 Instituto Infnet	Avaliação	Nota:
		Visto do Professor:
MIT em Inteligência Artificial, Machine Learning e Deep Learning		
Nome	Mateus Teixeira Ramos da Silva	
Link do repositório	https://github.com/GitMateusTeixeira/03-ml-modeling/tree/main/04_infnet_ml_engineering_pd	
Módulo	Engenharia de Machine Learning	
Prazo	14.04.2025	

1. 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.

R: O projeto seguiu a estrutura Kedro, invés de TDSP, conforme acertado com o professor nas aulas.

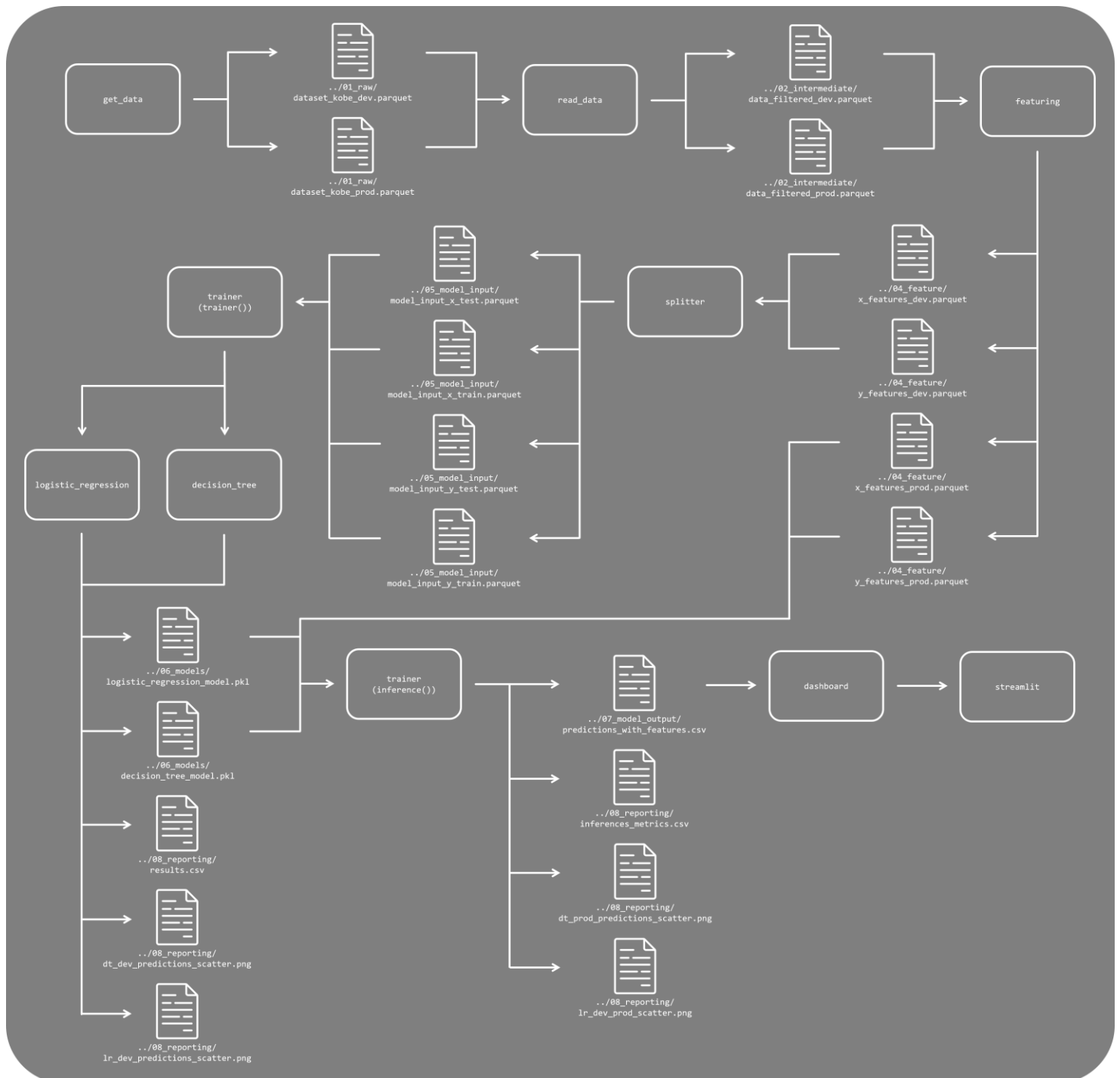
O projeto está disponível em: (https://github.com/GitMateusTeixeira/03-ml-modeling/tree/main/04_infnet_ml_engineering_pd).

