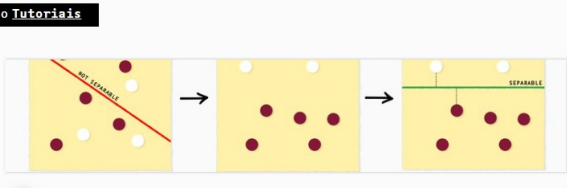


Autor: Bassim Eledath
Data Scientist

Introdução à engenharia de recursos para aprendizado de máquina com Python

A engenharia de recursos é o processo de transformação de dados para aumentar o desempenho preditivo dos modelos de aprendizado de máquina.

Conteúdo
Introdução
Importância
fluxo de trabalho
O que este artigo faz e o que não cobre
As duas abordagens para a engenharia de recursos
A Abordagem da Lista de Verificação

sua carreira.
graça

Receba as atualizações em sua caixa de entrada
Junte-se a mais de 7.500 alunos de ciência de dados.
Digite seu e-mail
Se inscrever

Artigos recentes:
- Melhores ofertas de cursos para Black Friday e Cyber Monday 2022
- Função Sigmoide
- Produto escalar
- 7 melhores cursos de inteligência artificial (IA) para 2022

Introdução

Você já deve saber:

Você já deve estar familiarizado com Python e Pandas . Você pode aprender ambos de forma interativa em Dataquest .

Você vai aprender:

Ao final deste artigo você deve saber:

- Dois métodos principais para engenharia de recursos
- Como usar Pandas e Numpy para executar várias tarefas de engenharia de recursos em Python
- Como aumentar o desempenho preditivo de um conjunto de dados real usando essas tarefas

A engenharia de recursos é indiscutivelmente a habilidade mais importante, embora negligenciada, na modelagem preditiva. Nós a empregamos em nosso dia a dia sem pensar nisso!

Deixe-me explicar - digamos que você é um barman e uma pessoa vem até você e pede uma vodca tônica. Você pede o ID e vê que o aniversário da pessoa é "09/12/1998". Esta informação não é inerentemente significativa, mas você soma o número de anos fazendo algumas contas mentais rápidas e descobre que a pessoa tem 22 anos (que está acima da idade legal para beber). O que aconteceu lá? Você pegou uma informação ("12/09/1998") e a transformou em outra variável (idade) para resolver a questão que você tinha ("Esta pessoa pode beber?").

A engenharia de recursos é exatamente isso, mas para modelos de aprendizado de máquina. Damos ao(s) nosso(s) modelo(s) a melhor representação possível de nossos dados - transformando-os e manipulando-os - para prever melhor nosso resultado de interesse. Se isso não estiver 100% claro agora, ficará muito mais claro à medida que percorremos exemplos reais neste artigo.

Definição

A engenharia de recursos é o processo de transformação de dados para aumentar o desempenho preditivo dos modelos de aprendizado de máquina.

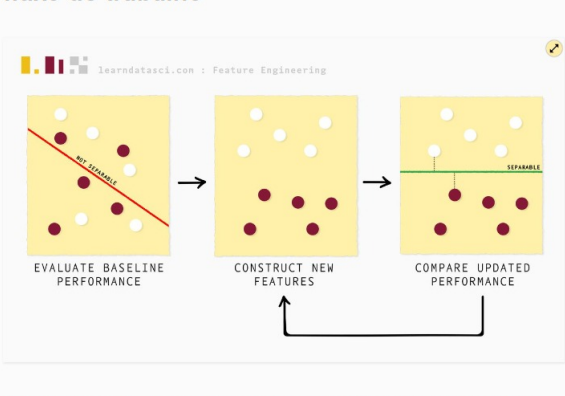
Importância

A engenharia de recursos é útil e necessária pelos seguintes motivos:

1. Freqüentemente, melhor precisão preditiva: as técnicas de engenharia de recursos, como padronização e normalização, geralmente levam a uma melhor ponderação de variáveis, o que melhora a precisão e, às vezes, leva a uma convergência mais rápida.
2. Melhor interpretabilidade dos relacionamentos nos dados: quando projetamos novos recursos e entendemos como eles se relacionam com nosso resultado de interesse, isso abre nossa compreensão dos dados. Se pularmos a etapa de engenharia de recursos e usarmos modelos complexos (que em grande parte automatizam a engenharia de recursos), ainda podemos obter uma alta pontuação de avaliação, ao custo de entender melhor nossos dados e sua relação com a variável de destino.

A engenharia de recursos é necessária porque a maioria dos modelos não pode aceitar certas representações de dados. Modelos como regressão linear, por exemplo, não podem lidar com valores ausentes por conta própria - eles precisam ser imputados (preenchidos). Veremos exemplos disso na próxima seção.

fluxo de trabalho


learndatasci.com : Feature Engineering
NOT SEPARABLE
SEPARABLE
EVALUATE BASELINE PERFORMANCE
CONSTRUCT NEW FEATURES
COMPARE UPDATED PERFORMANCE

Todo pipeline de ciência de dados começa com a Análise Exploratória de Dados (EDA), ou a análise inicial de nossos dados. O EDA é uma etapa crucial do pré-cursor, pois temos uma noção melhor de quais recursos precisamos criar/modificar. A próxima etapa geralmente é a limpeza/padronização de dados, dependendo de quão desestruturados ou confusos são os dados.

A engenharia de recursos segue em seguida e começamos esse processo avaliando o desempenho da linha de base dos dados em mãos. Em seguida, construímos recursos iterativamente e avaliamos continuamente o desempenho do modelo (e o comparamos com o desempenho da linha de base) por meio de um processo chamado seleção de recursos, até ficarmos satisfeitos com os resultados.

O que este artigo faz e o que não cobre

A engenharia de recursos é um campo vasto, pois há muitas tarefas específicas de domínio. Este artigo aborda algumas das técnicas populares empregadas na manipulação de conjuntos de dados tabulares. Não cobrimos a engenharia de recursos para processamento de linguagem natural (NLP), classificação de imagens, dados de séries temporais, etc.

As duas abordagens para a engenharia de recursos

Existem duas abordagens principais para a engenharia de recursos para a maioria dos conjuntos de dados tabulares:

1. **A abordagem da lista de verificação:** usando métodos testados e comprovados para construir recursos.
2. **A abordagem baseada em domínio:** incorporando o conhecimento de domínio do assunto do conjunto de dados na construção de novos recursos.

