SEMANA DO PYTHON DA HASHTAG

Apostila Completa
Aula 2

Aprenda como fazer uma análise de dados
que vai deixar seu chefe impressionado!

Impressionador do absoluto zero!

Parte 1

Introdução

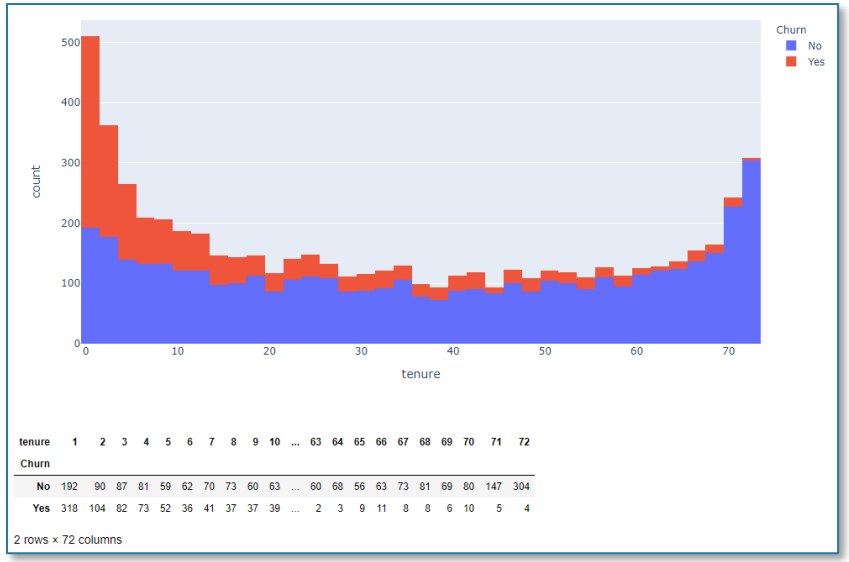
O que vamos aprender

Na segunda aula do Intensivão do Python você vai aprender a criar um código de análise de dados. No dia a dia das empresas, é muito comum dúvidas sobre os resultados da empresa. Um conceito que cada dia mais cresce nas empresas é o *data driven*. Basicamente, é dizer que ações são tomadas com base nos dados e não em achismos.
Aprenda como **fazer uma super análise do zero** com os conceitos abaixo:

- Importando dados de bases .csv
- Tratar dados usando a biblioteca Pandas
- Importação de bibliotecas
- Criação de gráficos usando o plotly

Após todos esses conhecimentos, seremos capazes de transformar uma tabela cheia de informações, nem um pouco fáceis de serem interpretadas ...

... em uma análise super aprofundada que servirão de base para tomada de decisão da gerência. Tudo graças a você! ☺
Prepare-se para **ver além do óbvio.**



tenure	1	2	3	4	5	6	7	8	9	10	...	63	64	65	66	67	68	69	70	71	72
Churn																					
No	192	90	87	81	59	62	70	73	60	63	...	60	68	56	63	73	81	69	80	147	304
Yes	318	104	82	73	52	36	41	37	37	39	...	2	3	9	11	8	8	6	10	5	4

2 rows × 72 columns

Entendendo a base de dados

As informações que vão alimentar nossa análise, foram extraídas do site Kaggle(link). Os dados são referentes a clientes serviços de telecomunicações e seus hábitos de consumo, produtos, etc.

A imagem ao lado, mostra os dados extraídos em modelo .csv. Como podemos ver, os dados não estão formatados o que nos dificulta um pouco entender corretamente o que temos aqui...

A situação:

Analisando o histórico dos clientes dos últimos anos, você percebeu que mais de 26% dos clientes que entraram na empresa, cancelaram o contrato.

A única informação que você tem é um arquivo .csv extraído do sistema da empresa (apresentado ao lado).

Entendendo a solução final

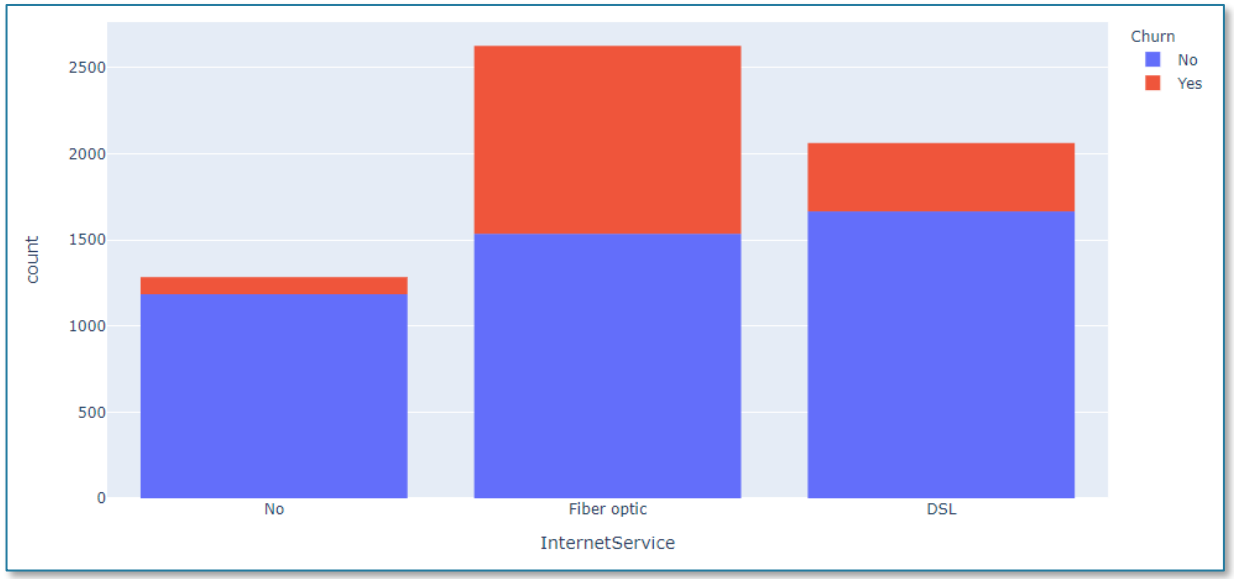
Nesse caso, a solução final podem ser diversas!

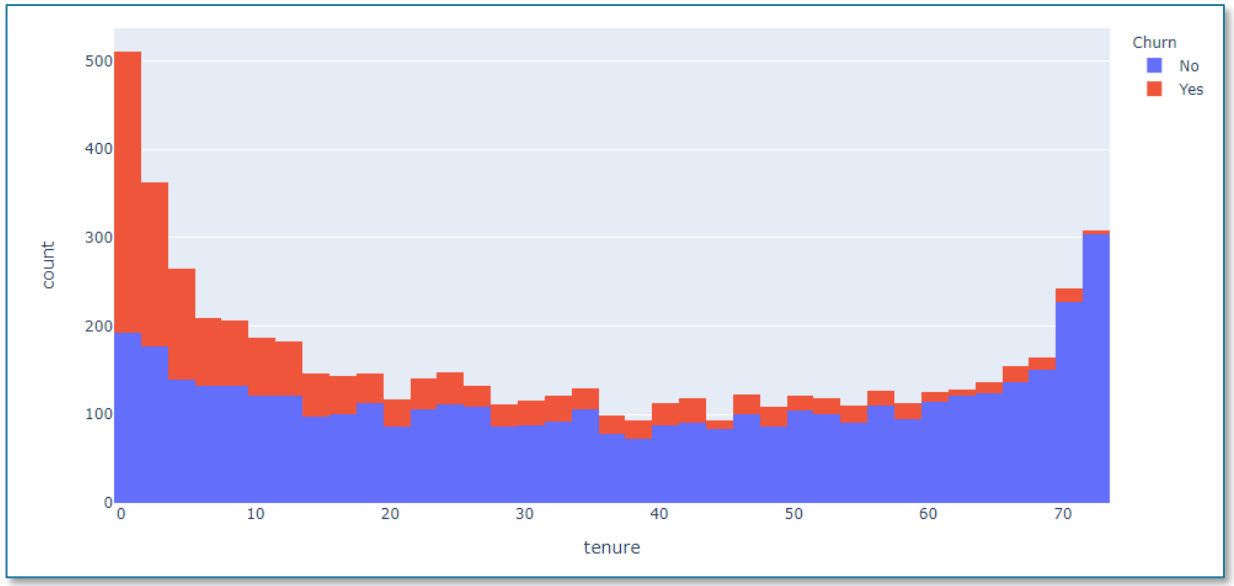
Estamos aqui tratando de análise de dados. Boa parte da solução aqui é não conhecida nesse estágio.