2. 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.

R: Os dados são baixados através do nó 'get_data.py'. O diagrama está disponível na pasta 'docs/' (https://github.com/GitMateusTeixeira/03-ml-modeling/tree/main/04_infnet_ml_engineering_pd/docs) e explicado no README.md ():



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

3.1. Rastreamento de experimentos;

3.2. Funções de treinamento;


















3.3. Monitoramento da saúde do modelo;

3.4. Atualização de modelo;

3.5. Provisionamento (Deployment).

R: As ferramentas Streamlit, MLFlow, PyCaret e Scikit-Learn desempenharam papéis muito importantes na construção das pipelines descritas no projeto, onde cada uma contribuiu em partes diferentes do projeto.

Para o rastreamento de experimentos, o MLFlow é uma ferramenta essencial, registrando automaticamente parâmetros, métricas, artefatos e versões dos modelos. Essa rastreabilidade garante uma visão clara de cada iteração e experimentação durante o desenvolvimento.

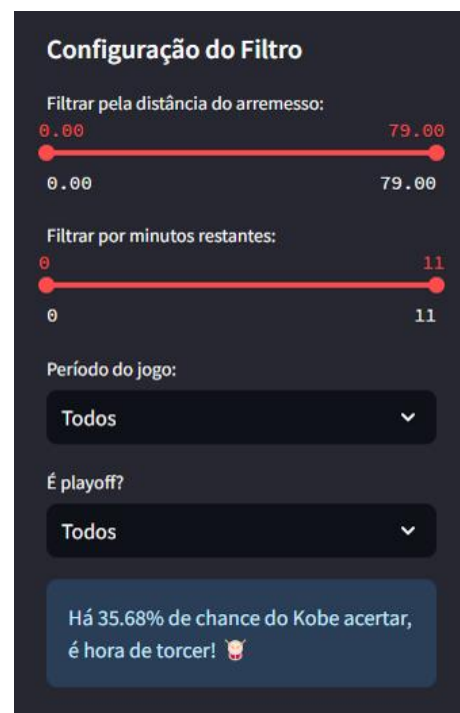
Run Name
  <code>_default_</code>
 <code>inference</code>
  <code>trainer</code>
 <code>rare-crab-852</code>
  <code>split_dev_data_node</code>
 <code>process_prod_data_node</code>
 <code>process_dev_data_node</code>
 <code>get_data</code>
  <code>extract_prod_features_node</code>
 <code>youthful-sheep-617</code>
  <code>extract_dev_features_node</code>
 <code>kindly-foal-322</code>

Em relação às funções de treinamento, tanto PyCaret quanto Scikit-Learn proporcionam pipelines bem estruturadas. O PyCaret se destaca pela simplicidade ao automatizar processos de pré-processamento, seleção de modelos e validação cruzada, enquanto o Scikit-Learn oferece flexibilidade e controle para implementar modelos personalizados, permitindo a aplicação de técnicas como Árvore de Decisão e Regressão Logística com ajustes detalhados em seus hiperparâmetros.

No monitoramento da saúde do modelo, o MLFlow novamente entra em cena ao registrar métricas de desempenho após o treinamento e a inferência, facilitando a análise contínua do comportamento dos modelos, mesmo em produção.

Já a atualização de modelos é simplificada com o PyCaret e Scikit-Learn, permitindo retreinar rapidamente os modelos com novos dados e registrar as versões atualizadas no MLFlow, mantendo todo o histórico das mudanças.

Por fim, o provisionamento é facilitado com o Streamlit, que transforma os modelos treinados e suas previsões em dashboards interativos e acessíveis, permitindo que os usuários visualizem os resultados em tempo real e interajam com os dados.



Essa integração de ferramentas não só torna o processo mais eficiente e confiável, como também promove uma colaboração intuitiva entre desenvolvedores e tomadores de decisão.

4. 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.

R: Requisito atendido no arquivo 'mflow_documentation.md' (https://github.com/GitMateusTeixeira/03-ml-modeling/tree/main/04_infnet_ml_engineering_pd/docs).

