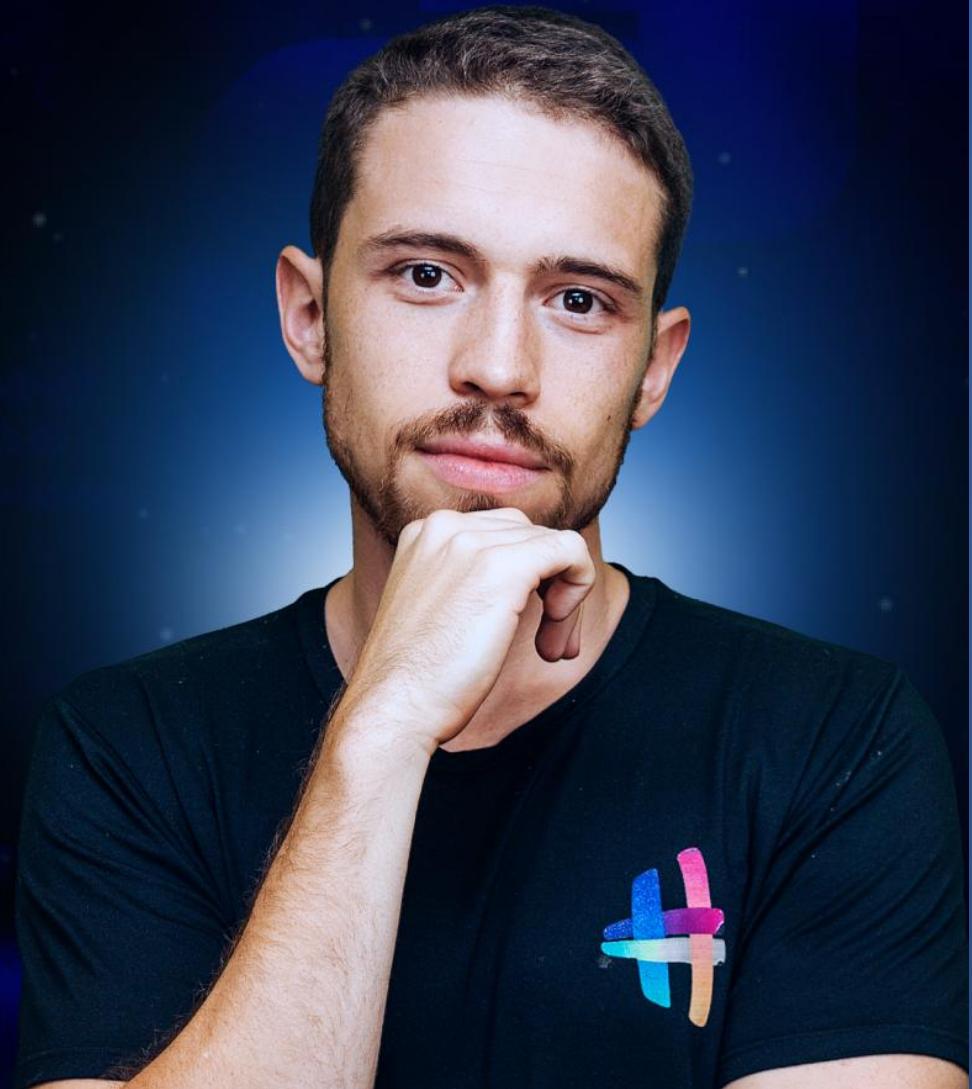




AULA 03

PROJETO DE PREVISÃO COM MACHINE LEARNING



Parte 1

Introdução

O que Vamos Aprender?

Na aula **Projeto de Previsão com Machine Learning** nós vamos fazer uma análise de **Score de cliente em um banco**. Imagine que você trabalha em um banco e tem uma base de dados de **100.000 clientes** e precisa verificar o score (pontuação) do cliente para saber se é um bom cliente ou não.

Isso vai definir se o banco vai emprestar dinheiro, se ele vai ter crédito entre outros benefícios dentro do banco. Então o nosso objetivo é fazer um tratamento na base de dados e criar alguns algoritmos de classificação e verificar qual deles é o melhor.

Com isso nós vamos conseguir fazer uma previsão dos clientes para saber se ele vai ser um cliente com um bom score ou não. Isso é muito importante, pois o banco não vai querer um cliente que não paga e que fica devendo não é mesmo?

Esse já é um projeto mais voltado para análise de dados, então, novamente você vai ver a importância de fazer o tratamento e a análise dos dados para chegar em um resultado aceitável para o seu projeto.