Veremos agora essas abordagens em detalhes usando conjuntos de dados reais. Observe que esses exemplos são bastante procedimentais e se concentram em mostrar como você pode implementá-lo em Python. O estudo de caso a seguir à seção mostrará um caso real de uso de cenário de ponta a ponta das práticas de engenharia de recursos.

Antes de carregarmos o conjunto de dados, importamos as seguintes dependências conforme abaixo:

```
# Dependencies

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()
sns.set_palette(sns.color_palette(['#851836', '#edbd17']))
sns.set_style("darkgrid")

from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

Aprenda ciência de dados com DATAQUEST

Vamos agora demonstrar a abordagem de lista de verificação usando um conjunto de dados sobre vendas em supermercados. O conjunto de dados original e mais informações sobre ele estão vinculados aqui . Observe que o conjunto de dados foi ligeiramente modificado para este tutorial.

Expanda para ver as descrições das colunas +

```
df = pd.read_csv('data/supermarket_sales.csv')
df.head()
```

FORA:

	ID da fatura	Ramo	Cidade	Tipo de Cliente	Gênero	Linha de produto	Preço unitário	Qu
0	750-67-8428	UMA	Yangon	Membro	Fêmea	Saúde e beleza	74,69	7
1	226-31-3081	C	Naypyitaw	Normal	Fêmea	acessórios eletrônicos	15.28	5
2	631-41-3108	UMA	Yangon	Normal	Macho	Casa e estilo de vida	46.33	7
3	123-19-1176	UMA	Yangon	Membro	Macho	Saúde e beleza	58.22	8
4	373-73-7910	UMA	Yangon	Normal	Macho	esportes e viagens	86.31	7

A Abordagem da Lista de Verificação

Agregações Numéricas

A agregação numérica é uma abordagem de engenharia de recursos comum para dados longitudinais ou de painel - dados em que os assuntos são repetidos. Em nosso conjunto de dados, temos variáveis categóricas com observações repetidas (por exemplo, temos várias entradas para cada filial de supermercado).

A agregação numérica envolve três parâmetros:

1. *Coluna categórica*
2. Coluna(s) numérica(s) a ser(em) agregada(s)
3. Tipo de agregação: média, mediana, moda, desvio padrão, variância, contagem, etc.

O trecho de código abaixo mostra três exemplos de agregações numéricas com base na média, desvio padrão e contagem, respectivamente.

No bloco seguinte, nossos três parâmetros são:

1. *Branch* – coluna categórica, que estamos agrupando por
2. *Imposto de 5%, preço unitário, linha de produto e gênero* - colunas numéricas a serem agregadas
3. *Média, desvio padrão e contagem* – agregações a serem usadas nas colunas numéricas

Abaixo, agrupamos por Ramo e realizamos três agregações estatísticas (média, desvio padrão e contagem) transformando as colunas numéricas de interesse. Por exemplo, na atribuição da primeira coluna, calculamos o *imposto médio de 5%* e o *preço unitário médio* para cada filial, o que nos dá duas novas colunas - `tax_branch_mean` e `unit_price_mean` no quadro de dados.

```
# Numeric aggregations

grouped_df = df.groupby('Branch')

df[['tax_branch_mean','unit_price_mean']] = grouped_df[['Tax 5%', 'Unit price']].transf
df[['tax_branch_std','unit_price_std']] = grouped_df[['Tax 5%', 'Unit price']].transfo
df[['product_count','gender_count']] = grouped_df[['Product line', 'Gender']].transfor
```

Aprenda ciência de dados com DATAQUEST

E vemos os recursos que acabamos de criar abaixo.

```
df[['Branch', 'tax_branch_mean', 'unit_price_mean', 'tax_branch_std',
    'unit_price_std', 'product_count', 'gender_count']].head(10)
```

FORA:

	Ramo	tax_branch_mean	unit_price_mean	tax_branch_std	unit_price_st
0	UMA	14.894393	54.937935	11.043263	26.203576
1	C	16.052367	56.645583	12.531470	27.247291
2	UMA	14.894393	54.937935	11.043263	26.203576
3	UMA	14.894393	54.937935	11.043263	26.203576
4	UMA	14.894393	54.937935	11.043263	26.203576
5	C	16.052367	56.645583	12.531470	27.247291
6	UMA	14.894393	54.937935	11.043263	26.203576
7	C	16.052367	56.645583	12.531470	27.247291
8	UMA	14.894393	54.937935	11.043263	26.203576
9	B	15.277808	55.743474	11.557958	26.136309

Observação: como estamos visualizando um subconjunto de colunas do full `df`, parece que há linhas duplicadas. Quando o restante das colunas estiver visível, você notará que não há linhas duplicadas, mas ainda há valores duplicados. Isso ocorre por design.

Escolhendo parâmetros de agregação numérica

Como escolhemos quais três parâmetros usar? Bem, isso dependerá do seu conhecimento de domínio e da sua compreensão do conjunto de dados. Por exemplo, neste conjunto de dados, se você sentir que a variação na **média** (tipo de agregação) **Classificação** (variável numérica) com base na **Filial** (coluna categórica) é importante para prever a **receita bruta** (variável de destino), crie o recurso! Se você acha que a contagem dos produtos na Linha de Produtos, por gênero, é importante para prever a receita bruta, codifique isso como um recurso!

Agora, se você puder testar quantas combinações dos três parâmetros - vá em frente - contanto que seja relativo ao selecionar apenas os recursos que têm poder preditivo suficiente, ou seja, certifique-se de ter um processo rigoroso de seleção de recursos.

Abaixo podemos ver algumas das colunas que criamos (`tax_branch_mean` e `unit_price_mean`). São agregações baseadas na `Branch` variável.