Quando encontramos esses casos é muito importante, mais até do que a solução em um primeiro momento, definir qual o problema.

Na aula e na apostila vamos fazer uma análise exploratória de dados e buscar direcionadores de causas raíz que podem ser atacadas visando o maior retorno com o menor esforço.

Para isso utilizaremos o Python para nos ajudar em análises gráficas dos dados como este aqui do lado ☺





Parte 2

Importando e visualizando os dados

INTENSIVÃO DE PYTHON {#}
100% ONLINE & GRATUITO

Importando base de dados (1/2)

Como vimos na aula 1 do Intensivão, vamos usar bibliotecas que nos facilitem importar dados de planilhas Excel, arquivos .csv, etc.

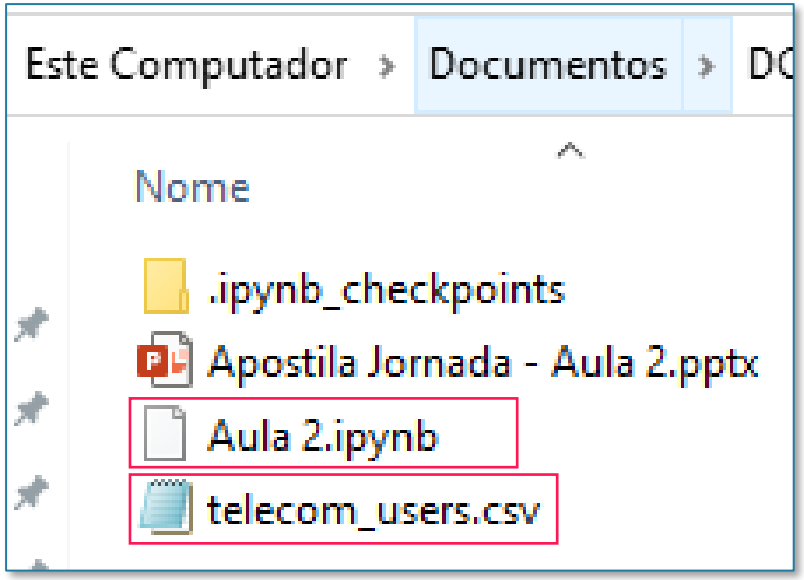
Novamente usaremos o PANDAS. Caso você não saiba do que estamos falando aqui, dá uma olhadinha na apostila da Aula 1 do intensivão!! Lá a gente explica o que são bibliotecas e para que servem ☺.

Vamos começar importando o PANDAS como pd.

Feito isso, precisamos agora buscar o arquivo no nosso pc.

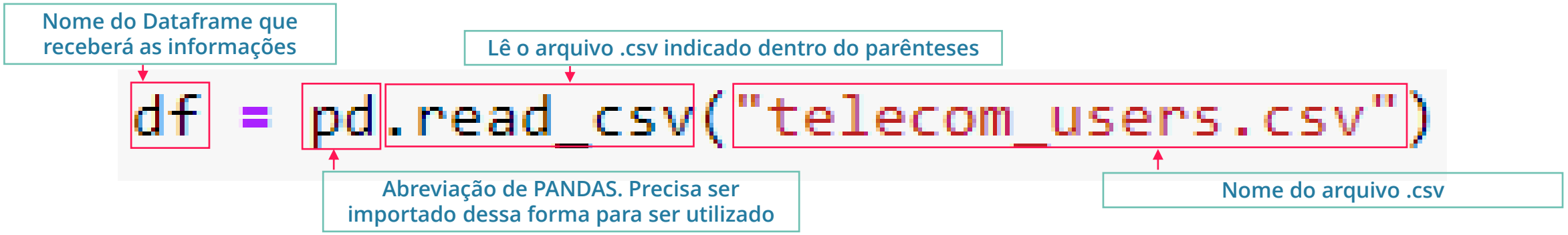
IMPORTANTE: Usaremos nessa apostila o arquivo .csv na MESMA PASTA do nosso notebook.

```
import pandas as pd
```



Importando base de dados (2/2)

Se você acompanhou a aula 1 do Intensivão, vai lembrar que lá usamos a função read_excel() do Pandas. Aqui, temos uma diferença. Como se trata de um arquivo .csv, precisamos usar a **fórmula .read_csv()** conforme apresentado abaixo:



Até aqui, nada muito diferente...

Vamos para a próxima parte ☺

Priorizando os dados importados (1/2)

Cada vez mais, saber que dados são úteis ou não é algo fundamental no dia a dia do trabalho.

É muito comum, termos bases de dados ENORMES extraídas de sistemas.

Saber **separar o que é útil do que não é**, é fundamental para uma boa análise de dados.

Ao usarmos **display(df)** conseguimos avaliar um pouco melhor nossa base de dados. No entanto, uma das colunas parece não nos ajudar muito na nossa análise...

Consegue identifica-la? A coluna **Unnamed:0,** além de não ter um rótulo que nos ajude a entender de que se trata, possui números que não seguem nenhum tipo de padrão. Portanto, pelo menos por enquanto, não nos é interessante pois é uma informação irrelevante para nosso estudo.

Sendo assim, podemos retirá-la da nossa tabela. Vamos ver como, a seguir.

A coluna **Unnamed:0** não nos é relevante.
Podemos retirá-la de nossa base para aumentar a eficiência do código.

```
display(df)
```

	Unnamed: 0	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
0	1869	7010-BRBUU	Male	0	Yes	Yes	72	Yes	Yes	No
1	4528	9688-YGXVR	Female	0	No	No	44	Yes	No	Fiber optic
2	6344	9286-DOJGF	Female	1	Yes	No	38	Yes	Yes	Fiber optic
3	6739	6994-KERXL	Male	0	No	No	4	Yes	No	DSL
4	432	2181-UAESM	Male	0	No	No	2	Yes	No	DSL
...	...	...	...	...	...	...	...	...	...	...
5981	3772	0684-AOSIH	Male	0	Yes	No	1	Yes	No	Fiber optic
5982	5191	5982-PSMKW	Female	0	Yes	Yes	23	Yes	Yes	DSL
5983	5226	8044-BGWPI	Male	0	Yes	Yes	12	Yes	No	No
5984	5390	7450-NWRTR	Male	1	No	No	12	Yes	Yes	Fiber optic
5985	860	4795-UXVCJ	Male	0	No	No	26	Yes	No	No

5986 rows × 23 columns

Priorizando os dados importados (2/2)

Para retirarmos a coluna Unnamed:0, vamos usar o método abaixo:

.drop()

Este método será aplicado na variável clientes_df criada na primeira linha do código para receber os dados do arquivo .csv.

Este método necessitará de alguns argumentos:

- Nome da coluna ou código da linha a ser removida: ('Unnamed:0')
- Qual dos eixos deve ser excluído:
  - 0 ou 'index' será apagada a linha indicada;
  - 1 ou 'columns' será apagada a coluna indicada.

```
import pandas as pd
df = pd.read_csv("telecom_users.csv")
df = df.drop(["Unnamed: 0"], axis=1)
display(df)
```

	Unnamed: 0	customerID	gender	SeniorCitizen
0	1869	7010-BRBUU	Male	0
1	4528	9688-YGXVR	Female	0
2	344	9286-DOJGF	Female	1
3	6739	6994-KERXL	Male	0
4	432	2181-UAESM	Male	0
...	...	...	...	...
5981	3772	0684-AOSIH	Male	0

	customerID	gender	SeniorCitizen
0	7010-BRBUU	Male	0
1	9688-YGXVR	Female	0
2	9286-DOJGF	Female	1
3	6994-KERXL	Male	0
4	2181-UAESM	Male	0
...	...	...	...
5981	0684-AOSIH	Male	0

Coluna Unnamed:0 retirada

Visualizar a base de dados importada

Bem, já importamos nossa base de dados...

Agora vamos tentar visualizá-la !

Usamos a função **DISPLAY()** para exibir nossos dados coletados.

Lembra como era difícil entender os dados quando estavam em .csv?

Com o pandas essa visualização fica bem mais amigável e prática.

Perceba também que já excluímos a coluna Unnamed:0 que não queríamos.