Lembrando que nunca vamos conseguir ter um modelo com 100% de acerto, isso é inviável, mas o objetivo é chegar em um valor aceitável que seja bom tanto para quem está construindo o projeto quanto para quem está solicitando o projeto!

Os passos que vamos fazer são:

- Importar a base de dados
- Verificar informações vazias
- Fazer o tratamento na base de dados
- Selecionar as colunas de treino para o modelo
- Treinar 2 modelos
- Verificar o melhor modelo
- Verificar quais as características mais importantes para definir o score do cliente

Parte 2

Importando a Base de Dados

Importando a Base de Dados

Importando a Base de Dados

Como você já deve saber, quando trabalhamos com dados no Python nós já pensamos na **biblioteca pandas**, pois é a biblioteca mais utilizada para essa área.

Vamos importar a biblioteca, importar a base de dados e visualizá-la. Você vai notar que não conseguimos ver todas as informações, pois temos **100.000 linhas e 25 colunas**, então temos muitas informações.

```
import pandas as pd

tabela = pd.read_csv("clientes.csv") # importa a base de dados
display(tabela)
```

	id_cliente	mes	idade	profissao	salario_anual	num_contas	num_cartoes	juros_emprestimo	num_emprestimos	dias_atraso	...	idade_historico_credito	investimento_mensal	comportamento_pagamento
0	3392	1	23.0	cientista	19114.12	3.0	4.0	3.0	4.0	3.0	...	265.0	21.465380	alto_gasto_pagamento_baixos
1	3392	2	23.0	cientista	19114.12	3.0	4.0	3.0	4.0	3.0	...	266.0	21.465380	baixo_gasto_pagamento_alto
2	3392	3	23.0	cientista	19114.12	3.0	4.0	3.0	4.0	3.0	...	267.0	21.465380	baixo_gasto_pagamento_medio
3	3392	4	23.0	cientista	19114.12	3.0	4.0	3.0	4.0	5.0	...	268.0	21.465380	baixo_gasto_pagamento_baixo
4	3392	5	23.0	cientista	19114.12	3.0	4.0	3.0	4.0	6.0	...	269.0	21.465380	alto_gasto_pagamento_medio
...
99995	37932	4	25.0	mecanico	39628.99	4.0	6.0	7.0	2.0	23.0	...	378.0	24.028477	alto_gasto_pagamento_alto
99996	37932	5	25.0	mecanico	39628.99	4.0	6.0	7.0	2.0	18.0	...	379.0	24.028477	alto_gasto_pagamento_medio
99997	37932	6	25.0	mecanico	39628.99	4.0	6.0	7.0	2.0	27.0	...	380.0	24.028477	alto_gasto_pagamento_alto
99998	37932	7	25.0	mecanico	39628.99	4.0	6.0	7.0	2.0	20.0	...	381.0	24.028477	baixo_gasto_pagamento_alto
99999	37932	8	25.0	mecanico	39628.99	4.0	6.0	7.0	2.0	18.0	...	382.0	24.028477	alto_gasto_pagamento_medio

100000 rows × 25 columns

Parte 3

Tratamento da Base de Dados

Verificando Valores Vazios

```
# verificar se temos valores vazios ou valores reconhecidos em formato errado
print(tabela.info())
print(tabela.columns)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 25 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   id_cliente       100000 non-null   int64  
 1   mes              100000 non-null   int64  
 2   idade             100000 non-null   float64 
 3   profissao        100000 non-null   object  
 4   salario_anual    100000 non-null   float64 
 5   num_contas       100000 non-null   float64 
 6   num_cartoes      100000 non-null   float64 
 7   juros_emprestimo 100000 non-null   float64 
 8   num_emprestimos  100000 non-null   float64 
 9   dias_atraso       100000 non-null   float64 
 10  num_pagamentos_atrasados 100000 non-null   float64 
 11  num_verificacoes_credito 100000 non-null   float64 
 12  mix_credito      100000 non-null   object  
 13  divida_total     100000 non-null   float64 
 14  taxa_uso_credito 100000 non-null   float64 
 15  idade_historico_credito 100000 non-null   float64 
 16  investimento_mensal 100000 non-null   float64 
 17  comportamento_pagamento 100000 non-null   object  
 18  saldo_final_mes  100000 non-null   float64 
 19  score_credito    100000 non-null   object  
...
      'comportamento_pagamento', 'saldo_final_mes', 'score_credito',
      'emprestimo_carro', 'emprestimo_casa', 'emprestimo_pessoal',
      'emprestimo_credito', 'emprestimo_estudantil'],
      dtype='object')
```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

Tendo importado a base de dados, a primeira coisa que vamos fazer é verificar se temos valores vazios, pois eles podem atrapalhar a nossa análise.