```
df[['Tax 5%', 'Unit price', 'Branch', 'tax_branch_mean', 'unit_price_mean']]
```

FORA:

	Imposto 5%	Preço unitário	Ramo	tax_branch_mean	unit_price_mean
0	26.1415	74.69	UMA	14.894393	54.937935
1	3.8200	15.28	C	16.052367	56.645583
2	16.2155	46.33	UMA	14.894393	54.937935
3	23.2880	58.22	UMA	14.894393	54.937935
4	30.2085	86.31	UMA	14.894393	54.937935
...	...	...	...	...	...
998	3.2910	65.82	UMA	14.894393	54.937935
999	30.9190	88.34	UMA	14.894393	54.937935
1000	30.9190	88.34	UMA	14.894393	54.937935
1001	5.8030	NaN	UMA	14.894393	54.937935
1002	30.4780	87.08	B	15.277808	55.743474

1003 linhas × 5 colunas

Mas por que tudo isso é necessário?

Agora, antes de prosseguir, você pode estar se perguntando por que isso é necessário - bons modelos não são projetados para levar em consideração todas essas agregações? Até certo ponto, sim, mas nem sempre. Depende muito do tamanho e dimensionalidade (número de colunas) do seu conjunto de dados. Quanto maior o conjunto de dados, mais recursos (em várias ordens de magnitude) você pode criar. Quando muitos recursos são criados, o modelo tem muitos sinais concorrentes para prever a variável de destino.

A engenharia de recursos tenta focar explicitamente a atenção do modelo em determinados recursos. Para resumir, a engenharia de recursos não é criar "novos" informações, mas sim direcionar o foco do modelo em certas informações que você, como cientista de dados, julga serem importantes.

Variáveis Indicadoras e Termos de Interação

Seguindo o mesmo padrão de pensamento das agregações numéricas, podemos construir variáveis indicadoras e termos de interação.

As variáveis indicadoras assumem apenas o valor 0 ou 1 para indicar a ausência ou presença de alguma informação.

Por exemplo, abaixo definimos uma variável indicadora `unit_price_50` para indicar se o produto tem preço unitário maior que 50. Para colocar em perspectiva, pense em um e-commerce com frete grátis em todos os pedidos acima de $ 50; isso pode ser uma informação útil para prever o comportamento do cliente e merece uma definição explícita para o modelo.

Os termos de interação são criados com base na presença de efeitos de interação entre duas ou mais variáveis. Isso é amplamente impulsionado pela experiência no domínio, embora existam testes estatísticos para ajudar a determiná-los (o que está além do escopo deste artigo). Por exemplo, embora o frete grátis possa afetar a classificação do cliente, o frete grátis combinado com a quantidade pode ter um efeito diferente na classificação do cliente, o que seria útil codificar (supondo que a classificação do cliente seja a variável de destino neste caso). Abaixo codificamos a variável `unit_price_50 * qty` para ser exatamente isso.

Usamos `np.where()` para criar uma variável indicadora `unit_price_50` que codifica 1 quando o preço unitário está acima de 50 e 0 caso contrário.

```
df['unit_price_50'] = np.where(df['Unit price'] > 50, 1, 0)
df['unit_price_50 * qty'] = df['unit_price_50'] * df['Quantity']
```

Aprenda ciência de dados com DATAQUEST

```
df[['unit_price_50', 'unit_price_50 * qty']].head()
```

FORA:

	unit_price_50	unit_price_50 * quantidade
0	1	7.0
1	0	0.0
2	1	7.0
3	1	8.0
4	1	7.0

Transformações Numéricas

Alguns conjuntos de dados não consideram que as transformações numéricas se enquadram na engenharia de recursos. Isso ocorre porque muitos modelos, especialmente os mais novos, como modelos baseados em árvore (árvores de decisão, florestas aleatórias, etc.), não são afetados por essas transformações. Em outras palavras, realizar essas transformações não melhora o desempenho preditivo. Mas para outros modelos, como a regressão linear, essas transformações podem fazer uma grande diferença, pois eles são sensíveis à escala de suas variáveis.

Abaixo, construímos uma nova variável `log_cogs` para corrigir a inclinação correta na variável `cogs`. O efeito é mostrado nos gráficos abaixo do trecho de código.

Também podemos fazer outras transformações, como elevar ao quadrado uma variável (mostrada no trecho de código abaixo) se acreditarmos que a relação entre um preditor e a variável de destino não é linear, mas de natureza quadrática (conforme uma variável preditora muda, a variável alvo muda em uma ordem de 2).

Podemos até ter variáveis cúbicas ou qualquer termo polinomial de n graus - fica a seu critério e conhecimento do domínio.

```
# Numeric transformations

df['log_cogs'] = np.log(df['cogs'] + 1)
df['gross income squared'] = np.square(df['gross income'])
```

Aprenda ciência de dados com DATAQUEST

```
df[['cogs', 'log_cogs', 'gross income', 'gross income squared']].head()
```

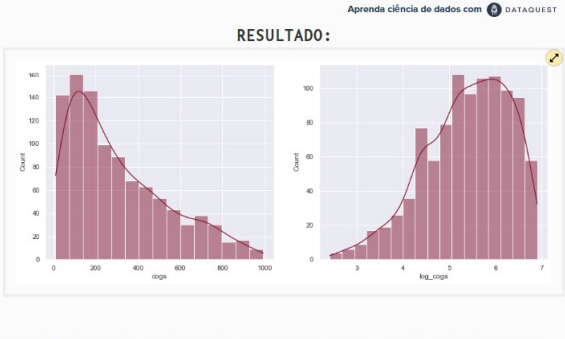
FORA:

	engrenagens	log_engrenagens	renda bruta	renda bruta ao quadrado
0	522.83	6.261167	26.1415	683.378022
1	76.40	4.348987	3.8200	14.592400
2	324.31	5.784779	16.2155	262.942440
3	465.76	6.145815	23.2880	542.330944
4	604.17	6.405509	30.2085	912.553472

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,6))

sns.histplot(df['cogs'], ax=ax1, kde=True)
sns.histplot(df['log_cogs'], ax=ax2, kde=True)
```

Aprenda ciência de dados com DATAQUEST

RESULTADO:



Como podemos ver, a transformação logarítmica tornou a distribuição do custo das mercadorias vendidas (cogs) mais normalmente distribuída (ou menos assimétrica à direita). Isso beneficiará modelos como regressão linear, pois seus pesos/coeficientes não serão fortemente influenciados por valores discrepantes que causaram a assimetria inicial.

Como um aparte, já que estaremos comparando gráficos um ao lado do outro várias vezes durante o artigo, usaremos apenas esta função auxiliar a partir de agora:

```
def plot_hist(data1, data2):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,6))
    sns.histplot(data1, ax=ax1, kde=True)
    sns.histplot(data2, ax=ax2, kde=True);
```

Aprenda ciência de dados com DATAQUEST

escala numérica

As colunas em um conjunto de dados geralmente estão em escalas diferentes. Em nosso conjunto de dados, por exemplo, 'receita bruta' e 'Rating' estão em escalas muito diferentes (como visto abaixo). Para corrigir isso, podemos executar a 'normalização' para colocar ambas as colunas em uma escala de 0-1.

porque nós fazemos isso? Quando as variáveis preditoras estão em escalas muito diferentes, modelos como regressão linear podem enviesar coeficientes para variáveis em uma escala maior. Portanto, corrigimos isso normalizando essas variáveis numéricas.

