Projeto da Disciplina: Engenharia de Machine Learning

- A solução criada nesse projeto deve ser disponibilizada em repositório git e disponibilizada em servidor de repositórios (Github (recomendado), Bitbucket ou Gitlab). O projeto deve obedecer o Framework TDSP da Microsoft (estrutura de arquivos, arquivo requirements.txt e arquivo README - com as respostas pedidas nesse projeto, além de outras informações pertinentes). Todos os artefatos produzidos deverão conter informações referentes a esse projeto (não serão aceitos documentos vazios ou fora de contexto). Escreva o link para seu repositório.
- Repos:

  https://github.com/pmneto/engenhariaml-docker.git

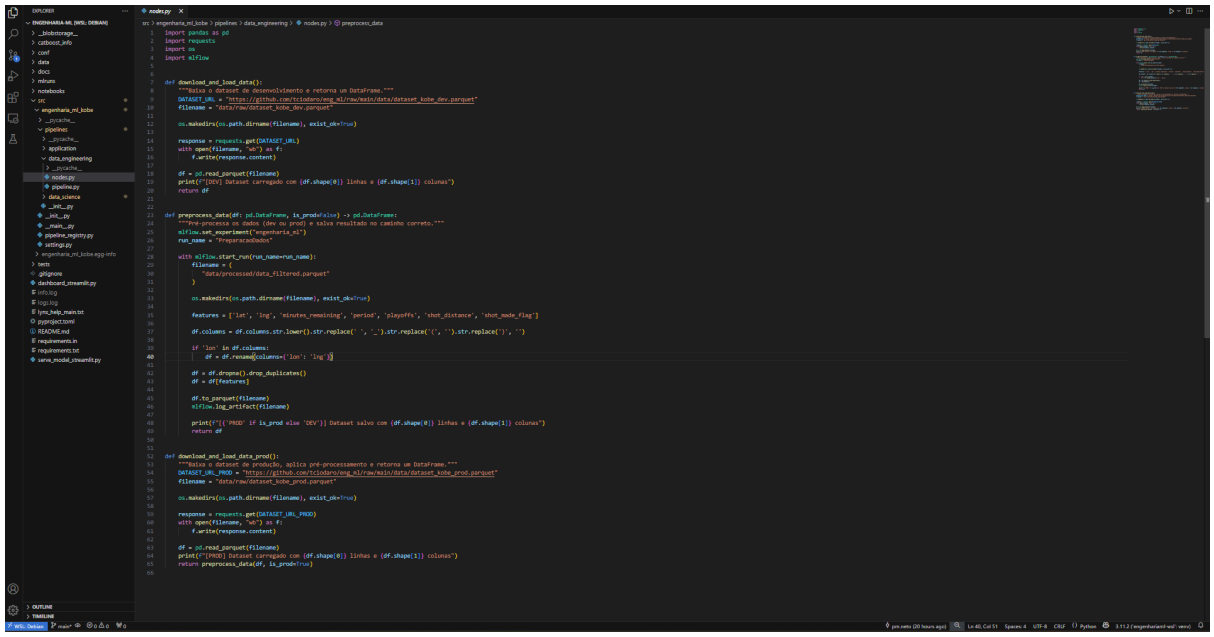
  https://github.com/pmneto/pd_engenharia_ml.git

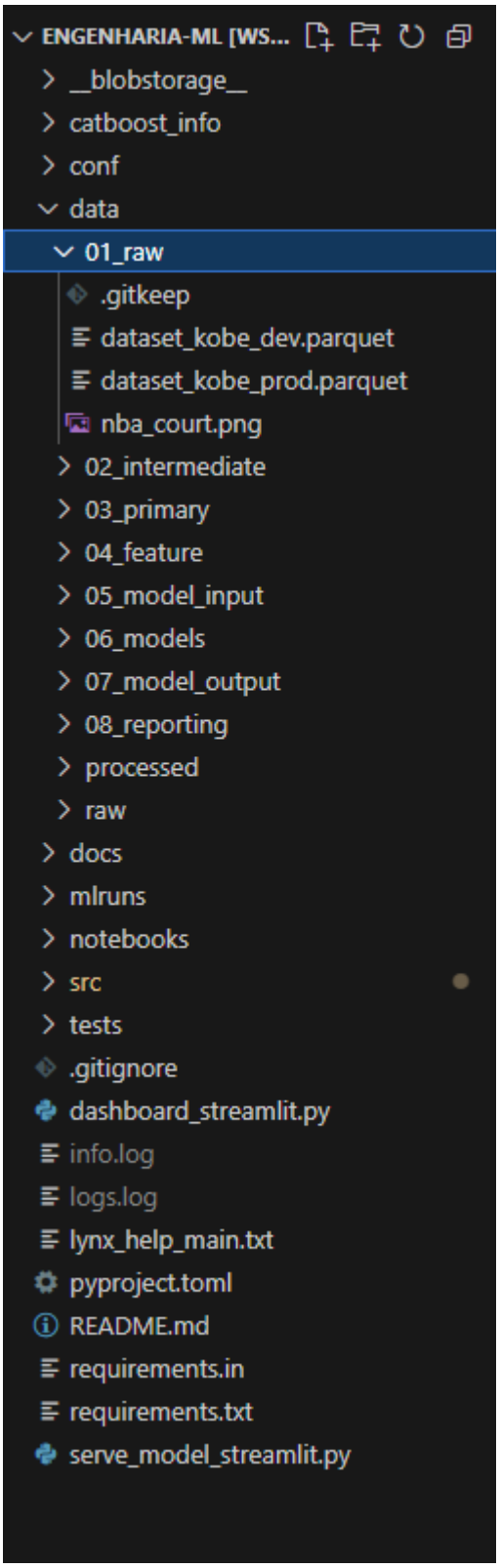
- Iremos desenvolver um preditor de arremessos usando duas abordagens (regressão e classificação) para prever se o "Black Mamba" (apelido de Kobe) acertou ou errou a cesta.

  Baixe os dados de desenvolvimento e produção aqui (datasets: dataset_kobe_dev.parquet e dataset_kobe_prod.parquet). Salve-os numa pasta /data/raw na raiz do seu repositório.

  Para começar o desenvolvimento, desenhe um diagrama que

demonstra todas as etapas necessárias para esse projeto, desde a aquisição de dados, passando pela criação dos modelos, indo até a operação do modelo.




ENGENHARIA-ML [WS...
__blobstorage__
catboost_info
conf
data
01_raw
.gitkeep
dataset_kobe_dev.parquet
dataset_kobe_prod.parquet
nba_court.png
02_intermediate
03_primary
04_feature
05_model_input
06_models
07_model_output
08_reporting
processed
raw
docs
mlruns
notebooks
src
tests
.gitignore
dashboard_streamlit.py
info.log
logs.log
lynx_help_main.txt
pyproject.toml
README.md
requirements.in
requirements.txt
serve_model_streamlit.py