Durante o desenvolvimento do projeto, diversos artefatos foram gerados ao longo das etapas do pipeline. Esses artefatos desempenham papéis importantes na análise, validação e inferência dos modelos. São eles:

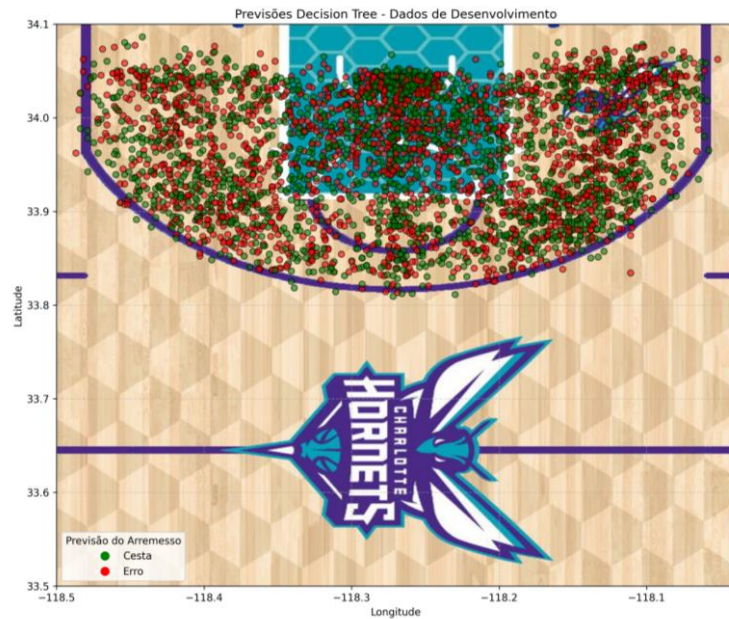
- Métricas de validação: Capturam os resultados das avaliações de desempenho dos modelos com os dados de desenvolvimento, como acurácia, F1-score e Logloss, e são salvos em formatos como '.csv'. Esses arquivos permitem análises e comparações entre os modelos, auxiliando na escolha do melhor candidato;

Metrics (10)

Search metrics	
Metric	Value
dt_accuracy	0.5342945417095778
dt_f1_score	0.602566356125857
dt_log_loss	14.468161635689158
dt_precision	0.5800338409475465
dt_recall	0.6269202633504023
lr_accuracy	0.5631307929969104
lr_f1_score	0.6693686671862822
lr_log_loss	0.6750466627873236
lr_precision	0.5832654170062483
lr_recall	0.7852962692026335

- Modelos treinados: Os modelos de Regressão Logística e Árvore de Decisão são armazenados em arquivos '.pkl'. Esses artefatos são fundamentais para inferências futuras, garantindo reprodutibilidade e acesso às versões validadas.

- Gráficos de predições: Gerados para visualizar os resultados das inferências, os gráficos exibem a distribuição das predições e são salvos como imagens 'scatter.png'. Eles fornecem uma representação visual clara da performance dos modelos.



- Arquivos de inferência: São arquivos '.csv' que contêm as previsões feitas pelos modelos aplicados aos dados de produção:

Metrics (10)

Search metrics	
Metric	Value
dt_accuracy	0.5273887332710863
dt_f1_score	0.4032226370603262
dt_log_loss	16.997724406588233
dt_precision	0.4474487570867859
dt_recall	0.3669527896995708
lr_accuracy	0.5644257703081232
lr_f1_score	0
lr_log_loss	0.6810783568959857
lr_precision	0
lr_recall	0

Além dos resultados das métricas de validação obtidos:

inference

Overview Model metrics System metrics Traces Artifacts

predictions_with_features.csv 273.71KB

Path: file:///C:/Users/pesso/Documents/DevProjects/GitHubRepositories/ml_models/infnet_04_ml_engineering_pd/mlruns/159606787888690177/38a8f27b46b94af8a867bdf09524a4/artifacts/predictions...

Previewing the first 500 rows

minutes_remaining	period	playoffs	shot_distance	decision_tree_hat	logistic_regression_hat
1	3	False	25	False	False
8	2	False	25	True	False

OBS: Note que o modelo de Regressão Logística obteve valor 0 em 'Precisão', 'Recall' e 'F1-Score', isso por que todos os dados de produção referem-se a arremessos longos, cujo resultado obtido pelo modelo foi de 100% de erros.

