PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

NÚCLEO DE EDUCAÇÃO A DISTÂNCIA

Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

José Nunes da Silva Junior

ANÁLISE CHURN RATE TELECOM

Belo Horizonte

2023

José Nunes da Silva Junior

ANÁLISE DE CHURN RATE TELECOM

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte

2023

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

1.1 Contextualização

Nos últimos anos o setor de telecomunicações tem observado um enorme avanço tecnológico, avanço este, que possibilitou novas oportunidades de negócios para as empresas. A facilidade de contratar determinados serviços com poucos cliques, também é encontrada ao se tentar cancelar este mesmo serviço. Dada a crescente competição no mercado, bem como o aumento da demanda dos consumidores por serviços mais personalizados e acessíveis, um elemento vem se tornando o foco das atenções: o *churn rate*.

O "*churn rate*" (taxa de rotatividade de clientes), é uma métrica crucial para empresas no ramo de telecomunicações, diferentemente de um modelo de compra e venda onde se obtêm o lucro quando o processo se concretiza, o modelo de negócios de telecomunicações se caracteriza por um investimento inicial feito pela empresa, seja em infraestrutura, equipamentos e/ou instalação na residência do cliente. Nesse modelo de negócio, o cliente só passa a dar lucro meses ou até mesmo anos depois de contratar o serviço.

Diante disso, tornou-se fundamental para as empresas o foco nesse indicador, uma vez que a retenção de clientes é crucial para o sucesso e a sustentabilidade dos negócios.

1.2 O problema proposto

Devido as características de alto investimento inicial e lucro futuro, a presente análise terá como foco buscar a identificação de padrões de comportamento, além de análises demográficas, de ofertas, tempo de empresa, valor pago entre outras informações que ajudem a prever um potencial cancelamento, fornecendo *insights*[1]

[1] Compreensão repentina sobre um problema ou situação específica.

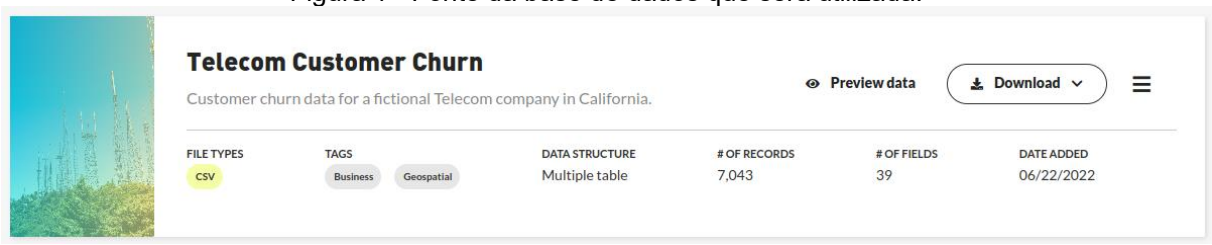
que ajude a empresa em suas estratégias de retenção de clientes. Na etapa de medição dos resultados dos algoritmos de aprendizado de máquina, iremos priorizar a métrica revocação[2] (*recall*). A justificativa para isso é que o custo de perder clientes propensos ao *churn* é alto, devemos buscar minimizar o máximo possível de falsos negativos, mesmo que isso gere alguns falsos positivos.

[2] Métrica que indica a capacidade do modelo de classificar corretamente uma determinada classe.

2 COLETA DOS DADOS

O conjunto de dados *Telecom Customer Churn Prediction*, objeto de análise deste projeto pode ser encontrado através do sítio da *Maven Analytics*, plataforma de desafios relacionados a análises de dados.

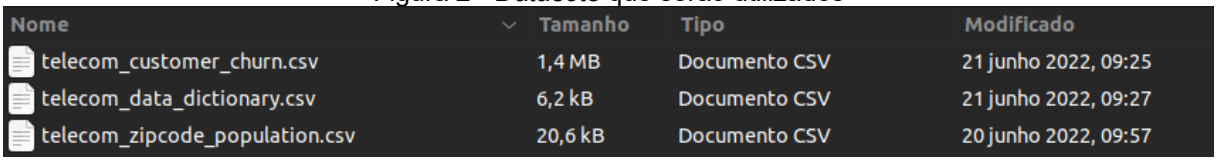
Figura 1 - Fonte da base de dados que será utilizada.



Fonte: Maven Analytics (2022)

Na Figura 1 é possível observar os nomes e propriedades dos arquivos que serão utilizados. O conjunto em questão consiste em 3 (três) *datasets*[3] que possuem informações e finalidades distintas (Figura 2): *telecom_customer_churn* contêm a base principal dos dados que serão analisados neste artigo, *telecom_zipcode_population* possui dados adicionais relacionados a população por código postal, e *telecom_data_dictionary*, como o próprio nome sugere, trata-se de um dicionário que contém as informações do que cada coluna representa em nossa base principal.

Figura 2 - *Datasets* que serão utilizados



Fonte: Autoria própria.

Para uma maior facilidade de interpretação dos leitores deste artigo, os nomes das colunas originalmente em inglês, serão traduzidos para o português na etapa de tratamento de dados. Na Tabela 1 está descrito de forma detalhada as informações que cada coluna representa no arquivo *telecom_customer_churn*.

[3] Conjunto de dados similar a uma tabela.

Tabela 1 – Detalhamento do *dataset telecom_customer_churn*

Nome Original	Nome traduzido	Descrição	Tipo
CustomerID	ID_cliente	ID única que identifica cada cliente.	Texto
Gender	Genero	Gênero do cliente.	Texto
Age	Idade	Idade do cliente.	Inteiro
Married	Casado	Indica se o cliente é casado.	Texto
Number of Dependents	Qtd_dependentes	Indica a quantidade de dependentes que moram com o cliente.	Inteiro
City	Cidade	Cidade em que se localiza a residência do cliente.	Texto
Zip Code	Codigo_postal	Código postal da residência do cliente.	Inteiro
Latitude	Latitude	Latitude da residência do cliente.	Real
Longitude	Longitude	Longitude da residência do cliente.	Real
Number of Referrals	Qtd_indicações	Quantidade de indicações que o cliente realizou a amigos ou parentes.	Inteiro
Tenure in Months	Meses_na_base	Indica o total de meses que o cliente permaneceu na empresa.	Inteiro
Offer	Ultima_oferta	Indica a última oferta que o cliente aceitou.	Texto
Phone Service	Servico_telefone	Indica se o cliente possui o serviço telefônico da empresa.	Texto
Avg Monthly Long Distance Charges	Recarga_longa_distancia	Indica o custo médio total de recargas de longa distância.	Real
Multiple Lines	Multiplas_linhas	Indica se o cliente possui mais de uma linha telefônica.	Texto
Internet Service	Servico_internet	Indica se o cliente possui o serviço de internet da empresa.	Texto
Internet Type	Tipo_internet	Indica o tipo de conexão que o cliente utiliza.	Texto
Avg Monthly GB Download	Media_mensal_download_GB	Indica o volume médio de download do cliente em gigabytes.	Real
Online Security	Servico_seguranca	Indica se o cliente possui um serviço adicional de segurança provido pela empresa.	Texto
Online Backup	Servico_backup	Indica se o cliente possui um serviço adicional de *backup* provido pela empresa.	Texto
Device Protection Plan	Servico_protecao_dispositivo	Indica se o cliente possui um serviço adicional de proteção de dispositivo.	Texto
Premium Tech Support	Servico_suporte_preferencial	Indica se o cliente possui um serviço adicional de atendimento preferencial.	Texto

Streaming TV	Servico_tv	Indica se o cliente possui algum serviço de *streaming* de TV.	Texto
Streaming Movies	Servico_filmes	Indica se o cliente possui algum serviço de *streaming* de filmes.	Texto
Streaming Music	Servico_musica	Indica se o cliente possui algum serviço de *streaming* de músicas.	Texto
Unlimited Data	Servico_ilimitado_dados	Indica se o cliente possui algum serviço de dados ilimitados.	Texto
Contract	Tipo_contrato	Indica o tipo de contrato ao qual o cliente está fidelizado.	Texto
Paperless Billing	Faturamento_sem_papel	Indica se o cliente escolheu pelo tipo de cobrança sem papel.	Texto
Payment Method	Tipo_pagamento	Indica o método de pagamento escolhido pelo cliente.	Texto
Monthly Charge	Cobranca_mensal	Indica o total mensal pago pelo cliente por todos os serviços.	Real
Total Charges	Total_cobranca	Indica a soma total paga pelo cliente em todo o tempo na empresa.	Real
Total Refunds	Total_reembolsos	Indica a soma total de reembolso recebida pelo cliente em todo o tempo na empresa.	Real
Total Extra Data Charges	Qtd_dados_extras	Indica a quantidade total de dados extras contratados pelo cliente.	Inteiro
Total Long Distance Charges	Cobrancas_longa_distancia	Indica a soma total paga pelo cliente em todo o tempo na empresa referente a recargas para longas distancias.	Real
Total Revenue	Total_cobranca_geral	Indica a soma total paga pelo cliente de forma geral somando o valor mensal com todas as cobranças de recargas extras.	Real
Customer Status	Situacao_cliente	Indica o status do cliente ao final período em que os dados foram coletados: *Churned* (cancelou), *Stayed* (permaneceu cliente) e *Joined* (se tornou cliente).	Texto
Churn Category	Categoria_cancelamento	Indica uma categoria que classifica o motivo de cancelamento do cliente.	Texto
Churn Reason	Motivo_cancelamento	Indica o motivo do cancelamento do cliente.	Texto

Fonte: Maven Analytics (2022)

O mesmo processo será realizado no arquivo *telecom_zipcode_population*, como pode ser observado na Tabela 2.

Tabela 2 - Detalhamento do *dataset telecom_data_dictionary*

Nome Original	Nome traduzido	Descrição	Tipo
Zip Code	Codigo_postal	Código postal em que o cliente reside.	Texto
Population	Populacao	População total estimada do código postal.	Inteiro

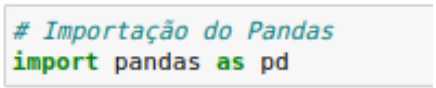
Fonte: Maven Analytics (2022)

3 TRATAMENTO DOS DADOS

Para a execução do projeto, foi utilizada a linguagem de programação interpretada Python, na sua versão 3.9.13, além do *Jupyter Notebook*, ambiente de desenvolvimento interativo baseado na Web.

Nesta etapa do projeto, será utilizada majoritariamente a biblioteca *Pandas* (Figura 3) para manejo dos dados, bem como de eventuais ajustes que possam ser necessários para deixar a base de dados mais bem estruturada, e com isso, buscar melhores interpretações dos algoritmos de *Machine Learning* (aprendizado de máquina).

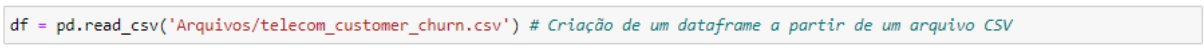
Figura 3 - Importação do *pandas*



Fonte: Autoria própria

Foi utilizado para a leitura dos dados, uma função própria do *pandas*, a *pd.read_csv* (Figura 4), através dela é possível ler um arquivo *csv* e já transforma-lo em um *dataframe*, que nada mais é do que uma estrutura de dados composta por duas dimensões: linhas e colunas, similar a uma planilha.

Figura 4 - Leitura do arquivo com a biblioteca *pandas*



Fonte: Autoria própria

Em seguida, por meio do comando *display* é possível visualizar o *dataframe* criado (Figura 5).

Figura 5 - Visualização do *dataframe*

```
display(df)
```

	Customer ID	Gender	Age	Married	Number of Dependents	City	Zip Code	Latitude	Longitude	Number of Referrals	...	Payment Method	Monthly Charge	Total Charges	Total Refunds	Total Extra Data Charges
0	0002-ORFBO	Female	37	Yes	0	Frazier Park	93225	34.827662	-118.999073	2	...	Credit Card	65.60	593.30	0.00	0
1	0003-MKNFE	Male	46	No	0	Glendale	91206	34.162515	-118.203869	0	...	Credit Card	-4.00	542.40	38.33	10
2	0004-TLHLJ	Male	50	No	0	Costa Mesa	92627	33.645672	-117.922613	0	...	Bank Withdrawal	73.90	280.85	0.00	0
3	0011-IGKFF	Male	78	Yes	0	Martinez	94553	38.014457	-122.115432	1	...	Bank Withdrawal	98.00	1237.85	0.00	0
4	0013-EXCHZ	Female	75	Yes	0	Camarillo	93010	34.227846	-119.079903	3	...	Credit Card	83.90	267.40	0.00	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7038	9987-LUTYD	Female	20	No	0	La Mesa	91941	32.759327	-116.997260	0	...	Credit Card	55.15	742.90	0.00	0
7039	9992-RRAMN	Male	40	Yes	0	Riverbank	95367	37.734971	-120.954271	1	...	Bank Withdrawal	85.10	1873.70	0.00	0
7040	9992-UJOEL	Male	22	No	0	Elk	95432	39.108252	-123.645121	0	...	Credit Card	50.30	92.75	0.00	0
7041	9993-LHIEB	Male	21	Yes	0	Solana Beach	92075	33.001813	-117.263628	5	...	Credit Card	67.85	4627.65	0.00	0
7042	9995-HOTOH	Male	36	Yes	0	Sierra City	96125	39.600599	-120.636358	1	...	Bank Withdrawal	59.00	3707.60	0.00	0

7043 rows × 38 columns

Fonte: Autoria própria

Um detalhe importante sobre a função display é que, ela por padrão exibe no máximo 20 colunas, para que ela exiba todas as colunas e assim consigamos uma visão geral do nosso *dataframe*, iremos realizar uma configuração através do comando *pd.set_option* alterando o parâmetro *display.max_columns*, como mostrado na Figura 6.

Figura 6 - Ajuste no máximo de colunas da função *display*

```
print("Quantidade padrão de colunas do display:",pd.options.display.max_columns)
pd.set_option('display.max_columns', None)
print("Quantidade alterada para:",pd.options.display.max_columns)

display(df)
```

```
Quantidade padrão de colunas do display: 20
Quantidade alterada para: None
```

	Customer ID	Gender	Age	Married	Number of Dependents	City	Zip Code	Latitude	Longitude	Number of Referrals	Tenure in Months	Offer	Phone Service	Avg Monthly Long Distance Charges	Multiple Lines	Internet Service	I
0	0002-ORFBO	Female	37	Yes	0	Frazier Park	93225	34.827662	-118.999073	2	9	None	Yes	42.39	No	Yes	
1	0003-MKNFE	Male	46	No	0	Glendale	91206	34.162515	-118.203869	0	9	None	Yes	10.69	Yes	Yes	
2	0004-TLHLJ	Male	50	No	0	Costa Mesa	92627	33.645672	-117.922613	0	4	Offer E	Yes	33.65	No	Yes	
3	0011-IGKFF	Male	78	Yes	0	Martinez	94553	38.014457	-122.115432	1	13	Offer D	Yes	27.82	No	Yes	
4	0013-EXCHZ	Female	75	Yes	0	Camarillo	93010	34.227846	-119.079903	3	3	None	Yes	7.38	No	Yes	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7038	9987-LUTYD	Female	20	No	0	La Mesa	91941	32.759327	-116.997260	0	13	Offer D	Yes	46.68	No	Yes	
7039	9992-RRAMN	Male	40	Yes	0	Riverbank	95367	37.734971	-120.954271	1	22	Offer D	Yes	16.20	Yes	Yes	
7040	9992-UJOEL	Male	22	No	0	Elk	95432	39.108252	-123.645121	0	2	Offer E	Yes	18.62	No	Yes	
7041	9993-LHIEB	Male	21	Yes	0	Solana Beach	92075	33.001813	-117.263628	5	67	Offer A	Yes	2.12	No	Yes	
7042	9995-HOTOH	Male	36	Yes	0	Sierra City	96125	39.600599	-120.636358	1	63	None	No	NaN	NaN	Yes	

7043 rows × 38 columns

Fonte: Autoria própria

Dando continuidade, agora realizaremos a leitura e criação do *dataframe* de população por código postal (Figura 7), esses dados serão agregados ao nosso *dataframe* principal com o objetivo de enriquecer nossa base de dados, a fim de encontrar algum padrão com a quantidade de pessoas de uma determinada região, o código usado para realizar essa junção está representado na Figura 8.

Figura 7 - Criando *dataframe* da população por código postal

```
df_populacao = pd.read_csv('Arquivos/telecom_zipcode_population.csv')
display(df_populacao)
```

	Zip Code	Population
0	90001	54492
1	90002	44586
2	90003	58198
3	90004	67852
4	90005	43019
...	...	...
1666	96145	4002
1667	96146	942
1668	96148	678
1669	96150	33038
1670	96161	15783

1671 rows × 2 columns

Fonte: Autoria própria

Figura 8 - Junção dos *dataframes*

```
df = pd.merge(df, df_populacao,
              left_on=['Zip Code'],
              right_on=['Zip Code'],
              how='left') # Mesclando as duas tabelas com base na coluna 'Zip Code'

### Reordenando a posição da coluna 'Population' dentro do dataframe ###
colunas = df.columns.tolist()
colunas.insert(7, colunas.pop(colunas.index('Population')))
df = df[colunas]
```

Fonte: Autoria própria

Agora temos o nosso *dataframe* consolidado e pronto para ser realizado os devidos ajustes.

Como informado no tópico 2 deste artigo, iremos realizar o ajuste dos nomes das colunas do nosso *dataframe*, traduzindo e alterando para um padrão onde a primeira letra é maiúscula e as palavras são separadas pelo caractere *underline* (_). O comando utilizado para realizar tal procedimento pode ser visualizado na Figura 9.

Figura 9 – Tradução dos nomes das colunas

```
df = df.rename(columns={'Customer ID':'ID_cliente',
                        'Gender': 'Genero',
                        'Age': 'Idade',
                        'Married': 'Casado',
                        'Number of Dependents':'Qtd_dependentes',
                        'City': 'Cidade',
                        'Zip Code':'Codigo_postal',
                        'Population': 'Populacao',
                        'Number of Referrals':'Qtd_indicações',
                        'Tenure in Months':'Meses_na_base',
                        'Offer':'Ultima_oferta',
                        'Phone Service':'Servico_telefone',
                        'Avg Monthly Long Distance Charges':'Recarga_longa_distancia',
                        'Multiple Lines':'Multiplas_linhas',
                        'Internet Service':'Servico_internet',
                        'Internet Type':'Tipo_internet',
                        'Avg Monthly GB Download':'Media_mensal_download_GB',
                        'Online Security':'Servico_seguranca',
                        'Online Backup':'Servico_backup',
                        'Device Protection Plan':'Servico_protecao_dispositivo',
                        'Premium Tech Support':'Servico_suporte_preferencial',
                        'Streaming TV':'Servico_tv',
                        'Streaming Movies':'Servico_filmes',
                        'Streaming Music':'Servico_musica',
                        'Unlimited Data':'Servico_ilimitado_dados',
                        'Contract':'Tipo_contrato',
                        'Paperless Billing':'Faturamento_sem_papel',
                        'Payment Method':'Tipo_pagamento',
                        'Monthly Charge':'Cobranca_mensal',
                        'Total Charges':'Total_cobranca',
                        'Total Refunds':'Total_reembolsos',
                        'Total Extra Data Charges':'Qtd_dados_extras',
                        'Total Long Distance Charges':'Cobrancas_longa_distancia',
                        'Total Revenue':'Total_cobranca_geral',
                        'Customer Status':'Situacao_cliente',
                        'Churn Category':'Categoria_cancelamento',
                        'Churn Reason':'Motivo_cancelamento' }, inplace=False)
```

Fonte: Autoria própria

Também será realizada a tradução dos valores contidos em todas as colunas de texto da nossa base de dados, com exceção das colunas Cidade e ID_cliente. No que diz respeito coluna Cidade, não se faz necessário a tradução desses valores por se tratar de nomes de cidades, já a coluna ID_cliente, possui apenas valores que identificam o cliente dentro da empresa, entretanto, para nós trata-se de uma informação sem valor, dessa forma, esta coluna será removida da nossa base de dados. Na Figura 10 está exposto o comando utilizado para remover a coluna ID_cliente.

Figura 10 - Removendo a coluna ID_cliente

```
df = df.drop('ID_cliente', axis=1)
```

Fonte: Autoria própria

Já na Figura 11, iremos realizar um comando para nos retornar os valores únicos contidos em cada coluna de texto, tornando mais fácil a identificação dos valores que irão necessitar serem traduzidos.

Figura 11 - Identificando os valores que precisarão ser traduzidos

```
val_unicos = {}
colunas = df.columns.tolist()
itens_removidos = ['Qtd_dependentes',
                   'Qtd_indicações',
                   'Qtd_dados_extras',
                   'Media_mensal_download_GB',
                   'Idade','Meses_na_base',
                   'Total_reembolsos',
                   'Cidade',
                   'Populacao',
                   'Cobranca_mensal',
                   'Longitude',
                   'Latitude',
                   'Codigo_postal',
                   'Recarga_longa_distancia',
                   'Cobrancas_longa_distancia',
                   'Total_cobranca',
                   'Total_cobranca_geral']

for item in itens_removidos:
    colunas.remove(item)

for col in colunas:
    val_unicos[col] = pd.DataFrame(df[col].unique(), columns=[col])

for col, val_unicos in val_unicos.items():
    print(f"Valores únicos para a coluna '{col}':")
    print(val_unicos)
    print()
```

```
Valores únicos para a coluna 'Genero':
   Genero
0  Female
1    Male

Valores únicos para a coluna 'Casado':
  Casado
0    Yes
1     No

Valores únicos para a coluna 'Ultima_oferta':
  Ultima_oferta
0          None
1       Offer E
2       Offer D
3       Offer A
4       Offer B
5       Offer C
```

Fonte: Autoria própria

A partir da identificação desses valores, podemos agora realizar a sua tradução com o comando *replace* (Figura 12).

Figura 12 - Renomeando valores da base de dados

```
subst = {
    'Male': 'Masculino',
    'Female': 'Feminino',
    'Yes': 'Sim',
    'No': 'Não',
    'Offer A': 'Oferta A',
    'Offer B': 'Oferta B',
    'Offer C': 'Oferta C',
    'Offer D': 'Oferta D',
    'Offer E': 'Oferta E',
    'None': 'Nenhuma oferta',
    'Cable': 'Cabo',
    'Fiber Optic': 'Fibra óptica',
    'DSL': 'Cabo DSL',
    'One Year': 'Um ano',
    'Month-to-Month': 'Mensal',
    'Two Year': 'Dois anos',
    'Credit Card': 'Cartão de credito',
    'Bank Withdrawal': 'Transferência Bancária',
    'Mailed Check': 'Cheque enviado pelo correio',
    'Stayed': 'Continua cliente',
    'Churned': 'Cancelou',
    'Joined': 'Se tornou cliente',
    'Competitor': 'Concorrente',
    'Dissatisfaction': 'Insatisfação',
    'Other': 'Outros',
    'Price': 'Preço',
    'Attitude': 'Atitude',
    'Competitor had better devices': 'Concorrente possui dispositivos melhores',
    'Product dissatisfaction': 'Insatisfação com o produto',
    'Network reliability': 'Confiabilidade da rede',
    'Limited range of services': 'Gama limitada de serviços',
    'Competitor made better offer': 'Concorrente fez uma oferta melhor',
    "Don't know": 'Não sabe dizer',
    'Long distance charges': 'Tarifas de longa distancia',
    'Attitude of service provider': 'Atitude do provedor do serviço',
    'Attitude of support person': 'Atitude do suporte',
    'Competitor offered higher download speeds': 'Concorrente ofereceu maiores velocidades de download',
    'Competitor offered more data': 'Concorrente ofereceu mais limite de dados',
    'Lack of affordable download/upload speed': 'Falta de velocidade de download/upload por um valor acessível',
    'Deceased': 'Faleceu',
    'Moved': 'Mudou-se',
    'Service dissatisfaction': 'Insatisfação com o serviço',
    'Price too high': 'Preço muito alto',
    'Lack of self-service on Website': 'Falta de autoatendimento no site',
    'Poor expertise of online support': 'Experiencia ruim no suporte online',
    'Extra data charges': 'Tarifas extras de dados',
    'Poor expertise of phone support': 'Experiencia ruim no suporte por telefone',
}
df = df.replace(subst)
```

Fonte: Autoria própria

Dando continuidade ao nosso processo de tratamento dos dados, iniciaremos agora algumas análises iniciais em cima da nossa base de dados, com o comando *df.info* (Figura 13) iremos visualizar algumas informações sobre o nosso *dataframe*.