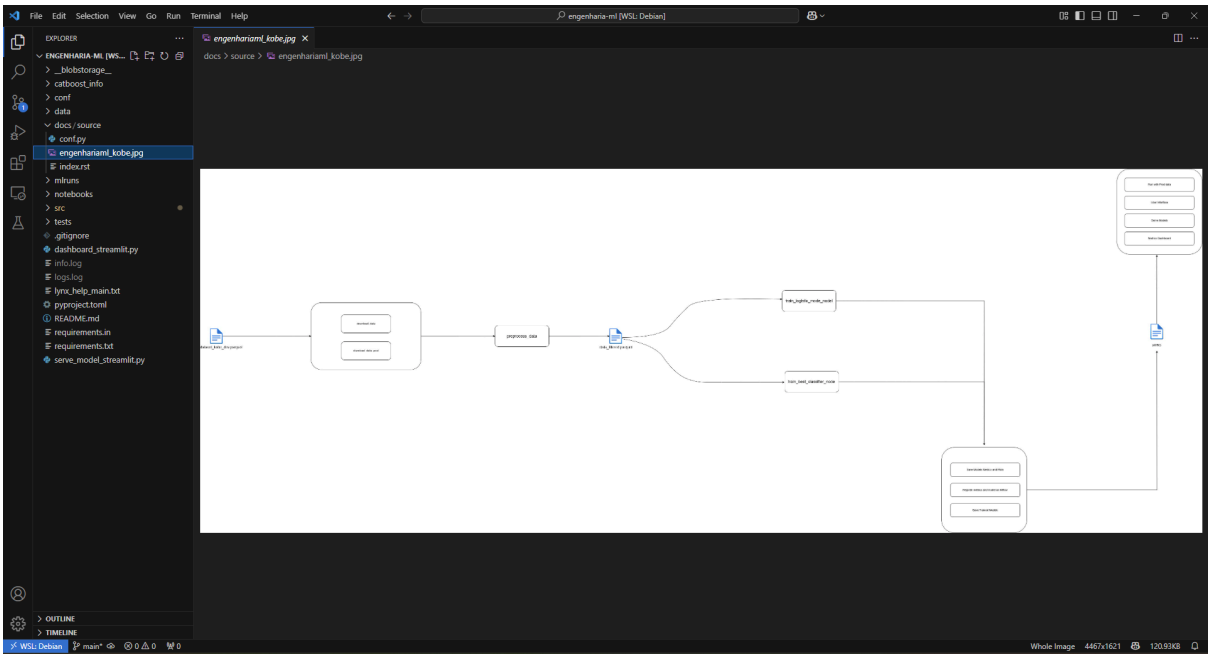
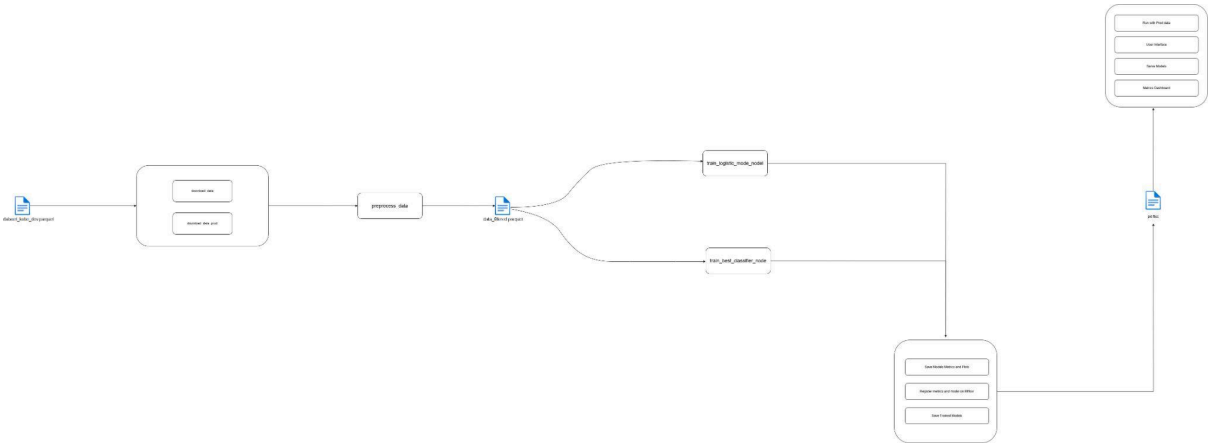


Diagrama com fluxo das implementações:



- Como as ferramentas Streamlit, MLFlow, PyCaret e Scikit-Learn auxiliam na construção dos pipelines descritos anteriormente? A resposta deve abranger os seguintes aspectos:

    a. Rastreamento de experimentos;

    O MLFlow tem a capacidade de fazer todo o rastreamento do ciclo de vida da aplicação, no contexto de modelagem de inteligência artificial. As ferramentas PyCaret, Scikit-Learn e Streamlit facilitam este processo também por possuirem integração com o MLFlow, permitindo inclusão de objetos e metadados e muitas outras facilidades na correta implementação de processos de engenharia de software dedicados a machine learning.

    b. Funções de treinamento;

    O PyCaret ajuda no contexto de simplificação dos processos para emprego de modelagem de machine learning. Usando a biblioteca, é possível selecionar de forma bastante otimizada modelos e parametros mais adequados para seu caso de uso, permitindo uma diminuição de esforço para testagem de suposições e implementações complexas. Neste sentido, o MLFlow colabora por conseguir manter históricos e facilidades, além de apis dedicadas para finalidades como servir modelos.

    c. Monitoramento da saúde do modelo;

O MLflow permite o registro de métricas como *log loss*, *F1-score* e outras que refletem o desempenho dos modelos ao longo do tempo. Isso facilita a identificação de possíveis quedas de performance (model drift).

Além disso, os dados de produção podem ser comparados com os dados de treino por meio de análises em dashboards construídos no Streamlit, que oferece uma maneira rápida e interativa de visualizar a "saúde" do modelo em operação. Assim, é possível criar alertas e mecanismos para reavaliação do modelo com base em métricas e distribuição de dados.

d. Atualização de modelo;

Com os experimentos e métricas versionadas no MLflow, é possível revalidar ou retreinar modelos com novas amostras de dados com facilidade. O pipeline de engenharia permite reexecutar etapas específicas, como pré-processamento, seleção de features ou tuning de hiperparâmetros, minimizando esforço e maximizando rastreabilidade. O PyCaret, com suas funções de reavaliação automática e tuning, facilita a atualização sem necessidade de reescrever código. Esse ciclo de retreinamento pode ser tanto reativo (após perda de performance) quanto preditivo (antecipando possíveis desvios detectados no monitoramento).

e. Provisionamento (Deployment).

O MLflow oferece uma interface para servir modelos como APIs REST de forma simples e rápida com o comando `mlflow models serve`. Essa funcionalidade permite integrar o modelo em qualquer aplicação que consiga fazer requisições HTTP. Já o Streamlit pode ser usado para criar interfaces interativas para usuários finais acessarem os resultados dos modelos ou simular predições em tempo real. Por fim, com o uso de Docker, toda a infraestrutura do projeto (incluindo Kedro, MLflow, e Streamlit) pode ser provisionada e mantida com consistência em qualquer ambiente, seja local ou em nuvem.

- Com base no diagrama realizado na questão 2, aponte os artefatos que serão criados ao longo de um projeto. Para cada artefato, a descrição detalhada de sua composição.

dataset_kobe_dev.parquet - Dataset de desenvolvimento baixado do link indicado do Github no descritivo do projeto.

dataset_kobe_prod.parquet - Dataset de produção baixado do link indicado do Github no descritivo do projeto.

data_filtered.parquet - filtragem solicitada em 5.b

confusion_matrix_reg_log.png - Matriz de Confusão da Regressão logística.

confusion_matrix.png - Matriz de confusão do modelo de classificação.

prediction_map_prod.png - Mapa de predição com os resultados de predict.

prediction_map_raw_data.png - Mapa de predição usando os dados originais.