IMPORTANTE: A base original, NÃO foi afetada.

```
import pandas as pd
df = pd.read_csv("telecom_users.csv")
df = df.drop(["Unnamed: 0"], axis=1)
display(df)
```

Função display que apresenta os dados armazenados na variável df

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	Onl
0	7010-BRBUU	Male	0	Yes	Yes	72	Yes	Yes	No	
1	9688-YGXVR	Female	0	No	No	44	Yes	No	Fiber optic	
2	9286-DOJGF	Female	1	Yes	No	38	Yes	Yes	Fiber optic	
3	6994-KERXL	Male	0	No	No	4	Yes	No	DSL	
4	2181-UAESM	Male	0	No	No	2	Yes	No	DSL	
...	...	...	...	...	...	...	...	...	...	
5981	0684-AOSIH	Male	0	Yes	No	1	Yes	No	Fiber optic	
5982	5982-PSMKW	Female	0	Yes	Yes	23	Yes	Yes	DSL	
5983	8044-BGWPI	Male	0	Yes	Yes	12	Yes	No	No	
5984	7450-NWRTR	Male	1	No	No	12	Yes	Yes	Fiber optic	
5985	4795-UXVCJ	Male	0	No	No	26	Yes	No	No	

5986 rows × 22 columns

5986 linhas
22 colunas

Parte 3

Tratamento e visão geral dos dados

INTENSIVÃO DE PYTHON
100% ONLINE & GRATUITO

Limpando a base de dados (1/4)

É muito comum que base de dados extraídas de sistemas possuam dados faltantes e/ou dados que não são corretos.

Todos esses dados influenciam diretamente nos resultados obtidos na nossa análise.

Imagine um **caso genérico** em que preciso calcular a quantos dos meus clientes possuem conta digital. Caso existam dados faltantes na coluna PaperlessBilling, meu cálculo de porcentagem será afetado diretamente.

Assim, é sempre importante antes de qualquer análise avaliar se precisamos tratar esta base de dados ou não.

Analisando nossa base conseguimos perceber que temos 2 problemas:

- Dados faltantes na coluna Código;
- Coluna **TotalCharges** está classificada como **OBJECT** e por se tratar de números, deveriam ser classificadas como tal.

```
print(df.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5986 entries, 0 to 5985
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        5986 non-null   object
 1   gender            5986 non-null   object
 2   SeniorCitizen     5986 non-null   int64
 3   Partner           5986 non-null   object
 4   Dependents        5985 non-null   object
 5   tenure            5986 non-null   int64
 6   PhoneService      5986 non-null   object
 7   MultipleLines     5986 non-null   object
 8   InternetService   5986 non-null   object
 9   OnlineSecurity    5986 non-null   object
 10  OnlineBackup      5986 non-null   object
 11  DeviceProtection  5986 non-null   object
 12  TechSupport       5986 non-null   object
 13  StreamingTV       5986 non-null   object
 14  StreamingMovies   5986 non-null   object
 15  Contract          5986 non-null   object
 16  PaperlessBilling  5986 non-null   object
 17  PaymentMethod     5986 non-null   object
 18  MonthlyCharges    5986 non-null   float64
 19  TotalCharges      5986 non-null   object
 20  Churn             5985 non-null   object
 21  Codigo            0 non-null      float64
dtypes: float64(2), int64(2), object(18)
memory usage: 1.0+ MB
None
```

Número de linhas

Número de colunas

Classificação OBJECT não é correta

Coluna totalmente vazia

Limpando a base de dados (2/4)

Para corrigir a base de dados, vamos precisar novamente usar o pandas e seus métodos.

O primeiro método que usaremos será o :

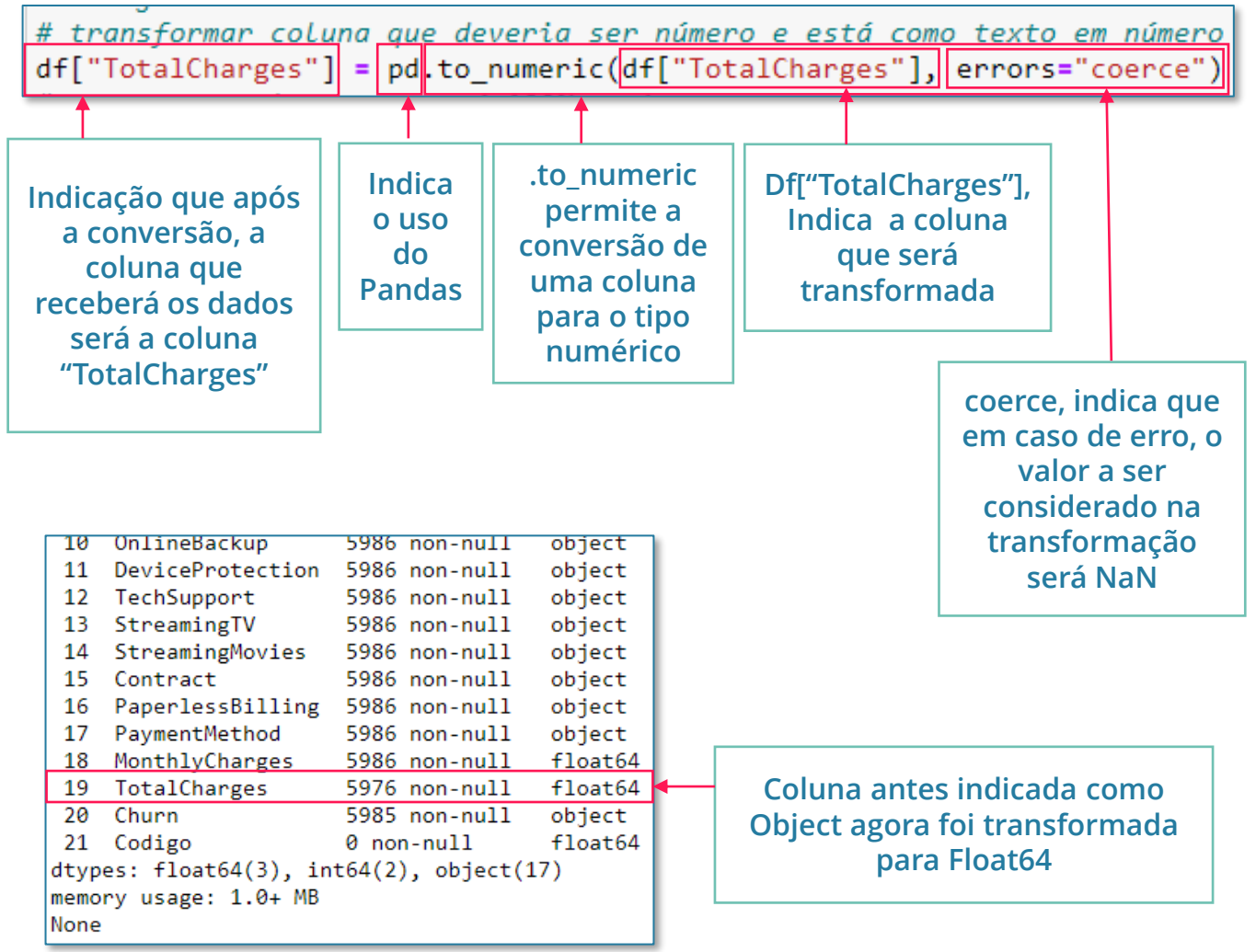
pd.to_numeric (documentação)

Como podemos ver na imagem ao lado, esse método se utiliza de alguns argumentos (valores que ficam entre os parênteses do método):

- Coluna que deverá ser transformada;
- Definição do que será feito em caso de erro.

Dica: Sempre consulte a documentação de uma biblioteca ou método antes de utilizá-la em um código ☺

Como podemos ver na extração da tabela de informações, a coluna **TotalCharges** agora é classificada como Float64.



Limpando a base de dados (3/4)

Agora precisamos remover a coluna Código que apresentou apenas valores Vazios.

Para isso, vamos utilizar outro método do pandas muito próximo do .drop que usamos anteriormente.

Esse método será o:

.dropna() (documentação)

Ele nos permite excluir, todas as linhas que possuam a linha toda completa de dados do tipo NA.

Assim como no caso anterior, o método dropna possui 2 argumentos que precisam ser fornecidos para que a exclusão possa acontecer:

- How = Indica quando uma linha ou coluna deve ser excluída. Apenas com a presença de um dado do tipo NA ou apenas se toda a linha ou coluna apresentar esse tipo de dado;
- Axis= Indica se o que deve ser excluído são linhas ou colunas. Axis=0 representam linhas e Axis=1 representam colunas.

```
# remover a coluna que está 100% vazia
df = df.dropna(how='all', axis=1)
```

Dataframe que receberá o dataframe transformado. Pode ser ou não o mesmo dataframe

Método para retirada de colunas com valores NA

How='all', Indica que só será removida a coluna que só possuir valores NA

axis=1 , Indica serão removidas colunas

```
 9   OnlineSecurity    5986 non-null   object
 10  OnlineBackup      5986 non-null   object
 11  DeviceProtection  5986 non-null   object
 12  TechSupport       5986 non-null   object
 13  StreamingTV       5986 non-null   object
 14  StreamingMovies   5986 non-null   object
 15  Contract          5986 non-null   object
 16  PaperlessBilling  5986 non-null   object
 17  PaymentMethod     5986 non-null   object
 18  MonthlyCharges    5986 non-null   float64
 19  TotalCharges      5976 non-null   float64
 20  Churn             5985 non-null   object
dtypes: float64(2), int64(2), object(17)
memory usage: 982.2+ KB
```

Coluna 'Código' Removida

Limpando a base de dados (4/4)

Ainda utilizando o método .dropna, vamos agora garantir que não temos dados faltantes em outras linhas do nosso dataframe.