Podemos normalizar uma variável de várias maneiras, mas a maneira mais comum de fazer isso é usando o escalonador min-max (mostrado abaixo dos gráficos). A fórmula é mostrada abaixo - para cada valor na coluna, subtraímos o valor mínimo da coluna e dividimos o número resultante com o intervalo da coluna ($max - min$).

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

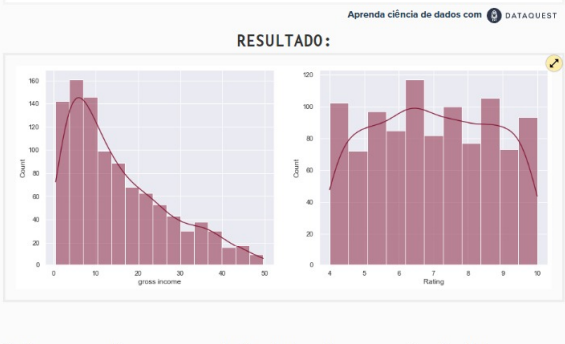
Podemos ver o intervalo de `gross income` e `Rating` atualmente em nosso conjunto de dados:

```
gincome = df["gross income"]
rating = df["Rating"]

print(f'Gross income range: {gincome.min()} to {gincome.max()}')
print(f'Rating range: {rating.min()} to {rating.max()}')

plot_hist(gincome, rating)
```

```
Gross income range: 0.5085 to 49.65
Rating range: 4.0 to 10.0
```

Aprenda ciência de dados com DATAQUEST

RESULTADO:

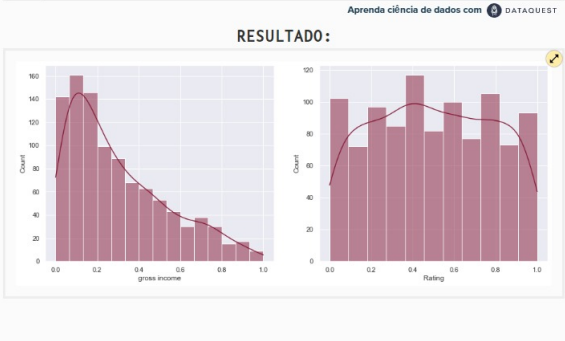


Podemos ver a diferença na escala depois de aplicar a normalização abaixo.

```
[["gross income", "Rating"]] = MinMaxScaler().fit_transform(df[["gross income", "Rating"]])

t_hist(df['gross income'], df['Rating'])
```

Aprenda ciência de dados com DATAQUEST

RESULTADO:



Observe que os gráficos parecem iguais, mas a escala no eixo x está entre 0 e 1 agora.

Tratamento de variáveis categóricas

Codificação única

Os modelos de aprendizado de máquina só podem lidar com variáveis numéricas. Portanto, devemos codificar variáveis categóricas como numéricas. A maneira mais fácil de fazer isso é "codificá-los com um hot-encode", o que significa que criamos variáveis indicadoras para uma coluna categórica com categorias. O código abaixo mostra como podemos codificar duas colunas categóricas - `Gender` e `Payment` .

```
pd.get_dummies(df[['Gender','Payment']]).head()
```

FORA:

	Gênero feminino	Gênero masculino	Pagamento_Dinheiro	Pagamento_Cartão de crédito	Payment_E
0	1	0	0	0	1
1	1	0	1	0	0
2	0	1	0	1	0
3	0	1	0	0	1
4	0	1	0	0	1

Mas há problemas com essa abordagem. Se tivermos uma coluna com 1.000 categorias, a codificação one-hot dessa coluna criará 1.000 novas colunas! Isso é muito! Você está alimentando o modelo com muita informação e, naturalmente, é muito mais difícil encontrar padrões. Quando temos dimensionalidade demais, nosso modelo levará muito mais tempo para treinar e encontrar os pesos preditores ideais.

Codificação de destino

Para resolver isso, podemos usar a codificação de destino. A codificação de destino não cria colunas adicionais. A ideia é simples - para cada categoria exclusiva, o valor médio da variável de destino (supondo que seja contínua ou binária) é calculado e se torna o valor da respectiva categoria na coluna categórica.

Vejamos um exemplo simples antes de aplicá-lo ao nosso conjunto de dados. Temos duas colunas - o alvo e a variável preditora. **Nosso objetivo é codificar a variável preditora (uma coluna categórica) em uma variável numérica que pode ser usada pelo modelo. Para fazer isso, simplesmente agrupamos pela variável preditora para obter o valor alvo médio para cada categoria preditora.** Portanto, para o preditor `a`, o valor codificado será a média de 1 e 5, que é 3. Pois `b` é a média de 4 e 6, que é 5. Agora, nossa coluna categórica é uma coluna numérica!

```
target = [1, 4, 5, 6]
predictor = ['a', 'b', 'a', 'b']

target_enc_df = pd.DataFrame(data={'target':target, 'predictor':predictor})

means = target_enc_df.groupby('predictor')['target'].mean()

target_enc_df['predictor_encoded'] = target_enc_df['predictor'].map(means)

target_enc_df
```

FORA:

	alvo	preditor	previsor_encoded
0	1	uma	3
1	4	b	5
2	5	uma	3
3	6	b	5

Em seguida, usamos a codificação de destino em nosso conjunto de dados de supermercado. Para o exemplo abaixo, usamos `Product line` como a coluna categórica que é codificada pelo destino e `Rating` é a variável de destino, que é uma variável contínua.

```
means = df.groupby('Product line')['Rating'].mean()

df['Product line target encoded'] = df['Product line'].map(means)
df[['Product line','Product line target encoded','Rating']]
```

FORA:

	Linha de produto	Meta de linha de produto codificada	Avaliação
0	Saúde e beleza	0,506134	0,850000
1	acessórios eletrônicos	0,481313	0,933333
2	Casa e estilo de vida	0,479139	0,566667
3	Saúde e beleza	0,506134	0,733333
4	esportes e viagens	0,484151	0,216667
...	...	...	...
998	Casa e estilo de vida	0,479139	0,766667
999	Acessórios fashions	0,514341	0,433333
1000	Acessórios fashions	0,514341	0,433333
1001	acessórios eletrônicos	0,481313	0,800000
1002	acessórios eletrônicos	0,481313	0,250000

1003 linhas × 3 colunas

A codificação de destino tem suas desvantagens - quando uma categoria aparece apenas uma vez, o valor médio dessa categoria é o próprio valor da média de um número é o próprio número). Em geral, nem sempre é uma boa ideia confiar em uma média quando o número de valores usados na média é baixo. Isso leva a problemas com a generalização dos resultados no conjunto de dados de treinamento para o conjunto de dados de teste ou dados nos quais o modelo não é treinado.