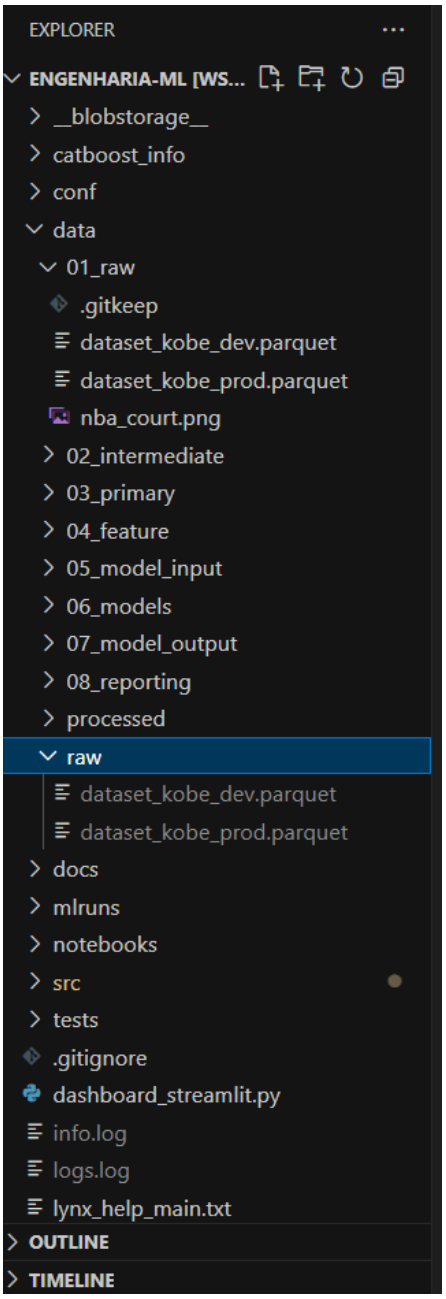
prediction_map_reg_log.png - Mapa de predição levando em consideração a regressão logística.

roc_curve_reg_log.png - Curva ROC da Regressão Logística

roc_curve.png - Curva ROC da Classificação

predictions_prod.parquet - Resultado da inferência em produção

- Implemente o pipeline de processamento de dados com o mlflow, rodada (run) com o nome "PreparacaoDados":
    a. Os dados devem estar localizados em "/data/raw/dataset_kobe_dev.parquet" e "/data/raw/dataset_kobe_prod.parquet"



b. Observe que há dados faltantes na base de dados! As linhas que possuem dados faltantes devem ser desconsideradas. Para esse exercício serão apenas consideradas as colunas:
    i. lat
    ii. lng

iii. minutes remaining
iv. period
v. playoffs
vi. shot_distance

A variável shot_made_flag será seu alvo, onde 0 indica que Kobe errou e 1 que a cesta foi realizada. O dataset resultante será armazenado na pasta "/data/processed/data_filtered.parquet". Ainda sobre essa seleção, qual a dimensão resultante do dataset?

```python
def preprocess_data(df: pd.DataFrame, is_prod=False) -> pd.DataFrame:
    """Pré-processa os dados (dev ou prod) e salva resultado no caminho correto."""
    mlflow.set_experiment("engenharia_ml")
    run_name = "PreparacaoDados"

    with mlflow.start_run(run_name=run_name):
        filename = (
            "data/processed/data_filtered_prod.parquet"
            if is_prod
            else "data/processed/data_filtered.parquet"
        )

        os.makedirs(os.path.dirname(filename), exist_ok=True)

        features = ['lat', 'lng', 'minutes_remaining', 'period', 'playoffs', 'shot_distance', 'shot_made_flag']

        df.columns = df.columns.str.lower().str.replace(' ', '_').str.replace('(', '').str.replace(')', '')

        if 'lon' in df.columns:
            df = df.rename(columns={'lon': 'lng'})

        df = df.dropna().drop_duplicates()
        df = df[features]

        df.to_parquet(filename)
        mlflow.log_artifact(filename)

        print(f"[{'PROD' if is_prod else 'DEV'}] Dataset salvo com {df.shape[0]} linhas e {df.shape[1]} colunas")
        return df
```

vii. Separe os dados em treino (80%) e teste (20 %) usando uma escolha aleatória e estratificada.

Armazene os datasets resultantes em "/Data/processed/base_{train|test}.parquet . Explique como a escolha de treino e teste afetam o resultado do modelo final. Quais estratégias ajudam a minimizar os efeitos de viés de dados.

```python
def setup_experiment(data, session_id: int, cv_folds: int) -> ClassificationExperiment:
    exp = ClassificationExperiment()
    exp.setup(
        data=data,
        target="shot_made_flag",
        session_id=session_id,
        train_size=0.8,
        fold=cv_folds,
        fold_shuffle=True,
        html=False,
        log_experiment=False,
        log_plots=False,
        verbose=False
    )
    return exp
```

```python
def train_logistic_model(data, session_id: int, cv_folds: int):

    mlflow.set_experiment("kobe_classificacao")
    with mlflow.start_run(run_name="Treinamento"):
        exp = setup_experiment(data, session_id, cv_folds)

        logistic = exp.create_model("lr", fold=cv_folds)
        tuned_logistic = exp.tune_model(logistic, n_iter=10, fold=cv_folds, optimize="AUC")
        pred = exp.predict_model(tuned_logistic)
        plot_previsoes_modelo('1',pred)



        y_test, y_pred_label, y_pred_proba = extract_metrics(pred)
        mlflow.log_metric("log_loss", log_loss(y_test, y_pred_proba))
        mlflow.log_metric("f1_score", f1_score(y_test, y_pred_label))


        metricas = {
            "model_type": type(tuned_logistic).__name__,
            "log_loss": log_loss(y_test, y_pred_proba),
            "accuracy": accuracy_score(y_test, y_pred_label),
            "precision": precision_score(y_test, y_pred_label, zero_division=0),
            "recall": recall_score(y_test, y_pred_label, zero_division=0),
            "f1_score": f1_score(y_test, y_pred_label),
            "cv_folds": cv_folds,
            "train_rows": len(data),
            "timestamp":datetime.now().isoformat()
        }

        salvar_metricas_csv('1',metricas)

        plot_confusion_matrix('1',y_test, y_pred_label)
        plot_roc_curve('1',y_test, y_pred_proba)

        os.makedirs("data/06_models", exist_ok=True)
        X_input = pred.drop(columns=["prediction_label", "prediction_score", "Label", "Score", "shot_made_flag"], errors="ignore")
        exp.save_model(tuned_logistic, "data/06_models/logistic_model")


        mlflow.sklearn.log_model(
            sk_model=tuned_logistic,           # seu modelo treinado
            artifact_path="model",             # o nome do diretório no MLflow
            input_example=X_input.iloc[:1],     # opcional mas recomendado
            signature=mlflow.models.infer_signature(X_input, tuned_logistic.predict(X_input))
        )

        mlflow.log_param("model_type", "LogisticRegression")
        mlflow.log_param("cv_folds", cv_folds)
        mlflow.log_metric("train_rows", len(data))

    return tuned_logistic
```

```python
def train_best_classifier(data, session_id: int, cv_folds: int):

    mlflow.set_experiment("kobe_classificacao")
    with mlflow.start_run(run_name="Treinamento"):
        exp = setup_experiment(data, session_id, cv_folds)

        best_model = exp.compare_models(exclude=["lr"], fold=cv_folds, n_select=1)
        tuned_best = exp.tune_model(best_model, n_iter=10, fold=cv_folds, optimize="AUC")
        pred = exp.predict_model(tuned_best)
        plot_previsoes_modelo('c',pred)

        y_test, y_pred_label, y_pred_proba = extract_metrics(pred)
        mlflow.log_metric("log_loss", log_loss(y_test, y_pred_proba))
        mlflow.log_metric("f1_score", f1_score(y_test, y_pred_label))


        metricas = {
            "model_type": type(tuned_best).__name__,
            "log_loss": log_loss(y_test, y_pred_proba),
            "accuracy": accuracy_score(y_test, y_pred_label),
            "precision": precision_score(y_test, y_pred_label, zero_division=0),
            "recall": recall_score(y_test, y_pred_label, zero_division=0),
            "f1_score": f1_score(y_test, y_pred_label),
            "cv_folds": cv_folds,
            "train_rows": len(data),
            "timestamp":datetime.now().isoformat()
        }

        salvar_metricas_csv('c',metricas)

        plot_confusion_matrix('c',y_test, y_pred_label)
        plot_roc_curve('c',y_test, y_pred_proba)

        os.makedirs("data/06_models", exist_ok=True)
        # Salva o modelo treinado
        X_input = pred.drop(columns=["prediction_label", "prediction_score", "Label", "Score", "shot_made_flag"], errors="ignore")
        _, model_path = exp.save_model(tuned_best, "data/06_models/best_classifier")

        # Faz o log do artefato corretamente
        mlflow.sklearn.log_model(
            sk_model=best_model,
            artifact_path="model",
            input_example=X_input.iloc[:1],
            signature=infer_signature(X_input, best_model.predict(X_input))
        )

        mlflow.log_param("model_type", type(tuned_best).__name__)
        mlflow.log_param("cv_folds", cv_folds)
        mlflow.log_metric("train_rows", len(data))

    return tuned_best
```

viii. Registre os parâmetros (% teste) e métricas (tamanho de cada base) no MlFlow