No entanto, é possível que ao olharmos a imagem do código ao lado, surja a dúvida " Por que o parênteses estão vazios?".

No exemplo anterior vimos que precisávamos utilizar os argumentos axis e how para definir como e quais dados seriam excluídos.

O que acontece nesse caso, é que o método **.dropna()** possui valores padrões para os argumentos how e axis:

- How = 'any'
- Axis = '0'(linhas)

Assim, ao usarmos o método sem argumentos, o Python assumirá os argumentos padrões do método.

```
# remover a linha que tem um item vazio
df = df.dropna()
```

Dataframe que receberá o dataframe transformado. Pode ser ou não o mesmo dataframe

Método para retirada de colunas com valores NA

```
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        5974 non-null   object
 1   gender            5974 non-null   object
 2   SeniorCitizen     5974 non-null   int64
 3   Partner           5974 non-null   object
 4   Dependents        5974 non-null   object
 5   tenure            5974 non-null   int64
 6   PhoneService      5974 non-null   object
 7   MultipleLines     5974 non-null   object
 8   InternetService   5974 non-null   object
 9   OnlineSecurity    5974 non-null   object
 10  OnlineBackup      5974 non-null   object
 11  DeviceProtection  5974 non-null   object
 12  TechSupport       5974 non-null   object
 13  StreamingTV       5974 non-null   object
 14  StreamingMovies   5974 non-null   object
 15  Contract          5974 non-null   object
 16  PaperlessBilling  5974 non-null   object
 17  PaymentMethod     5974 non-null   object
 18  MonthlyCharges    5974 non-null   float64
 19  TotalCharges      5974 non-null   float64
 20  Churn             5974 non-null   object
dtypes: float64(2), int64(2), object(17)
```

Perceba que o número de números Não nulos em relação a tabela anterior diminuiu

Parte 4

Analisando os dados

INTENSIVÃO DE PYTHON {#}
100% ONLINE & GRATUITO

Como seguir?

O que vamos ver daqui para a frente, vai além do Python em si.

Poderíamos explicar milhões de formas de analisar os dados do Python, mas uma coisa é essencial:

O QUE EU QUERO RESPONDER?

Entender seu problema é fundamental. Assim, será possível orientar sua análise para resolver o problema.

Então vamos lá!

Nosso problema é:

"Alto índice de cancelamento de contratos"

O que eu quero:

"Entender os principais motivos que levam ao cancelamento para assim gerar um plano de ação"

Calculando encerramentos de contratos (1/2)

A primeira análise que iremos realizar é de caráter exploratório. Entender quantos de fato cancelaram o contrato e quantos ainda estão ativos.

Para isso, usaremos a coluna Churn do nosso dataframe.

O conceito de *Churn* em poucas palavras e simplificando o entendimento é o número de clientes que cancelaram o serviço prestado pela empresa.

Para isso, usaremos um novo método do pandas:

.value_counts() (documentação)

Perceba que usamos dentro do parênteses um argumento **normalize=True**. Ele nos permite transformar esses dados em percentuais que representam a proporção dos dados em relação ao número total de dados.

Baseado nos valores da coluna irá contar quantas vezes cada um aparece

```
display(df['Churn'].value_counts())
display(df['Churn'].value_counts(normalize=True))
```

Normalize =True como argumento permite transformar os números em valores percentuais. Ou seja qual a participação deles em relação ao todo

```
No     4387
Yes    1587
Name: Churn, dtype: int64
```

Na coluna ['Churn'] possuímos apenas os valores 'Yes' e 'No'.
Os números apresentados representam a quantidade de vezes que eles aparecem na base de dados

```
No     0.734349
Yes    0.265651
Name: Churn, dtype: float64
```

Proporção dos valores em relação ao total de dados. No entanto, ainda podemos melhorar a exibição desses dados

Calculando encerramentos de contratos (2/2)

Para melhorarmos a exibição dos valores vamos precisar acrescentar no nosso código.

Algo importante de ser entendido quando trabalhamos com python é que os códigos que estão na mesma linha se utilizam do resultado dos códigos anteriores para o resultado final.

O que isso significa?

Vamos analisar a segunda linha:

df['Churn'] -> é a base de tudo, coluna do dataframe que será envolvida na análise;

.Value_counts(normalize=True) -> Da coluna escolhida, conta os dados em um formato normalizado;

.map('{:.1%}'.format) -> Essa etapa se utiliza dos valores calculados na etapa anterior e simplesmente formata os números para um padrão específico. Nesse caso, percentual com apenas 1 casa decimal.

```
display(df['Churn'].value_counts())
display(df['Churn'].value_counts(normalize=True).map('{:.1%}'.format))
```

map() nos permite criar uma função dentro do parênteses. O que está ocorrendo aqui é:
"Do valor calculado até aqui, formate-o para um percentual com apenas 1 casa decimal"

```
No     4387
Yes    1587
Name: Churn, dtype: int64

No     73.4%
Yes    26.6%
Name: Churn, dtype: object
```

Dados calculados e formatados como percentual ☺

Análise gráfica – importando o Plotly

Um dos caminhos mais comuns e usuais para analisarmos os dados é através de uma análise gráfica.

Temos um total de 20 colunas, cada uma delas nos fornece uma informação distinta. Fazer análise gráfica destes dados pode ser um tanto quanto repetitiva por serem muitos dados.

Para fazermos isso um pouco mais rápido vamos criar um código permita gerar gráficos automaticamente via Python.

Vamos importar mais uma biblioteca para nos ajudar neste processo: **plotly.express**

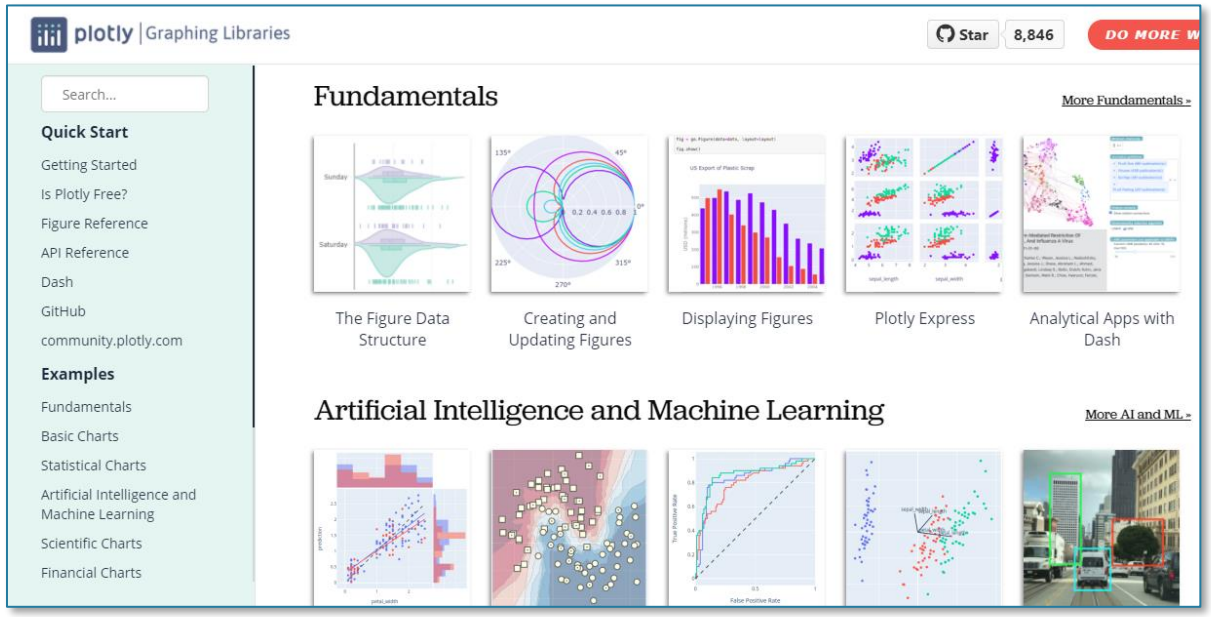
Caso você tenha interesse para mais informações é só acessar:

Para todos tipos de gráficos:
https://plotly.com/python/

Para nosso caso específico de histograma:
https://plotly.com/python/histograms/

Importando biblioteca plotly.express

```
import plotly.express as px
```



Análise gráfica – criando gráficos automaticamente (1/3)

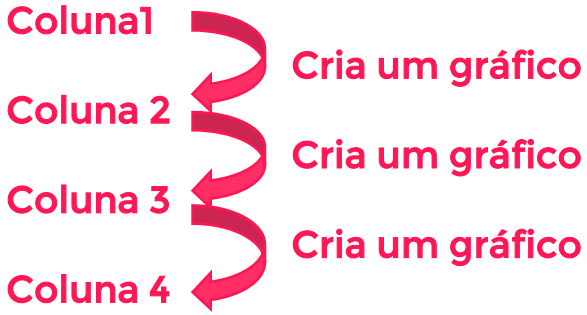
Como temos muitas colunas e a criação de gráficos essencialmente será sempre a mesma, podemos usar uma estrutura de repetição.