5. Implemente o pipeline de processamento de dados com o mlflow, rodada (run) com o nome "PreparacaoDados":

R: A run está nomeada como "infnet_04_ml_engineering_pd".

```
1 artifact_uri: file:///C:/Users/pesso/Documents/DevProjects/GitHubRepositories/ml_models/infnet_04_ml_engineering_pd/mlruns/1
2 creation_time: 1744156690094
3 experiment_id: '17'
4 last_update_time: 1744156690094
5 lifecycle_stage: 'active'
6 name: 'infnet_04_ml_engineering_pd'
```

5.1. Os dados devem estar localizados em "/data/raw/dataset_kobe_dev.parquet" e "/data/raw/dataset_kobe_prod.parquet"

R: A pipeline 'get_data' baixa os arquivos do repositório e os salva na pasta './data/01_raw/', conforme descrita na documentação 'pipelines_documentation.md'.

```
4 def get_data() -> None:
5     """
6     Baixa os arquivos de um repositório GitHub fixo e os salva no diretório especificado.
7     """
8     # Parâmetros fixos
9     repo_url = "https://github.com/tciodaro/eng_ml/raw/main/data/"
10    files = ["dataset_kobe_dev.parquet", "dataset_kobe_prod.parquet"]
11    save_path = Path(__file__).resolve().parents[4] / 'data' / '01_raw'
12
13    # Criação do diretório se não existir
14    save_path.mkdir(parents=True, exist_ok=True)
15    downloaded_files = []
16
17    for file in files:
18        file_url = repo_url + file
19        file_path = save_path / file
```

```
▼ data
  ▼ 01_raw
    ◆ .gitkeep
    ≡ dataset_kobe_dev.parquet
    ≡ dataset_kobe_prod.parquet
```

5.2. 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: ['lat', 'lon', 'minutes remaining', 'period', 'playoffs', 'shot_distance']

R: A pipeline 'read_data' lê os arquivos em '../data/01_raw/', apaga as linhas nulas e salva em '../data/02_intermediate/':

```
27 # Tratar os dados: ajustar tipos e remover valores ausentes
28 data = data.assign(
29     playoffs=lambda x: x['playoffs'].astype(bool),
30     shot_made_flag=lambda x: x['shot_made_flag'].astype(bool)
31 ).dropna(subset=['shot_made_flag'])
```

E a pipeline 'featuring' lê os arquivos em '../data/02_intermediate/', seleciona as colunas necessárias e salva os arquivos em '../data/04_feature/':

```
35 # Selecionar as colunas que serão usadas para a extração de features
36 features = ['lat', 'lon', 'minutes_remaining', 'period', 'playoffs', 'shot_distance', 'shot_made_flag']
```

Tudo conforme descrito na documentação 'pipelines_documentation.md' (https://github.com/GitMateusTeixeira/03-ml-modeling/tree/main/04_infnet_ml_engineering_pd/docs).

5.2.1. 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?

R: A pipeline 'featuring' ainda separa os arquivos 'X' e 'y' e salva em '../data/04_feature/':

```
43 # Separar X e y usando as colunas existentes
44 X = data[features[:-1]].copy() # Variáveis independentes (todas exceto a última)
45 y = data[[features[-1]]].copy() # Variável dependente (última coluna)
46
47 # Salvar os arquivos (opcional para rastreabilidade)
48 try:
49     X.to_parquet(X_FEATURES_FILE_PATH, index=False)
50     y.to_parquet(Y_FEATURES_FILE_PATH, index=False) # Converte Series para DataFrame
51
52 except Exception as e:
53     raise ValueError(f"Erro ao salvar os arquivos de X e y: {e}")
```

```
✓ 04_feature
  .gitkeep
  x_features_dev.parquet
  x_features_prod.parquet
  y_features_dev.parquet
  y_features_prod.parquet
```


5.3. 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.

R: A pipeline 'splitter' lê os arquivos de '../data/04_feature/' , separa apenas os arquivos de desenvolvimento em 'train' e 'test' e salva em '../data/05_model_input/':

```
46 test_size = 0.2
47 X_train, X_test, y_train, y_test = train_test_split(
48     X, y,
49     test_size=test_size,
50     random_state=17,
51     stratify=y
52 )
```

```
▼ 05_model_input
  ◆ .gitkeep
  ≡ model_input_x_test.parquet
  ≡ model_input_x_train.parquet
  ≡ model_input_y_test.parquet
  ≡ model_input_y_train.parquet
```

5.4. Registre os parâmetros (% teste) e métricas (tamanho de cada base) no MLFlow.

R: A pipeline 'splitter' registra o percentual de teste e o tamanho dos dataframes de treino e teste no MLFlow:

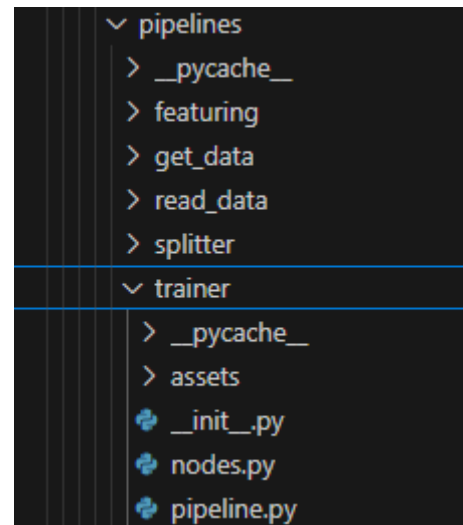
```
72 # Registrar shapes no MLflow
73 mlflow.log_metric("x_train_shape", X_train.shape[0])
74 mlflow.log_metric("x_test_shape", X_test.shape[0])
75 mlflow.log_metric("y_train_shape", y_train.shape[0])
76 mlflow.log_metric("y_test_shape", y_test.shape[0])
77 mlflow.log_metric("test_size", test_size)
```

Metrics (5)

Q Search metrics	
Metric	Value
test_size	0.2
x_test_shape	4855
x_train_shape	19416
y_test_shape	4855
y_train_shape	19416

6. Implementar o pipeline de treinamento do modelo com o MlFlow usando o nome "Treinamento".

R: O treinamento foi realizado na pipeline 'trainer' no nó 'trainer()':



Que lê os dados de ‘../data/05_model_input/’ (model_input_x_test.parquet, model_input_y_test.parquet, model_input_x_train.parquet e model_input_y_train.parquet) e realiza o treino dos modelos.

```

48 # Inicializar os modelos
49 dt_model = DecisionTreeClassifier(
50     |         |         |         |         |         |         |         |         random_state=17
51     |         |         |         |         |         |         |         )
52
53 lr_model = LogisticRegression(
54     |         |         |         |         |         |         |         random_state=17,
55     |         |         |         |         |         |         max_iter=1000
56     |         |         |         |         |         )
57
58 print("Trainer parte 02 - Modelos inicializados.")
59
60 # Treinar os modelos
61 dt_model.fit(X_train, y_train)
62 lr_model.fit(X_train, y_train)
63
64 print("Trainer parte 03 - Modelos treinados com sucesso.")
65
66 # Fazer previsões
67 dt_predictions = dt_model.predict(X_test)
68 lr_predictions = lr_model.predict(X_test)

```

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

R: A pipeline 'trainer' registra 'logloss', bem como outras métricas no MLFlow:

```
72 # Registrar métricas no MLflow
73 mlflow.log_metric("dt_accuracy", accuracy_score(y_test, dt_predictions))
74 mlflow.log_metric("lr_accuracy", accuracy_score(y_test, lr_predictions))
75 mlflow.log_metric("dt_log_loss", log_loss(y_test, dt_model.predict_proba(X_test)))
76 mlflow.log_metric("lr_log_loss", log_loss(y_test, lr_model.predict_proba(X_test)))
77 mlflow.log_metric("dt_precision", precision_score(y_test, dt_predictions, zero_division=0))
78 mlflow.log_metric("lr_precision", precision_score(y_test, lr_predictions, zero_division=0))
79 mlflow.log_metric("dt_recall", recall_score(y_test, dt_predictions, zero_division=0))
80 mlflow.log_metric("lr_recall", recall_score(y_test, lr_predictions, zero_division=0))
81 mlflow.log_metric("dt_f1_score", f1_score(y_test, dt_predictions, zero_division=0))
82 mlflow.log_metric("lr_f1_score", f1_score(y_test, lr_predictions, zero_division=0))
```

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

R: A pipeline 'trainer' treina os modelos de Árvore de Decisão e Regressão Logística:

```
48 # Inicializar os modelos
49 dt_model = DecisionTreeClassifier(
50     |
51     |
52     |
53     |
54     |
55     |
56     |
57     |
58     random_state=17
59 )
60
61 lr_model = LogisticRegression(
62     |
63     |
64     |
65     |
66     |
67     |
68     |
69     |
70     random_state=17,
71     max_iter=1000
72 )
73
74 print("Trainer parte 02 - Modelos inicializados.")
75
76 # Treinar os modelos
77 dt_model.fit(X_train, y_train)
78 lr_model.fit(X_train, y_train)
79
80 print("Trainer parte 03 - Modelos treinados com sucesso.")
81
82 # Fazer previsões
83 dt_predictions = dt_model.predict(X_test)
84 lr_predictions = lr_model.predict(X_test)
```

