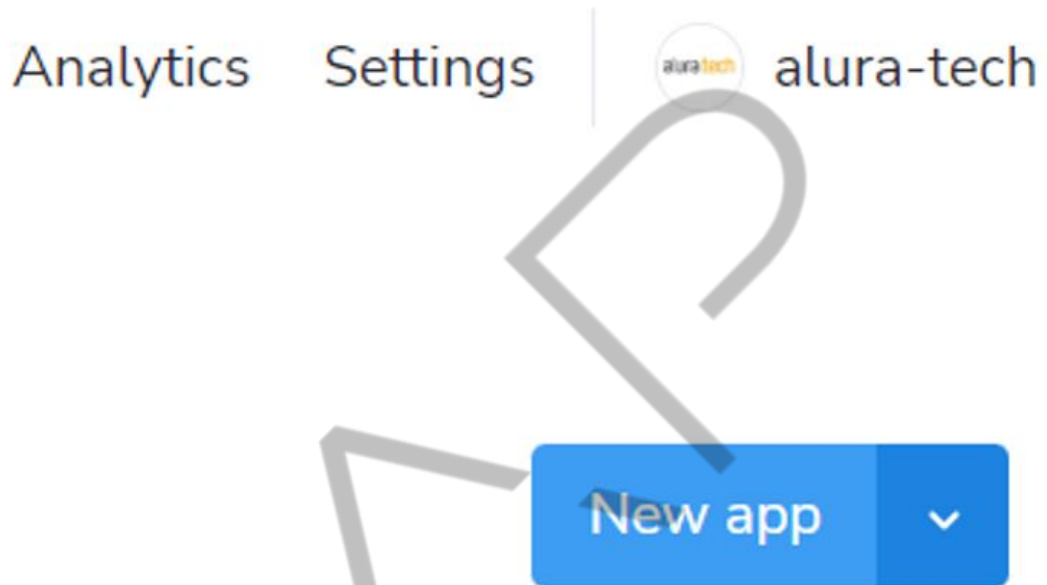


Figura 13 - Criando um app no Streamlit
Fonte: Elaborado pela autora (2023)

Na próxima página, coloque o nome do repositório, diga em qual branch está e qual é o arquivo:

[← Back](#)

Deploy an app

Repository

[Paste GitHub URL](#)

alura-tech/repo

Branch

master

Main file path

streamlit_app.py

[Advanced settings...](#)

Deploy!

Figura 14 - Deploy do app no Streamlit
Fonte: Elaborado pela autora (2023)

Você também pode ir em [Paste GitHub URL](#). Basta colar a url e clicar em Deploy:

Deploy an app

GitHub URL

[Switch to interactive picker](#)

https://github.com/username/repository/blob/master/streamlit_app.py

[Advanced settings...](#)

Deploy!

Figura 15 - Deploy com GitHub URL
Fonte: Elaborado pela autora (2023)

Agora é só aguardar um tempinho e a aplicação estará pronta! Para compartilhar com seus e suas colegas e publicar no LinkedIn, vá em Share e depois em Copy link:



Figura 16 - Deploy da aplicação
Fonte: Elaborado pela autora (2023)

O QUE VOCÊ VIU NESTA AULA?

Nessa aula, você aprendeu a criar uma aplicação no Streamlit e realizar o deploy de um modelo de Machine Learning. Agora, você já tem conhecimentos para realizar suas próprias aplicações e testar vários modelos de negócio. Que tal testar com diferentes cases?

Tem alguma dúvida ou quer conversar sobre o tema desta aula? Entre em contato conosco pela comunidade do Discord! Lá você pode fazer networking, receber avisos, tirar dúvidas e muito mais.

REFERÊNCIAS

ALENCAR, Valquiria. **Machine Learning**: Criando a Aplicação. 2023. São Paulo. Notas de aula.

STREAMLIT LIBRARY. **Streamlit**. 2023. Disponível em: <<https://docs.streamlit.io/library/get-started>>. Acesso em: 21 ago. 2023.

EMEND

PALAVRAS-CHAVE

Deploy, Streamlit, Pipeline.

EMEND

The background is a dark blue field filled with numerous small, light blue dots. Overlaid on this are several large, wavy, translucent lines in shades of blue and yellow. A vertical line on the left side has a small 'x' mark near the bottom. A circle containing the number '7' is positioned in the upper left quadrant. A hexagon is located in the lower right quadrant. The text 'POSTECH' is centered in the middle of the image.

POSTECH