Você pode notar que não temos nenhuma informação vazia ou formatos errados. Isso quer dizer que nossa base de dados já está pronta para uso.

OBS: Caso tenha informações vazias ou qualquer outro problema é necessário que você faça o tratamento antes de começar a trabalhar para não perder o que fez, pois essas informações podem prejudicar o seu resultado.

Um ponto **IMPORTANTE** para se observar, é que nós temos algumas colunas com o tipo de informação “**object**”. Isso na verdade é um texto, só que os modelos de classificação não conseguem trabalhar com textos.

Então nós vamos precisar fazer um tratamento nessas colunas antes de continuar com o nosso projeto.

Transformando as Colunas de Texto

```
from sklearn.preprocessing import LabelEncoder
# vai transformar as colunas de texto em números, ex: profissões vai sair de cientista, professor, mecânico, etc para 0, 1, 2, etc
codificador = LabelEncoder()

# só não aplicamos na coluna de score_credito que é o nosso objetivo
for coluna in tabela.columns:
    if tabela[coluna].dtype == "object" and coluna != "score_credito":
        tabela[coluna] = codificador.fit_transform(tabela[coluna])

# verificando se realmente todas as colunas foram modificadas
for coluna in tabela.columns:
    if tabela[coluna].dtype == "object" and coluna != "score_credito":
        print(coluna)
✓ 2.5s
```

Aqui nós vamos utilizar a [biblioteca sklearn](#), então se você ainda não instalou basta escrever **pip install -U scikit-learn** no seu terminal, assim vai conseguir utilizar a biblioteca para prosseguirmos.

O objetivo de utilizar o codificador [LabelEncoder](#) é para transformar as informações de texto em números para que possamos trabalhar com os modelos de classificação, pois como falei anteriormente, eles não funcionam com informações em texto.

Assim como temos no texto explicativo, nós vamos pegar cada informação de texto tipo: cientista, professor, mecânico... E transformar isso em números: 0, 1, 2...

A única coluna de texto que nós não vamos alterar é a coluna de **score_credito**, pois é a coluna que vamos querer prever. Então transformamos todas as outras e por fim basta verificar se todas as colunas foram de fato modificadas.

Transformando as Colunas de Texto

```
display(tabela)
✓ 0.0s
```

	id_cliente	mes	idade	profissao	salario_anual	num_contas	num_cartoes	juros_emprestimo	num_emprestimos	dias_atraso	...	idade_historico_credito	investimento_mensal	comportamento_pagamento
0	3392	1	23.0	2	19114.12	3.0	4.0	3.0	4.0	3.0	...	265.0	21.465380	1
1	3392	2	23.0	2	19114.12	3.0	4.0	3.0	4.0	3.0	...	266.0	21.465380	3
2	3392	3	23.0	2	19114.12	3.0	4.0	3.0	4.0	3.0	...	267.0	21.465380	5
3	3392	4	23.0	2	19114.12	3.0	4.0	3.0	4.0	5.0	...	268.0	21.465380	4
4	3392	5	23.0	2	19114.12	3.0	4.0	3.0	4.0	6.0	...	269.0	21.465380	2
...
99995	37932	4	25.0	11	39628.99	4.0	6.0	7.0	2.0	23.0	...	378.0	24.028477	0
99996	37932	5	25.0	11	39628.99	4.0	6.0	7.0	2.0	18.0	...	379.0	24.028477	2
99997	37932	6	25.0	11	39628.99	4.0	6.0	7.0	2.0	27.0	...	380.0	24.028477	0
99998	37932	7	25.0	11	39628.99	4.0	6.0	7.0	2.0	20.0	...	381.0	24.028477	3
99999	37932	8	25.0	11	39628.99	4.0	6.0	7.0	2.0	18.0	...	382.0	24.028477	2

100000 rows × 25 columns

Essa imagem é para que você possa observar que as colunas de texto todas foram modificadas e agora possuem números para que os modelos consigam trabalhar de forma correta sem nenhum problema.

Isso é muito importante, pois esses modelos não trabalham com textos, então precisamos fazer essa transformação para que consiga fazer as classificações.