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

R: A pipeline 'trainer' registra 'logloss', bem como outras métricas no MLFlow tanto para Árvore de Decisão quanto para Regressão Logística:

```
72 # Registrar métricas no MLflow
73 mlflow.log_metric("dt_accuracy", accuracy_score(y_test, dt_predictions))
74 mlflow.log_metric("lr_accuracy", accuracy_score(y_test, lr_predictions))
75 mlflow.log_metric("dt_log_loss", log_loss(y_test, dt_model.predict_proba(X_test)))
76 mlflow.log_metric("lr_log_loss", log_loss(y_test, lr_model.predict_proba(X_test)))
77 mlflow.log_metric("dt_precision", precision_score(y_test, dt_predictions, zero_division=0))
78 mlflow.log_metric("lr_precision", precision_score(y_test, lr_predictions, zero_division=0))
79 mlflow.log_metric("dt_recall", recall_score(y_test, dt_predictions, zero_division=0))
80 mlflow.log_metric("lr_recall", recall_score(y_test, lr_predictions, zero_division=0))
81 mlflow.log_metric("dt_f1_score", f1_score(y_test, dt_predictions, zero_division=0))
82 mlflow.log_metric("lr_f1_score", f1_score(y_test, lr_predictions, zero_division=0))
```

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

R: A Árvore de Decisão foi escolhida para o dashboard devido à melhor performance nos dados de produção, que contêm exclusivamente arremessos de 3 pontos.

Enquanto a Regressão Logística previa todos os arremessos como erro, a Árvore de Decisão conseguiu identificar alguns acertos, trazendo maior equilíbrio e interatividade para a visualização dos resultados. Isso garantiu que o dashboard fosse mais representativo da realidade e informativo para o usuário. Conforme teste feito em '08_predictions_exploring.ipynb':

```
# Contar os valores únicos em cada coluna
decision_tree_counts = df['decision_tree_hat'].value_counts()
logistic_counts = df['logistic_regression_hat'].value_counts()

# Criar o DataFrame combinando as contagens
counts_df = pd.DataFrame(
    {
        "Predições da Regressão Logística": logistic_counts,
        "Predições da Árvore de Decisão": decision_tree_counts
    }
)

counts_df
```

	Predições da Regressão Logística	Predições da Árvore de Decisão
False	6423	4133
True	3	2293

Conforme documentado em 'streamlit_documentation.md' (https://github.com/GitMateusTeixeira/03-ml-modeling/tree/main/04_infnet_ml_engineering_pd/docs).

7. 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.

R: A pipeline 'trainer' registra ambos os modelos em 'arquivos.pkl' no MLFlow:

```
95 # Registrar os arquivos .pkl no MLflow como artefatos
96 mlflow.log_artifact(str(DT_MODEL_PATH))
97 mlflow.log_artifact(str(LR_MODEL_PATH))
98
99 print("Trainer parte 08 - Modelos salvos registrados como artefatos no MLflow.")
```

infnet_04_ml_engineering_pd >

funny-koi-922

Overview Model metrics System metrics Traces Artifacts

decision_tree_model.pkl
logistic_regression_model.pkl

A pipeline 'trainer', no nó 'inference()' é responsável por carregar os dados de produção, lendo de './data/04_feature/':

```
19 # Node para realizar inferência nos dados de produção
20 node(
21     func=nodes.inference,
22     inputs=[
23         "decision_tree_model", # Modelo de árvore de decisão
24         "logistic_regression_model", # Modelo de regressão logística
25         "x_features_prod", # Dados de produção (X)
26         "y_features_prod"
27     ],
28     outputs="predictions",
29     name="inference",
30 ),
31 ]
32 )
```

```
17 x_features_dev:
18 |   type: kedro_datasets.pandas.ParquetDataset
19 |   filepath: data/04_feature/x_features_dev.parquet
20
21 y_features_dev:
22 |   type: kedro_datasets.pandas.ParquetDataset
23 |   filepath: data/04_feature/y_features_dev.parquet
24
25 x_features_prod:
26 |   type: kedro_datasets.pandas.ParquetDataset
27 |   filepath: data/04_feature/x_features_prod.parquet
28
29 y_features_prod:
30 |   type: kedro_datasets.pandas.ParquetDataset
31 |   filepath: data/04_feature/y_features_prod.parquet
```

E registrar uma tabela com os resultados, tanto dos dados de desenvolvimento:

infnet_04_ml_engineering_pd >

trainer

OverviewModel metricsSystem metricsTracesArtifacts

results.csv

results.csv276B

Path: file:///C:/Users/pesso/Documents/DevProjects/GitHubRepositories/ml_models/infnet_04_ml_engineering_pd/mlruns/159606787888690177/b9b2527aebce4858b4c0f495ec85c00a/artifacts/results...