A conclusão desta seção é tentar a codificação one-hot se a dimensionalidade não for um problema. Se for um problema, você pode usar outras abordagens, como codificação de destino.

Tratamento de valores ausentes

A modelagem preditiva pode ser considerada como a extração dos sinais corretos de um conjunto de dados. Os valores ausentes podem ser uma fonte de sinal (quando os valores não estão ausentes aleatoriamente) ou podem ser uma ausência de sinal (quando os valores estão ausentes aleatoriamente).

Observação

Nota: os dados foram modificados para conter valores ausentes para que pudéssemos discutir este tópico. Se você obtiver uma nova cópia do Kaggle, ela não deve ter nenhum valor ausente.

Por exemplo, digamos que temos alguns dados populacionais e adicionamos uma coluna chamada `has_license` indicando se uma pessoa tem carteira de motorista ou não. Notaremos valores ausentes - uma quantidade desproporcional deles sendo pessoas com menos de 18 anos. Este é um caso em que os valores NÃO estão ausentes aleatoriamente. Agora, se tivermos alguns valores ausentes na `Gender` coluna causados por problemas de entrada de dados, esses valores provavelmente estarão ausentes aleatoriamente.

Por que isso é importante? Se houver dados ausentes que não são aleatórios, sabemos por que os valores estão ausentes e isso pode ser explicado pelo conjunto de dados, podemos simplesmente codificar isso como uma variável indicadora. Isso permitirá que o modelo descubra facilmente. Se houver uma explicação de por que eles estão faltando não for explicada pelo conjunto de dados, então estamos em território obscuro e o tratamento do tal caso requer atenção mais avançada.

Quando faltam dados aleatoriamente, temos uma perda de informações, as esperamos poder preencher essas lacunas com base nas informações de outros recursos.

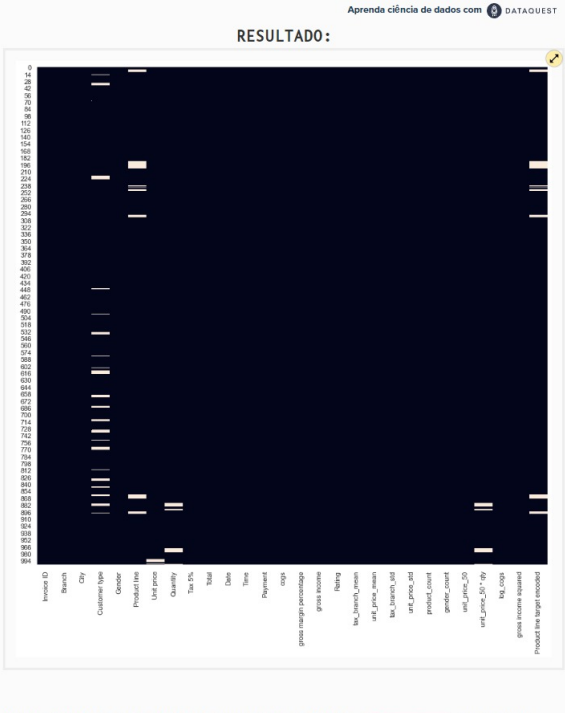
O mínimo que podemos fazer é remover as linhas com dados ausentes, pois a maioria dos modelos não lida com dados ausentes. Como as colunas com muitos valores ausentes geralmente não fornecem um sinal útil, podemos removê-las com base em uma condição de limite para ausência (mostrada abaixo).

Mas antes de preenchermos os valores ausentes, pode ser útil primeiro visualizar os valores ausentes usando o Seaborn.

```
plt.figure(figsize=(15, 15))
sns.heatmap(df.isnull(), cbar=False);
```

Aprenda ciência de dados com DATAQUEST

RESULTADO:



Vemos que há valores ausentes em algumas colunas - `Customer type`, uma coluna categórica, com o maior número de valores ausentes. Normalmente, colunas com muitos valores ausentes não fornecem sinal suficiente para previsão - então alguns profissionais decidem remover essas colunas definindo um limite para "ausência". No trecho de código abaixo, definimos o limite como 70% e removemos as colunas e linhas que atendem a essas condições.

Eu não recomendo essa estratégia - ainda pode haver informações úteis nessas colunas/linhas e eu deixaria o processo de seleção de recursos decidir se deve ou não manter/remover colunas. Independentemente disso, se você simplesmente deseja criar um modelo de linha de base rápido, pode empregar essa estratégia.

Veja como você pode remover valores ausentes para um determinado limite:

```
threshold = 0.7

# Dropping columns with missing value rate higher than threshold
df = df[df.columns[df.isnull().mean() < threshold]]

# Dropping rows with missing value rate higher than threshold
df = df.loc[df.isnull().mean(axis=1) < threshold]
```

Aprenda ciência de dados com DATAQUEST

Alternativamente (de preferência), podemos **imputar** valores ausentes com um único valor, como a média ou mediana da coluna. Para colunas categóricas, podemos imputar valores ausentes com a moda ou categoria mais frequente na coluna.

```
# Filling missing values with medians of the columns
df = df.fillna(df.median())

# Fill remaining columns - categorical columns - with mode
df = df.apply(lambda x:x.fillna(x.value_counts().index[0]))
```

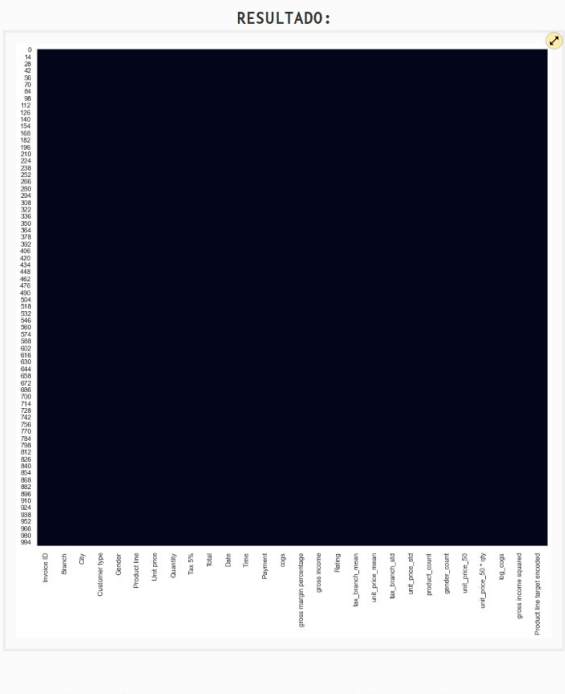
Aprenda ciência de dados com DATAQUEST

Agora não vemos mais valores ausentes no conjunto de dados!

```
plt.figure(figsize=(15, 15))
sns.heatmap(df.isnull(), cbar=False);
```