Parte 4

Modelos de Classificação

Informações para Treinar o Modelo

```
# escolhendo quais colunas vamos usar para treinar o modelo
# y é a coluna que queremos que o modelo calcule
# x vai todas as colunas que vamos usar para prever o score de credito, não vamos usar a coluna id_cliente porque ela é um numero qualquer que nao ajuda a previsao
x = tabela.drop(["score_credito", "id_cliente"], axis=1)
y = tabela["score_credito"]

from sklearn.model_selection import train_test_split

# separamos os dados em treino e teste. Treino vamos dar para os modelos aprenderem e teste vamos usar para ver se o modelo aprendeu corretamente
x_treino, x_teste, y_treino, y_teste = train_test_split(x, y, test_size=0.3, random_state=1)
```

✓ 0.3s

Agora nós vamos escolher quais as colunas vamos utilizar para treinar o nosso modelo de classificação, que é o modelo que vai pegar algumas das informações que temos (vamos separar em treino e teste).

Isso quer dizer que vamos separar nossa base de dados em 2, uma base de treino, para treinar o nosso modelo para que ele consiga fazer as previsões. E uma base de teste, que é a base que vamos utilizar para ver como está fazendo essa previsão.

É importante fazer essa separação, porque não faz muito sentido utilizar a mesma base para treinar e para testar, pois isso pode levar o modelo a “gravar” os resultados e acabar não fazendo uma previsão correta. Com essa separação podemos de fato verificar quão bem o modelo prevê o score dos nossos clientes.

Lembrando que você pode acessar a documentação do **sklearn.model_selection.train_test_split** caso tenha alguma dúvida ou queira saber mais sobre essa ferramenta! [[Clique aqui para acessar a documentação](#)]

Importando e Treinando os Modelos

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

modelo_arvore = RandomForestClassifier() # modelo arvore de decisao
modelo_knn = KNeighborsClassifier() # modelo do KNN (nearest neighbors - vizinhos mais proximos)

# treinando os modelos
modelo_arvore.fit(x_treino, y_treino)
modelo_knn.fit(x_treino, y_treino)

✓ 28.2s
```

Depois de ter feito o passo para separar a base de dados em treino e teste nós vamos importar 2 modelos de classificação para treiná-los e verificar qual deles tem o melhor resultado para esse caso.

IMPORTANTE: Não quer dizer que um modelo sendo melhor nesse caso, vai ser melhor para todos os casos. Como cada projeto é diferente eles terão resultados diferentes.

Para esse projeto nós vamos utilizar o [**Modelo Árvore de Decisão**](#) e o [**Modelo Vizinhos Mais Próximos**](#). Você pode clicar nos nomes dos modelos que será redirecionado à documentação de cada um deles. Assim pode tirar suas dúvidas e aprofundar mais em cada um deles se precisar.

Depois de importar os modelos, nós vamos treinar cada uma deles utilizando a nossa base de treino. Com isso eles estarão preparados para serem testados com a nossa base de teste e poderemos comparar o resultado entre os dois modelos.

Acurácia do Modelo

```
# se o nosso modelo chutasse tudo "Standard", qual seria a acurácia do modelo?  
contagem_scores = tabela["score_credito"].value_counts()  
print(contagem_scores['Standard'] / sum(contagem_scores))  
✓ 0.0s  
0.53174
```

Antes de explicar o código acima é importante que você entenda o que é **Acurácia**. É uma das métricas de avaliação que utilizamos para verificar se um modelo é bom em relação a outro.

A acurácia nada mais é do que a performance do modelo, ou seja, vai verificar quantas informações foram classificadas de forma correta.

Esse exemplo acima, vai fazer uma verificação de acurácia do modelo se ele tivesse “chutado” todos os scores dos clientes como **standard**. Lembrando que temos 3 tipos de score: **Poor**, **Standard** e **Good**. (Nessa ordem, do menor para o maior).

Veja que a acurácia seria de 53%. Isso quer dizer que ele acertou 53% das classificações que fez. Só que se você analisar esse número sozinho não faz tanto sentido.

Por isso que quando trabalhamos em problemas de classificação vamos utilizar pelo menos 2 modelos para verificar qual deles tem o melhor resultado. E é exatamente isso que vamos fazer no próximo passo.

Acurácia do Modelo

```
from sklearn.metrics import accuracy_score

# calculamos as previsões
previsão_arvore = modelo_arvore.predict(x_teste)
previsão_knn = modelo_knn.predict(x_teste.to_numpy())

# comparamos as previsões com o y_teste
# esse score queremos o maior (maior acurácia, mas tb tem que ser maior do que o chute de tudo Standard)
print(accuracy_score(y_teste,visão_arvore))
print(accuracy_score(y_teste,visão_knn))