- Implementar o pipeline de treinamento do modelo com o MlFlow usando o nome "Treinamento"
    a. Com os dados separados para treinamento, treine um modelo com regressão logística do sklearn usando a biblioteca pyCaret.

```python
def train_logistic_model(data, session_id: int, cv_folds: int):

    mlflow.set_experiment("kobe_classificacao")
    with mlflow.start_run(run_name="Treinamento"):
        exp = setup_experiment(data, session_id, cv_folds)

        logistic = exp.create_model("lr", fold=cv_folds)
        tuned_logistic = exp.tune_model(logistic, n_iter=10, fold=cv_folds, optimize="AUC")
        pred = exp.predict_model(tuned_logistic)
        plot_previsoes_modelo('1',pred)



        y_test, y_pred_label, y_pred_proba = extract_metrics(pred)
        mlflow.log_metric("log_loss", log_loss(y_test, y_pred_proba))
        mlflow.log_metric("f1_score", f1_score(y_test, y_pred_label))


        metricas = {
            "model_type": type(tuned_logistic).__name__,
            "log_loss": log_loss(y_test, y_pred_proba),
            "accuracy": accuracy_score(y_test, y_pred_label),
            "precision": precision_score(y_test, y_pred_label, zero_division=0),
            "recall": recall_score(y_test, y_pred_label, zero_division=0),
            "f1_score": f1_score(y_test, y_pred_label),
            "cv_folds": cv_folds,
            "train_rows": len(data),
            "timestamp":datetime.now().isoformat()
        }

        salvar_metricas_csv('1',metricas)

        plot_confusion_matrix('1',y_test, y_pred_label)
        plot_roc_curve('1',y_test, y_pred_proba)

        os.makedirs("data/06_models", exist_ok=True)
        X_input = pred.drop(columns=["prediction_label", "prediction_score", "Label", "Score", "shot_made_flag"], errors="ignore")
        exp.save_model(tuned_logistic, "data/06_models/logistic_model")


        mlflow.sklearn.log_model(
            sk_model=tuned_logistic,           # seu modelo treinado
            artifact_path="model",             # o nome do diretório no MLflow
            input_example=X_input.iloc[:1],     # opcional mas recomendado
            signature=mlflow.models.infer_signature(X_input, tuned_logistic.predict(X_input))
        )

        mlflow.log_param("model_type", "LogisticRegression")
        mlflow.log_param("cv_folds", cv_folds)
        mlflow.log_metric("train_rows", len(data))
        mlflow.log_metric("test_rows", len(y_test))


    return tuned_logistic
```

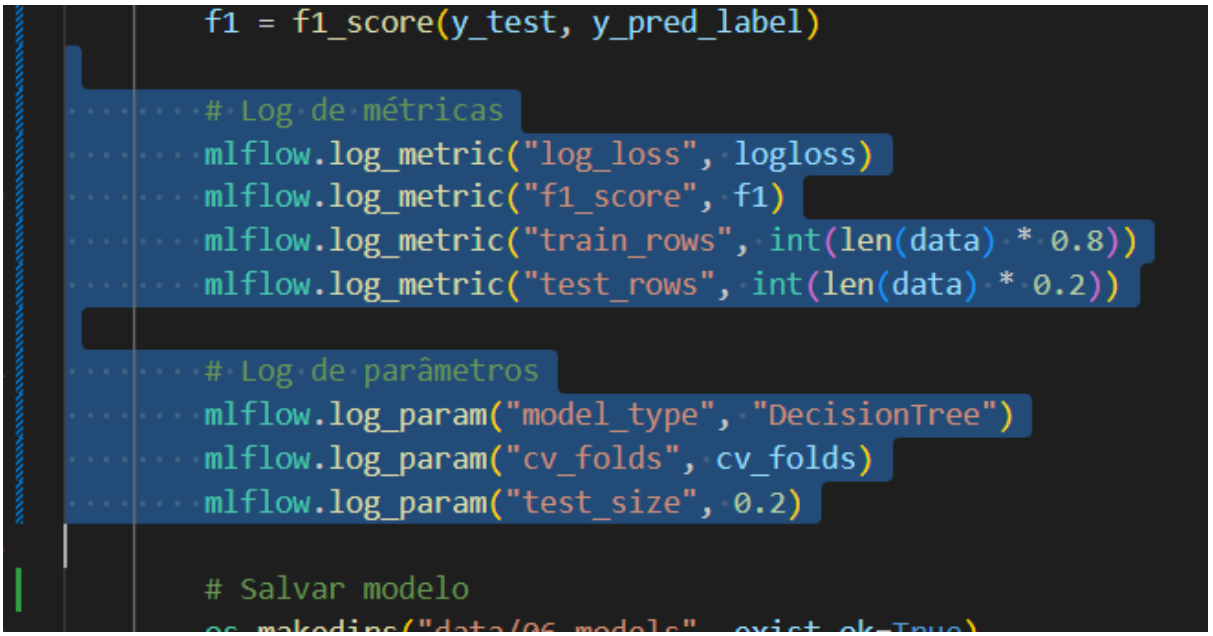
b. Registre a função custo "log loss" usando a base de teste

```
y_test, y_pred_label, y_pred_proba = extract_metrics(pred)
mlflow.log_metric("log_loss", log_loss(y_test, y_pred_proba))
mlflow.log_metric("f1_score", f1_score(y_test, y_pred_label))
```

c. Com os dados separados para treinamento, treine um modelo de árvore de decisão do sklearn usando a biblioteca pyCaret.

```python
def train_decision_tree_model(data: pd.DataFrame, session_id: int, cv_folds: int):
    """Treina e registra um modelo de árvore de decisão com PyCaret e MLflow."""


    mlflow.set_experiment("kobe_classificacao")

    with mlflow.start_run(run_name="Treinamento_ArvoreDecisao"):
        # Setup do experimento
        exp = ClassificationExperiment()
        exp.setup(
            data=data,
            target="shot_made_flag",
            session_id=session_id,
            train_size=0.8,
            fold=cv_folds,
            fold_shuffle=True,
            html=False,
            log_experiment=False,
            verbose=False
        )

        # Criação e tuning do modelo de árvore de decisão
        dt_model = exp.create_model("dt", fold=cv_folds)
        tuned_dt = exp.tune_model(dt_model, n_iter=10, fold=cv_folds, optimize="F1")

        # Predição
        pred = exp.predict_model(tuned_dt)

        # Extração de métricas
        y_test = pred["shot_made_flag"]
        y_pred_label = pred["prediction_label"]
        y_pred_proba = pred["prediction_score"]

        logloss = log_loss(y_test, y_pred_proba)
        f1 = f1_score(y_test, y_pred_label)

        # Log de métricas
        mlflow.log_metric("log_loss", logloss)
        mlflow.log_metric("f1_score", f1)
        mlflow.log_metric("train_rows", int(len(data) * 0.8))
        mlflow.log_metric("test_rows", int(len(data) * 0.2))

        # Log de parâmetros
        mlflow.log_param("model_type", "DecisionTree")
        mlflow.log_param("cv_folds", cv_folds)
        mlflow.log_param("test_size", 0.2)

        # Salvar modelo
        os.makedirs("data/06_models", exist_ok=True)
        X_input = pred.drop(columns=["prediction_label", "prediction_score", "Label", "Score", "shot_made_flag"], errors="ignore")
        exp.save_model(tuned_dt, "data/06_models/decisionTree")

        # Log do modelo completo
        mlflow.sklearn.log_model(
            sk_model=tuned_dt,
            artifact_path="model",
            input_example=X_input.iloc[:1],
            signature=infer_signature(X_input, tuned_dt.predict(X_input))
        )

        print("[OK] Modelo de árvore de decisão treinado e registrado no MLflow.")
        return tuned_dt
```