Figura 13 – Informações do *dataframe*

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7043 entries, 0 to 7042
Data columns (total 38 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Genero                          7043 non-null   object
 1   Idade                           7043 non-null   int64
 2   Casado                          7043 non-null   object
 3   Qtd_dependentes                 7043 non-null   int64
 4   Cidade                          7043 non-null   object
 5   Codigo_postal                   7043 non-null   int64
 6   Populacao                       7043 non-null   int64
 7   Latitude                        7043 non-null   float64
 8   Longitude                       7043 non-null   float64
 9   Qtd_indicações                  7043 non-null   int64
 10  Meses_na_base                   7043 non-null   int64
 11  Ultima_oferta                   7043 non-null   object
 12  Servico_telefone                7043 non-null   object
 13  Recarga_longa_distancia         6361 non-null   float64
 14  Multiplas_linhas                6361 non-null   object
 15  Servico_internet                7043 non-null   object
 16  Tipo_internet                   5517 non-null   object
 17  Media_mensal_download_GB        5517 non-null   float64
 18  Servico_seguranca               5517 non-null   object
 19  Servico_backup                  5517 non-null   object
 20  Servico_protecao_dispositivo    5517 non-null   object
 21  Servico_suporte_preferencial    5517 non-null   object
 22  Servico_tv                      5517 non-null   object
 23  Servico_filmes                  5517 non-null   object
 24  Servico_musica                  5517 non-null   object
 25  Servico_ilimitado_dados         5517 non-null   object
 26  Tipo_contrato                   7043 non-null   object
 27  Faturamento_sem_papel           7043 non-null   object
 28  Tipo_pagamento                  7043 non-null   object
 29  Cobranca_mensal                 7043 non-null   float64
 30  Total_cobranca                  7043 non-null   float64
 31  Total_reembolsos                7043 non-null   float64
 32  Qtd_dados_extras                7043 non-null   int64
 33  Cobrancas_longa_distancia       7043 non-null   float64
 34  Total_cobranca_geral            7043 non-null   float64
 35  Situacao_cliente                7043 non-null   object
 36  Categoria_cancelamento          1869 non-null   object
 37  Motivo_cancelamento             1869 non-null   object
dtypes: float64(9), int64(7), object(22)
memory usage: 2.1+ MB
```

Fonte: Autoria própria

É possível observar que em nosso *dataframe*, algumas colunas possuem valores faltantes, em uma breve análise, nota-se que essas colunas são condicionadas ao valor contido em outra coluna, por exemplo: o valor presente na coluna Tipo_internet está relacionado ao fato de o cliente possuir (ou não) o serviço de internet (coluna Servico_internet).

O mesmo ocorre com colunas que dependem do valor contido na coluna Servico_telefone, que tem informações se o cliente possui (ou não) o serviço. Na Figura 14 foi utilizado o comando *groupby* para agrupar os dados contidos nas colunas Servico_internet e Servico_telefone junto do comando *count* para que o agrupamento seja feito realizando a contagem dos valores.

Figura 14 - Contagem dos valores das colunas

```
print(df.groupby(['Servico_telefone'])['Servico_telefone'].count())
print('---------------------------------------')
print(df.groupby(['Servico_internet'])['Servico_internet'].count())
```

```
Servico_telefone
Não     682
Sim    6361
Name: Servico_telefone, dtype: int64
---------------------------------------
Servico_internet
Não    1526
Sim    5517
Name: Servico_internet, dtype: int64
```

Fonte: Autoria própria

Se visualizarmos os valores gerados na saída do comando da Figura 14, é possível observar que a quantidade de clientes que não possuem o serviço de telefone, é a mesma quantidade de clientes com valores faltantes nas colunas que detalham especificamente o serviço de telefone (Recarga_longa_distancia e Multiplas_linhas), o mesmo ocorre com clientes que não possuem o serviço de internet e as colunas que estratificam esse serviço (Tipo_internet, Media_mensal_download_GB, Servico_seguranca, Servico_backup, Servico_protecao_dispositivo, Servico_suporte_preferencial, Servico_tv, Servico_filmes, Servico_musica e Servico_ilimitado_dados), isso pode ser melhor observado na Figura 15, onde o comando realiza um filtro na base de dados, que retorna a quantidade de valores presentes em todas as linhas onde o cliente não possui o serviço de telefone, na Figura 16, o mesmo processo é realizado nos clientes que não possuem o serviço de internet. Em ambos os casos, as colunas que detalham o serviço aparecem sem valor preenchido.

Figura 15 - Filtro de clientes sem o serviço de telefone

```
print(df[df['Servico_telefone']=='Não'].count().sort_values(ascending=True))
```

```
Multiplas_linhas                0
Recarga_longa_distancia         0
Motivo_cancelamento           170
Categoria_cancelamento        170
Servico_suporte_preferencial  682
Servico_tv                    682
Servico_filmes                682
Servico_musica                682
Servico_ilimitado_dados       682
Tipo_contrato                 682
```

Fonte: Autoria própria

Figura 16 - Filtro de clientes sem o serviço de internet

```
print(df[df['Servico_internet']=='Não'].count().sort_values(ascending=True))
```

```
Servico_seguranca                0
Servico_ilimitado_dados          0
Servico_musica                   0
Servico_filmes                   0
Servico_tv                       0
Servico_suporte_preferencial     0
Servico_protecao_dispositivo     0
Servico_backup                   0
Media_mensal_download_GB         0
Tipo_internet                    0
Categoria_cancelamento         113
Motivo_cancelamento            113
Servico_telefone              1526
Situacao_cliente              1526
Total_cobranca_geral          1526
Cobrancas_longa_distancia     1526
Qtd_dados_extras              1526
Total_reembolsos              1526
Total_cobranca                1526
```

Fonte: Autoria própria

A falta de campos preenchidos se torna um problema para a nossa análise dos dados, bem como para a utilização de algoritmos de *machine learning*, uma vez que isso pode acarretar numa perda de contexto, e consequentemente, em uma análise incompleta e enviesada. Além disso, a presença de valores ausentes pode afetar a qualidade dos modelos de aprendizado de máquina, visto que muitos algoritmos não podem lidar diretamente com lacunas nos dados. Em algumas situações, pode ser considerada a remoção completa da linha que contém o dado faltante como uma solução, este não é o nosso caso, além de resultar numa perda quantitativa de dados, estaríamos excluindo da nossa base todos os clientes que não possuem os serviços de telefone e os serviços de internet.

Antes de escolhermos uma estratégia para solucionar nosso problema, é necessário entendermos os dados já preenchidos nessas colunas e seus contextos. Com exceção das colunas Recarga_longa_distancia e Media_mensal_download_GB que possuem valores numéricos informando o custo médio total de recargas de longa distância e o volume médio de download do cliente em gigabytes respectivamente, as outras colunas possuem os valores binários “Não” e “Sim”.

Para resolver esta situação, atribuiremos o valor “Não” nas colunas que possuem valores binários e o valor “0” nas duas colunas que possuem valores numéricos. A justificativa para essa escolha é que, uma vez que o cliente não possua o serviço de internet, consequentemente ele não terá um serviço de filmes ou músicas por exemplo, e isso pode ser replicado para as outras colunas que detalham esses

dois serviços, a Figura 17 mostra o comando realizado para imputar estes dados nas respectivas colunas.

Figura 17 - atribuindo dados nos campos com valores nulos

```
df.loc[df['Recarga_longa_distancia'].isnull(),'Recarga_longa_distancia'] = 0
df.loc[df['Multiplas_linhas'].isnull(),'Multiplas_linhas'] = 'Não'

df.loc[df['Tipo_internet'].isnull(),'Tipo_internet'] = 'Não'
df.loc[df['Media_mensal_download_GB'].isnull(),'Media_mensal_download_GB'] = 0
df.loc[df['Servico_seguranca'].isnull(),'Servico_seguranca'] = 'Não'
df.loc[df['Servico_backup'].isnull(),'Servico_backup'] = 'Não'
df.loc[df['Servico_protecao_dispositivo'].isnull(),'Servico_protecao_dispositivo'] = 'Não'
df.loc[df['Servico_suporte_preferencial'].isnull(),'Servico_suporte_preferencial'] = 'Não'
df.loc[df['Servico_tv'].isnull(),'Servico_tv'] = 'Não'
df.loc[df['Servico_filmes'].isnull(),'Servico_filmes'] = 'Não'
df.loc[df['Servico_musica'].isnull(),'Servico_musica'] = 'Não'
df.loc[df['Servico_ilimitado_dados'].isnull(),'Servico_ilimitado_dados'] = 'Não'
```

Fonte: Autoria própria

Continuando com nossa análise, agora temos apenas duas colunas com dados ausentes, que são Categoria_cancelamento e Motivo_cancelamento. A própria nomenclatura dessas colunas sugere que estão relacionadas à decisão do cliente de cancelar o serviço ou não. Essa informação pode ser encontrada na coluna Situacao_cliente, no entanto, antes de prosseguirmos com o processo de correção de dados faltantes, é necessário fazer um ajuste. Conforme indicado na Tabela 1, a coluna Situacao_cliente contém três valores distintos: “Cancelou”, “Continua cliente” e “Se tornou cliente”.

Dado que nosso principal objetivo é analisar o *churn* rate e tentar prever quais clientes têm maior propensão a solicitar o cancelamento, os dados relacionados aos clientes que aderiram ao serviço se tornam um elemento de interferência em nossa base de dados. Isso ocorre porque, quando se trata de análise dos cancelamentos, estamos particularmente interessados em analisar o histórico do cliente e em detectar padrões que o levaram a solicitar o término da prestação do serviço. No caso dos recém-contratantes, simplesmente não dispomos de um período de observação suficiente para discernir esses padrões, uma vez que não tiveram tempo de estabelecer um histórico de interações significativo com o serviço.

Na Figura 18 podemos observar a quantidade de clientes por sua situação, no momento que os dados foram coletados.

Figura 18 - Quantidade de clientes por situação

```
print(df.groupby(['Situacao_cliente'])['Situacao_cliente'].count())
```

```
Situacao_cliente
Cancelou              1869
Continua cliente      4720
Se tornou cliente      454
Name: Situacao_cliente, dtype: int64
```

Fonte: Autoria própria

Na Figura 19 pode ser visualizado o comando realizado para remover estes clientes.

Figura 19 - Removendo clientes que contrataram o serviço no período analisado

```
df = df[df['Situacao_cliente']!='Se tornou cliente']
df.reset_index(drop=True, inplace=True)
```

Fonte: Autoria própria

Retornando agora para as colunas Categoria_cancelamento e Motivo_cancelamento, podemos realizar um agrupamento contabilizando os valores contidos nessa coluna para entender melhor os seus dados, o comando utilizando está contido na Figura 20.

Figura 20 – Agrupamento e contagem dos valores das colunas

```
print(df.groupby(['Categoria_cancelamento'])['Categoria_cancelamento'].count())
print(df.groupby(['Motivo_cancelamento'])['Motivo_cancelamento'].count())
```

```
Categoria_cancelamento
Atitude         314
Concorrente     841
Insatisfação    321
Outros          182
Preço           211
Name: Categoria_cancelamento, dtype: int64
Motivo_cancelamento
Atitude do provedor do serviço                                94
Atitude do suporte                                           220
Concorrente fez uma oferta melhor                            311
Concorrente ofereceu maiores velocidades de download         100
Concorrente ofereceu mais limite de dados                    117
Concorrente possui dispositivos melhores                     313
Confiabilidade da rede                                        72
Experiencia ruim no suporte online                            31
Experiencia ruim no suporte por telefone                      12
Faleceu                                                        6
Falta de autoatendimento no site                              29
Falta de velocidade de download/upload por um valor acessível 30
Gama limitada de serviços                                     37
Insatisfação com o produto                                    77
Insatisfação com o serviço                                    63
Mudou-se                                                      46
Não sabe dizer                                               130
Preço muito alto                                              78
Tarifas de longa distancia                                    64
Tarifas extras de dados                                       39
Name: Motivo_cancelamento, dtype: int64
```

Fonte: Autoria própria

Com base nas informações contidas no arquivo *telecom_data_dictionary*, é possível inferir que os valores contidos na coluna Motivo_cancelamento são uma estratificação dos valores contidos na coluna Categoria_cancelamento, isto pode ser comprovado com a saída do comando mostrado na Figura 21, neste comando também foi realizado o agrupamento por Situacao_cliente, comprovando que somente os clientes que cancelaram possuem dados nestas colunas.

Figura 21 - Agrupamento em conjunto das colunas

```
print(df.groupby(['Situacao_cliente','Categoria_cancelamento','Motivo_cancelamento'])['Motivo_cancelamento'].count())
```

```
Situacao_cliente  Categoria_cancelamento  Motivo_cancelamento
Cancelou          Atitude                 Atitude do provedor do serviço                          94
                                          Atitude do suporte                                     220
                  Concorrente             Concorrente fez uma oferta melhor                      311
                                          Concorrente ofereceu maiores velocidades de download   100
                                          Concorrente ofereceu mais limite de dados              117
                                          Concorrente possui dispositivos melhores               313
                  Insatisfação            Confiabilidade da rede                                  72
                                          Experiencia ruim no suporte online                      31
                                          Experiencia ruim no suporte por telefone                12
                                          Falta de autoatendimento no site                        29
                                          Gama limitada de serviços                               37
                                          Insatisfação com o produto                              77
                                          Insatisfação com o serviço                              63
                  Outros                  Faleceu                                                  6
                                          Mudou-se                                                46
                                          Não sabe dizer                                         130
                  Preço                   Falta de velocidade de download/upload por um valor acessível  30
                                          Preço muito alto                                        78
                                          Tarifas de longa distancia                              64
                                          Tarifas extras de dados                                 39
Name: Motivo_cancelamento, dtype: int64
```

Fonte: Autoria própria

Uma vez que os valores em branco nas colunas mencionadas anteriormente são referentes aos clientes que permaneceram com o serviço, atribuiremos o valor “Permaneceu cliente” em ambas as colunas (Figura 22), pois assim teremos todos os valores preenchidos sem nenhuma perca de contexto ou risco de enviesar a nossa base de dados, já que o cliente de fato permaneceu com o serviço.

Figura 22 - Atribuição dos valores nas linhas com dados ausentes

```
df.loc[df['Categoria_cancelamento'].isnull(),'Categoria_cancelamento'] = 'Permaneceu cliente'
df.loc[df['Motivo_cancelamento'].isnull(),'Motivo_cancelamento'] = 'Permaneceu cliente'
```

Fonte: Autoria própria

Finalizando esta etapa, ao executar o comando *df.count* (Figura 23), sua saída nos mostra uma base de dados com 6.589 registros em que todas as colunas estão devidamente ajustadas e sem valores faltantes.

Figura 23 - Contagem dos valores por coluna

```
df.count()
```

```
Genero                          6589
Idade                           6589
Casado                          6589
Qtd_dependentes                 6589
Cidade                          6589
Codigo_postal                   6589
Populacao                       6589
Latitude                        6589
Longitude                       6589
Qtd_indicações                  6589
Meses_na_base                   6589
Ultima_oferta                   6589
Servico_telefone                6589
Recarga_longa_distancia         6589
Multiplas_linhas                6589
Servico_internet                6589
Tipo_internet                   6589
Media_mensal_download_GB        6589
Servico_seguranca               6589
Servico_backup                  6589
Servico_protecao_dispositivo    6589
Servico_suporte_preferencial    6589
Servico_tv                      6589
Servico_filmes                  6589
Servico_musica                  6589
Servico_ilimitado_dados         6589
Tipo_contrato                   6589
Faturamento_sem_papel           6589
Tipo_pagamento                  6589
Cobranca_mensal                 6589
Total_cobranca                  6589
Total_reembolsos                6589
Qtd_dados_extras                6589
Cobrancas_longa_distancia       6589
Total_cobranca_geral            6589
Situacao_cliente                6589
Categoria_cancelamento          6589
Motivo_cancelamento             6589
dtype: int64
```

Fonte: Autoria própria

4 ANÁLISE E EXPLORAÇÃO DOS DADOS

Partindo agora para a etapa de análise exploratória dos dados, iremos avançar na investigação detalhada do nosso *dataframe* com o objetivo de identificar padrões, tendências e fatores que moldam a decisão dos clientes de permanecerem ou cancelaram seus serviços. Esta análise aprofundada, permitirá a formulação de estratégias de retenção de clientes mais eficazes e aprimoramento da qualidade dos serviços oferecidos pelas empresas de telecomunicações.

Primeiramente, iremos importar a biblioteca *matplotlib*, ela será fundamental nesta etapa pois através dela conseguiremos construir gráficos, que facilitarão tanto nossas análises quanto as tomadas de decisões, o comando utilizado para importar esta biblioteca pode ser visualizado na Figura 24.

Figura 24 - Importando o *Matplotlib*

```
import matplotlib.pyplot as plt
```

Fonte: Autoria própria

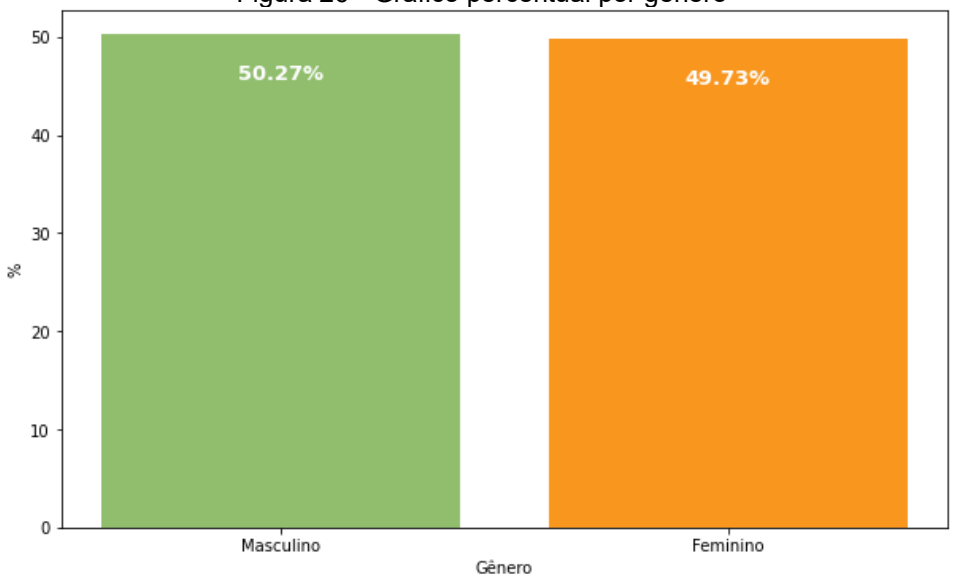
Com a biblioteca importada, podemos iniciar a construção dos nossos gráficos, iremos começar visualizando o percentual de clientes por gênero em nossa base, o código utilizado está registrado na Figura 25. Já na Figura 26 temos o gráfico gerado pelo código em questão.

Figura 25 - Código para geração do gráfico de percentual por gênero

```
#### Contabilizando os valores que serão plotados no gráfico ###
cont = df['Genero'].value_counts()
total = cont.sum()
porcentagens = (cont / total) * 100

### Ajustes de cores e tamanho do gráfico ###
cores_personalizadas = ['#90BE6D', '#F8961E']
fig, ax = plt.subplots(figsize=(10, 6))
bars = plt.bar(cont.index, porcentagens.values, color=cores_personalizadas)

### Ajustes de design do rótulo das colunas do grafico ###
for bar, percentage in zip(bars, porcentagens):
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() * 0.90,
        f'{percentage:.2f}%',
        ha='center',
        va='bottom',
        fontsize=13,
        color='white',
        fontweight='bold')

### Edição de titulos e labels do gráfico ###
plt.title('Percentual por Gênero')
plt.xlabel('Gênero')
plt.ylabel('%')
ax.grid(visible=False)

### Comando para exibição do gráfico ###
plt.show()
```

Fonte: Autoria própria

Figura 26 - Gráfico percentual por gênero



Fonte: Autoria própria

É possível observar que possuímos uma base de dados muito equilibrada nesse aspecto. Se cruzarmos esses dados com outras colunas, podemos estratificar ainda mais essa informação, na Figura 27 por exemplo, utilizaremos um código que irá criar faixas etárias a partir da coluna “Idade”, a fim de agrupar os dados em categorias mais amplas. Esse procedimento simplificará a análise, permitindo-nos identificar padrões de maneira mais eficaz.

Figura 27 - Criação de faixas etárias

```
df_default = df.loc[0:,['Genero','Idade']] ### Separando as colunas que serão utilizadas

faixa_etaria_bins = [0,18, 20, 25, 30, 35, 40,45, 50,55, 60,65, 70,75,80, 100] ### Definindo agrupamentos das idades
faixa_etaria_labels = ['Menos de 18', '18-20', '21-25','26-30', '31-35', '36-40', '41-45',
                       '46-50', '51-55', '56-60', '61-65','66-70','71-75','76-80', 'Mais de 80']

df_default['Faixa Etária'] = pd.cut(df_default['Idade'], bins=faixa_etaria_bins, labels=faixa_etaria_labels)