Aprenda ciência de dados com DATAQUEST

RESULTADO:



Existem técnicas de imputação mais complicadas além do escopo deste artigo, mas isso deve ser suficiente para você começar. Se estiver interessado em uma exploração mais aprofundada no tratamento de dados ausentes, recomendo fortemente a leitura de **Missing Data** , de Paul D. Allison.

Decomposição Data-Hora

A decomposição data-hora é simplesmente quebrar uma variável de data em seus constituintes. Fazemos isso porque o modelo precisa trabalhar com variáveis numéricas.

```
# Convert to datetime object
df['Date'] = pd.to_datetime(df['Date'])
df[['Date']].head()
```

FORA:

	Encontro
0	2019-01-05
1	2019-03-08
2	2019-03-03
3	2019-01-27
4	2019-02-08

```
# Decomposition
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
df[['Year','Month','Day']].head()
```

FORA:

	Ano	Mês	Dia
0	2019	1	5
1	2019	3	8
2	2019	3	3
3	2019	1	27
4	2019	2	8

O que acabamos de fazer é separar a coluna de data que estava no formato "ano-mês-dia" em colunas individuais, ou seja, ano, mês e dia. Essas são informações que o modelo agora pode usar para fazer previsões, pois as novas colunas são numéricas.

Abordagem baseada em domínio

Não há um limite estrito entre abordagens baseadas em domínio e baseadas em lista de verificação para a engenharia de recursos. A distinção, eu diria, é bastante subjetiva - com recursos baseados em domínio, você ainda aplica muitas das técnicas que já discutimos, mas com grande ênfase no conhecimento do domínio.

Os recursos baseados em domínio envolverão muitas métricas ad hoc, como proporções, fórmulas. etc. Veremos exemplos disso no exemplo de estudo de caso abaixo.

Exemplo de estudo de caso - dados de bilheteria de filmes

Agora que aprendemos várias técnicas de engenharia de recursos, vamos aplicá-las!

Para nosso estudo de caso, trabalharemos com dados de bilheteria de filmes. Você pode encontrar mais informações sobre o conjunto de dados clicando **aqui** .

Normalmente, nosso primeiro passo seria realizar uma análise exploratória de dados no conjunto de dados, mas como este é um artigo sobre engenharia de recursos, vamos nos concentrar nisso. Observe que muitas das ideias para engenharia de recursos mostradas abaixo foram inspiradas por um kernel Kaggle vinculado **aqui** .

```
df = pd.read_csv('data/movies.csv')
df.head()
```

FORA:

	Eu iria	pertence_à_coleção	orçamento	gêneros	pagina inicial
0	1	[{'id': 313576, 'name': 'Hot Tub Time Machine ...	14000000	[{'id': 35, 'nome': 'Comédia'}]	NaN
1	2	[{'id': 107674, 'name': 'O Diário da Princesa...	40000000	[{'id': 35, 'name': 'Comédia'}, {'id': 18, 'nam...	NaN
2	3	NaN	3300000	[{'id': 18, 'nome': 'Drama'}]	http://sonyclassics.com/whip
3	4	NaN	1200000	[{'id': 53, 'name': 'Thriller'}, {'id': 18, 'n...	http://kahaanithefilm.com/
4	5	NaN	0	[{'id': 28, 'nome': 'Ação'}, {'id': 53, 'nome...	NaN

5 linhas × 23 colunas

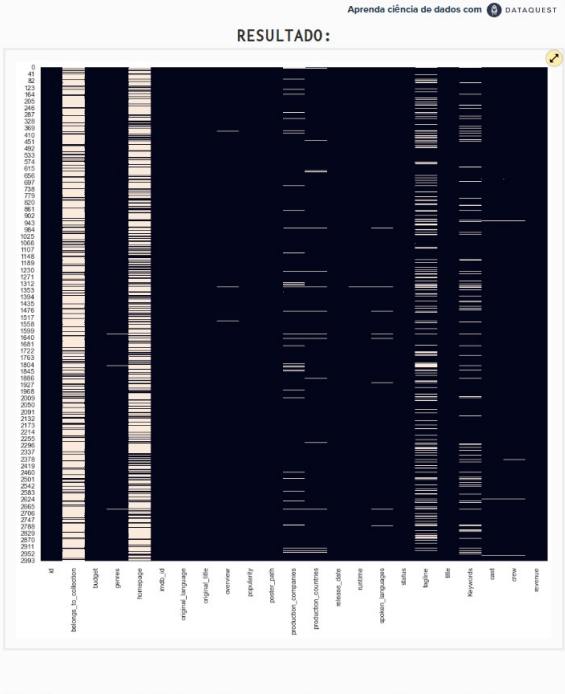
Preenchendo valores ausentes

Primeiro, vamos lidar com os valores ausentes. Nós os visualizamos usando Seaborn e, em seguida, preenchemos os valores numéricos ausentes com a mediana e os valores categóricos ausentes com a moda.

```
plt.figure(figsize=(15, 15))
sns.heatmap(df.isnull(), cbar=False);
```

Aprenda ciência de dados com DATAQUEST

RESULTADO:



Preencheremos as variáveis numéricas ausentes com a mediana e a coluna categórica com a moda. Abordaremos os valores categóricos ausentes depois que terminarmos a engenharia de recursos de outras colunas (bem no final).

Não há uma ciência difícil para escolher qual abordagem de imputação de valor ausente você adota. A maioria dos praticantes testa várias técnicas de imputação de valores ausentes e decide aquela que obtém a melhor pontuação de avaliação.

Data de Decomposição

E agora podemos decompor a coluna de data em seus atributos. Observe que codificamos mês e dia como variáveis de string, pois não há uma relação numérica entre elas. Dias e meses têm limites fixos (o mês não ultrapassa 12, o dia não ultrapassa 31). Os dias 10 e 31 são simplesmente dias diferentes (pense neles como categorias).