Essa estrutura de repetição é muito comum em linguagens de programação.

Uma das mais comuns é a função FOR.

Essencialmente o que essa função faz é repetir alguma atividade baseadas em uma indicação de início e fim.

O que acontecerá aqui, é o seguinte. Coluna a coluna, o python irá criar um gráfico assim como o diagrama abaixo:



...

Estrutura de repetição for:

coluna: variável que terá a função de receber os nomes das colunas passando por todas as 20 colunas dentro do nosso dataframe;

In: indica que são os valores QUE EXISTEM no dataframe df;

df: Base que as colunas que serão "lidas" existem.

```
for coluna in df:
    if coluna != "customerID":
        # criar a figura
        fig = px.histogram(df, x=coluna, color="Churn")
```

O espaço indicado se chama INDENTAÇÃO. Ele nos permite identificar quais são os códigos existentes DENTRO do for.

No Python, especificamente, a INDENTAÇÃO é OBRIGATÓRIA.

Análise gráfica – criando gráficos automaticamente (2/3)

Vamos entender agora outra função utilizada dentro do código.

A função IF. Essa função é uma função que nos permite tomar ações baseados em condições.

No nossos caso, o que queremos saber é se a coluna que está sendo usada naquela iteração do FOR é DIFERENTE da coluna CUSTOMERID.

Por que? Não faz muito sentido para nós criarmos um gráfico com o número dos clientes. Logo, vamos deixar essa coluna de fora quando passarmos por ela.

O que acontecerá aqui, é o seguinte. Coluna a coluna, o python irá criar um gráfico assim como o diagrama abaixo:

Coluna 1 é DIFERENTE de CostumerID? → Sim! Cria Gráfico

Coluna 2 é DIFERENTE de CostumerID? → Não! Próximo!

Coluna 3 é DIFERENTE de CostumerID? → Sim! Cria Gráfico

...

Estrutura do IF:

coluna: variável que JÁ RECEBEU uma coluna no FOR. Cada Iteração coluna terá um valor distinto.

(!=) : Sinal lógico que representa DIFERENTE;

"customerID": Valor que COLUNA será comparado a cada iteração.

```
for coluna in df:
    if coluna != "customerID":
        # criar a figura
        fig = px.histogram(df, x=coluna, color="Churn")
```

O espaço indicado se chama INDENTAÇÃO. Ele nos permite identificar quais são os códigos existentes DENTRO do IF.

No Python, especificamente, a INDENTAÇÃO é OBRIGATÓRIA.

Análise gráfica – criando gráficos automaticamente (3/3)

Continuando com as demais linhas de código chegamos na parte onde de fato o gráfico é criado.

Primeiramente, vamos precisar criar uma variável **fig** que receberá os dados a serem printados.

Esses dados (gráfico) serão calculados a partir do uso do método **.histogram().**

Podemos ver que os argumentos necessários são :

- **Tabela:** argumento da função, deverá ser fornecido no momento de ativação da função pelo usuário;
- **X=coluna:** Serão dados do eixo X. Os valores que existem na coluna fornecida pelo usuário no momento da ativação da função;
- **Color='Churn':** O gráfico terá cores diferentes para diferentes valores da coluna 'Churn'

df: Indica que a base dos dados que usaremos será df;

X=coluna: indica que o rótulo do eixo X será o nome da coluna que estiver naquela iteração.

color="Churn" : Aqui informamos o Python que a coluna Churn deverá ser diferenciada por uma cor específica.

```
for coluna in df:
    if coluna != "customerID":
        # criar a figura
        fig = px.histogram(df, x=coluna, color="Churn")
```

Variável fig armazenará o gráfico criado

Método da biblioteca pyplot. Nesse caso um gráfico do tipo histograma

Indica o uso do pyplot. Lembrando que na sua importação usamos a estrutura as px

Análise gráfica – Exibindo os gráficos (1/3)

Agora que temos uma variável **fig** que receberá os gráficos, precisamos criar uma linha de código que nos permita exibi-los.

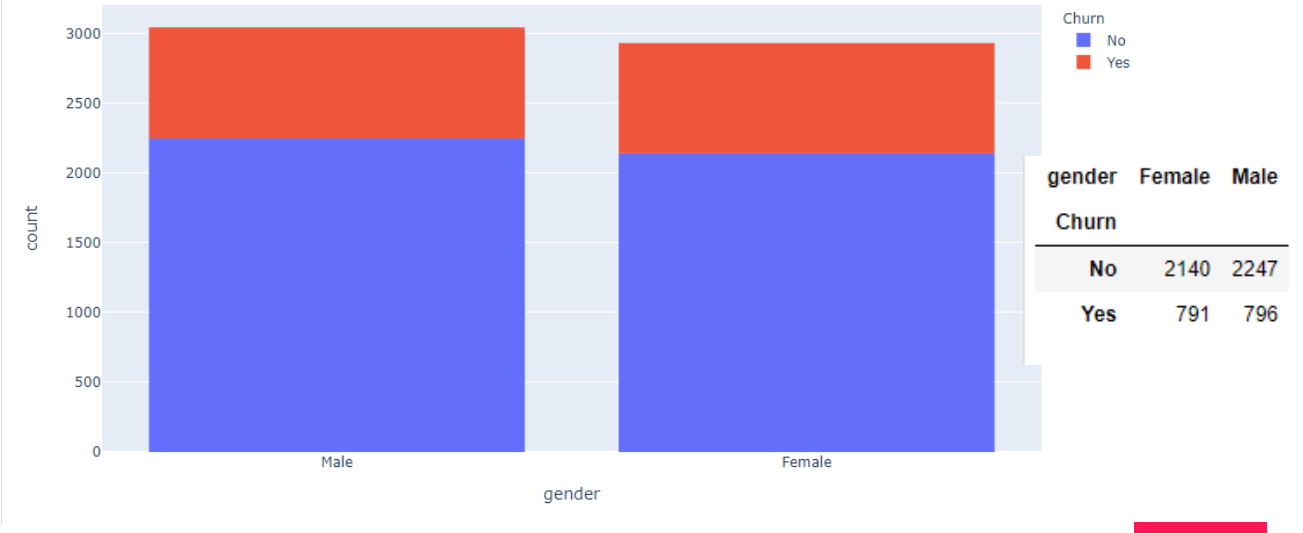
Essa linha de código será criada DENTRO do nosso FOR. Assim, a cada iteração, teremos um gráfico criado.