### Contabilizando as quantidades em colunas separadas
result = df_default.groupby(['Faixa Etária', 'Genero']).size().unstack().reset_index()
result = result.rename_axis(None, axis=1)

result
```

	Faixa Etária	Feminino	Masculino
0	Menos de 18	0	0
1	18-20	124	106
2	21-25	319	302
3	26-30	280	293
4	31-35	290	292
5	36-40	276	320
6	41-45	313	291
7	46-50	301	310
8	51-55	288	299
9	56-60	296	290
10	61-65	278	287
11	66-70	184	164
12	71-75	167	180
13	76-80	161	178
14	Mais de 80	0	0

Fonte: Autoria própria

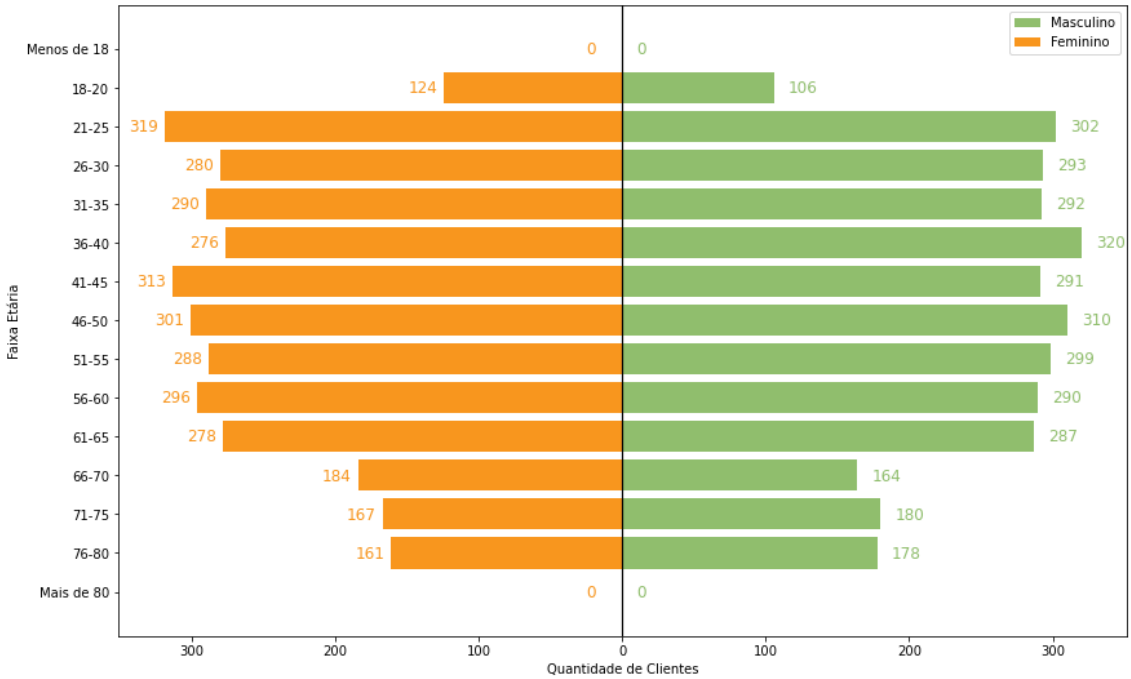
Posteriormente, exploraremos a composição de gênero dos clientes dentro de cada faixa etária, permitindo-nos visualizar essas informações de maneira mais representativa por meio de um gráfico de pirâmide. Na Figura 28 podemos visualizar o código utilizado para criação gráfico e na Figura 29 o gráfico em si.

Figura 28 - Código para criação do gráfico de pirâmide do gênero do cliente

```
fig, ax = plt.subplots(figsize=(14, 9))
ax.barh(result['Faixa Etária'], result['Masculino'], color='#90BE6D', label='Masculino')
ax.barh(result['Faixa Etária'], [-x for x in result['Feminino']], color='#F8961E', label='Feminino')

### Atribuindo linha vertical no meio
ax.axvline(0, color='black', linewidth=1.2)

### Rótulo dos eixos
ax.set_xlabel('Quantidade de Clientes')
ax.set_ylabel('Faixa Etária')

ax.legend() ### Adicionando uma legenta
ax.invert_yaxis() ### Invertendo a ordem do eixo y para as idades mais jovens aparecerem em cima

for i in range(len(result['Faixa Etária'])): ### Ajuste dos rótulos das colunas
    ax.text(result['Masculino'][i] + 10, i, str(result['Masculino'][i]), va='center', fontsize=12, color='#90BE6D')
    ax.text(-result['Feminino'][i] - 25, i, str(result['Feminino'][i]), va='center', fontsize=12, color='#F8961E')

custom_ticks = [-300, -200, -100, 0, 100, 200, 300 ]
custom_labels = [abs(i) for i in custom_ticks]  ### Ajuste para que os valores fiquem postivos
plt.xticks(custom_ticks, custom_labels)
ax.grid(visible=False)
plt.show()
```

Fonte: Autoria própria

Figura 29 - Gráfico de gênero por idade



Fonte: Autoria própria

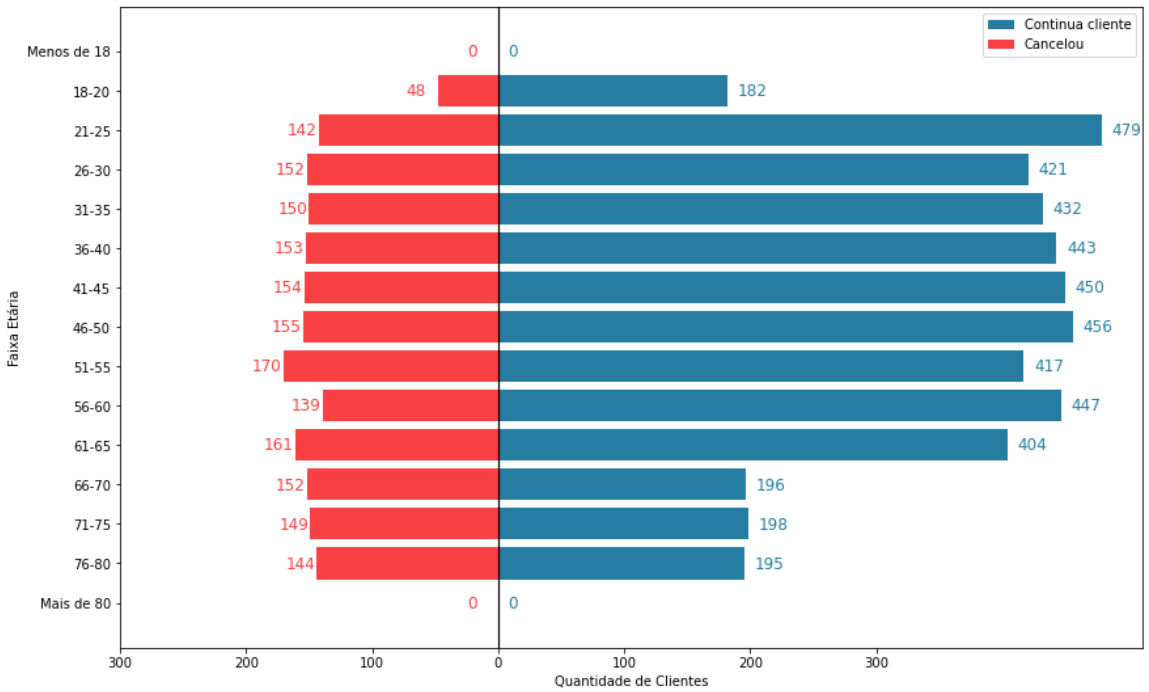
Neste nível de estratificação conseguimos observar que as faixas etárias de 66 até 80 anos possuem uma quantidade de clientes menor em relação as outras. Iremos agora utilizar esta mesma estrutura e estratificar a coluna “Situacao_cliente”, o código para criar este gráfico pode ser visualizado na Figura 30.

Figura 30 - Código para criação do gráfico de pirâmide da situação do cliente

```
df_default = df.loc[0:,['Situacao_cliente','Idade']]
df_default['Faixa Etária'] = pd.cut(df_default['Idade'], bins=faixa_etaria_bins, labels=faixa_etaria_labels)
result = df_default.groupby(['Faixa Etária', 'Situacao_cliente']).size().unstack().reset_index()
result = result.rename_axis(None, axis=1)

fig, ax = plt.subplots(figsize=(14, 9))
ax.barh(result['Faixa Etária'], result['Continua cliente'], color='#277DA1', label='Continua cliente')
ax.barh(result['Faixa Etária'], [-x for x in result['Cancelou']], color='#F94144', label='Cancelou')
ax.axvline(0, color='black', linewidth=1.2)
ax.set_xlabel('Número de Pessoas')
ax.set_ylabel('Faixa Etária')
ax.legend()
ax.invert_yaxis()

for i in range(len(result['Faixa Etária'])):
    ax.text(result['Continua cliente'][i] + 8, i, str(result['Continua cliente'][i]),
            va='center', fontsize=12, color='#277DA1')
    ax.text(-result['Cancelou'][i] - 25, i, str(result['Cancelou'][i]),
            va='center', fontsize=12, color='#F94144')

custom_ticks = [-300, -200, -100, 0, 100, 200, 300 ]
custom_labels = [abs(i) for i in custom_ticks]  ### Ajustando para que os valores fiquem positivos nos rótulos

plt.xticks(custom_ticks, custom_labels)
ax.grid(visible=False)
plt.show()
```

Fonte: Autoria própria

Figura 31 - Gráfico de pirâmide da situação do cliente



Fonte: Autoria própria

Na Figura 31 temos o gráfico gerado, nele é possível observar que temos uma quantidade bem menor de clientes que cancelaram em relação aos que continuam com o serviço, algo que já tínhamos observado anteriormente (Figura 18), porém, é importante nos atentarmos que o volume referente as faixas de 66 a 80 anos

diminuíram pouco em relação as outras, o que pode indicar uma maior tendência desse público em cancelar o serviço.

Para melhor visualizarmos essa hipótese, iremos verificar a proporção de clientes que cancelam por faixa etária. Primeiramente iremos identificar qual o valor que corresponde a proporção geral da nossa base (Figura 32).

Figura 32 - Encontrando o valor proporcional de clientes que cancelaram

```
clientes_cancelados = df[df['Situacao_cliente']=='Cancelou']['Situacao_cliente'].count()
clientes_retidos = df[df['Situacao_cliente']=='Continua cliente']['Situacao_cliente'].count()
val_med = clientes_cancelados/(clientes_cancelados+clientes_retidos)
val_med
```

```
0.2836545758081651
```

Fonte: Autoria própria

Em seguida, realizaremos o mesmo procedimento para todos os valores no *dataframe* com as faixas etárias (Figura 33).

Figura 33 - Calculando valor proporcional por faixa etária

```
df_rel = result[['Cancelou','Continua cliente']]

df_rel['proporcional'] = df_rel['Cancelou']/(df_rel['Cancelou']+df_rel['Continua cliente'])
df_rel['proporcional'] = df_rel['proporcional'].fillna(0)
df_rel['Faixa Etária'] = result['Faixa Etária']

faixas_vazias = ['Menos de 18','Mais de 80']
df_rel = df_rel[~df_rel['Faixa Etária'].isin(faixas_vazias)]### Retirando faixas etarias que não possuem clientes

df_rel
```

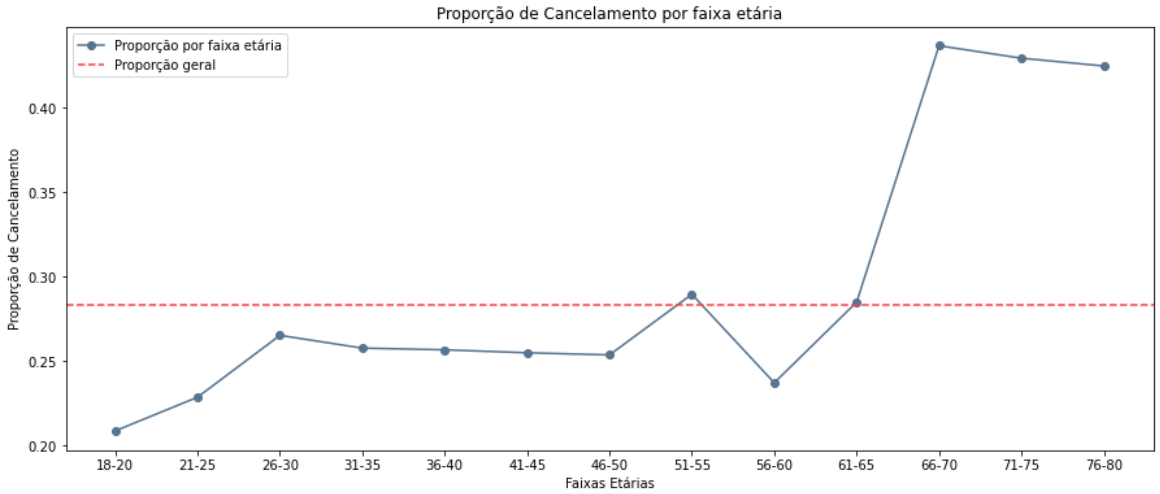
	Cancelou	Continua cliente	proporcional	Faixa Etária
1	48	182	0.208696	18-20
2	142	479	0.228663	21-25
3	152	421	0.265271	26-30
4	150	432	0.257732	31-35
5	153	443	0.256711	36-40
6	154	450	0.254967	41-45
7	155	456	0.253682	46-50
8	170	417	0.289608	51-55
9	139	447	0.237201	56-60
10	161	404	0.284956	61-65
11	152	196	0.436782	66-70
12	149	198	0.429395	71-75
13	144	195	0.424779	76-80

Fonte: Autoria própria

Com isso, podemos então criar um gráfico comparando o proporcional de cancelamento de cada faixa etária com o proporcional de cancelamento geral da nossa base de dados (Figura 34).

Figura 34 - Gráfico de proporção de cancelamento por faixa etária

```
plt.figure(figsize=(15, 6))
plt.plot(df_rel['Faixa Etária'], df_rel['proporcional'],
         label='Proporção por faixa etária', marker='o', linestyle='-', color='#577590')
plt.axhline(y=val_med, color='#F94144', linestyle='--', label='Proporção geral')

plt.xlabel('Faixas Etárias')
plt.ylabel('Proporção de Cancelamento')
plt.title('Proporção de Cancelamento por faixa etária')

plt.legend()

plt.grid(visible=False)
plt.show()
```

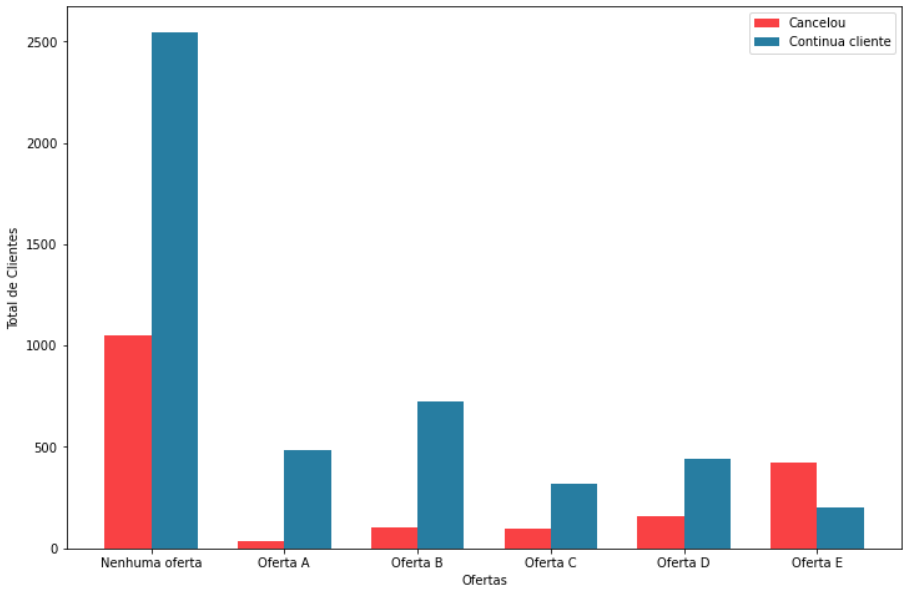


Fonte: Autoria própria

Através do gráfico, fica bem perceptivo o quanto os proporcionais das últimas três faixas etárias são bem mais altos em relação aos outros, chegando a valores acima de 40%. Tendo isso em mente, pode ser interessante para a empresa buscar a retenção deste perfil de cliente. Se analisarmos a retenção do cliente baseado na última oferta recebida temos o seguinte cenário (Figura 35).

Figura 35 - Retenção do cliente por oferta recebida

```
df_default = df.loc[0:,['Situacao_cliente','Ultima_oferta']]
result = df_default.groupby(['Ultima_oferta', 'Situacao_cliente']).size().unstack(fill_value=0)
result = result.reset_index()

lrg_bar = 0.35
índices = range(len(result['Ultima_oferta']))
plt.figure(figsize=(12, 8))
plt.bar([i - lrg_bar/2 for i in índices], result['Cancelou'], lrg_bar, color='#F94144',label='Cancelou')
plt.bar([i + lrg_bar/2 for i in índices], result['Continua cliente'], lrg_bar, color='#277DA1',label='Continua cliente')

plt.xticks(índices, result['Ultima_oferta'])
plt.legend()
plt.ylabel('Total de Clientes')
plt.xlabel('Ofertas')
plt.show()
```



Fonte: Autoria própria

Uma ponderação se faz necessária nesse momento em relação aos dados da coluna “Ultimas_ofertas”, onde, por mais que os nomes das ofertas possam sugerir isto, com as informações que temos em mãos atualmente não é possível concluir que as ofertas possuem uma relação de ordem entre si, dessa forma iremos trata-las como distintas uma da outra.

Voltando a atenção para o gráfico na Figura 35 algo que chama a atenção é a alta quantidade de clientes que cancelam o serviço sem nenhuma oferta recebida, sugerindo que esses clientes nem mesmo aceitaram negociar o cancelamento, isso por si só já é uma informação valiosa. Posteriormente iremos analisar a coluna “Motivos_cancelamento” para entendermos melhor esse dado. Por enquanto, iremos nos aprofundar um pouco mais na retenção do cliente, dado sua faixa etária e última oferta recebida.

Com o código apresentado na Figura 36, obtemos como saída uma tabela com o percentual de clientes que permaneceram na base após uma determinada oferta.

Figura 36 - Geração de tabela com a retenção por faixa etária e última oferta

```
df_default = df.loc[0:,['Ultima_oferta','Situacao_cliente','Idade']]
df_default['Faixa Etária'] = pd.cut(df_default['Idade'], bins=faixa_etaria_bins, labels=faixa_etaria_labels)

result = df_default.groupby(['Ultima_oferta','Faixa Etária', 'Situacao_cliente']).size().unstack().reset_index()
result = result.rename_axis(None, axis=1)

faixas_vazias = ['Menos de 18','Mais de 80']
result = result[~result['Faixa Etária'].isin(faixas_vazias)] ### Removendo faixas etarias sem clientes
result = result[result['Ultima_oferta']!='Nenhuma oferta'] ### Removendo clientes que nao receberam ofertas

result['% clientes que permaneceram'] = result['Continua cliente']/(result['Continua cliente']+result['Cancelou'])*100
result
```

	Ultima_oferta	Faixa Etária	Cancelou	Continua cliente	% clientes que permaneceram
16	Oferta A	18-20	0	18	100.000000
17	Oferta A	21-25	6	61	91.044776
18	Oferta A	26-30	6	42	87.500000
19	Oferta A	31-35	3	37	92.500000
20	Oferta A	36-40	1	46	97.872340
...	...	...	...	...	...
84	Oferta E	56-60	34	13	27.659574
85	Oferta E	61-65	47	15	24.193548
86	Oferta E	66-70	20	13	39.393939
87	Oferta E	71-75	27	4	12.903226
88	Oferta E	76-80	20	7	25.925926

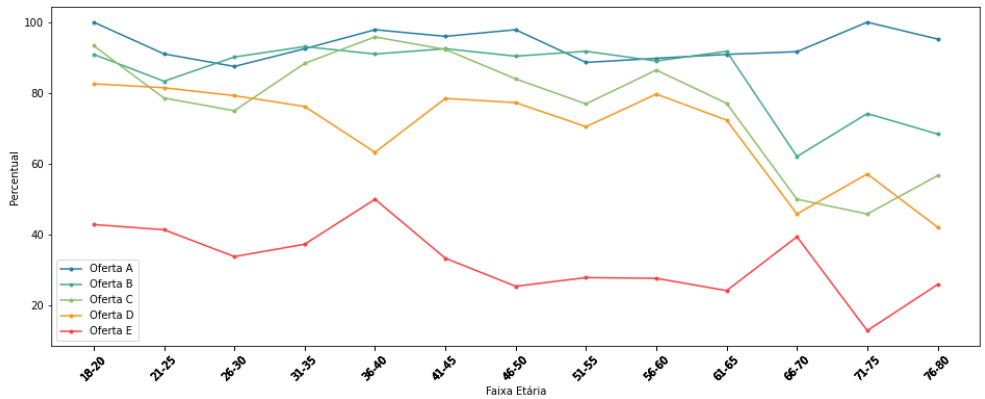
65 rows × 5 columns

Fonte: Autoria própria

Em seguida, a partir da tabela gerada é possível construir um gráfico (Figura 37) onde cada linha representa o percentual de retenção atingido por cada plano através das faixas etárias.

Figura 37 - Gráfico de retenção de cliente por faixa etária e último plano ofertado

```
plt.figure(figsize=(16, 6))
cores_paleta = ['#277DA1','#43AA8B','#90BE6D','#F8961E','#F94144']
categorias_unicas = result['Ultima_oferta'].unique()

for i, (categoria) in enumerate(categorias_unicas):
    cor = cores_paleta[i % len(cores_paleta)]
    dados_categoria = result[result['Ultima_oferta'] == categoria]

    plt.plot(dados_categoria['Faixa Etária'], dados_categoria['% clientes que permaneceram'],
             label=categoria,marker='.',color=cor)

plt.xlabel('Faixa Etária')
plt.ylabel('Percentual')
plt.legend()
plt.xticks(result['Faixa Etária'], rotation=45)
plt.grid(False)
plt.show()
```



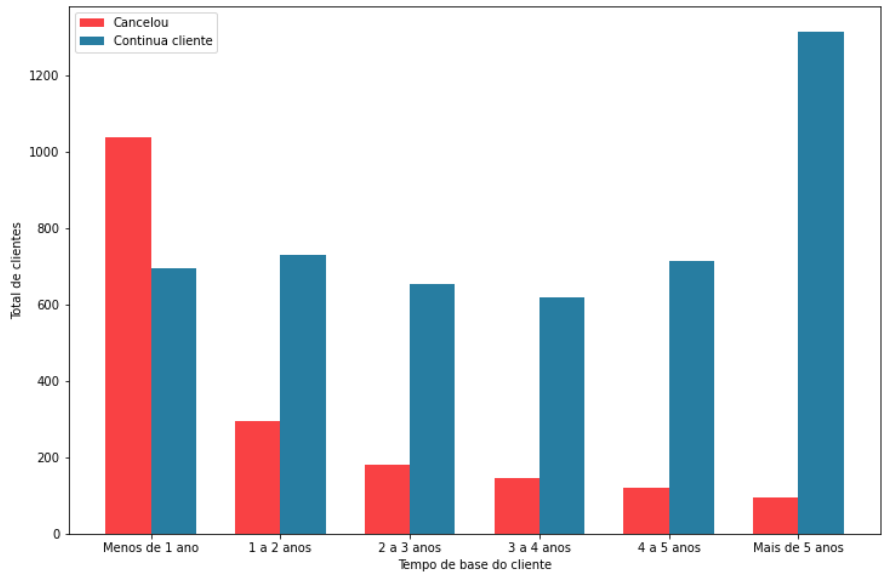
Fonte: Autoria própria

O gráfico nos mostra que os planos têm uma efetividade diferente de acordo com a faixa etária do cliente ao qual está sendo ofertado, em alguns casos chegando próximo aos 100% de retenção. Se analisarmos as três últimas faixas etárias por exemplo, faz mais sentido tentar fidelizar o cliente com a “oferta A”, já que possui uma probabilidade de retenção do cliente maior que as outras ofertas. Um fato importante que deve ser levado em consideração, é que esta é uma análise puramente estatística, pois como não temos informações sobre as diferenças dos planos não é possível fazer uma análise mais profunda. É possível que a “oferta A” seja montada com serviços que atendem melhor este público de maior idade, o que explicaria essas diferenças nas retenções entre as ofertas.

Continuando com a análise da nossa tabela, temos a coluna que informa o total de meses do cliente na base da empresa, estratificando esse dado e detalhando entre clientes que cancelaram e clientes que permaneceram temos a seguinte situação (Figura 38).

Figura 38 - Situação do cliente x Tempo de base