✓ 3.1s

c:\Users\Heitor\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:464: UserWarning: X
warnings.warn(
0.8242
0.7324
```

Agora nós vamos de fato calcular as previsões utilizando os modelos, a **árvore de decisão** e o **KNN**. Para isso vamos utilizar o **.predict** com cada um dos modelos para fazer essa previsão.

Feita a previsão, vamos utilizar a métrica de acurácia para verificar quanto cada um dos modelos acertou.

Para a Árvore de Decisão tivemos uma acurácia de **82%** enquanto que para o modelo KNN tivemos **73%** de acurácia.

Então para esse projeto em específico entre os dois modelos que nós temos, o modelo Árvore de Decisão teve o melhor resultado.

Isso quer dizer que podemos utilizar esse modelo para prever o score dos próximos clientes.

Claro, que não vamos acertar em 100% dos casos, mas uma precisão de 82% já é um valor excelente.

Então a empresa já conseguiria prever com uma boa acurácia o score dos clientes e com isso pode fazer suas decisões de forma mais eficiente.

Características Importantes para Definir o Score

```
# quais as características mais importantes para definir o score de crédito?  
colunas = list(x_teste.columns)  
importancia = pd.DataFrame(index=colunas, data=modelo_arvore.feature_importances_)  
importancia = importancia * 100  
print(importancia)  
  
✓ 0.0s  
  
          0  
mes           3.991855  
idade          4.252459  
profissao       3.281923  
salario_anual    5.096533  
num_contas        3.475716  
num_cartoes        4.403893  
juros_emprestimo   8.040536  
num_emprestimos     3.060944  
dias_atraso         6.475263  
num_pagamentos_atrasados 4.766166  
num_verificacoes_credito 4.345831  
mix_credito          8.451818  
divida_total         11.762552  
taxa_uso_credito      5.076572  
idade_historico_credito 7.417234  
investimento_mensal     4.769693  
comportamento_pagamento 2.353365  
saldo_final_mes        5.486327  
emprestimo_carro        0.699515  
emprestimo_casa          0.707023  
emprestimo_pessoal        0.707517  
emprestimo_credito        0.687107  
emprestimo_estudantil      0.690156
```

Além de definir qual o modelo é melhor para fazer a nossa classificação, nós ainda podemos utilizar esse mesmo modelo para verificar quais as características mais importantes para definir o score de crédito.

Assim você conseguaria melhorar ainda mais sua análise e mostrar para a empresa as características mais importantes para definir o score do cliente.

Na imagem você pode notar que as informações de **divida_total**, **mix_credito** e **juros_empréstimo** são características bem importantes.

Então elas vão ajudar bastante na hora de definir o score do cliente, mais do que as outras características.

São mais pontos a serem analisados e conversados para que a empresa consiga fazer uma melhor avaliação dos seus clientes, com o objetivo de minimizar os clientes com um score pobre (**Poor**), por exemplo.

Parte 5

Conclusão

Conclusão

Esse projeto foi muito importante para te mostrar como podemos fazer a importação, tratamento de dados e análise de dados.

Com isso, foi possível treinar dois modelos de classificação para auxiliar na previsão dos dados. Isso ajudaria a empresa a verificar com uma acurácia de 82% quais os clientes que possuem um bom score.

Outro ponto, é que foi possível analisar quais as características mais importantes para definir esse score do cliente.

Percebeu como todas as etapas são importantes dentro de um projeto? Cada etapa é fundamental para que você consiga melhores resultados.

Algo bem importante é que você percebe como é importante fazer a análise de dados em uma empresa, independente da área. Você pode inclusive, fazer uma análise de dados na empresa que trabalha para tirar algumas conclusões que seus colegas ou chefes não estão conseguindo enxergar.

Pode propor algumas soluções, pode criar seus próprios modelos de classificação para verificar vários tipos de informação para diminuir prejuízos e aumentar os lucros. Nesse projeto a ideia era diminuir a quantidade de clientes com um score baixo, então conseguimos criar um modelo para fazer uma previsão e ainda conseguimos analisar quais os pontos importantes precisamos levar em consideração para análise do score.

Isso você pode reproduzir para outros projetos, lembrando de fazer as etapas com calma e fazer uma análise mais detalhada dos dados para conseguir melhores resultados.

{JORNADA} PYTHON

100% GRATUITO E ONLINE

Ainda não segue a gente no Instagram e nem é inscrito no nosso canal do Youtube? Então corre lá!



@hashtagprogramacao



youtube.com/hashtag-programacao