Vamos colocar ano, mês e dia em suas próprias colunas no dataframe:

```
df['release_date'] = pd.to_datetime(df['release_date'])

# decomposition
df['Year'] = df['release_date'].dt.year
df['Month'] = df['release_date'].dt.month.astype(str)
df['Day'] = df['release_date'].dt.day.astype(str)

df[['Year','Month','Day']].head()
```

FORA:

	Ano	Mês	Dia
0	2015	2	20
1	2004	8	6
2	2014	10	10
3	2012	3	9
4	2009	2	5

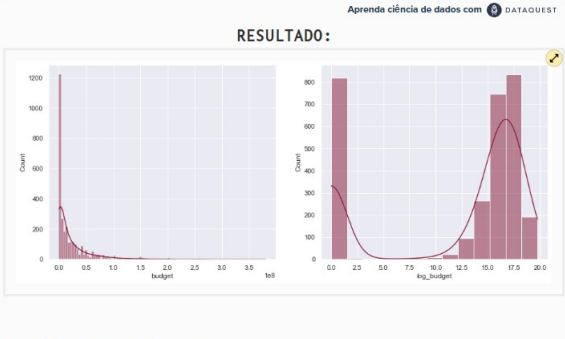
Ajustando o orçamento

Como o orçamento é altamente assimétrico à direita, usamos o logaritmo do orçamento para ajustá-lo. Observe que usamos o logaritmo de `budget + 1` porque muitos filmes têm um orçamento de $ 0 e não podemos usar o logaritmo de 0.

```
df['log_budget'] = np.log(df['budget'] + 1)

plot_hist(df['budget'], df['log_budget'])
```

Aprenda ciência de dados com DATAQUEST

RESULTADO:



Codificação da inflação

Sabemos que o orçamento aumenta anualmente em certa medida devido à inflação. Podemos codificar isso usando uma fórmula de inflação simples da seguinte forma:

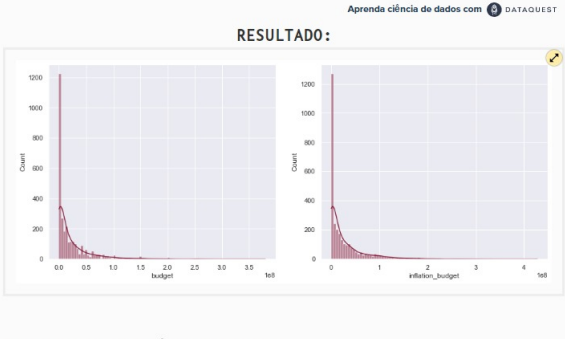
$$InflationBudget_i = Budget_i(1 + \frac{1.8}{100} \times (MaxYear - Year_i))$$

Ondeé cada linha e $MaxYear$ é o ano máximo do conjunto de dados (2018 no nosso caso). Aqui está criando-o para o nosso dataframe:

```
df['inflation_budget'] = df['budget'] * (1 + (1.8 / 100) * (2018 - df['Year']))

plot_hist(df['budget'], df['inflation_budget'])
```

Aprenda ciência de dados com DATAQUEST

RESULTADO:



Outros recursos interessantes

Com base no conhecimento do domínio, podemos criar algumas variáveis de proporção úteis, conforme mostrado abaixo.

```
df['budget_runtime_ratio'] = df['budget'] / df['runtime']

df['budget_popularity_ratio'] = df['budget'] / df['popularity']

df['budget_year_ratio'] = df['budget'] / (df['Year'] * df['Year'])

df['releaseYear_popularity_ratio'] = df['Year'] / df['popularity']
```

Aprenda ciência de dados com DATAQUEST

variáveis indicadoras

Codificamos uma variável indicadora que indica se um filme tem uma página inicial ou não e se o filme está em inglês:

```
# Has a homepage
df['has_homepage'] = 1
df.loc[pd.isnull(df['homepage']), "has_homepage"] = 0

# Was in English
df['is_english'] = np.where(df['original_language']=='en', 1, 0)
```

Aprenda ciência de dados com DATAQUEST

E agora podemos preencher os valores ausentes da coluna categórica.

```
# Fill remaining columns - categorical columns - with mode
df = df.apply(lambda x: x.fillna(x.value_counts().index[0]))
```

Aprenda ciência de dados com DATAQUEST

Subconfiguramos o quadro de dados para incluir apenas as variáveis que desejamos.

```
engineered_df = df[['budget_runtime_ratio',
                    'budget_popularity_ratio',
                    'budget_year_ratio',
                    'releaseYear_popularity_ratio',
                    'inflationBudget',
                    'Year',
                    'Month',
                    'is_english',
                    'has_homepage',
                    'budget',
                    'popularity',
                    'runtime',
                    'revenue']]
```

Aprenda ciência de dados com DATAQUEST

Codificamos one-hot as colunas categóricas. No nosso caso, temos apenas uma coluna categórica - mês.

```
engineered_df = engineered_df.replace([np.inf, -np.inf], np.nan).dropna(axis=1)
engineered_df = pd.get_dummies(engineered_df)
```

Aprenda ciência de dados com DATAQUEST

Nosso novo quadro de dados se parece com isso

```
engineered_df.head()
```

FORA:

	budget_popularity_ratio	orçamento_ano_ratio	releaseYear_popularity_ratio
0	2.129150e+06	3.448085	306.445562
1	4.849134e+06	9.960120	242.949630
2	5.132194e+04	0.813570	31.321933
3	3.779604e+05	0.296432	633.713561
4	0.000000e+00	0.000000	1749.893299

5 linhas × 23 colunas

Agora que nosso conjunto de dados está pronto, podemos passar pelo processo de seleção de recursos úteis (seleção de recursos) e fazer previsões.

Predição

Para provar que a engenharia de recursos funciona e melhora o desempenho do modelo, podemos construir um modelo de regressão simples para prever a receita dos filmes.

Normalmente, escolhemos quais recursos usar por meio de um processo chamado seleção de recursos; no entanto, como este artigo é focado na engenharia de recursos, empregaremos um processo simples de seleção de recursos: **análise de correlação** .