```
df_default = df.loc[0:,['Situacao_cliente','Meses_na_base']]
faixa_tempo_de_base = [0,12, 24, 36, 48, 60,1000]
faixa_tempo_labels = ['Menos de 1 ano', '1 a 2 anos', '2 a 3 anos','3 a 4 anos', '4 a 5 anos','Mais de 5 anos']
df_default['Faixa_tempo_base'] = pd.cut(df_default['Meses_na_base'], bins=faixa_tempo_de_base, labels=faixa_tempo_labels)
result = df_default.groupby(['Faixa_tempo_base', 'Situacao_cliente']).size().unstack().reset_index()
result = result.reset_index(drop=True)
tam_bar = 0.35
índices = range(len(result['Faixa_tempo_base']))
plt.figure(figsize=(12, 8))
plt.bar([i - tam_bar/2 for i in índices], result['Cancelou'], tam_bar, label='Cancelou',color='#F94144')
plt.bar([i + tam_bar/2 for i in índices], result['Continua cliente'], tam_bar, label='Continua cliente',color='#277DA1')
plt.xticks(índices, result['Faixa_tempo_base'])
plt.ylabel('Total de clientes')
plt.xlabel('Tempo de base do cliente')
plt.legend()
plt.show()
```



Fonte: Autoria própria

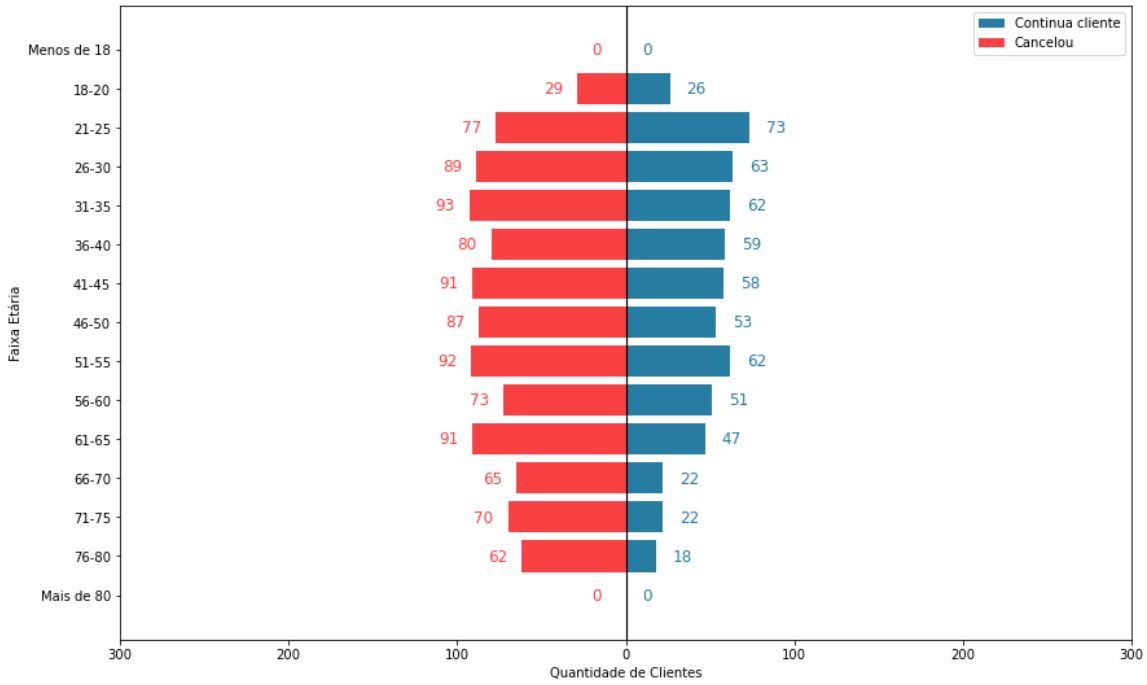
Algo bem notório no gráfico apresentado, é a alta quantidade de clientes cancelando o serviço com menos de um ano de serviço. Estratificando esse dado pelas faixas etárias (Figura 39) temos o seguinte cenário (Figura 40).

Figura 39 - Estratificando os clientes com menos de 12 meses

```
df_default = df[df['Meses_na_base']<12].loc[0:,['Situacao_cliente','Idade']]
df_default['Faixa Etária'] = pd.cut(df_default['Idade'], bins=faixa_etaria_bins, labels=faixa_etaria_labels)
result = df_default.groupby(['Faixa Etária', 'Situacao_cliente']).size().unstack().reset_index()
result = result.reset_index(drop=True)
fig, ax = plt.subplots(figsize=(14, 9))
ax.barh(result['Faixa Etária'], result['Continua cliente'], color='#277DA1', label='Continua cliente')
ax.barh(result['Faixa Etária'], [-x for x in result['Cancelou']], color='#F94144', label='Cancelou')
ax.axvline(0, color='black', linewidth=1.2)
ax.set_xlabel('Quantidade de Clientes')
ax.set_ylabel('Faixa Etária')
ax.legend()
ax.invert_yaxis()

for i in range(len(result['Faixa Etária'])):
    ax.text(result['Continua cliente'][i] + 10, i, str(result['Continua cliente'][i]),
            va='center', fontsize=12, color='#277DA1')
    ax.text(-result['Cancelou'][i] - 20, i, str(result['Cancelou'][i]),
            va='center', fontsize=12, color='#F94144')

custom_ticks = [-300, -200, -100, 0, 100, 200, 300 ]
custom_labels = [abs(i) for i in custom_ticks]
plt.xticks(custom_ticks, custom_labels)
ax.grid(visible=False)
plt.show()
```

Fonte: Autoria própria

Figura 40 - Situação de clientes com menos de 12 meses por faixa etária



Fonte: Autoria própria

Novamente temos uma situação, em que nas faixas etárias de 66 a 80 anos chama a atenção pela alta quantidade de cancelamentos em relação a quantidade de clientes que permanecem na base, comprovando que este perfil de cliente deve ser um dos focos de atenção da empresa nas suas estratégias de retenção.

Podemos aprofundar nossa análise considerando o perfil do cliente em relação à sua localidade e ao tamanho populacional dessa área. Para isso, é necessário realizar uma pequena modificação em nossa tabela. A coluna de população está atualmente vinculada à coluna de código postal, que por sua vez, possui uma relação de um para muitos com a coluna de cidades. Em outras palavras, um código postal está associado a uma única cidade, enquanto uma cidade pode abranger diversos códigos postais em sua região. Essa correção permitirá uma análise mais precisa do perfil do cliente com base na demografia da cidade em que reside. Com o ajuste realizado não será mais necessária a coluna de código postal, uma vez que teremos a população por cidade. O código utilizado para realizar esse ajuste está contido na Figura 41.

Figura 41 - Ajuste dos valores da população

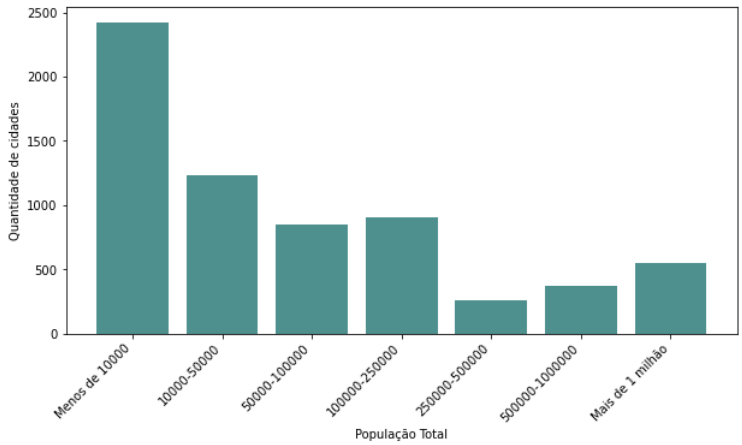
```
df_default = df.loc[0:,['Cidade','Codigo_postal','Populacao']]
df_cid = df_default.drop_duplicates(
    subset=['Cidade', 'Codigo_postal'])[['Cidade', 'Populacao']].groupby('Cidade').sum().reset_index()

df_cid = df_cid.sort_values('Populacao',ascending=False)
df = df.drop(['Codigo_postal','Populacao'], axis=1)
df = pd.merge(df, df_cid,left_on=['Cidade'],right_on=['Cidade'],how='left')
colunas = df.columns.tolist()
colunas.insert(5, colunas.pop(colunas.index('Populacao')))
df = df[colunas]
```

Fonte: Autoria própria

Em seguida, para uma melhor visualização dos dados, iremos agrupar as cidades com base na população total (Figura 42).

Figura 42 - Cidades agrupadas com base no total da população

```
df_default = df.loc[0:,['Cidade','Populacao']]
faixa_populacional = [0,10000, 50000, 100000, 250000, 500000,1000000,10000000000]
faixa_populacional_labels = ['Menos de 10000', '10000-50000', '50000-100000',
                             '100000-250000', '250000-500000', '500000-1000000','Mais de 1 milhão']
df_default['Faixa Populacional'] = pd.cut(df_default['Populacao'], bins=faixa_populacional, labels=faixa_populacional_labels)
result = df_default.groupby(['Faixa Populacional']).count().reset_index()
result = result.rename_axis(None, axis=1)

plt.figure(figsize=(10, 5))
plt.bar(result['Faixa Populacional'], result['Populacao'], color='#4D908E')
plt.ylabel('Quantidade de cidades')
plt.xlabel('População Total')
plt.xticks(rotation=45, ha='right')
plt.show()
```



Fonte: Autoria própria

Neste momento, usaremos essa estrutura de agrupamento para a criação de alguns gráficos, onde analisaremos a média de determinados valores baseados nesses grupos. Além disso, é preciso destacar que os gráficos a seguir serão feitos com base nesses agrupamentos, foi ajustado o valor inicial do eixo Y apenas para deixar mais visível as variações dos valores.

Para agilizar a criação dos gráficos, foi definida uma função (Figura 43) que receberá como parâmetros a coluna que irá utilizar e o valor inicial do eixo Y.

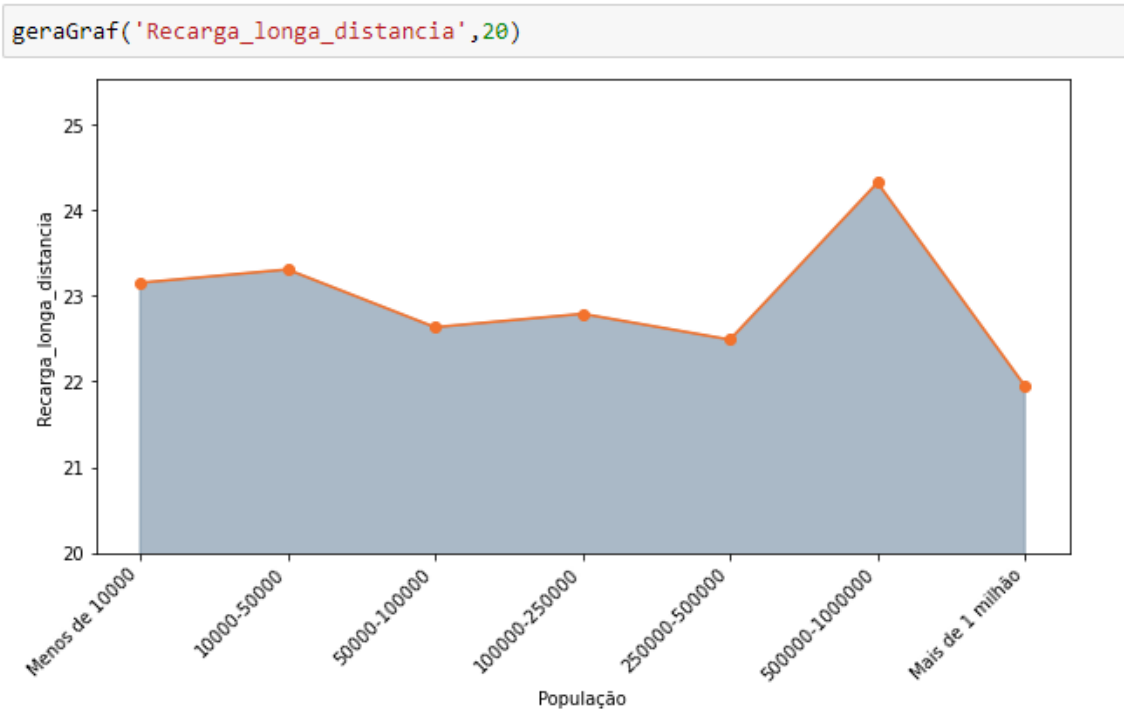
Figura 43 - Função para gerar os gráficos

```
def geraGraf(coluna,val_ini_y):
    df_default = df.loc[0:,['Cidade','Populacao',coluna]]
    faixa_populacional = [0,10000, 50000, 100000, 250000, 500000,1000000,10000000000]
    faixa_populacional_labels = ['Menos de 10000', '10000-50000', '50000-100000',
                                 '100000-250000', '250000-500000', '500000-1000000','Mais de 1 milhão']
    df_default['Faixa Populacional'] = pd.cut(df_default['Populacao'],
                                              bins=faixa_populacional, labels=faixa_populacional_labels)
    result = df_default.groupby(['Faixa Populacional'])[coluna].mean().reset_index()
    result = result.rename_axis(None, axis=1)
    plt.figure(figsize=(10, 5))
    plt.plot(result['Faixa Populacional'], result[coluna], color='#F3722C', marker='o')
    plt.fill_between(result['Faixa Populacional'], result[coluna], color='#577590', alpha=0.5)
    plt.ylim(bottom=val_ini_y)  ### Definindo o valor inicial do eixo Y
    plt.ylabel(coluna)
    plt.xlabel('População')
    plt.xticks(rotation=45, ha='right')
    plt.show()
```

Fonte: Autoria própria

Começando pela coluna de recargas de longa distância obtemos o seguinte gráfico (Figura 44):

Figura 44 - Ticket médio de recarga de longa distância



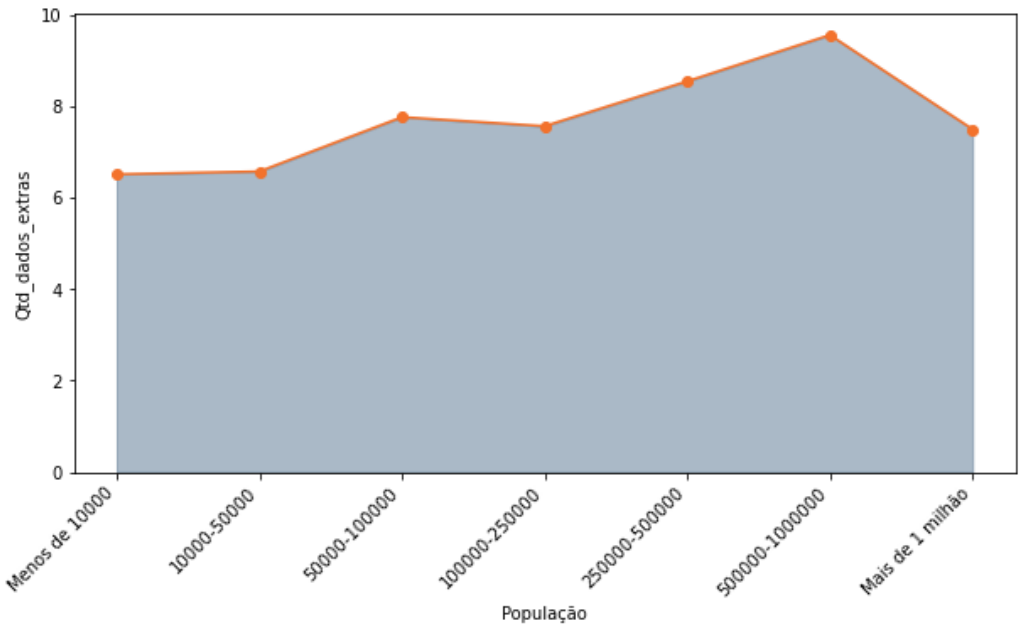
Fonte: Autoria própria

Com exceção do valor registrado no agrupamento de cidades com população entre 500000 e 1000000, o gráfico apresenta uma tendência de diminuição do ticket médio gasto em recarga de longa distância, em relação ao total da população da cidade, uma possível explicação para isto pode ser que, em teoria, um cliente de uma cidade pequena pode precisar ligar para outras cidades mais do que um cliente em uma cidade grande.

Por outro lado, a solicitação para pacotes de dados extras apresenta o comportamento contrário, como pode ser visto na Figura 45.

Figura 45 – Quantidade de pacotes de dados extras solicitados

```
geraGraf('Qtd_dados_extras',0)
```

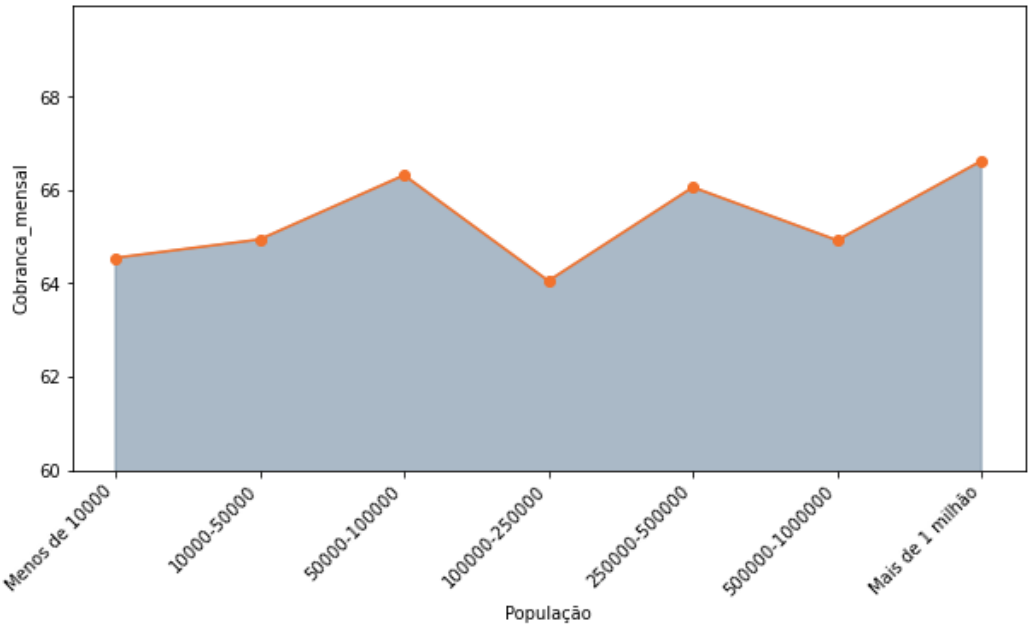


Fonte: Autoria própria

Referente ao ticket médio pago pelo cliente (Figura 46) também é possível perceber uma pequena tendência de crescimento em cidades mais populosas.

Figura 46 - Ticket médio pago pelo cliente

```
geraGraf('Cobranca_mensal',60)
```

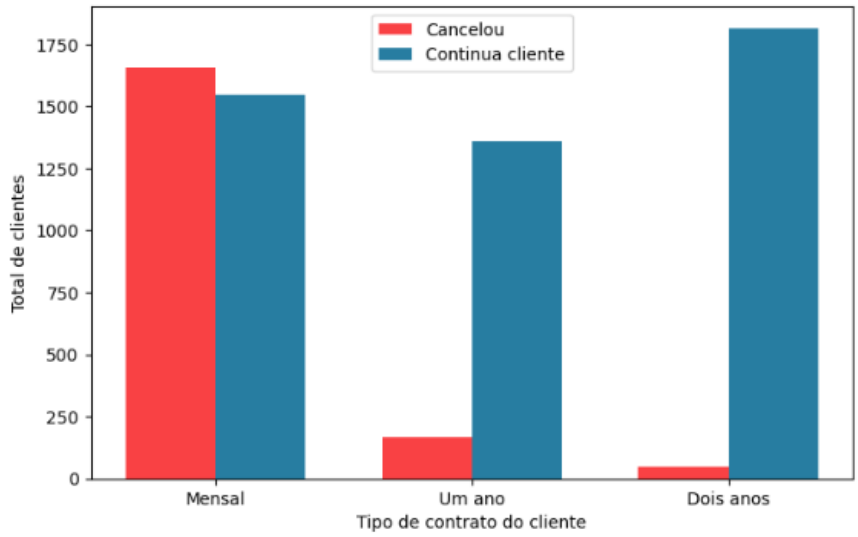


Fonte: Autoria própria

Em seguida, iremos analisar a coluna com as informações do tipo do contrato, a Figura 47 mostra o gráfico bem como o código utilizado para gerá-lo.

Figura 47 - Geração de gráfico de situação de cliente dado seu tipo de contrato

```
df_default = df.loc[0:,['Tipo_contrato','Situacao_cliente','Cidade']]
result = df_default.groupby(['Tipo_contrato','Situacao_cliente'])['Cidade'].count().reset_index()
result = result.rename(columns={'Cidade':'Qtd'},inplace=False)
result = result.pivot(index='Tipo_contrato', columns='Situacao_cliente', values='Qtd').reset_index()
result.columns.name = None  # Para remover o nome da coluna 'Regiao'
result.columns = ['Tipo_contrato', 'Qtd_cancelado', 'Qtd_permanece_cliente']
ordem = ['Mensal', 'Um ano','Dois anos']
result['Tipo_contrato'] = pd.Categorical(result['Tipo_contrato'], categories=ordem, ordered=True)
result = result.sort_values(by='Tipo_contrato')
tam_bar = 0.35
índices = range(len(result['Tipo_contrato']))
plt.figure(figsize=(8, 5))
plt.bar([i - tam_bar/2 for i in índices], result['Qtd_cancelado'], tam_bar, label='Cancelou',color='#F94144')
plt.bar([i + tam_bar/2 for i in índices], result['Qtd_permanece_cliente'], tam_bar, label='Continua cliente',color='#277DA1')
plt.xticks(índices, result['Tipo_contrato'])
plt.ylabel('Total de clientes')
plt.xlabel('Tipo de contrato do cliente')
plt.legend()
plt.show()
```



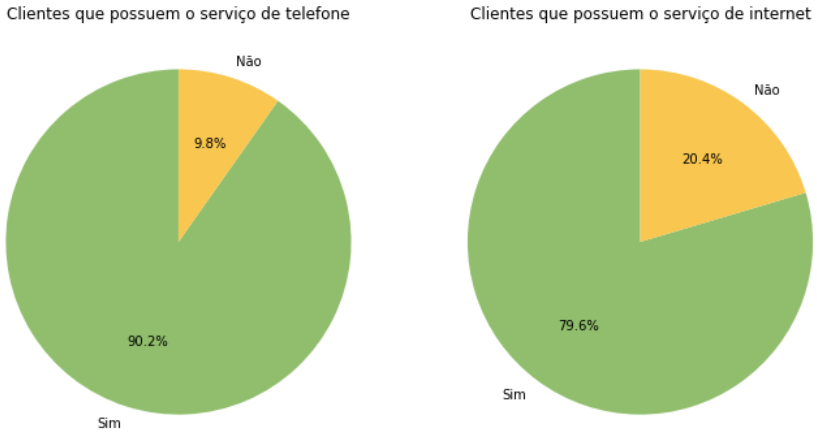
Fonte: Autoria própria

Através deste gráfico, é possível observar uma enorme relação de cancelamentos com o tipo de contrato mensal.

Partindo agora para a análise dos serviços vendidos, iremos primeiro visualizar como se dá a distribuição dos principais serviços oferecidos pela empresa, o código expresso na Figura 48 nos mostra a distribuição percentual dos serviços de telefone e internet respectivamente.

Figura 48 - Distribuição de clientes por serviço possuído