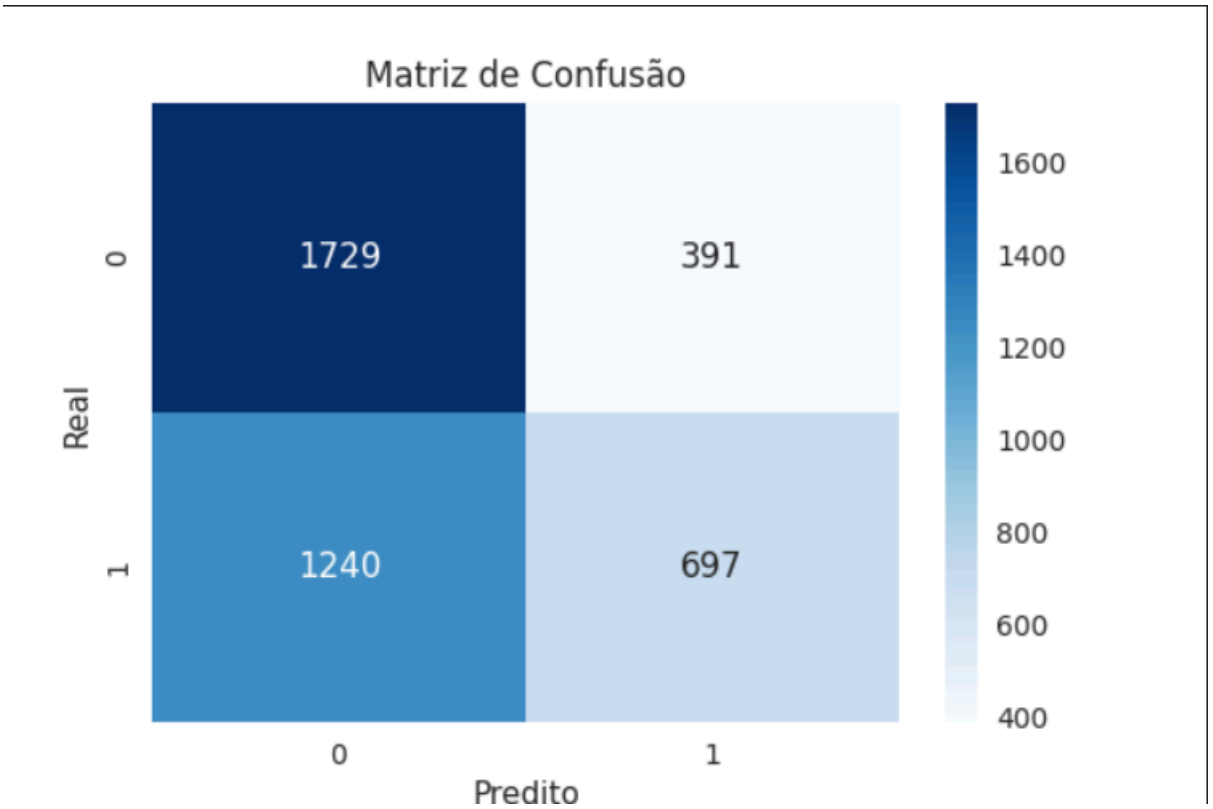
d. Registre a função custo "log loss" e F1_score para o modelo de árvore.



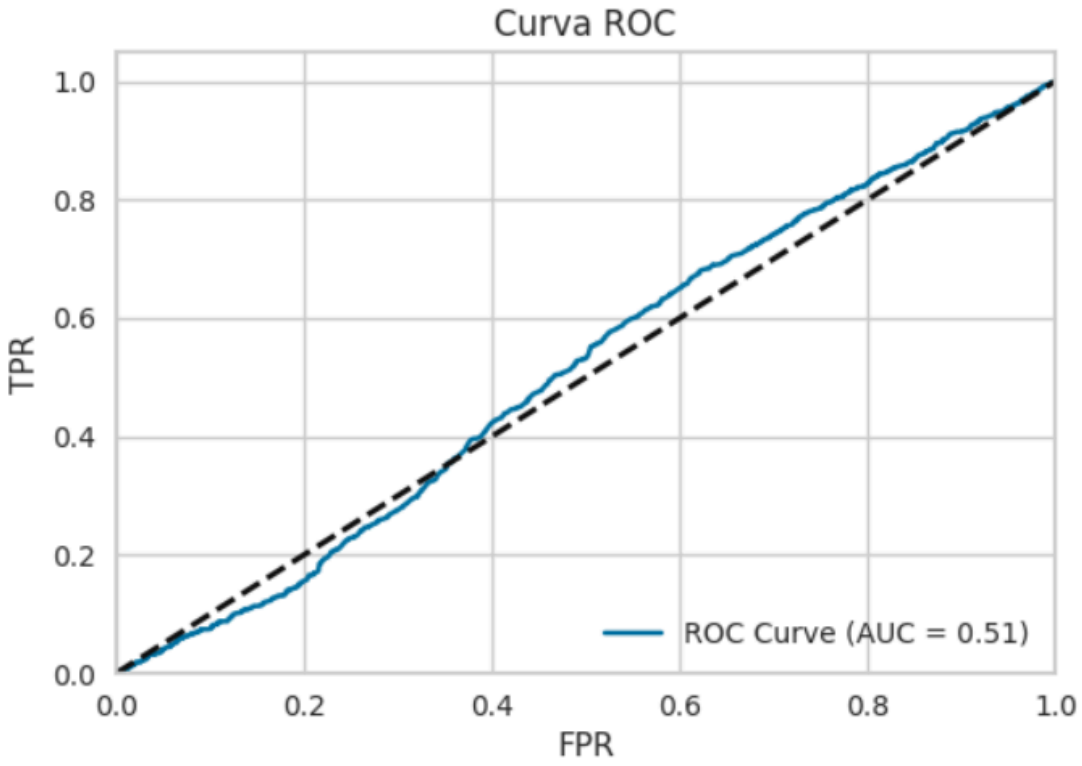
e. Selecione um dos dois modelos para finalização e justifique sua escolha.

Escolhi a Regressão Logística por que apresentou melhores métricas com o modelo treinado:

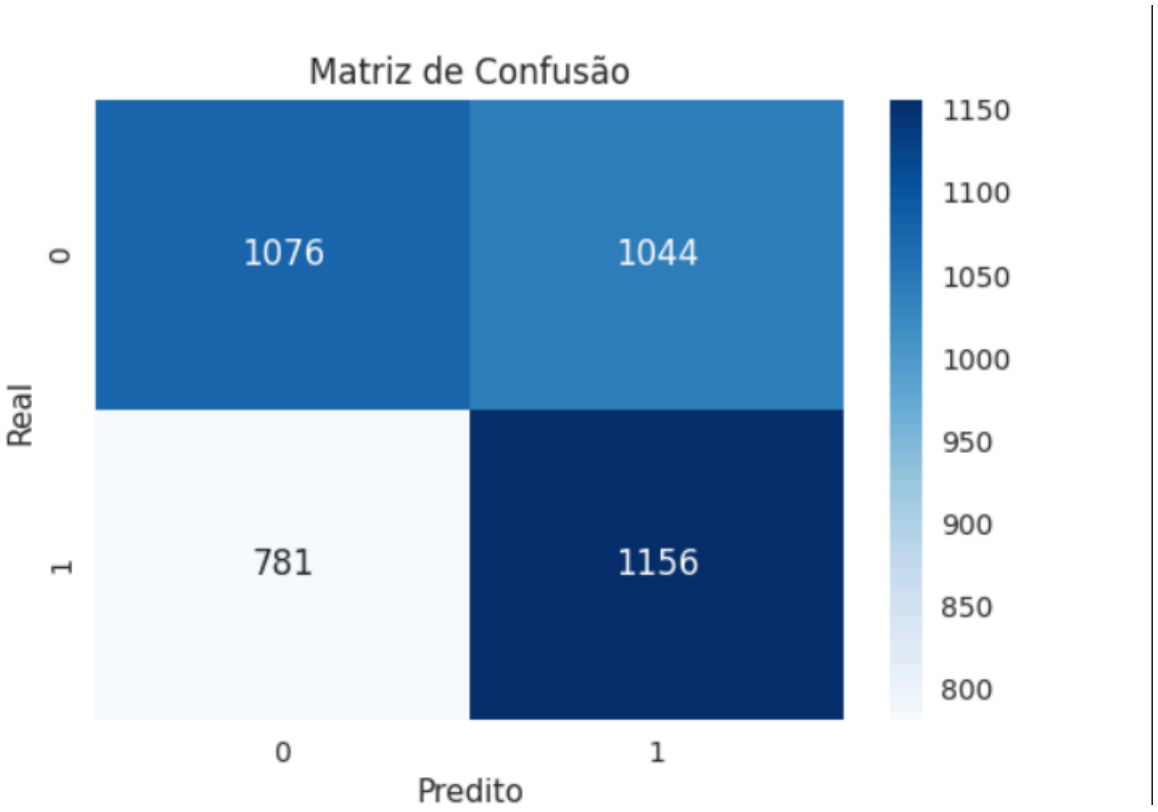
Regressão Logística:

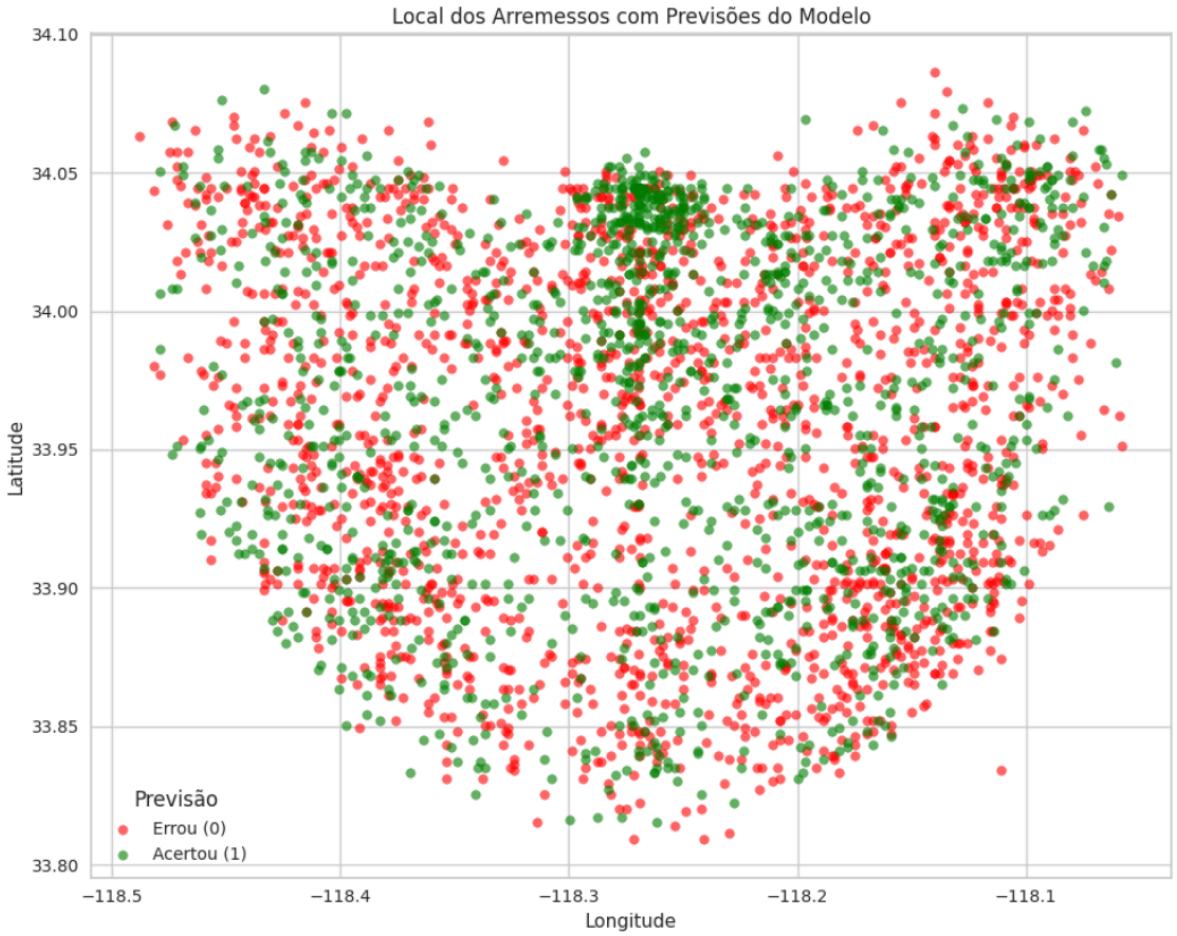


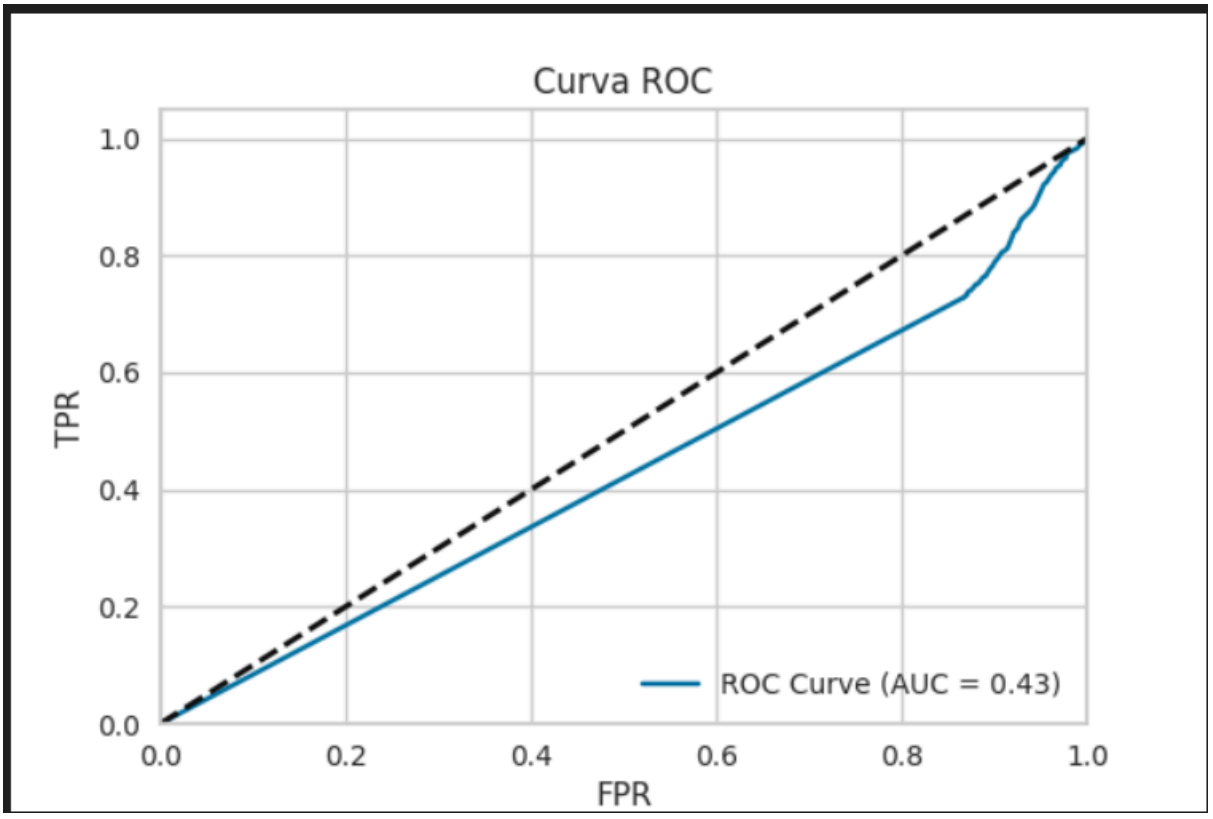

Local dos Arremessos com Previsões do Modelo
Latitude
Longitude
34.10
34.05
34.00
33.95
33.90
33.85
33.80
−118.5
−118.4
−118.3
−118.2
−118.1
Previsão
Errou (0)
Acertou (1)