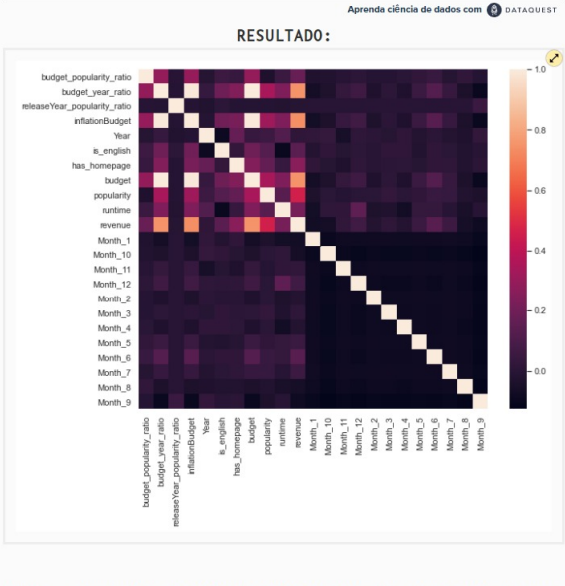
Ao plotar a matriz de correlação (abaixo), vemos que a maioria dos recursos que criamos não são tão preditivos de receita. Isso é o que acontece na maioria das vezes - você cria uma tonelada de recursos, mas apenas alguns acabam sendo úteis - mas os recursos que são úteis fazem a diferença.

No gráfico abaixo, usaremos `has_homepage` , `budget_year_ratio` , e `is_english` em nosso modelo, além de recursos que vieram antes da engenharia de recursos - `budget` , `runtime` e `popularity` .

```
plt.figure(figsize=(10, 8))
sns.heatmap(engineered_df.corr());
```

Aprenda ciência de dados com DATAQUEST

RESULTADO:



Usaremos aproximadamente 80% do conjunto de dados para treinar a linha de base e os modelos de engenharia de recursos e comparar seus desempenhos no conjunto de teste oculto.

```
train_engineered = engineered_df[['budget','runtime','popularity',
                                  'has_homepage','budget_year_ratio','is_english']].iloc[

train_baseline = engineered_df[['budget','runtime','popularity']].iloc[:2500]

test_engineered = engineered_df[['budget','runtime','popularity',
                                 'has_homepage','budget_year_ratio','is_english']].iloc[

test_baseline = engineered_df[['budget','runtime','popularity']].iloc[2500:]

target_train = engineered_df['revenue'].iloc[:2500]
target_test = engineered_df['revenue'].iloc[2500:]
```

Aprenda ciência de dados com DATAQUEST

```
reg_baseline = LinearRegression().fit(train_baseline, target_train)
reg_predict_baseline = reg_baseline.predict(test_baseline)

reg_engineered = LinearRegression().fit(train_engineered, target_train)
reg_predict_engineered = reg_engineered.predict(test_engineered)

rmse_baseline = np.sqrt(mean_squared_error(target_test, reg_predict_baseline))
rmse_engineered = np.sqrt(mean_squared_error(target_test, reg_predict_engineered))

rmse_difference = rmse_baseline - rmse_engineered

print ("The difference in RMSE is", round(rmse_difference, 3), "dollars")
```

```
The difference in RMSE is 909146.83 dollars
```

Aprenda ciência de dados com DATAQUEST

A diferença é bem gritante! As previsões do modelo de linha de base - que usa apenas `budget` , `runtime` e `popularity` como recursos - são, em média, $ 909.146,83 piores do que o modelo em que usamos nossos recursos construídos. Chegamos a essa conclusão comparando a raiz do erro quadrático médio (RMSE) de ambos os modelos no conjunto de teste.

Ao usar a engenharia de recursos, permitimos que nosso modelo entendesse melhor nosso conjunto de dados e, portanto, fizesse previsões melhores.

Conclusão

Armadilhas da engenharia de recursos

Algumas das armadilhas comuns da engenharia de recursos são:

1. **Overfitting:** Quando construímos muitos recursos, corremos o risco de superajustar os dados. Isso é muitas vezes referido como a *maldição da dimensionalidade* . Resumidamente, quanto mais recursos um modelo tiver, mais flexibilidade ele terá para estabelecer relacionamentos entre o preditor e a variável de destino. Isso pode parecer uma coisa boa, mas se um modelo tiver muita flexibilidade, ele irá, de certa forma, otimizar demais os dados nos quais é treinado. Isso resultará em uma pontuação de alto desempenho, mas terá um desempenho ruim em dados ocultos ou novos dados, pois os modelos terão diferenças não observadas nos dados de treinamento. Devemos estar atentos a isso e levar em consideração os testes de amostra ao avaliar os recursos (durante a etapa de seleção de recursos).
2. **Vazamento de informações:** se a engenharia de recursos não for feita corretamente, isso pode levar ao vazamento de informações. Isso geralmente envolve a construção de novos recursos usando a variável de destino. A engenharia de recursos sempre deve ser feita independentemente da variável de destino e deve incluir apenas variáveis preditoras de recursos.

A mentalidade de engenharia de recursos

A mentalidade de engenharia de recursos é muito experimental. Geralmente, a quantidade é mais valorizada do que a qualidade. A qualidade entra em jogo quando lidamos com a seleção de recursos após a engenharia de recursos. Podemos ter alguma orientação sobre quais recursos podem ser úteis, mas não devemos deixar que nosso viés entre em jogo - construa o máximo possível de recursos relevantes a partir de seus dados (computação e tempo, é claro) e siga-o com um recurso robusto processo de seleção para eliminar características ruins.

APRENDIZADO ADICIONAL:

Muitos cursos de ciência de dados e aprendizado de máquina contêm seções sobre engenharia de recursos. Aqui está um ótimo curso para conferir:

Engenharia de recursos com o Google Cloud Aprenda sobre a engenharia de recursos no contexto do Tensorflow e do Google Cloud Platform

Receba as atualizações em sua caixa de entrada
Junte-se a mais de 7.500 alunos de ciência de dados.
Digite seu e-mail
Se inscrever

Conheça os Autores

Bassim Eledath
cientista de dados
Bassim é Cientista de Dados na NoviSci, onde ajuda a resolver problemas difíceis de epidemiologia usando várias ferramentas estatísticas.

Editor: Brendan Martin
Fundador da LearnDataSci

Voltar ao índice do blog

Carregar comentários

© 2022 LearnDataSci. Todos os direitos reservados.

O uso e/ou registro em qualquer parte deste site constitui a aceitação de nossa Política de Privacidade . O material deste site não pode ser reproduzido, distribuído, transmitido, armazenado em cache ou usado de outra forma, exceto com a permissão prévia por escrito da LearnDataSci.com.