```
qtd_telefone = df['Servico_telefone'].value_counts()
qtd_internet = df['Servico_internet'].value_counts()
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
axes[0].pie(qtd_telefone, labels=qtd_telefone.index, autopct='%1.1f%%', startangle=90, colors=['#90BE6D','#F9C74F'])
axes[0].set_title('Clientes que possuem o serviço de telefone')
axes[1].pie(qtd_internet, labels=qtd_internet.index, autopct='%1.1f%%', startangle=90, colors=['#90BE6D','#F9C74F'])
axes[1].set_title('Clientes que possuem o serviço de internet')
plt.tight_layout()
plt.show()
```

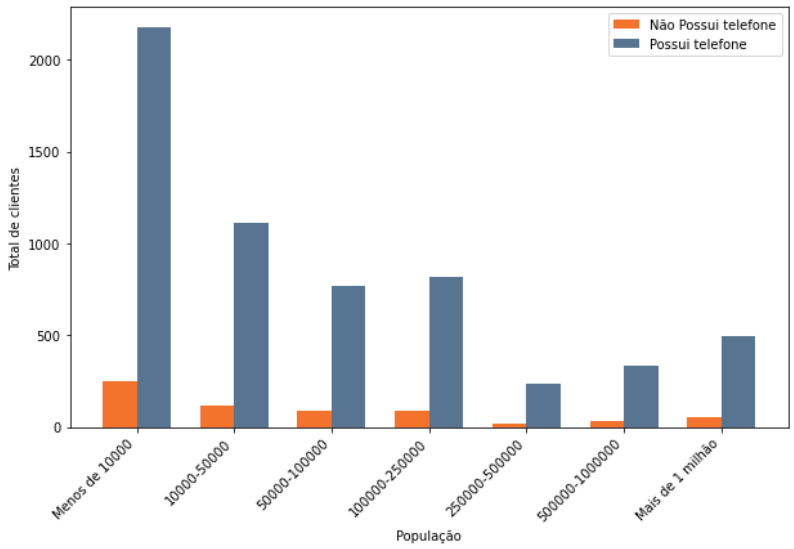


Fonte: Autoria própria

Em seguida, podemos estratificar pela população das cidades. Começando pelo serviço de telefone, na Figura 49 temos a quantidade de clientes que possuem ou não o serviço, dado a população da cidade.

Figura 49 - Estratificação de clientes com o serviço de telefone por população

```
df_default = df.loc[0:,['Cidade','Populacao','Servico_telefone']]
faixa_populacional = [0,10000, 50000, 100000, 250000, 500000,1000000,10000000000]
faixa_populacional_labels = ['Menos de 10000', '10000-50000', '50000-100000',
                             '100000-250000', '250000-500000', '500000-1000000','Mais de 1 milhão']
df_default['Faixa Populacional'] = pd.cut(df_default['Populacao'],
                                          bins=faixa_populacional, labels=faixa_populacional_labels)
result = df_default.groupby(['Faixa Populacional', 'Servico_telefone']).size().unstack().reset_index()
result = result.rename_axis(None, axis=1)
índices = range(len(result['Faixa Populacional']))
plt.figure(figsize=(10, 6))
plt.bar([i - tam_bar/2 for i in índices], result['Não'], tam_bar, label='Não Possui telefone',color='#F3722C')
plt.bar([i + tam_bar/2 for i in índices], result['Sim'], tam_bar, label='Possui telefone',color='#577590')
plt.xticks(índices, result['Faixa Populacional'])
plt.ylabel('Total de clientes')
plt.xlabel('População')
plt.xticks(rotation=45, ha='right')
plt.legend()
plt.show()
```

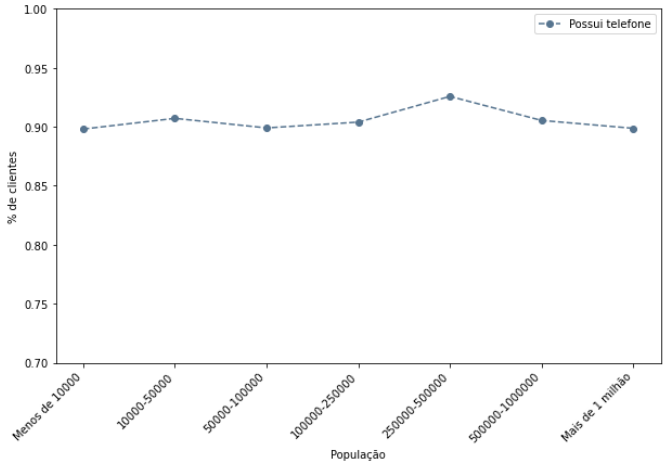


Fonte: Autoria própria

Em um primeiro, momento nota-se uma alta quantidade de clientes em cidades com menos de 10000 habitantes, o que de fato é verdade, nos mostrando que a empresa em questão possui sua base de clientes bem distribuída em várias cidades menores, mas algo que também deve-se notar é a proporção de clientes que possuem ou não o serviço, a Figura 50 elucida isso.

Figura 50 - Proporcional de clientes que possuem o serviço de telefone

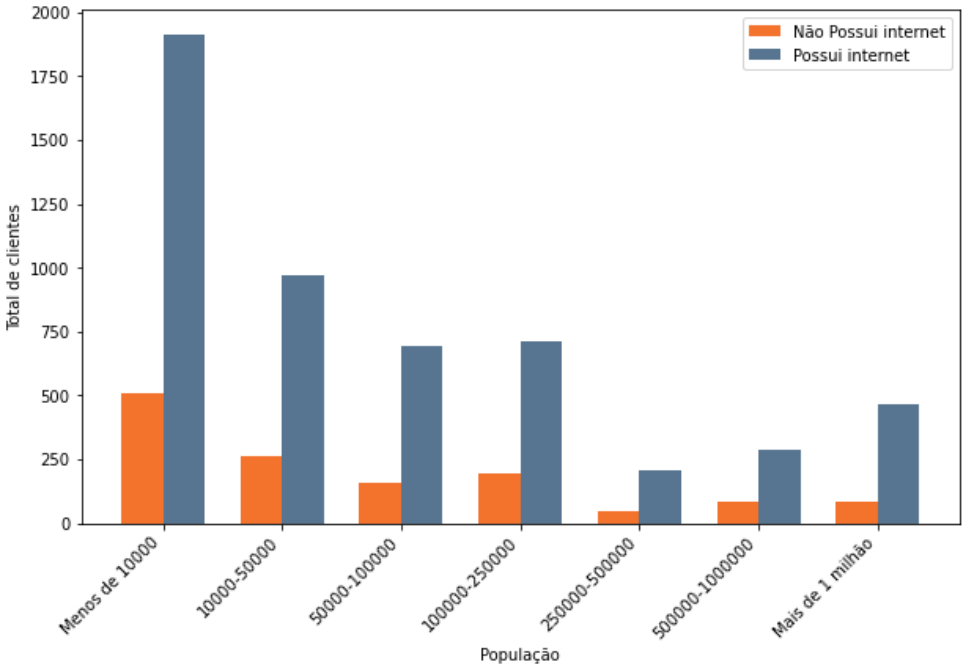
```
result['proporção'] = result['Sim']/(result['Sim']+result['Não'])
plt.figure(figsize=(10, 6))
plt.plot(result['Faixa Populacional'], result['proporção'], label='Possui telefone', marker='o',linestyle='--',color='#577590')
plt.xticks(índices, result['Faixa Populacional'])
plt.ylabel('% de clientes')
plt.xlabel('População')
plt.xticks(rotation=45, ha='right')
plt.ylim(bottom=0.7,top=1)
plt.legend()
plt.show()
```



Fonte: Autoria própria

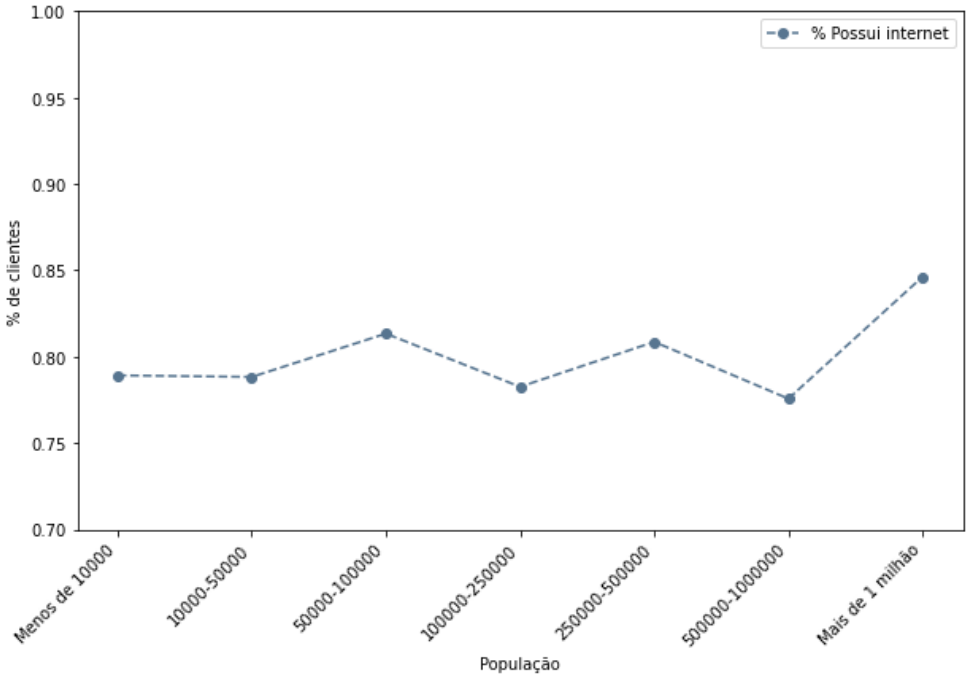
Fica nítido que as proporções se mantêm quase as mesmas, independente da faixa populacional. O mesmo gráfico será gerado agora com a coluna referente ao serviço de internet, a Figura 51 mostra o quantitativo de clientes e a Figura 52 mostra o seu respectivo proporcional.

Figura 51 - Estratificação de clientes com o serviço de internet por população



Fonte: Autoria própria

Figura 52 - Proporcional de clientes que possuem o serviço de internet

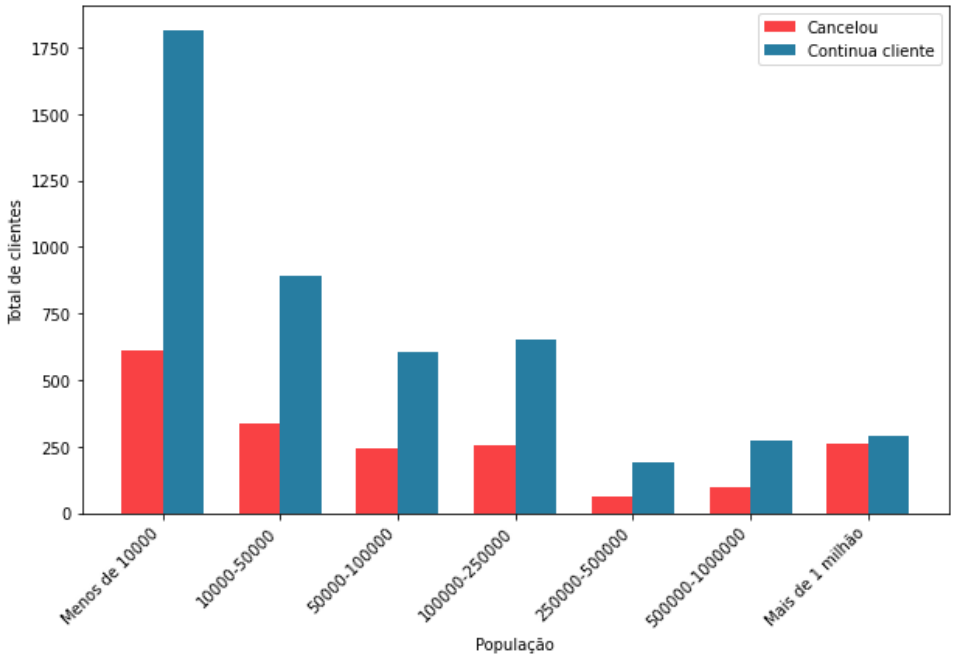


Fonte: Autoria própria

Já neste cenário, é possível observar que no último agrupamento, a taxa de clientes que possuem o serviço de internet é um pouco mais elevada em relação as demais, se encontrando próxima a 85%.

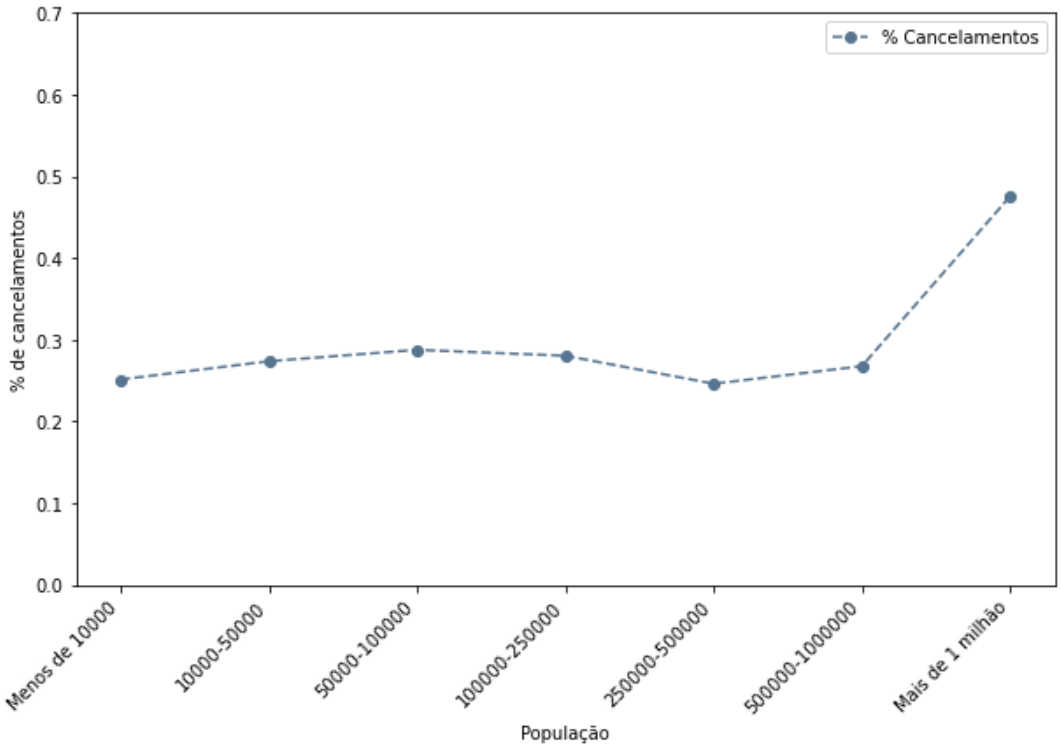
Podemos ainda usar essa mesma estrutura para a visualização da distribuição de cancelamentos por população, na Figura 53 pode-se visualizar a estratificação por quantidade de clientes, e na Figura 54 o percentual proporcional de clientes que cancelaram.

Figura 53 - Estratificação da situação do cliente por população



Fonte: Autoria própria

Figura 54 - Proporcional de clientes que cancelaram



Fonte: Autoria própria

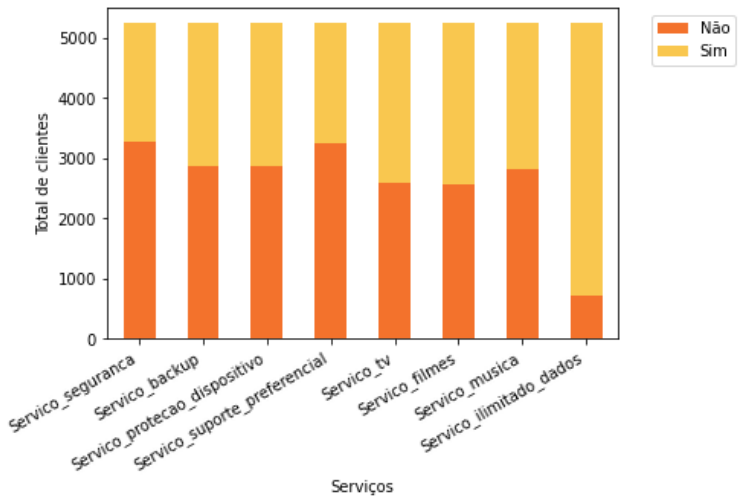
Referente ao proporcional de clientes que cancelaram, ele se manteve estável com exceção do último agrupamento, onde o percentual de clientes que cancelaram sobe para aproximadamente 50%.

Outros dados que podemos explorar, são os dados relacionados às vendas de serviços adicionais, que estão condicionados à posse ou ausência de um serviço específico, que no nosso contexto, se trata dos serviços de internet e telefone.

Começando pelo serviço de internet, iremos primeiramente filtrar da nossa base de dados as colunas de serviços adicionais relacionadas à internet, selecionando exclusivamente os clientes que utilizam esse serviço específico. Em seguida, podemos gerar um gráfico mostrando a relação de clientes que possuem ou não o serviço adicional. Este gráfico bem como o código utilizado para sua criação pode ser visualizado na Figura 55.

Figura 55 - Distribuição de serviços adicionais de internet

```
df_default = df[df['Servico_internet']=="Sim"].loc[0:,['Servico_seguranca','Servico_backup',
                                                       'Servico_protecao_dispositivo',
                                                       'Servico_suporte_preferencial',
                                                       'Servico_tv','Servico_filmes',
                                                       'Servico_musica','Servico_ilimitado_dados']]
counts = df_default.apply(pd.Series.value_counts)
counts.T.plot(kind='bar', stacked=True, color=['#F3722C','#F9C74F'])
plt.xlabel('Serviços')
plt.ylabel('Total de clientes')
plt.legend(bbox_to_anchor=(1.05, 1))
plt.xticks(rotation=30,ha='right')
plt.show()
```



Fonte: Autoria própria

Através do gráfico podemos visualizar como o serviço ilimitado de dados se destaca frente aos outros, mostrando como uma grande parte dos clientes que possuem o serviço de internet também possui este serviço adicional.

Em seguida, iremos relacionar estes dados com a retenção ou não do cliente. No código apresentado na Figura 56, estaremos para cada serviço adicional, agrupando os dados de forma que seja contabilizado a quantidade de cancelamentos

e retenções, dado o fato do cliente ter ou não o serviço adicional e em seguida concatenando tudo isso em uma única tabela onde realizaremos o cálculo do percentual de clientes retidos para cada linha da tabela. Dessa forma, teremos para cada serviço, o percentual de clientes que permaneceram na empresa com o serviço adicional contratado ou não.

Figura 56 - Gerando percentual de clientes que permaneceram na base dado o serviço adicional

```
lista_colunas = ['Servico_seguranca','Servico_backup','Servico_protecao_dispositivo','Servico_suporte_preferencial',
                 'Servico_tv','Servico_filmes','Servico_musica','Servico_ilimitado_dados']
result = pd.DataFrame()
res = pd.DataFrame()
for item in lista_colunas:
    df_default = df[df['Servico_internet']=="Sim"].loc[0:,['Servico_internet','Servico_seguranca','Servico_backup',
                                                           'Servico_protecao_dispositivo','Servico_suporte_preferencial',
                                                           'Servico_tv','Servico_filmes','Servico_musica',
                                                           'Servico_ilimitado_dados','Situacao_cliente']]
    df_default = df_default.groupby([item,'Situacao_cliente'])['Servico_internet'].count().reset_index()
    res = df_default.groupby([item,'Situacao_cliente'])['Servico_internet'].sum().unstack().reset_index()
    res = res.rename_axis(None, axis=1)
    res['serviço'] = item
    res = res.rename(columns={item:'possui'},inplace=False)
    result = pd.concat([result,res], axis=0, ignore_index=True)

result['proporcao'] = result['Continua cliente']/(result['Cancelou']+result['Continua cliente'])
result = result.groupby(['serviço','possui'])['proporcao'].sum().unstack().reset_index()
result = result.rename_axis(None, axis=1)
result['Não'] = result['Não'] * 100
result['Sim'] = result['Sim'] * 100
display(result)
```

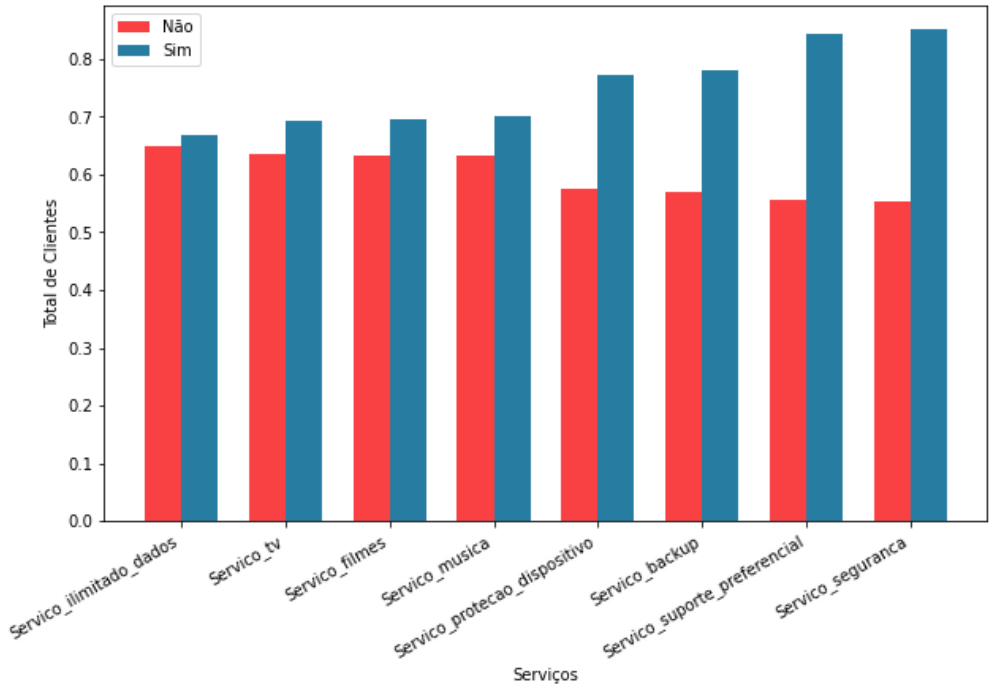
	serviço	Não	Sim
0	Servico_backup	57.038328	77.978947
1	Servico_filmes	63.387978	69.511741
2	Servico_ilimitado_dados	64.917127	66.777262
3	Servico_musica	63.403346	70.114943
4	Servico_protecao_dispositivo	57.583187	77.196653
5	Servico_seguranca	55.348411	85.048150
6	Servico_suporte_preferencial	55.480296	84.476715
7	Servico_tv	63.587167	69.375470

Fonte: Autoria própria

Para deixar a informação mais visual, foi utilizado o código da Figura 57 para criação de um gráfico de barras, que mostra a taxa de retenção por serviço.

Figura 57 - Percentual de clientes que retidos com base na posse ou não do serviço