Decision Tree:




Local dos Arremessos com Previsões do Modelo
Latitude
Longitude
34.10
34.05
34.00
33.95
33.90
33.85
33.80
−118.5
−118.4
−118.3
−118.2
−118.1
Previsão
Errou (0)
Acertou (1)


Curva ROC
TPR
FPR
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC Curve (AUC = 0.43)

Análise:

Desta forma temos que **Árvore de Decisão:** AUC = **0.43 e Regressão Logística:** AUC = **0.51.**

Claramente a Regressão Logística performa melhor em termos de treinamento do que a Árvore de Decisão, tanto na comparação da curva RoC quanto na Matriz de Confusão, a regressão se mostra superior. Tanto em termos de balanceamento, quanto em menor indicativo de ***overfitting***.

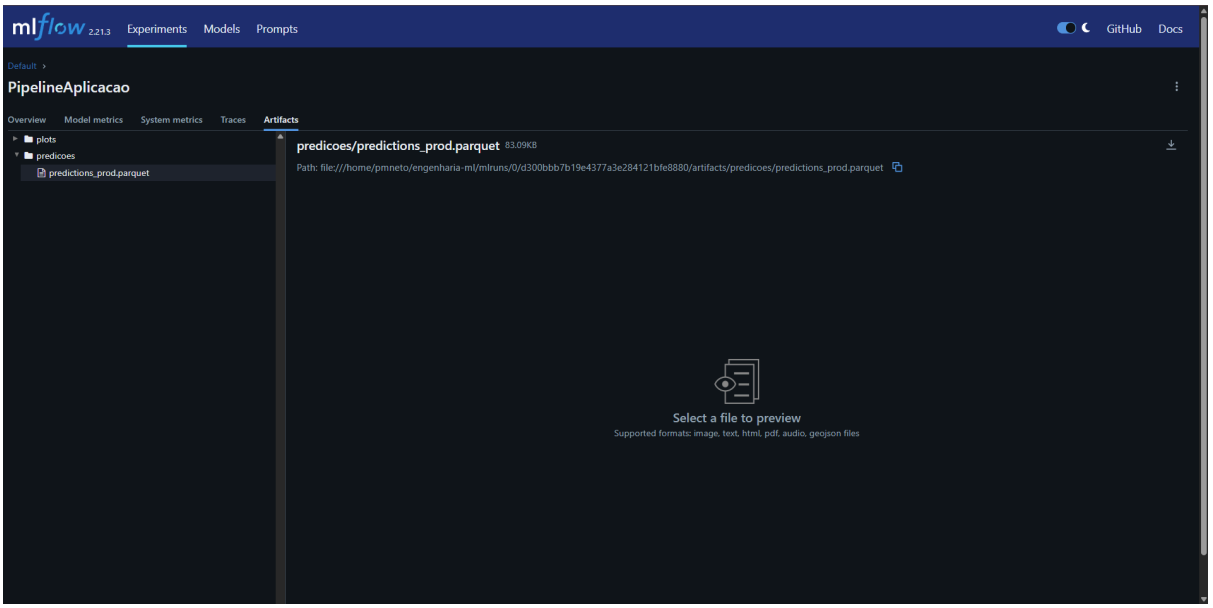
Em resumo, a Regressão Logística apresentou:

- Melhor **AUC**
- Melhor **distribuição de acertos reais**
- Melhor equilíbrio entre **falsos negativos e positivos**
- **Generalização mais robusta** no mapa de predições

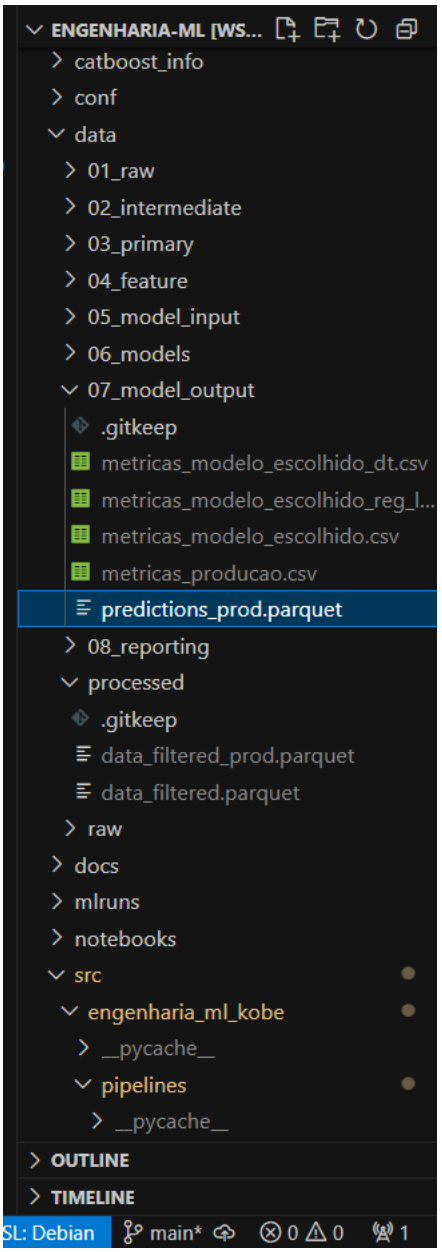
Mesmo que ambos tenham performances medianas, a árvore de decisão claramente sofre de sobreajuste ao treinamento.

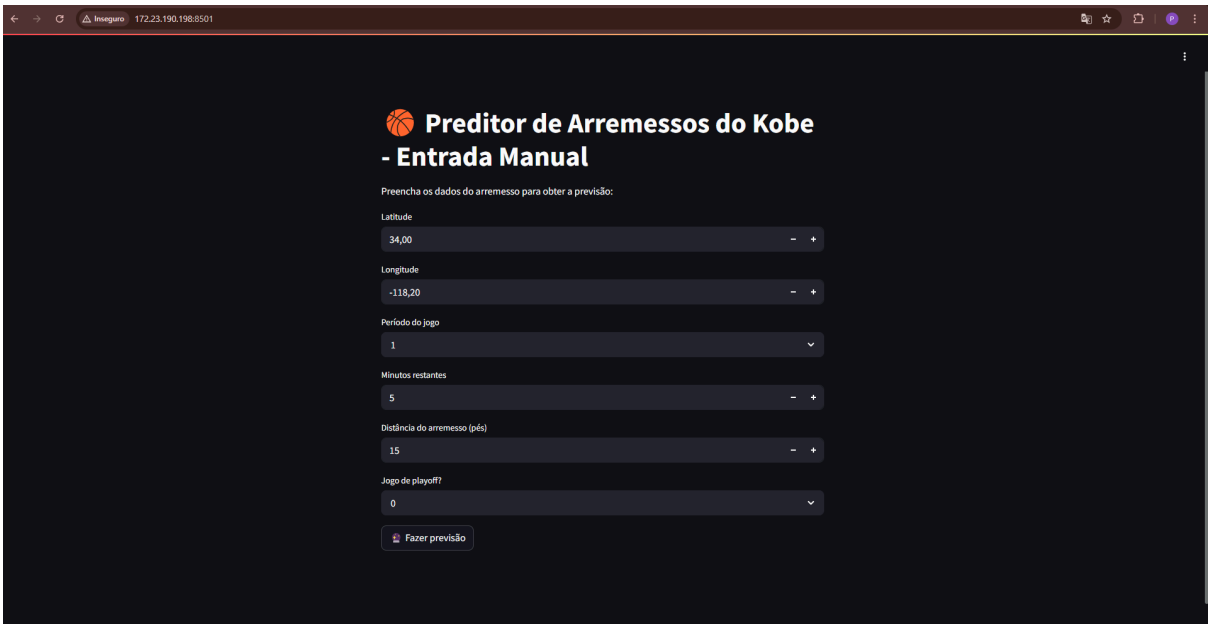
- Registre o modelo de classificação e o sirva através do MLFlow (ou como uma API local, ou embarcando o modelo na aplicação). Desenvolva um pipeline de aplicação (aplicacao.py) para carregar a base de produção (/data/raw/dataset_kobe_prod.parquet) e aplicar o modelo. Nomeie a rodada (run) do mlflow como “PipelineAplicacao” e publique, tanto uma tabela com os resultados obtidos (artefato como .parquet), quanto log as métricas do novo log loss e f1_score do modelo.