Para isso, usaremos o método **.show()**

Aproveitaremos para criar também uma tabela que acompanhe o gráfico com os dados que o compõe.

```
for coluna in df:
    if coluna != "customerID":
        # criar a figura
        fig = px.histogram(df, x=coluna, color="Churn")
        # exibir a figura
        fig.show()
        display(df.pivot_table(index="Churn", columns=coluna, aggfunc='count')["customerID"])
```

Cria a tabela com os dados do gráfico.



gender	Female	Male
Churn		
No	2140	2247
Yes	791	796

Análise gráfica – Exibindo os gráficos (2/3)

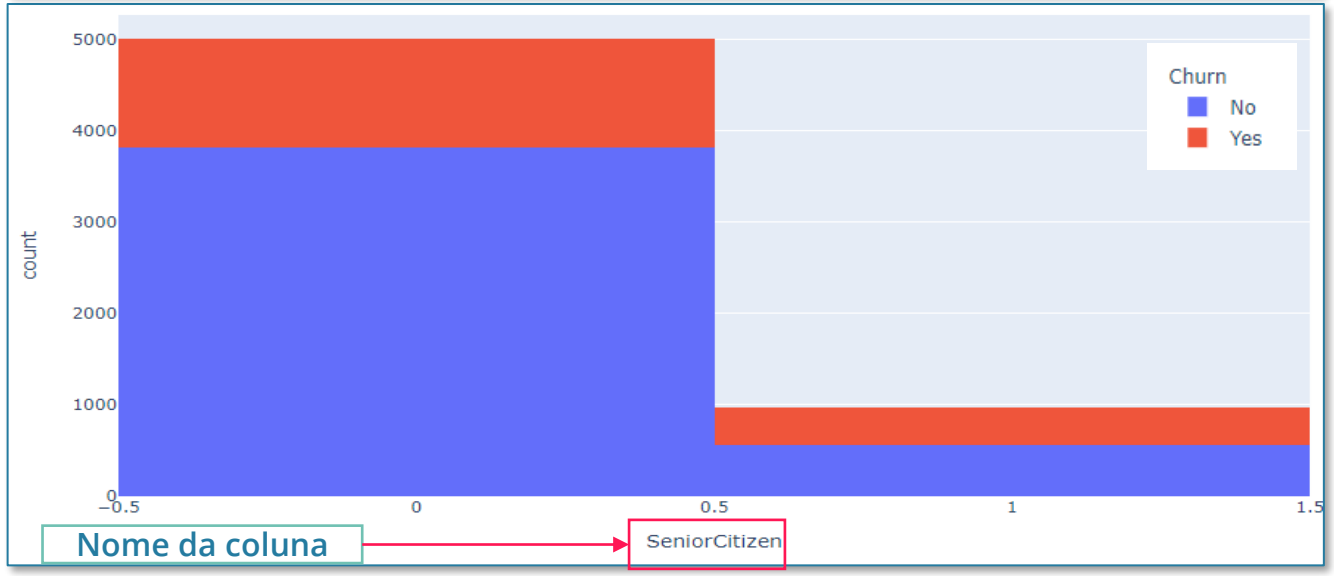
Como foi dito, em apenas um for conseguimos criar todos os gráficos disponíveis.

O eixo x sempre será o nome da coluna.
Isso já era esperado visto que ao fazermos a nossa função parametrizamos **x=coluna**:

```
# criar a figura
fig = px.histogram(df, x=coluna, color="Churn")
```

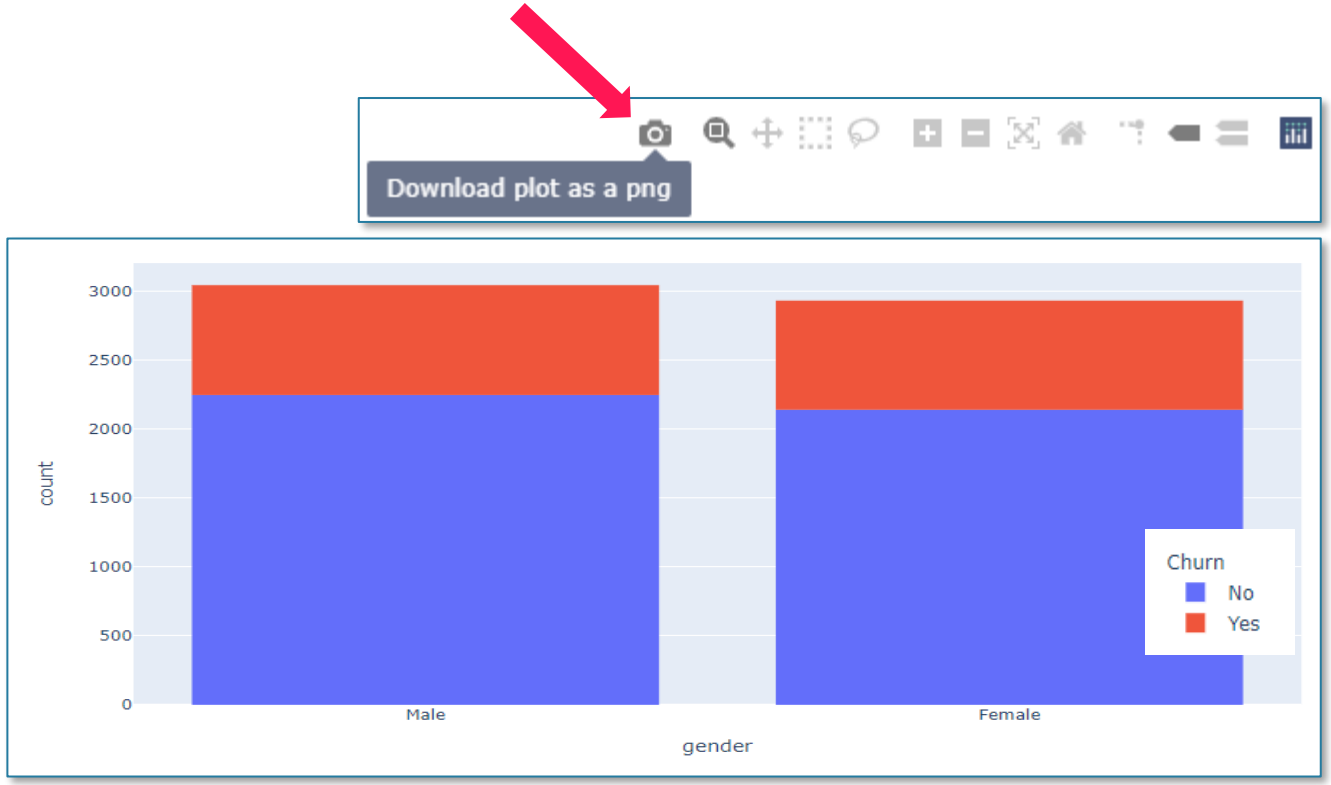
Além disso, podemos perceber que os dados que possuem valor da coluna Churn como "Yes", são representados em uma cor distinta aos clientes. Isso também era esperado visto a parametrização **color='Churn'**

```
# criar a figura
fig = px.histogram(df, x=coluna, color="Churn")
```



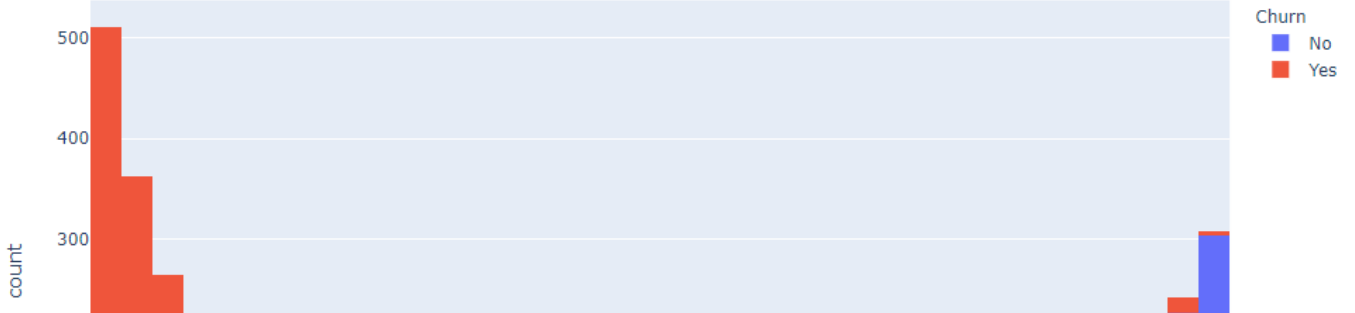
Análise gráfica – Exibindo os gráficos (3/3)

Uma funcionalidade muito interessante do Plotly é que podemos utilizar a barra de tarefas no topo superior do gráfico para aprofundar nossa análise.