```
lrg_bar = 0.35
result = result.sort_values(by=['Sim', 'Não'])
indices = range(len(result['serviço']))
plt.figure(figsize=(10, 6))
plt.bar([i - lrg_bar/2 for i in indices], result['Não'], lrg_bar, color='#F94144',label='Não')
plt.bar([i + lrg_bar/2 for i in indices], result['Sim'], lrg_bar, color='#277DA1',label='Sim')

plt.xticks(indices, result['serviço'])
plt.legend()
plt.ylabel('Total de Clientes')
plt.xlabel('Serviços')
plt.xticks(rotation=30,ha='right')

plt.show()
```



Fonte: Autoria própria

É relevante observar que, do ponto de vista estatístico, os quatro serviços vinculados à segurança e conforto apresentam uma tendência de elevar a probabilidade de retenção do cliente quando adquiridos.

Foi realizado o mesmo processo para o serviço de telefone, porém não foi encontrado nenhuma informação relevantemente expressiva.

Em seguida, iremos examinar os motivos que levaram ao cancelamento. Inicialmente, realizaremos uma filtragem para identificar exclusivamente os clientes que optaram pelo fim do contrato. Com base nesses dados selecionados, construiremos um gráfico de Pareto, proporcionando uma análise visual que destaca a frequência de cada motivo, dispostos de maneira decrescente bem como o

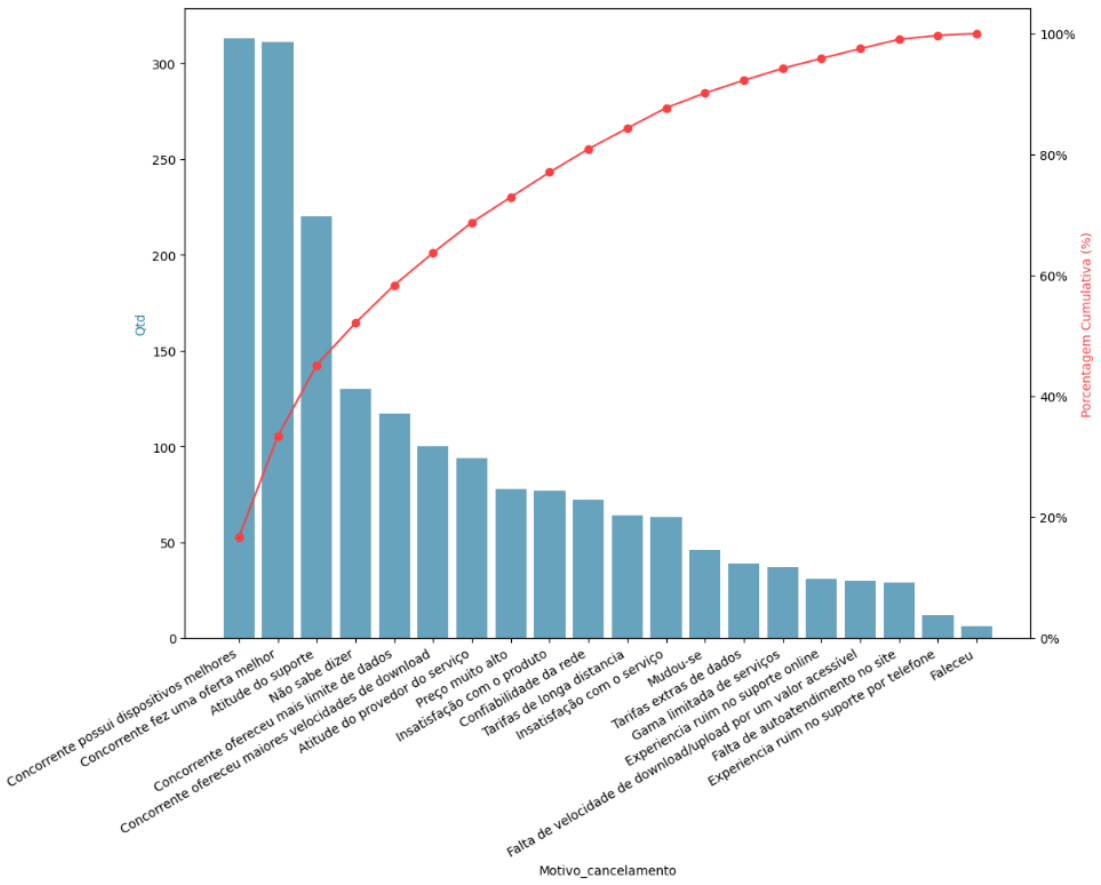
percentual acumulado associado a cada motivo. Na Figura 58 está expresso o código utilizado e na Figura 59, o gráfico gerado.

Figura 58 - Código para criação de um gráfico de Pareto

```
df_default = df.loc[0:,['Ultima_oferta','Categoria_cancelamento','Motivo_cancelamento']]
df_default = df_default[df_default['Categoria_cancelamento']!='Permaneceu cliente']
df_default = pd.DataFrame(df_default.groupby(['Categoria_cancelamento',
                                              'Motivo_cancelamento'])['Ultima_oferta'].count().reset_index())
df_default = df_default.rename(columns={'Ultima_oferta':'Qtd'},inplace=False)
df_default = df_default.sort_values('Qtd',ascending=False)
df_default = df_default.sort_values(by='Qtd', ascending=False)
df_default['Porcentagem Cumulativa'] = 100 * df_default['Qtd'].cumsum() / df_default['Qtd'].sum()
fig, ax1 = plt.subplots(figsize=(12,9))
plt.xticks(rotation=30,ha='right')
ax1.bar(df_default['Motivo_cancelamento'], df_default['Qtd'], color='b', alpha=0.7)
ax1.set_xlabel('Motivo_cancelamento')
ax1.set_ylabel('Qtd', color='b')
ax2 = ax1.twinx()
ax2.plot(df_default['Motivo_cancelamento'], df_default['Porcentagem Cumulativa'], color='r', marker='o')
ax2.set_ylabel('Porcentagem Cumulativa (%)', color='r')
ax2.set_ylim(0)
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())

plt.show()
```

Fonte: Autoria própria

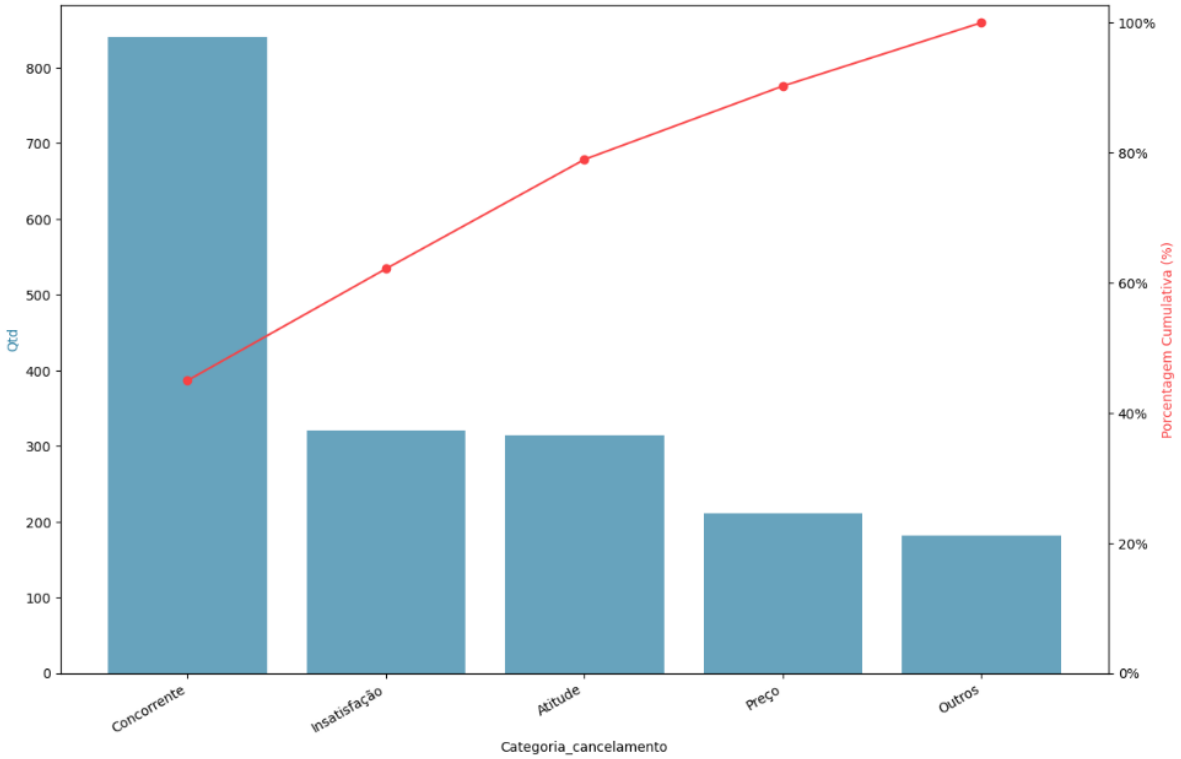
Figura 59 - Gráfico de Pareto dos motivos de cancelamento



Fonte: Autoria própria

É importante salientar que os motivos são informados pelos clientes no ato do cancelamento. Partindo do pressuposto que o cliente informou a verdade, essa pesquisa se torna uma fonte valiosa de *feedback* direto dos clientes sobre o que estão procurando em um produto, na situação em questão, podemos notar que quatro dos seis motivos mais comuns estão relacionados ao concorrente. Isso pode ser mais bem visualizado com o gráfico de Pareto das categorias de cancelamento realizadas pela empresa com base na informação repassada pelo cliente (Figura 60).

Figura 60 - Gráfico de Pareto das categorias de cancelamento



Fonte: Autoria própria

O princípio de Pareto nos diz que 80% dos resultados são gerados por 20% das causas. Obviamente, o princípio em questão trata-se de uma heurística e não uma equação matemática formal. Dessa forma, embora a proporção específica possa variar, a ideia fundamental é que uma minoria de fatores contribui de maneira desproporcional para a obtenção dos resultados observados. Em nosso cenário, somente as causas correlacionadas ao concorrente concentram 40% dos clientes.

5 CRIAÇÃO DE MODELOS DE MACHINE LEARNING

5.1 Preparando os dados para os modelos de aprendizado de máquina

Passaremos agora para a etapa onde utilizaremos algoritmos de aprendizado de máquina para tentar prever clientes que possam vir a solicitar o cancelamento do serviço. Para isso, alguns ajustes precisam ser feitos em nossa base.

Primeiramente, alguns algoritmos não lidam muito bem com dados em formato de texto (dados categóricos), dessa forma precisamos alterar os valores de algumas colunas. Nas colunas que possuem valores “sim” e “não” alteraremos para 1 e 0 respectivamente. Na coluna “Situacao_cliente” modificaremos os valores de “Continua cliente” para 0 e “Cancelou” para 1. Já na coluna de “Gênero” alteraremos o valor de “Masculino” para 1 e “Feminino” para 0, além disso, para manter consistência entre o título da coluna e a informação a ela associada, renomearemos a coluna para "Genero_masculino", na Figura 61 podemos visualizar o código utilizado. Importante destacar que iremos realizar estes ajustes em um novo *dataframe* “df1” para que em etapas futuras, possamos conseguir recuperar as informações antes de serem ajustadas caso haja necessidade.

Figura 61 - Ajustando valores das colunas

```
df1 = df.replace({'Genero': {'Masculino': 1, 'Feminino': 0},
            'Casado': {'Sim': 1, 'Não': 0},
            'Servico_telefone': {'Sim': 1, 'Não': 0},
            'Servico_internet': {'Sim': 1, 'Não': 0},
            'Multiplas_linhas': {'Sim': 1, 'Não': 0},
            'Servico_seguranca': {'Sim': 1, 'Não': 0},
            'Servico_backup': {'Sim': 1, 'Não': 0},
            'Servico_protecao_dispositivo': {'Sim': 1, 'Não': 0},
            'Servico_suporte_preferencial': {'Sim': 1, 'Não': 0},
            'Servico_tv': {'Sim': 1, 'Não': 0},
            'Servico_filmes': {'Sim': 1, 'Não': 0},
            'Servico_musica': {'Sim': 1, 'Não': 0},
            'Servico_ilimitado_dados': {'Sim': 1, 'Não': 0},
            'Faturamento_sem_papel': {'Sim': 1, 'Não': 0},
            'Situacao_cliente': {'Continua cliente': 0, 'Cancelou': 1}})

df1 = df1.rename(columns={'Genero':'Genero_masculino'}, inplace=False)
```

Fonte: Autoria própria

Ainda em relação as colunas com valores de texto, temos a coluna de cidades que apesar de possuir valores categóricos, não podemos aplicar a mesma estratégia usada anteriormente, pois isso resultaria em atribuir a coluna um sentido de ordem entre as cidades, que na realidade não existe. Por se tratar de um dado que não podemos simplesmente remover do nosso conjunto de dados, iremos criar categorias de cidades com base em sua população assim como na Figura 42, e assim, preservaremos de uma forma mais generalizada a relação do cliente e a cidade em que reside (Figura 62). Com o agrupamento realizado, serão removidas as colunas de cidade e população.

Figura 62 - Categorização das cidades com base na população

```
df_default = df.loc[0:,['Cidade','Populacao']]
faixa_populacional = [0,10000, 50000, 100000, 250000, 500000,1000000,10000000000]
faixa_populacional_labels = ['Menos de 10000', '10000-50000', '50000-100000',
                             '100000-250000', '250000-500000', '500000-1000000','Mais de 1 milhão']
df_default['Faixa Populacional'] = pd.cut(df_default['Populacao'],
                                          bins=faixa_populacional, labels=faixa_populacional_labels)
df_default = df_default.loc[0:,['Cidade','Faixa Populacional']]

df_default= df_default.drop_duplicates().reset_index(drop=True)
df1 = pd.merge(df1, df_default,
               left_on=['Cidade'],
               right_on=['Cidade'],
               how='left')
```

Fonte: Autoria própria

Além disso, temos algumas colunas que possuem mais de dois valores categóricos. Diferentemente da coluna cidades, essas colunas possuem quantidades pequenas de valores distintos, o que possibilita a utilização de outra estratégia. Para esses casos utilizaremos a biblioteca “*OneHotEnconder*”.

Essencialmente, a função dessa biblioteca é converter as variáveis categóricas em representações binárias, criando uma coluna adicional para cada categoria única presente nos dados originais. Esta configuração será realizada nas colunas “Tipo_contrato”, “Tipo_internet”, “Tipo_pagamento”, “Ultima_oferta” e “Faixa Populacional”, em seguida removeremos as colunas originais uma vez que não são mais necessárias, o código referente a esta configuração pode ser visto na Figura 63.

Figura 63 - Utilização do *OneHotEncoder*

```
from sklearn.preprocessing import OneHotEncoder
```

```
def ohe(df_padrao, coluna):
    ohencoder = OneHotEncoder(handle_unknown = 'ignore')
    ohencoder = ohencoder.fit(df_padrao[[coluna]])
    colunas = ohencoder.get_feature_names_out()
    ohe_df = pd.DataFrame(ohencoder.transform(df_padrao[[coluna]]).toarray(),
                          columns = colunas,
                          dtype='int32')
    return ohe_df
```

```
df_tipoContrato = ohe(df1, 'Tipo_contrato')
df_tipoInternet = ohe(df1, 'Tipo_internet')
df_tipoPagamento = ohe(df1, 'Tipo_pagamento')
df_tipoOferta = ohe(df1, 'Ultima_oferta')
df_tipoFaixaPopulacional = ohe(df1, 'Faixa Populacional')
```

```
df1 = pd.concat([df1,df_tipoContrato],axis=1)
df1 = pd.concat([df1,df_tipoInternet],axis=1)
df1 = pd.concat([df1,df_tipoPagamento],axis=1)
df1 = pd.concat([df1,df_tipoOferta],axis=1)
df1 = pd.concat([df1,df_tipoFaixaPopulacional],axis=1)

df1 = df1.drop(['Tipo_contrato','Tipo_internet','Tipo_pagamento',
                'Ultima_oferta','Cidade','Populacao','Faixa Populacional'], axis=1)
```

Fonte: Autoria própria

Em seguida foram também removidas as colunas mostradas na Figura 64, essa remoção é justificada pois as duas colunas em questão têm seus dados preenchidos apenas no caso de o cliente efetuar o cancelamento. Portanto, essas informações não podem ser utilizadas para prever o próprio cancelamento.

Figura 64 - Remoção de colunas de motivo e categoria do cancelamento

```
df1 = df1.drop(['Motivo_cancelamento','Categoria_cancelamento'], axis=1)
```

Fonte: Autoria própria

Com estes ajustes realizados, podemos gerar um gráfico para observar a correlação da coluna “Situacao_cliente” em relação as outras.

A correlação é uma medida estatística que expressa a força de um relacionamento entre duas colunas, seu valor varia de -1 a 1, onde um coeficiente de correlação próximo a 1 indica uma forte correlação positiva, o que significa que à medida que os valores de uma coluna aumentam, os valores correspondentes na outra coluna também tendem a aumentar. Por outro lado, um coeficiente próximo a -1 indica uma forte correlação negativa, indicando que à medida que os valores de uma coluna aumentam, os valores correspondentes na outra coluna tendem a diminuir. Já

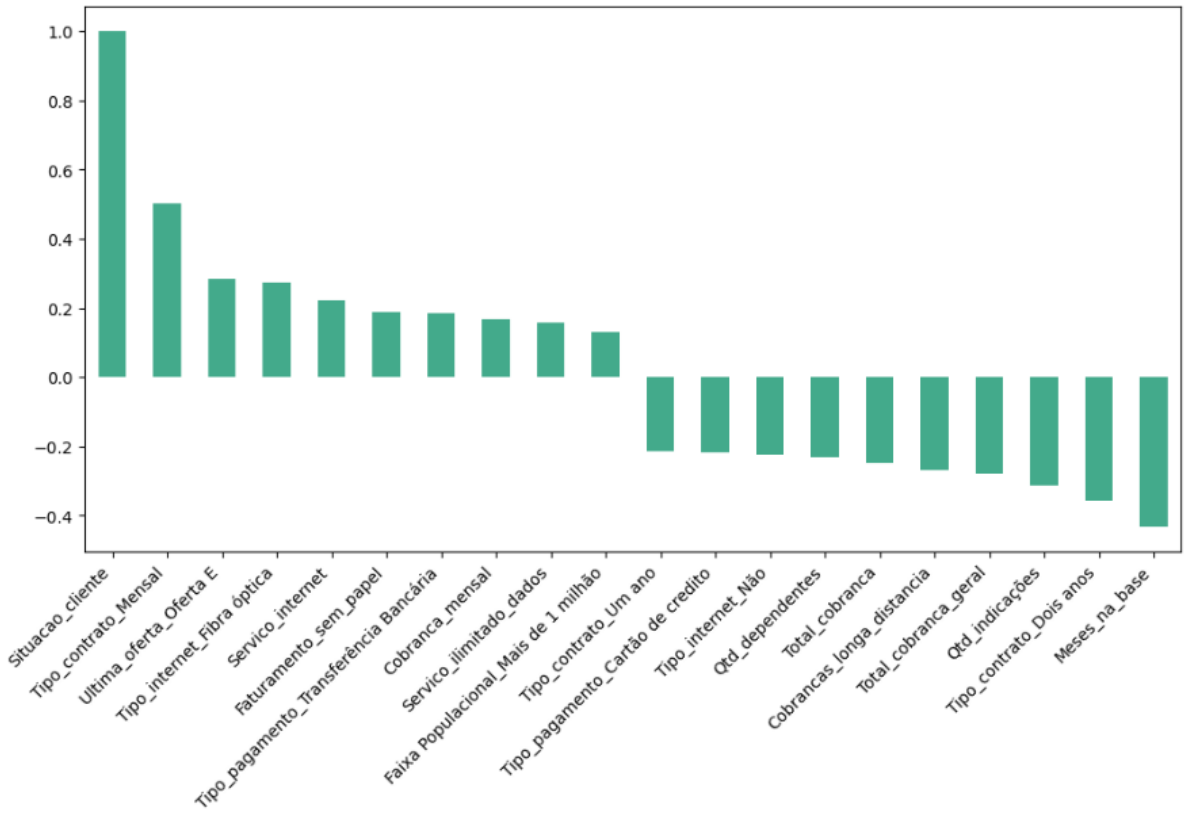
um coeficiente próximo a zero sugere uma correlação fraca ou inexistente. Na Figura 65 é possível visualizar o código utilizado para gerar um gráfico com as dez colunas com maiores correlações positivas, bem com as dez maiores correlações negativas, já na Figura 66 é possível ver o gráfico gerado.

Figura 65 - Código utilizado para gerar gráfico de correlação

```
correl = df1.corr()['Situacao_cliente']
colunasF = correl.nlargest(10).index.union(correl.nsmallest(10).index)
plt.figure(figsize=(12,6))
df1[colunasF].corr()['Situacao_cliente'].sort_values(ascending = False).plot(kind='bar', color='#43AA8B')
plt.xticks(rotation=45, ha='right')
plt.show()
```

Fonte: Autoria própria

Figura 66 - Gráfico com as principais correlações da base de dados



Fonte: Autoria própria

Antes de iniciarmos a aplicação de modelos de aprendizado de máquina, um último ajuste se faz necessário. Em nossa base temos algumas colunas que são redundantes, a chamada multicolinearidade, é o caso das colunas “Recarga_longa_distancia”, que contém a informação dos custos médios das recargas de longa distancias realizadas, e a coluna “Cobrancas_longa_distancia” que contém

a soma total desses custos, além disso a soma desta última com a coluna “Total_cobranca” resulta nos valores da coluna “Total_cobranca_geral”. Para corrigir esta situação, iremos simplificar um pouco o nosso conjunto de dados, porém sem uma perda significativa de informações. Será criada uma coluna que será a média dos valores pagos mensalmente pelo cliente ao longo do tempo com o serviço assinado, em seguida removeremos as colunas com informações relacionadas ao custo (Figura 67).

Figura 67 - Correção de Multicolinearidade

```
df1['Média_mensalidade']= df1['Total_cobranca_geral']/df1['Meses_na_base']
```

```
df1 = df1.drop(['Total_cobranca','Cobrancas_longa_distancia','Total_cobranca_geral',
                'Total_reembolsos','Cobranca_mensal','Recarga_longa_distancia'], axis=1)
```

Fonte: Autoria própria

5.2 Random Forest

Agora, iniciaremos a etapa de utilização de algoritmos de aprendizado de máquina, o primeiro a ser utilizado neste artigo será o “*Random Forest*”.

Sua escolha se dá pelo fato de se tratar de um algoritmo de aprendizado robusto, capaz de tratar base de dados complexas, além de lidar bem com *overfiting*[4] e ser menos sensível a ruídos.

Conforme pode ser visualizado na Figura 68, iremos importar as bibliotecas necessárias para execução do script, bem como de visualização das métricas pelas quais iremos conseguir avaliar a qualidade do modelo. Neste primeiro momento, após executarmos o algoritmo em suas configurações padrão, conseguimos uma acurácia[5] de 87,15%.

[4] Situação em que o modelo se ajuste aos dados de treinamento de tal maneira que ele não consegue generalizar para novos dados.
[5] Métrica calculada a partir da divisão do total de acertos de modelo pelo total de classes avaliadas.

Figura 68 – Primeira execução do algoritmo Random Forest

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
X = df1.drop(['Situacao_cliente'], axis=1)
y = df1['Situacao_cliente']
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.3, random_state=42)
modelo_rf = RandomForestClassifier(random_state=42)
modelo_rf.fit(X_treino, y_treino)
previsoes = modelo_rf.predict(X_teste)
acuracia = accuracy_score(y_teste, previsoes)
print(f'Acurácia: {acuracia:.2%}\n')
print(classification_report(y_teste, previsoes))
```

```
Acurácia: 86.90%

              precision    recall  f1-score   support

           0       0.88      0.95      0.91      1407
           1       0.84      0.67      0.75       570

    accuracy                           0.87      1977
   macro avg       0.86      0.81      0.83      1977
weighted avg       0.87      0.87      0.86      1977
```

Fonte: Autoria própria

Ainda há espaço para ajustes em nossa base de dados para conseguirmos melhores resultados nas métricas de avaliação. Tendo isso em vista, registraremos em um dataframe as métricas de precisão[6], revocação e f1-score[7] de cada ajuste realizado. Assim teremos um histórico mostrando a progressão dessas métricas. A Figura 69 mostra a função criada para salvar os dados no dataframe.