Previewing the first 2 rows

Model	Accuracy	Log Loss	Precision	Recall	F1-Score
Decision Tree	0.5342945417095778	14.468161635689158	0.5800338409475465	0.6269202633504023	0.602566356125857
Logistic Regression	0.5631307929969104	0.6750466627873236	0.5832654170062483	0.7852962692026335	0.6693686671862822

Quanto os dados de produção:

infnet_04_ml_engineering_pd >

inference

OverviewModel metricsSystem metricsTracesArtifacts

inferences_metrics.csv

predictions_with_features.csv

inferences_metrics.csv232B

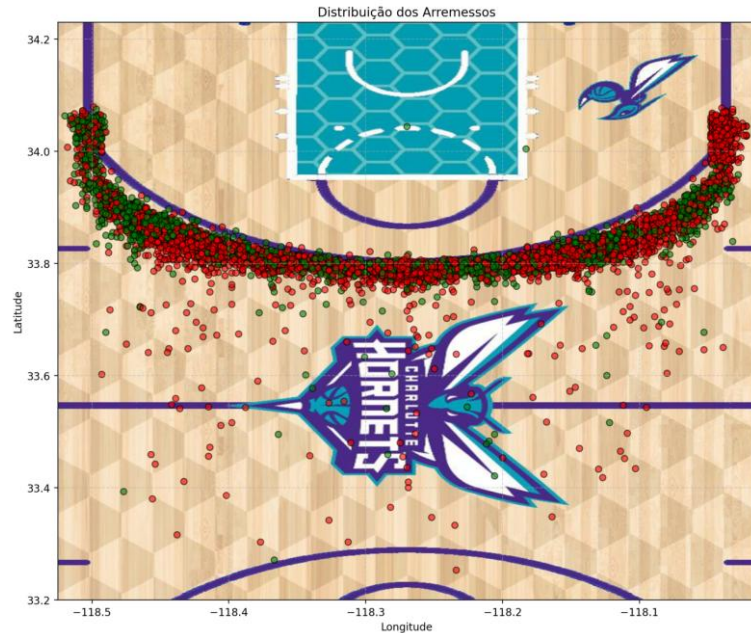
Path: file:///C:/Users/pesso/Documents/DevProjects/GitHubRepositories/ml_models/infnet_04_ml_engineering_pd/mlruns/159606787888690177/7849bf36b8a4437790045b6bf9fb36b7/artifacts/inferences...

Previewing the first 2 rows

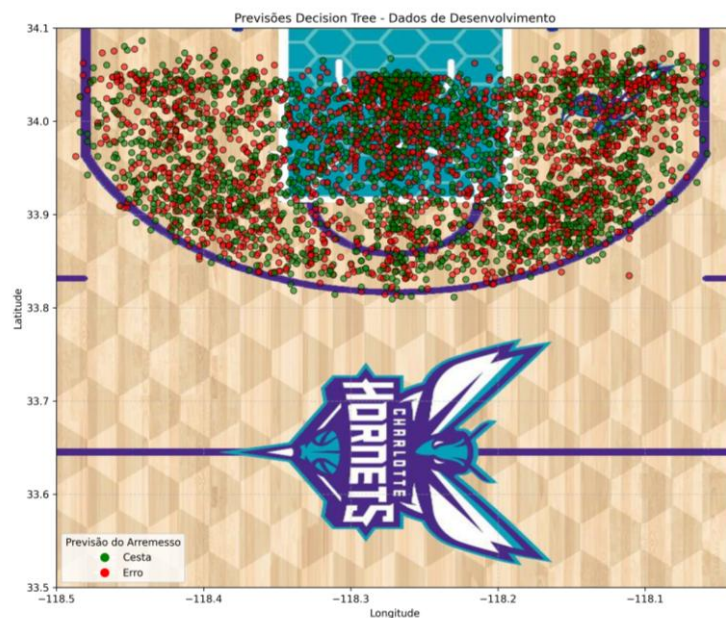
Model	Accuracy	Log Loss	Precision	Recall	F1-Score
Decision Tree	0.5273887332710863	16.997724406588233	0.4474487570867859	0.3669527896995708	0.4032226370603262
Logistic Regression	0.5644257703081232	0.6810783568959857	0	0	0

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

R: O modelo não é aderente a nova base, isso por que a nova base possui apenas dados de arremessos de 3 pontos (mais distantes):



Diferente dos dados de desenvolvimento, que possui apenas dados de arremessos de dois pontos:



Fazendo com que o modelo de Regressão Logística tivesse predito a maioria esmagadora dos arremessos como erros:

```
# Contar os valores únicos em cada coluna
decision_tree_counts = df['decision_tree_hat'].value_counts()
logistic_counts = df['logistic_regression_hat'].value_counts()

# Criar o DataFrame combinando as contagens
counts_df = pd.DataFrame({
    "Predições da Regressão Logística": logistic_counts,
    "Predições da Árvore de Decisão": decision_tree_counts
})

counts_df
```

[10]

	Predições da Regressão Logística	Predições da Árvore de Decisão
False	6423	4133
True	3	2293

Isso fica corroborado ainda pelas métricas salvas dos dados de produção, onde as métricas de ‘F1-score’, ‘precisão’ e ‘recall’ aparecem zeradas no modelo de Regressão Linear:

Metrics (10)

Q Search metrics	
Metric	Value
dt_accuracy	0.5273887332710863
dt_f1_score	0.4032226370603262
dt_log_loss	16.997724406588233
dt_precision	0.4474487570867859
dt_recall	0.3669527896995708
lr_accuracy	0.5644257703081232
lr_f1_score	0
lr_log_loss	0.6810783568959857
lr_precision	0
lr_recall	0

Isso demonstra a importância de uma boa base de dados na etapa de desenvolvimento, de modo que, quando forem carregados dados novos, os modelos consigam se adaptar melhor a nova realidade.

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

R: O monitoramento da saúde do modelo pode ser realizado de forma eficaz em ambos os cenários, com e sem a disponibilidade da variável resposta. Em cada caso, abordagens específicas são aplicadas para garantir a avaliação contínua da performance do modelo.

No cenário COM disponibilidade da variável, a avaliação da saúde do modelo é mais direta, pois podemos comparar as previsões geradas pelo modelo com os valores reais (ground truth).

Técnicas como cálculo de métricas padrão, como acurácia, recall, precisão, F1-score e Logloss, são utilizadas periodicamente para avaliar se o modelo está mantendo seu desempenho em dados de produção. Além disso, é possível utilizar validação cruzada com os novos dados e analisar possíveis desvios no comportamento do modelo, como aumento de erros sistemáticos ou degradação de métricas ao longo do tempo.

O uso de ferramentas como MLflow permite o registro histórico das métricas, possibilitando identificar tendências que indiquem a necessidade de re-treinamento ou ajustes.

Já no cenário SEM disponibilidade da variável, a análise se torna mais dependente de abordagens indiretas. Uma técnica comum é o monitoramento de distribuições das features e das probabilidades preditivas geradas pelo modelo.

Comparações podem ser feitas entre os dados do ambiente de treinamento e os dados utilizados em produção, verificando alterações significativas na distribuição das variáveis (drift).

O uso de métodos como análise de confiança das previsões também auxilia a identificar potenciais problemas no modelo, como alta confiança em previsões erradas.

Além disso, pode-se implementar monitoramento de métricas proxy, como a taxa de aceitação ou rejeição em sistemas de decisão, que refletem indiretamente o desempenho do modelo em produção.

Ferramentas como Streamlit permitem a construção de dashboards para acompanhar essas distribuições e probabilidades em tempo real.

Em ambos os cenários, o re-treinamento periódico do modelo, quando necessário, é essencial para garantir que ele continue representando bem os padrões atuais dos dados. Esses processos ajudam a manter a saúde e a eficácia do modelo, mesmo diante de alterações no ambiente de produção.

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

R: O retreinamento de modelos em operação pode ser feito de forma reativa ou preditiva.

A estratégia reativa ocorre após identificar uma queda no desempenho, com base em métricas como acurácia ou mudanças nos dados (data drift). Apesar de simples, ela pode levar a períodos de baixa performance antes que o problema seja corrigido.

Já a abordagem preditiva antecipa a necessidade de retreinamento ao monitorar sinais como alterações nas distribuições dos dados ou incertezas nas predições. Assim, o modelo é atualizado antes que haja impacto significativo, evitando falhas futuras. Ambas podem ser usadas de forma complementar, garantindo maior robustez e eficiência no ciclo de vida do modelo.

8. Implemente um dashboard de monitoramento da operação usando Streamlit.

R: O dashboard foi implementado, conforme descrito na documentação ‘streamlit_documentation.md’:



9. Assim que terminar, salve o seu arquivo PDF e poste no Moodle. Utilize o seu nome para nomear o arquivo, identificando também a disciplina no seguinte formato: “nomedoaluno_nomedadisciplina_pd.PDF”.

R: Link para o Github: (https://github.com/GitMateusTeixeira/03-ml-modeling/tree/main/04_infnet_ml_engineering_pd)