Analisando os gráficos - Tenure

Assim, temos vários outros gráficos que nos permitem analisar de maneira mais profunda o nosso problema.
Boa parte do problema daqui para a frente é muito mais uma questão de análise de dados do que Python propriamente dita.
Vamos para os destaques da análise.



tenure	1	2	3	4	5	6	7	8	9	10	...	63	64	65	66	67	68	69	70	71	72
Churn																					
No	192	90	87	81	59	62	70	73	60	63	...	60	68	56	63	73	81	69	80	147	304
Yes	318	104	82	73	52	36	41	37	37	39	...	2	3	9	11	8	8	6	10	5	4

2 rows × 72 columns

Análise:

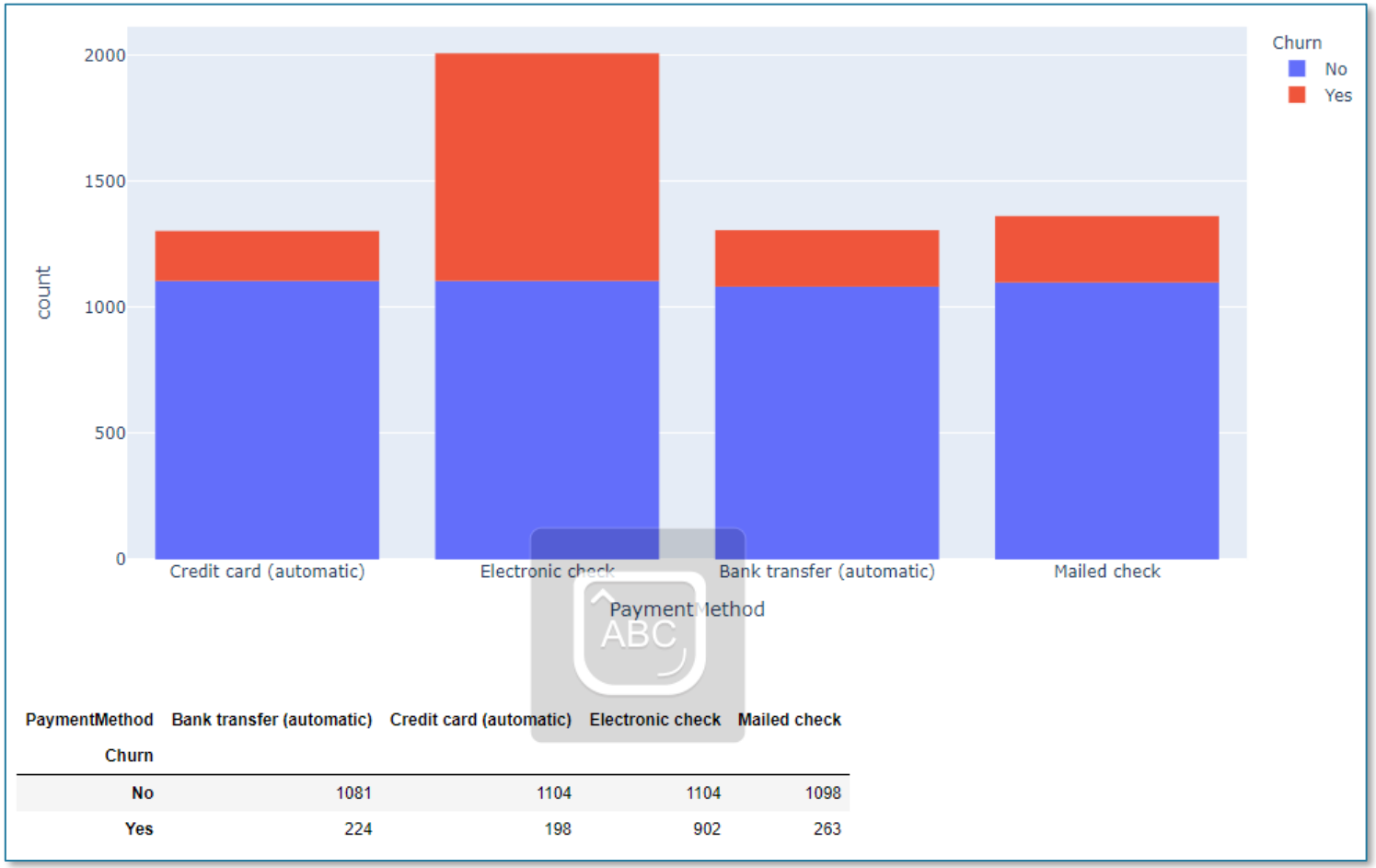
Proporção de Cancelamento Alta para valores de Tenure baixo.

Insights:

- Baixa qualidade no atendimento do pós venda;
- 1ª experiência com o cliente não é boa;
- Captação de clientes falha:
  - clientes desqualificados;
  - expectativas erradas ao adquirir o produto;

Analisando os gráficos – Payment Method

Assim, temos vários outros gráficos que nos permitem analisar de maneira mais profunda o nosso problema.
Boa parte do problema daqui para a frente é muito mais uma questão de análise de dados do que Python propriamente dita.
Vamos para os destaques da análise.



PaymentMethod	Bank transfer (automatic)	Credit card (automatic)	Electronic check	Mailed check
Churn				
No	1081	1104	1104	1098
Yes	224	198	902	263

Análise:

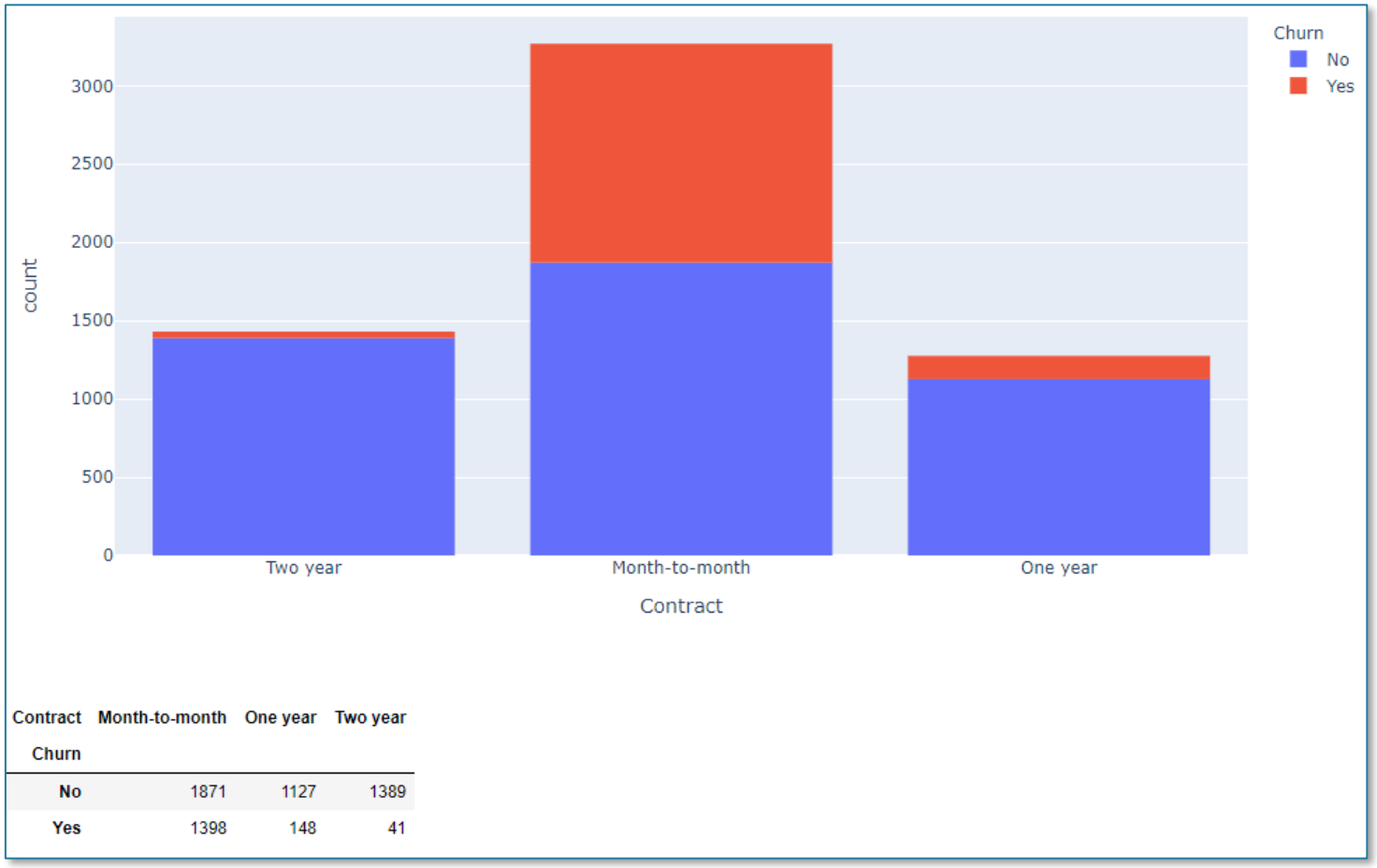
Os métodos automáticos tem uma taxa de churn menor

Insights:

- O Eletronic Check possui resultado muito ruim. Se possível evitar
- Ideia: oferecer benefícios caso a pessoa escolha um método automático;

Analisando os gráficos – Contract

Assim, temos vários outros gráficos que nos permitem analisar de maneira mais profunda o nosso problema.
Boa parte do problema daqui para a frente é muito mais uma questão de análise de dados do que Python propriamente dita.
Vamos para os destaques da análise.