[6] Métrica que avalia a proporção de classes positivas identificadas corretamente pelo modelo em relação a todas as classes que o modelo classificou como positivo.
[7] Métrica calculada a partir da média harmônica entre precisão e revocação.

Figura 69 - Função para registrar informações das métricas atingidas

```
df_relatorio = pd.DataFrame()
def geraRelatorio(df_resultado, dados_teste,modelo, obs):
    dados = classification_report(dados_teste, previsoes,output_dict=True)
    df_relatorio_default = pd.DataFrame(dados)
    df_relatorio_default = df_relatorio_default.iloc[:, :2].copy()
    df_relatorio_default['modelo'] = modelo
    df_relatorio_default['obs'] = obs
    df_relatorio_default = df_relatorio_default.reset_index()
    df_relatorio_default = df_relatorio_default.rename(columns={'index': 'parametro'})
    df_relatorio_default = df_relatorio_default[df_relatorio_default['parametro']!='support']
    df_resultado = pd.concat([df_resultado,df_relatorio_default], axis=0, ignore_index=True)
    return df_resultado
```

```
df_relatorio = geraRelatorio(df_relatorio,y_teste,'Random Forest','OneHotEncoding e ajuste de multicolinearidade')
df_relatorio
```

	parametro	0	1	modelo	obs
0	precision	0.876147	0.844789	Random Forest	OneHotEncoding e ajuste de multicolinearidade
1	recall	0.950249	0.668421	Random Forest	OneHotEncoding e ajuste de multicolinearidade
2	f1-score	0.911695	0.746327	Random Forest	OneHotEncoding e ajuste de multicolinearidade

Fonte: Autoria própria

Um dos ajustes que ainda precisam ser feitos, é em relação a escala dos dados. Atualmente, nossa base possui várias colunas com valores numéricos que possuem escalas de valores diferentes entre si, determinados algoritmos podem dar uma importância maior para uma determinada coluna que possui uma escala de dados maior que outra, isso acarretará a geração de resultados distorcidos. Para resolver isso, iremos realizar a normalização dos dados, ajustando os valores de cada coluna para que fiquem com a média igual a 0, e o desvio padrão igual a 1. A Figura 70 mostra o código que realiza tal processo.

Figura 70 - Realizando a normalização dos dados

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaler = scaler.fit(df1[['Idade']])
df1['Idade'] = scaler.transform(df1[['Idade']])

scaler = StandardScaler()
scaler = scaler.fit(df1[['Qtd_dependentes']])
df1['Qtd_dependentes'] = scaler.transform(df1[['Qtd_dependentes']])

scaler = StandardScaler()
scaler = scaler.fit(df1[['Qtd_indicações']])
df1['Qtd_indicações'] = scaler.transform(df1[['Qtd_indicações']])

scaler = StandardScaler()
scaler = scaler.fit(df1[['Meses_na_base']])
df1['Meses_na_base'] = scaler.transform(df1[['Meses_na_base']])

scaler = StandardScaler()
scaler = scaler.fit(df1[['Media_mensal_download_GB']])
df1['Media_mensal_download_GB'] = scaler.transform(df1[['Media_mensal_download_GB']])

scaler = StandardScaler()
scaler = scaler.fit(df1[['Qtd_dados_extras']])
df1['Qtd_dados_extras'] = scaler.transform(df1[['Qtd_dados_extras']])

scaler = StandardScaler()
scaler = scaler.fit(df1[['Média_mensalidade']])
df1['Média_mensalidade'] = scaler.transform(df1[['Média_mensalidade']])

scaler = StandardScaler()
scaler = scaler.fit(df1[['Latitude']])
df1['Latitude'] = scaler.transform(df1[['Latitude']])

scaler = StandardScaler()
scaler = scaler.fit(df1[['Longitude']])
df1['Longitude'] = scaler.transform(df1[['Longitude']])
```

Fonte: Autoria própria

Novamente podemos executar o *Random Forest* e salvar as métricas no *dataframe* de relatório. Como saída, temos o resultado mostrado na Figura 71.

Figura 71 - Registrando métricas após normalização dos dados

```
df_relatorio = geraRelatorio(df_relatorio,y_teste,'Random Forest','Ajuste de normalização dos dados')
df_relatorio
```

	parametro	0	1	modelo	obs
0	precision	0.876147	0.844789	Random Forest	OneHotEncoding e ajuste de multicolinearidade
1	recall	0.950249	0.668421	Random Forest	OneHotEncoding e ajuste de multicolinearidade
2	f1-score	0.911695	0.746327	Random Forest	OneHotEncoding e ajuste de multicolinearidade
3	precision	0.875654	0.846325	Random Forest	Ajuste de normalização dos dados
4	recall	0.950959	0.666667	Random Forest	Ajuste de normalização dos dados
5	f1-score	0.911755	0.745829	Random Forest	Ajuste de normalização dos dados

Fonte: Autoria própria

Outro fator que merece nossa atenção, é o fato de nossa base de dados estar desbalanceada. Assim como já mostrado na Figura 18, temos mais de 2,5 vezes mais casos de clientes que permanecem clientes do que clientes que solicitam o cancelamento.

Duas soluções possíveis seria fazer um "*undersampling*" ou um "*oversampling*". Na primeira opção, o excedente da classe majoritária é removido e assim iguala-se a quantidade de clientes que cancelaram com os clientes que não cancelaram. Essa solução foi descartada pois dado o grande desbalanceamento que existe entre as classes, estaríamos perdendo muitos dados.

A outra opção é justamente o contrário, utilizaremos uma técnica chamada *smote*, que consiste em gerar amostras sintéticas, acrescentando dados na classe minoritária e assim igualando as quantidades das duas classes.

Se faz necessário em um primeiro momento instalar a biblioteca *imblearn* para só então conseguirmos utilizar a *smote* (Figura 72).

Figura 72 - *Oversampling* com *smote*

```
!pip install -U imbalanced-learn
```

```
from imblearn.over_sampling import SMOTE
```

```
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.3, random_state=42)
df_counts = y_treino.value_counts().to_frame(name='Contagem de classes ANTES do SMOTE:')
print(df_counts)
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_treino, y_treino)
print('\n----------------------------------------\n')
df_counts = y_resampled.value_counts().to_frame(name='Contagem de classes DEPOIS do SMOTE:')
print(df_counts)
modelo_rf = RandomForestClassifier(random_state=42)
modelo_rf.fit(X_resampled, y_resampled)
previsoes = modelo_rf.predict(X_teste)
```

```
   Contagem de classes ANTES do SMOTE:
0                                 3313
1                                 1299

----------------------------------------

   Contagem de classes DEPOIS do SMOTE:
0                                 3313
1                                 3313
```

Fonte: Autoria própria

Em seguida novamente salvaremos as métricas resultantes no *dataframe* de relatório (Figura 73).

Figura 73 - Registrando métricas após balanceamento smote

```
df_relatorio = geraRelatorio(df_relatorio,y_teste,'Random Forest','Balanceamento com a técnica de SMOTE')
df_relatorio
```

	parametro	0	1	modelo	obs
0	precision	0.876147	0.844789	Random Forest	OneHotEncoding e ajuste de multicolinearidade
1	recall	0.950249	0.668421	Random Forest	OneHotEncoding e ajuste de multicolinearidade
2	f1-score	0.911695	0.746327	Random Forest	OneHotEncoding e ajuste de multicolinearidade
3	precision	0.875654	0.846325	Random Forest	Ajuste de normalização dos dados
4	recall	0.950959	0.666667	Random Forest	Ajuste de normalização dos dados
5	f1-score	0.911755	0.745829	Random Forest	Ajuste de normalização dos dados
6	precision	0.901767	0.766904	Random Forest	Balanceamento com a técnica de SMOTE
7	recall	0.906894	0.756140	Random Forest	Balanceamento com a técnica de SMOTE
8	f1-score	0.904323	0.761484	Random Forest	Balanceamento com a técnica de SMOTE

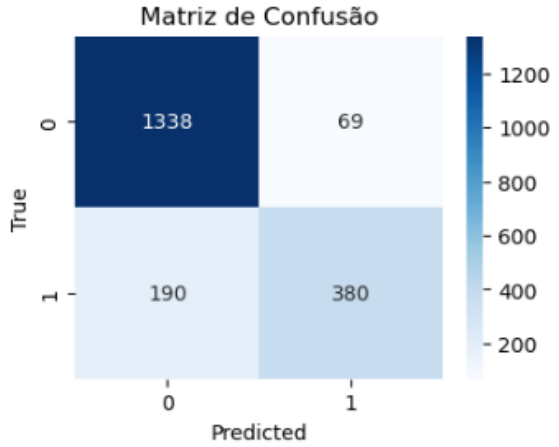
Fonte: Autoria própria

Posteriormente, podemos gerar uma matriz de confusão (Figura 74) para avaliar como estão dispostas as quantidades de erros e acertos do nosso modelo.

Figura 74 - Matriz de confusão

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

```
matriz_confusao = confusion_matrix(y_teste, previsoes)
plt.figure(figsize=(4, 3))
sns.heatmap(matriz_confusao, annot=True, fmt='d', cmap='Blues',
            xticklabels=modelo_rf.classes_,
            yticklabels=modelo_rf.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Matriz de Confusão')
plt.show()
```



Fonte: Autoria própria

Outro fator que merece nossa atenção, é em relação ao desempenho do nosso modelo e se ele está apresentando *overfitting*, no comando mostrado na Figura 75, foi aplicado a técnica do *k-fold cross validation*, que consiste em dividir a base de dados em k partes (ou "*folds*"), e em seguida, realizar “k” iterações de treinamento e teste. Em cada iteração realizada, foi retornado na saída a precisão e a revocação, no final das iterações foi calculado também a média e desvio padrão das métricas.

Figura 75 - Análise das métricas usando *kfold cross validation*

```
from sklearn.metrics import make_scorer, precision_score, recall_score
from sklearn.model_selection import cross_val_score, KFold, cross_val_predict
import numpy as np
```

```
k=5
kf = KFold(n_splits=k, shuffle=True, random_state=42)
modelo_rf = RandomForestClassifier(random_state=42)

# Métricas que serão availaidas
scoring = {'precision': make_scorer(precision_score),
           'recall': make_scorer(recall_score)}

# Executando a validação cruzada e obtendo as métricas para cada fold
precision_scores = cross_val_score(modelo_rf, X, y, cv=kf, scoring=scoring['precision'])
recall_scores = cross_val_score(modelo_rf, X, y, cv=kf, scoring=scoring['recall'])

# Impressão das métricas para cada fold
for i in range(k):
    print(f"Fold {i + 1} - Precisão: {precision_scores[i]}, Recall: {recall_scores[i]}")

# Métricas agregadas
print("\nMédia - Precisão:", np.mean(precision_scores))
print("Desvio Padrão - Precisão:", np.std(precision_scores))
print("Média - Recall:", np.mean(recall_scores))
print("Desvio Padrão - Recall:", np.std(recall_scores))
```

```
Fold 1 - Precisão: 0.8636363636363636, Recall: 0.6945169712793734
Fold 2 - Precisão: 0.8138801261829653, Recall: 0.7010869565217391
Fold 3 - Precisão: 0.8366013071895425, Recall: 0.6826666666666666
Fold 4 - Precisão: 0.8, Recall: 0.6720867208672087
Fold 5 - Precisão: 0.8557692307692307, Recall: 0.713903743315508

Média - Precisão: 0.8339774055556205
Desvio Padrão - Precisão: 0.024165210641887543
Média - Recall: 0.6928522117300993
Desvio Padrão - Recall: 0.014482405651326096
```

Fonte: Autoria própria

É possível observar na saída retornada no código anterior que o desvio padrão, tanto da precisão quanto da revocação são valores baixos, mostrando que o modelo se mantém bem consistente ao longo dos testes.

Como dito no início deste artigo, a métrica que iremos priorizar é a revocação, já que nossa preocupação é errar o mínimo possível de clientes que irão cancelar.

Tendo isso em vista, iremos prosseguir no processo de melhoria do nosso modelo e para isso iremos executar um código que percorrerá alguns laços de repetição, verificando em cada execução a combinação que retorna o melhor valor de revocação, na Figura 76 instanciamos algumas variáveis necessárias e na Figura 77 executamos os laços de repetições aninhados.

Figura 76 - Instancia das variáveis a serem usadas nos laços de repetição

```
# Relação de parametros que terão suas combinações testadas
parametros = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

high_recall    = 0
high_precision = 0
high_accuracy  = 0

melhor_recall = 0
melhores_parametros = {}
historico_recall = []  # Lista para armazenar o recall em cada iteração
historico_precision = []  # Lista para armazenar a precisão em cada iteração
historico_acuracia = []  # Lista para armazenar a acurácia em cada iteração
```

Fonte: Autoria própria

Figura 77 - Execução dos laços de repetições

```
# Loop através das combinações de hiperparâmetros
for estimators in parametros['n_estimators']:
    for depth in parametros['max_depth']:
        for split in parametros['min_samples_split']:
            for leaf in parametros['min_samples_leaf']:
                modelo_rf = RandomForestClassifier(n_estimators=estimators,
                                                   max_depth=depth,
                                                   min_samples_split=split,
                                                   min_samples_leaf=leaf,
                                                   random_state=42)

                modelo_rf.fit(X_resampled, y_resampled)
                previsoes = modelo_rf.predict(X_teste)
                recall = recall_score(y_teste, previsoes)
                precision = precision_score(y_teste, previsoes)
                acuracia = accuracy_score(y_teste, previsoes)

                # Salvando no historico apenas quando é encontrado um valor mais alto
                # caso contrario salva o valor anterior
                if high_recall<recall:
                    historico_recall.append(recall)
                    high_recall=recall
                else:
                    historico_recall.append(high_recall)

                if high_precision<precision:
                    historico_precision.append(precision)
                    high_precision=precision
                else:
                    historico_precision.append(high_precision)

                if high_accuracy<acuracia:
                    historico_acuracia.append(acuracia)
                    high_accuracy=acuracia
                else:
                    historico_acuracia.append(high_accuracy)

                # Verificando qual conjunto de hiperparametros retorna a melhor revocação
                if recall > melhor_recall:
                    melhor_recall = recall
                    melhores_parametros = {
                        'n_estimators': estimators,
                        'max_depth': depth,
                        'min_samples_split': split,
                        'min_samples_leaf': leaf
                    }
```

Fonte: Autoria própria

Após isso iremos exibir o melhor índice de revocação encontrado, bem como a combinação de parâmetros que retornou tal revocação (Figura 78).

Figura 78 - Impressão dos resultados obtidos

```
# Impressão das melhores métricas do conjunto repassado que podem ser usadas
print("Melhores hiperparâmetros para otimizar o Recall:", melhores_parametros)
print("Melhor Recall:", melhor_recall)
```

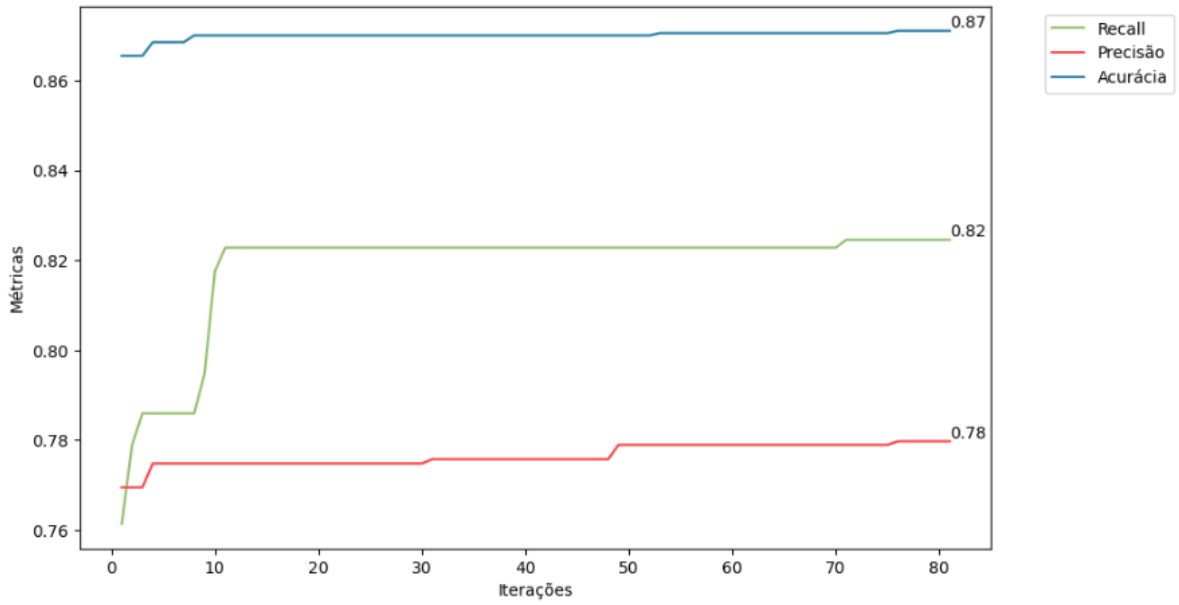
```
Melhores hiperparâmetros para otimizar o Recall: {'n_estimators': 150, 'max_depth': 10, 'min_samples_split': 10, 'min_samples_l
eaf': 2}
Melhor Recall: 0.8245614035087719
```

Fonte: Autoria própria

Podemos visualizar ainda, um gráfico que mostra a evolução dos indicadores conforme os laços eram executados (Figura 79).

Figura 79 - Evolução das métricas conforme execução dos laços de repetição

```
iteracoes = range(1, len(historico_recall) + 1)

# Plotando as linhas
plt.plot(iteracoes, historico_recall, label='Recall', color='#90BE6D')
plt.plot(iteracoes, historico_precision, label='Precisão', color='#F94144')
plt.plot(iteracoes, historico_acuracia, label='Acurácia', color='#277DA1')

# Adicionando rótulos apenas no último ponto de cada linha
plt.text(iteracoes[-1], historico_recall[-1], f'{historico_recall[-1]:.2f}', ha='left', va='bottom')
plt.text(iteracoes[-1], historico_precision[-1], f'{historico_precision[-1]:.2f}', ha='left', va='bottom')
plt.text(iteracoes[-1], historico_acuracia[-1], f'{historico_acuracia[-1]:.2f}', ha='left', va='bottom')
plt.gcf().set_size_inches(10, 6)

# Configurando o gráfico
plt.xlabel('Iterações')
plt.ylabel('Métricas')
plt.legend(bbox_to_anchor=(1.05, 1))
plt.show()
```



Fonte: Autoria própria

Lembrando que, como o nosso foco é a revocação, utilizaremos no modelo dentro do conjunto de parâmetros repassado os valores que retornaram a revocação mais alta.

Executaremos o modelo mais uma vez, mas agora com os parâmetros adicionados e em seguida, salvaremos as métricas em nossa tabela de relatório (Figura 80).

Figura 80 - Registrando métricas ajustadas na tabela de relatório

```
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.3, random_state=42)
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_treino, y_treino)
modelo_rf = RandomForestClassifier(n_estimators=150,max_depth=10,min_samples_split=10,
                                   min_samples_leaf=2,random_state=42)
modelo_rf.fit(X_resampled, y_resampled)
previsoes = modelo_rf.predict(X_teste)
df_relatorio = geraRelatorio(df_relatorio,y_teste,'Random Forest','mudança de hiperparametros melhorando o recall')
df_relatorio
```

	parametro	0	1	modelo	obs
0	precision	0.876147	0.844789	Random Forest	OneHotEncoding e ajuste de multicolinearidade
1	recall	0.950249	0.668421	Random Forest	OneHotEncoding e ajuste de multicolinearidade
2	f1-score	0.911695	0.746327	Random Forest	OneHotEncoding e ajuste de multicolinearidade
3	precision	0.875654	0.846325	Random Forest	Ajuste de normalização dos dados
4	recall	0.950959	0.666667	Random Forest	Ajuste de normalização dos dados
5	f1-score	0.911755	0.745829	Random Forest	Ajuste de normalização dos dados
6	precision	0.901767	0.766904	Random Forest	Balanceamento com a técnica de SMOTE
7	recall	0.906894	0.756140	Random Forest	Balanceamento com a técnica de SMOTE
8	f1-score	0.904323	0.761484	Random Forest	Balanceamento com a técnica de SMOTE
9	precision	0.924185	0.714286	Random Forest	mudança de hiperparametros melhorando o recall
10	recall	0.866382	0.824561	Random Forest	mudança de hiperparametros melhorando o recall
11	f1-score	0.894351	0.765472	Random Forest	mudança de hiperparametros melhorando o recall

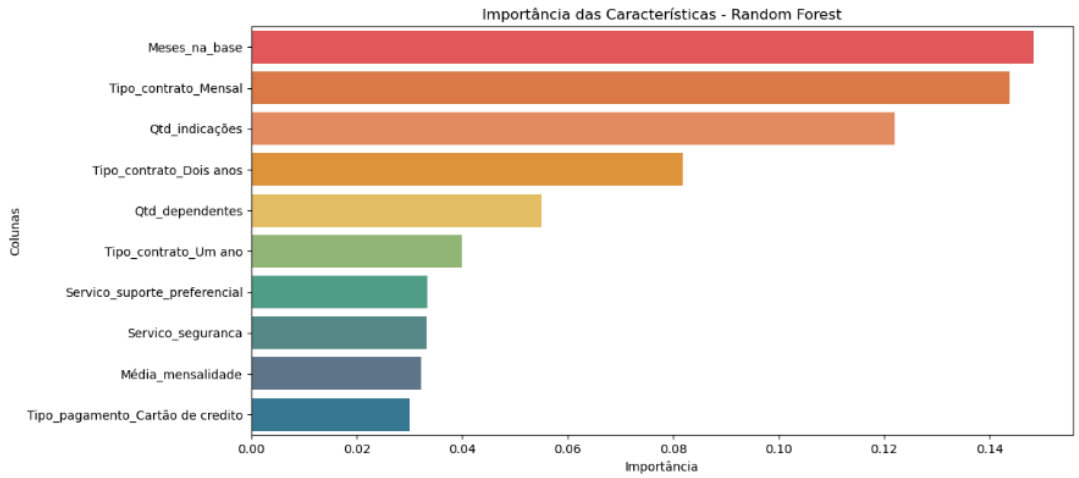
Fonte: Autoria própria

Uma informação interessante que podemos observar, se trata das colunas que mais contribuíram para o modelo. Essas colunas e seu nível de importância podem ser observados a partir do código executado na Figura 81.

Figura 81 - Importância das colunas no modelo