Artefato com saída dos dados do modelo:

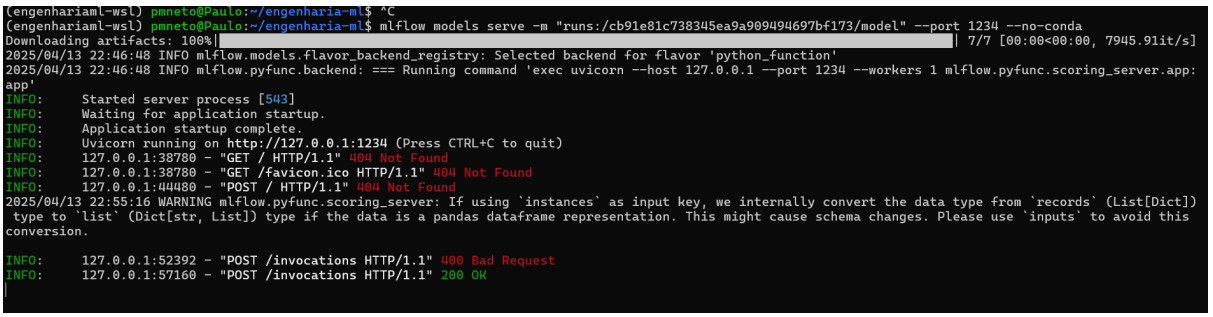


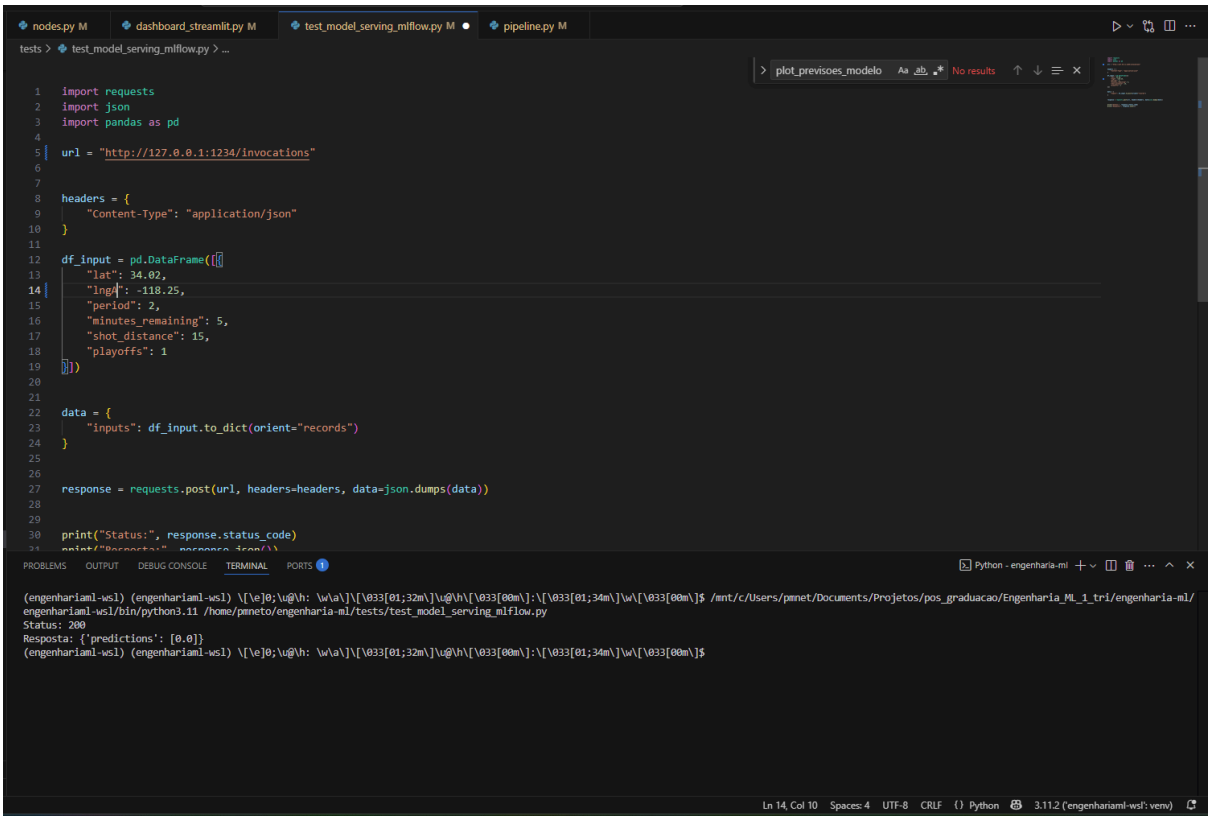
artefato predictions_prod.parquet


ENGENHARIA-ML [WS...
catboost_info
conf
data
01_raw
02_intermediate
03_primary
04_feature
05_model_input
06_models
07_model_output
.gitkeep
metricas_modelo_escolhido_dt.csv
metricas_modelo_escolhido_reg_l...
metricas_modelo_escolhido.csv
metricas_producao.csv
predictions_prod.parquet
08_reporting
processed
.gitkeep
data_filtered_prod.parquet
data_filtered.parquet
raw
docs
mlruns
notebooks
src
engenharia_ml_kobe
__pycache__
pipelines
__pycache__
OUTLINE
TIMELINE
SL: Debian
main*
0 0
1



servindo o modelo via streamlit, com página UI interativa.





Testando servir como api atraves do mlflow.

a. O modelo é aderente a essa nova base? O que mudou entre uma base e outra? Justifique.

Não, o perfil dos dados de produção é diferente. São registros com lat e lng diferentes das usadas no treino. No predict de produção, o modelo erra quase tudo.

b. Descreva como podemos monitorar a saúde do modelo no cenário com e sem a disponibilidade da variável resposta para o modelo em operação.

É possível acompanhar métricas como F1, Log Loss, AUC no tempo e queda de performance diretamente. Tanto pelo MlFlow quanto por um dashboard interativo modelado em streamlit.

c. Descreva as estratégias reativa e preditiva de retreinamento para o modelo em operação.

Reativa:

- Reavaliar periodicamente o modelo.
- Se F1 ou log loss degradarem acima de um limiar (ex.: 15%), disparar o retreinamento.

Preditiva:

- Implementar alertas de mudança de distribuição nas features (ex.: KS-test).
- Antecipar retreinamento antes que o desempenho piore.

- Implemente um dashboard de monitoramento da operação usando Streamlit.