Contract	Month-to-month	One year	Two year
Churn			
No	1871	1127	1389
Yes	1398	148	41

Análise:

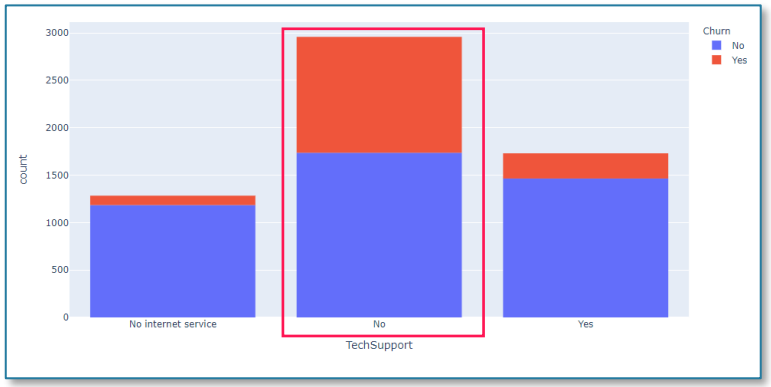
Contrato Mensal com taxa de cancelamento MUITO superior aos demais métodos de pagamento;

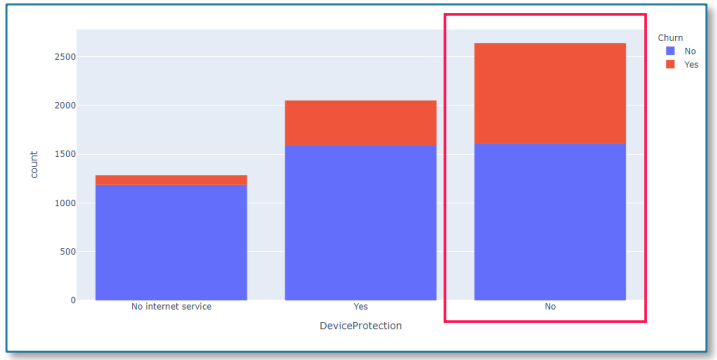
Insights:

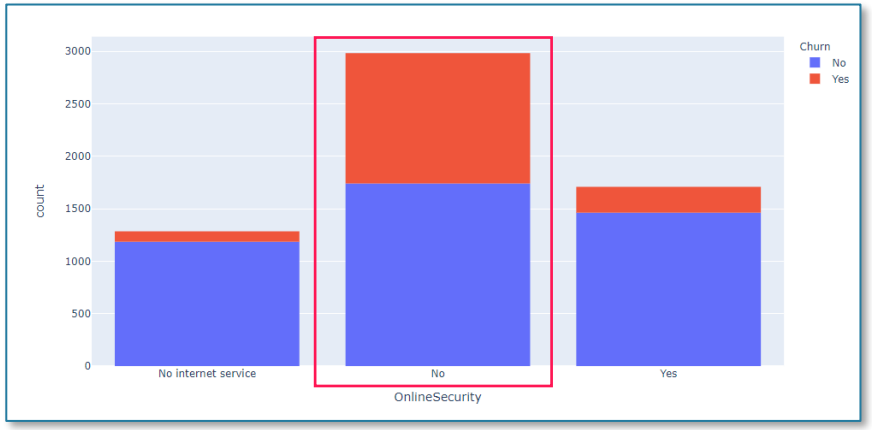
- Melhorar as condições comerciais dos planos anuais para direcionar a pessoa para estes casos;

Analisando os gráficos – Serviços adicionais

Assim, temos vários outros gráficos que nos permitem analisar de maneira mais profunda o nosso problema.
Boa parte do problema daqui para a frente é muito mais uma questão de análise de dados do que Python propriamente dita.
Vamos para os destaques da análise.







Análise:

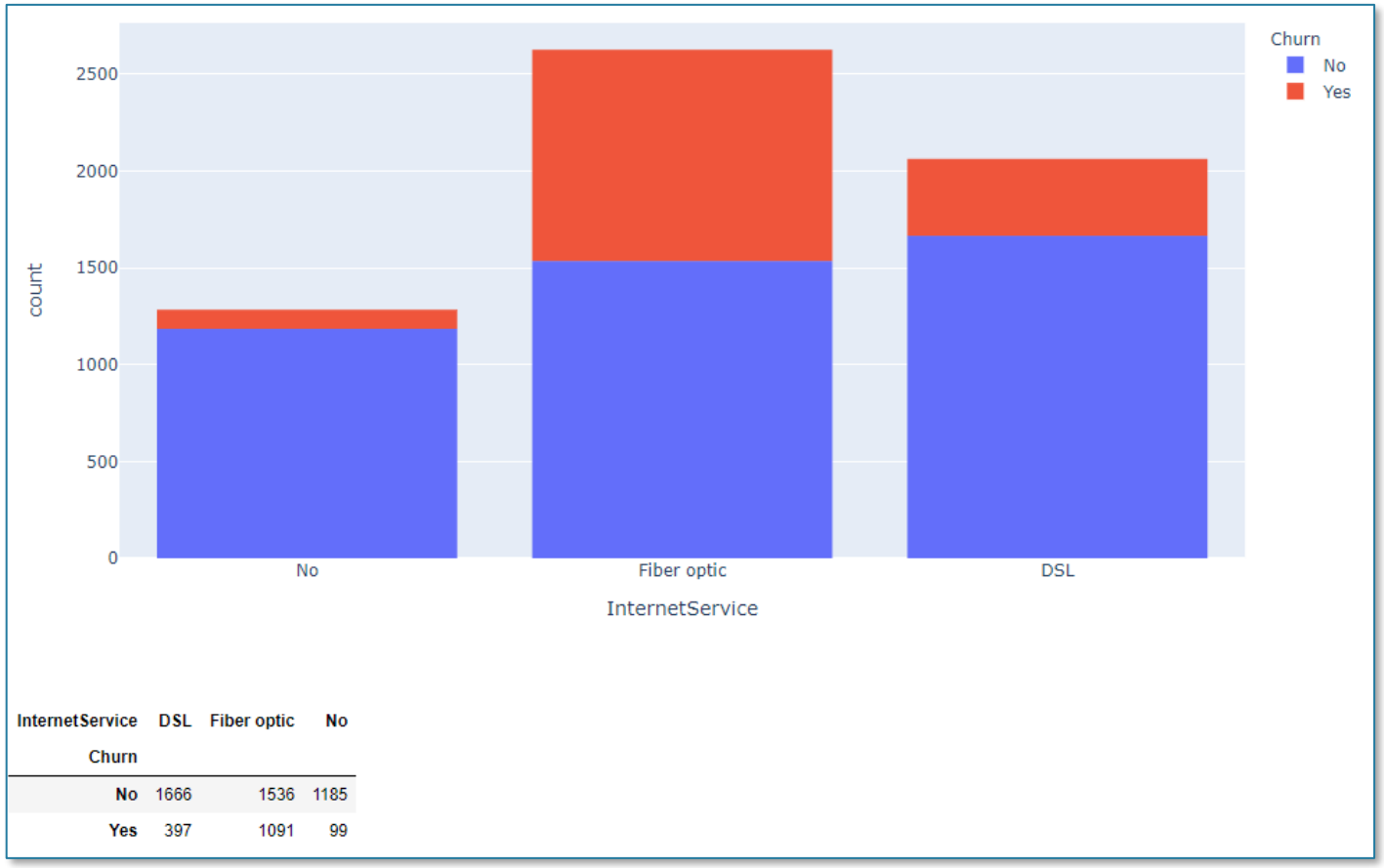
Clientes que NÃO possuem serviços adicionais de Suporte Técnico, Proteção a dispositivos e Segurança Online tendem a cancelar mais.

Insights:

- Todos os itens em algum nível falam sobre aumentar a confiabilidade/disponibilidade do serviço recebido pelo cliente.

Analisando os gráficos – Serviço de Internet

Assim, temos vários outros gráficos que nos permitem analisar de maneira mais profunda o nosso problema.
Boa parte do problema daqui para a frente é muito mais uma questão de análise de dados do que Python propriamente dita.
Vamos para os destaques da análise.



InternetService	DSL	Fiber optic	No
Churn			
No	1666	1536	1185
Yes	397	1091	99

Análise:

Clientes que NÃO possuem serviços adicionais de Suporte Técnico, Proteção a dispositivos e Segurança Online tendem a cancelar mais.

Insights:

- Todos os itens em algum nível falam sobre aumentar a confiabilidade/disponibilidade do serviço recebido pelo cliente.

INTENSIVÃO DE PYTHON

100% ONLINE & GRATUITO

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

@hashtagprogramacao

youtube.com/hashtag-programacao