```
cores_paleta = ['#F94144','#F3722C','#F9844A','#F8961E','#F9C74F',
                '#90BE6D','#43AA8B','#4D908E','#577590','#277DA1']
```

```
importancias = modelo_rf.feature_importances_
importancias_df = pd.DataFrame({'Feature': X.columns, 'Importance': importancias})
importancias_df = importancias_df.sort_values(by='Importance', ascending=False)
plt.figure(figsize=(12, 6))
sns.barplot(x='Importance', y='Feature', data=importancias_df.head(10), palette=cores_paleta)
plt.title('Importância das Características - Random Forest')
plt.ylabel('Colunas')
plt.xlabel('Importância')
plt.show()
```

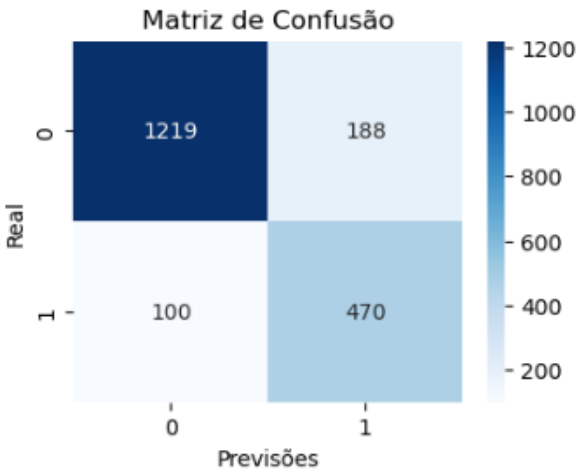


Fonte: Autoria própria

Por fim, geraremos novamente a matriz de confusão para observar os dados gerados com os últimos parâmetros configurados (Figura 82).

Figura 82 - Matriz de confusão com a melhor revocação

```
matriz_confusao = confusion_matrix(y_teste, previsoes)
plt.figure(figsize=(4, 3))
sns.heatmap(matriz_confusao, annot=True, fmt='d', cmap='Blues',
            xticklabels=modelo_rf.classes_,
            yticklabels=modelo_rf.classes_)
plt.xlabel('Previsões')
plt.ylabel('Real')
plt.title('Matriz de Confusão')
plt.show()
```



Fonte: Autoria própria

5.3 Regressão Logística

O próximo modelo de aprendizado de máquina que iremos utilizar, é o da regressão logística. Sua escolha se justifica pelo fato de possuir uma alta resistência a *outliers*[8], o que a torna mais robusta nas situações em que dados incomuns possam estar presentes, além de se encaixar em nossa situação por ser um modelo de classificação binária (0 ou 1). Podemos citar ainda sua capacidade de lidar com multicolinearidade e a não necessidade de normalização dos dados. Dito isto, como tais práticas já integram nosso *pipeline*[9], optamos por realizar esses ajustes como medida preventiva, em consonância com as boas práticas de pré-processamento de dados.

Para iniciar, iremos retornar o *dataframe* "df1" ao estado inicial, e então repassar os passos de pré-processamento, registrando em cada etapa as métricas de precisão, revocação e f1-scores alcançadas.

Em nossa primeira execução do algoritmo após o uso do *OneHotEncoding* e de ajustes de multicolinearidade conseguimos os seguintes resultados (Figura 83).

Figura 83 - Primeira execução da regressão logística

```
from sklearn.linear_model import LogisticRegression
```

```
X = df1.drop(['Situacao_cliente'], axis=1)
y = df1['Situacao_cliente']

X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.3, random_state=42)
modelo_lr = LogisticRegression(max_iter=1000,random_state=42)
modelo_lr.fit(X_treino, y_treino)
previsoes = modelo_lr.predict(X_teste)
acuracia = accuracy_score(y_teste, previsoes)
print('Acurácia:', acuracia)
print(classification_report(y_teste, previsoes))
```

```
Acurácia: 0.8492665655032878
              precision    recall  f1-score   support

           0       0.89      0.91      0.90      1407
           1       0.75      0.71      0.73       570

    accuracy                           0.85      1977
   macro avg       0.82      0.81      0.81      1977
weighted avg       0.85      0.85      0.85      1977
```

Fonte: Autoria própria

[8] Valores atípicos dentro de um conjunto de dados
[9] Processo de dividir as tarefas sequenciais em estágios distintos

Em seguida, após realizar as etapas de normalização dos dados e balanceamento das classes, obtemos respectivamente os resultados da Figura 84 e da Figura 85 respectivamente.

Figura 84 - Resultado após a normalização

Acurácia: 0.855336368234699

	precision	recall	f1-score	support
0	0.89	0.91	0.90	1407
1	0.76	0.72	0.74	570
accuracy			0.86	1977
macro avg	0.83	0.82	0.82	1977
weighted avg	0.85	0.86	0.85	1977

Fonte: Autoria própria

Figura 85 - Resultado após o balanceamento

Acurácia: 0.8457258472432979

	precision	recall	f1-score	support
0	0.91	0.87	0.89	1407
1	0.71	0.78	0.74	570
accuracy			0.85	1977
macro avg	0.81	0.82	0.82	1977
weighted avg	0.85	0.85	0.85	1977

Fonte: Autoria própria

É possível observar na Figura 84 como o modelo de regressão logística lida bem com dados não normalizados, pois mesmo após o ajuste, pouco mudou em relação as métricas de saída, diferentemente dos dados mostrados na Figura 85 que foram gerados após realizarmos o balanceamento das classes de clientes que permaneceram e que cancelaram.

Novamente iremos ainda realizar uma validação cruzada para testar o nosso modelo. Como visualizado na Figura 86, temos um modelo com um bom desempenho onde as métricas se mantem estáveis e com um desvio padrão baixo.

Figura 86 - Validação cruzada no modelo de regressão logística

```
k=5
kf = KFold(n_splits=k, shuffle=True, random_state=42)
modelo_lr = LogisticRegression(max_iter=1000,random_state=42)

# Métricas que serão availaidas
scoring = {'precision': make_scorer(precision_score),
           'recall': make_scorer(recall_score)}

# Executando a validação cruzada e obtendo as métricas para cada fold
precision_scores = cross_val_score(modelo_lr, X, y, cv=kf, scoring=scoring['precision'])
recall_scores = cross_val_score(modelo_lr, X, y, cv=kf, scoring=scoring['recall'])

# Impressão das métricas para cada fold
for i in range(k):
    print(f"Fold {i + 1} - Precisão: {precision_scores[i]}, Recall: {recall_scores[i]}")

# Métricas agregadas
print("\nMédia - Precisão:", np.mean(precision_scores))
print("Desvio Padrão - Precisão:", np.std(precision_scores))
print("Média - Recall:", np.mean(recall_scores))
print("Desvio Padrão - Recall:", np.std(recall_scores))
```

```
Fold 1 - Precisão: 0.7840909090909091, Recall: 0.720626631853786
Fold 2 - Precisão: 0.7556179775280899, Recall: 0.7309782608695652
Fold 3 - Precisão: 0.7245989304812834, Recall: 0.7226666666666667
Fold 4 - Precisão: 0.7239583333333334, Recall: 0.7533875338753387
Fold 5 - Precisão: 0.7857142857142857, Recall: 0.7352941176470589

Média - Precisão: 0.7547960872295802
Desvio Padrão - Precisão: 0.027120591550097547
Média - Recall: 0.732590642182483
Desvio Padrão - Recall: 0.011696513073167355
```

Fonte: Autoria própria

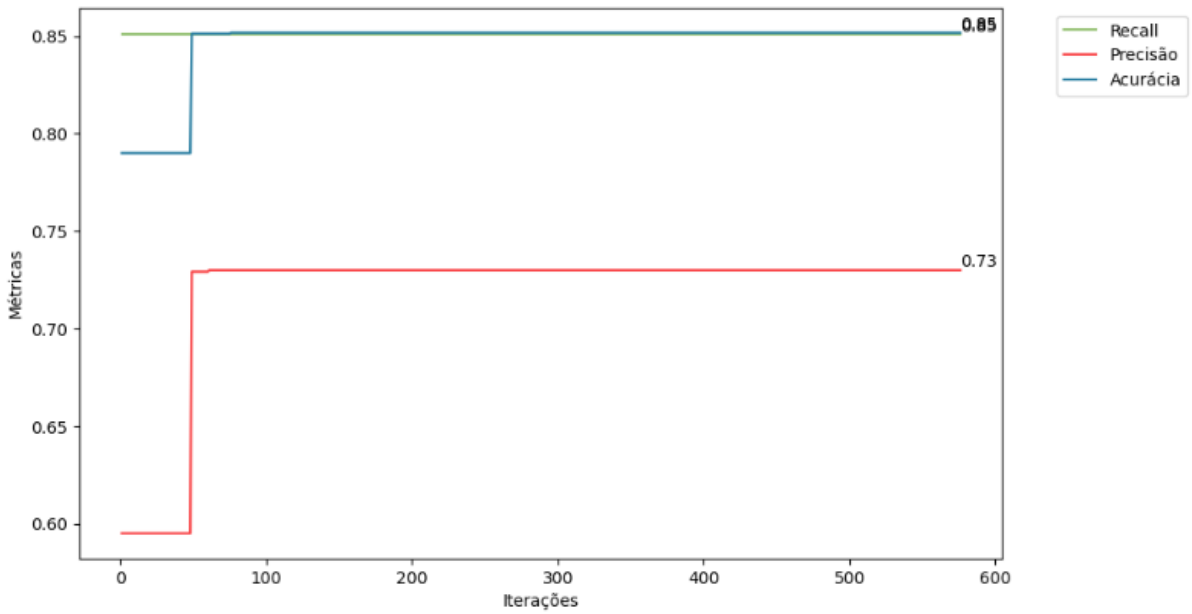
Em seguida, iremos fazer a verificação do melhor conjunto de parâmetros. Na Figura 87, temos os parâmetros que serão testados, e na Figura 88 o gráfico mostrando a evolução das métricas conforme os laços de repetição foram sendo executados.

Figura 87 - Conjunto de parâmetros testados para a regressão logística

```
# Relação de parametros que terão suas combinações testadas
parametros = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l2', 'none'],
    'solver': ['newton-cg', 'lbfgs', 'sag', 'saga'],
    'max_iter': [100, 200, 500, 1000],
    'tol': [1e-4, 1e-3, 1e-2]
}
```

Fonte: Autoria própria

Figura 88 - Evolução das métricas da regressão logística

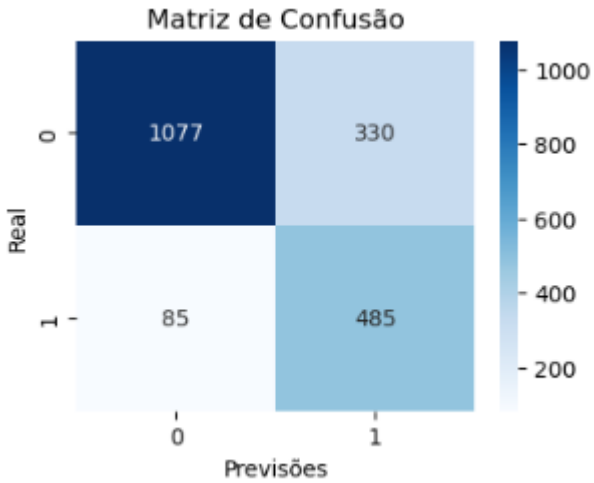


Fonte: Autoria própria

Um ponto que pode ser observado é que o valor mais alto da revocação foi atingido logo no início das execuções dos laços de repetição.

Novamente geraremos uma matriz de confusão para observarmos os quantitativos de acerto (Figura 89).

Figura 89 - Matriz de confusão da regressão logística

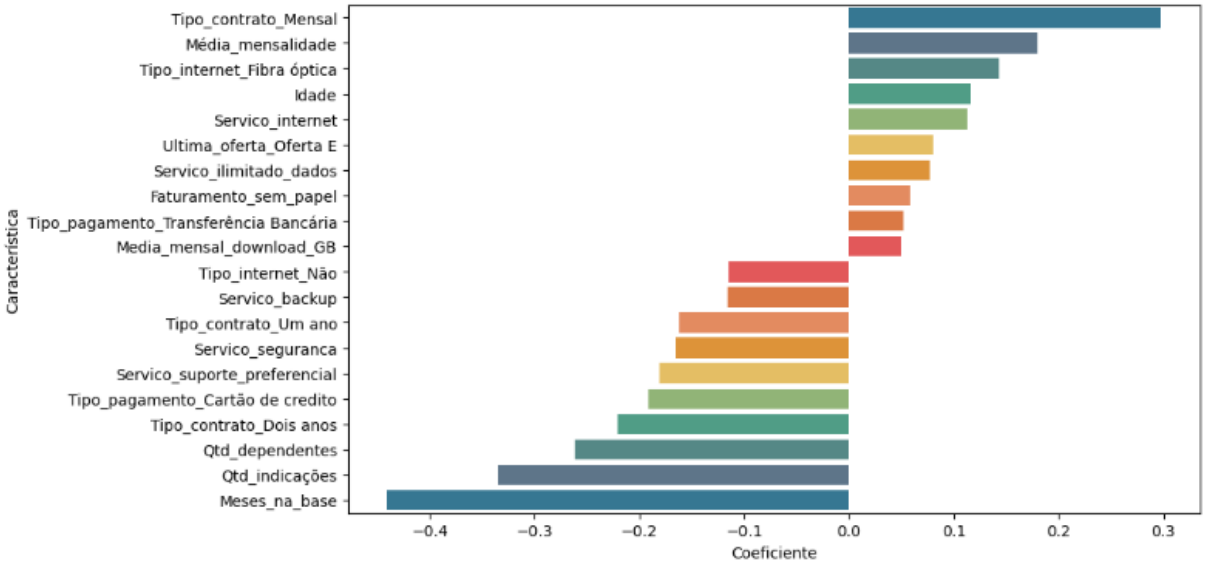


Fonte: Autoria própria

Nota-se que nossa escolha de priorizar a revocação também faz com que a precisão do modelo seja um pouco prejudicada, fazendo com que o modelo seja mais sensível ao determinar um cliente como potencial cancelamento.

Por fim, ao analisarmos os coeficientes do nosso modelo de regressão logística (Figura 90), temos algo próximo ao resultado das colunas que mais contribuíram no modelo de *Random Forest*.

Figura 90 - Coeficientes da regressão logística



Fonte: Autoria própria

5.4 SVM (Support Vector Machine)

O último algoritmo que testaremos será o SVM, sua escolha se justifica por ser um algoritmo que consegue lidar bem com base de dados com alto número de características (colunas) e ter robustez contra *overfitting*. Uma de suas desvantagens, porém, é sua sensibilidade ao lidar com dados com escalas muito distintas, algo que será ajustado no nosso *pipeline*.

Antes de tudo, iremos restaurar novamente a nossa base de dados df1 para uma fase em que temos apenas as colunas ajustadas com o *OneHotEnconding* e a remoção das colunas que causam multicolinearidade e em seguida, executaremos o algoritmo em suas configurações padrão (Figura 91).

Figura 91 – Primeira execução do SVM

```
from sklearn.svm import SVC
```

```
X = df1.drop(['Situacao_cliente'], axis=1)
y = df1['Situacao_cliente']
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.3, random_state=42)
modelo_svm = SVC(random_state=42)
modelo_svm.fit(X_treino, y_treino)
previsoes = modelo_svm.predict(X_teste)
accuracy = accuracy_score(y_teste, previsoes)
print('Acurácia:', accuracy)
print(classification_report(y_teste, previsoes))
```

```
Acurácia: 0.8002023267577137
              precision    recall  f1-score   support

           0       0.82      0.92      0.87      1407
           1       0.71      0.52      0.60       570

    accuracy                           0.80      1977
   macro avg       0.77      0.72      0.73      1977
weighted avg       0.79      0.80      0.79      1977
```

Fonte: Autoria própria

Ao realizarmos o ajuste da escala dos dados fica perceptível a melhoria nas métricas (Figura 92).

Figura 92 - Execução após o ajuste das escalas dos dados

```
Acurácia: 0.8573596358118362
              precision    recall  f1-score   support

           0       0.89      0.92      0.90      1407
           1       0.77      0.71      0.74       570

    accuracy                           0.86      1977
   macro avg       0.83      0.81      0.82      1977
weighted avg       0.85      0.86      0.86      1977
```

Fonte: Autoria própria

Em seguida, após a etapa de balanceamento das classes, conseguimos obter os seguintes resultados (Figura 93).

Figura 93 - Execução após o balanceamento das classes

```
Acurácia: 0.8492665655032878
              precision    recall  f1-score   support

           0       0.91      0.88      0.89      1407
           1       0.72      0.78      0.75       570

    accuracy                           0.85      1977
   macro avg       0.81      0.83      0.82      1977
weighted avg       0.85      0.85      0.85      1977
```

Fonte: Autoria própria

Ao realizarmos a validação cruzada conforme a Figura 94, podemos observar que o modelo apresenta um bom desempenho com um baixo valor de desvio padrão, se mostrando assim, consistente durante as execuções da validação cruzada.

Figura 94 - Aplicação da validação cruzada no SVM

```
k=5
kf = KFold(n_splits=k, shuffle=True, random_state=42)
modelo_svm = SVC(random_state=42)

# Métricas que serão availaidas
scoring = {'precision': make_scorer(precision_score),
           'recall': make_scorer(recall_score)}

# Executando a validação cruzada e obtendo as métricas para cada fold
precision_scores = cross_val_score(modelo_svm, X, y, cv=kf, scoring=scoring['precision'])
recall_scores = cross_val_score(modelo_svm, X, y, cv=kf, scoring=scoring['recall'])

# Impressão das métricas para cada fold
for i in range(k):
    print(f"Fold {i + 1} - Precisão: {precision_scores[i]}, Recall: {recall_scores[i]}")

# Métricas agregadas
print("\nMédia - Precisão:", np.mean(precision_scores))
print("Desvio Padrão - Precisão:", np.std(precision_scores))
print("Média - Recall:", np.mean(recall_scores))
print("Desvio Padrão - Recall:", np.std(recall_scores))
```

```
Fold 1 - Precisão: 0.7987987987987988, Recall: 0.6945169712793734
Fold 2 - Precisão: 0.7430167597765364, Recall: 0.7228260869565217
Fold 3 - Precisão: 0.7543352601156069, Recall: 0.696
Fold 4 - Precisão: 0.7401129943502824, Recall: 0.7100271002710027
Fold 5 - Precisão: 0.793002915451895, Recall: 0.7272727272727273

Média - Precisão: 0.765853345698624
Desvio Padrão - Precisão: 0.025056909854865236
Média - Recall: 0.7101285771559249
Desvio Padrão - Recall: 0.01340520694668742
```

Fonte: Autoria própria

Dando prosseguimento, realizaremos agora o código com laços de repetição, que irá retornar na saída o conjunto de parâmetros que resulta na melhor revocação dentre os parâmetros repassados, na Figura 95 podemos visualizar a lista de parâmetros e na Figura 96 podemos visualizar a saída resultante, com uma revocação de 85,26%.

Figura 95 - Conjunto de parâmetros testados para o SVM

```
parametros = {
    'gamma': [0.001, 0.01, 0.1],
    'degree': [2, 3, 4,],
    'coef0': [-1, 0, 1]
}
```

Fonte: Autoria própria

Figura 96 - Saída com o conjunto de parâmetros resultante

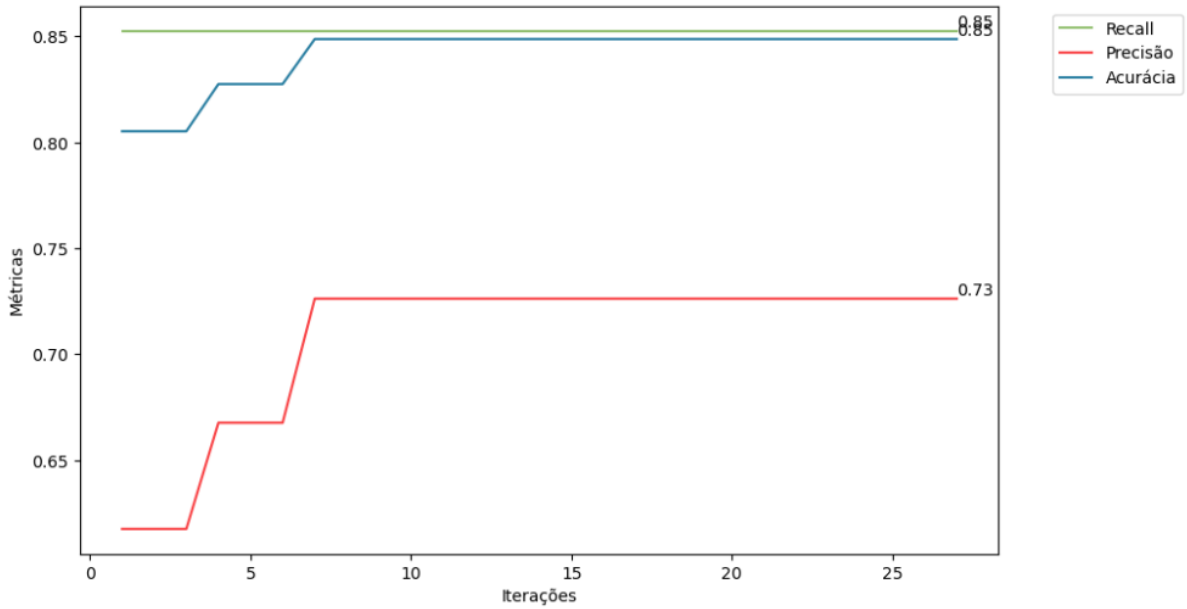
```
# Impressão das melhores métricas do conjunto repassado que podem ser usadas
print("Melhores hiperparâmetros para otimizar o Recall:", melhores_parametros)
print("Melhor Recall:", melhor_recall)
```

```
Melhores hiperparâmetros para otimizar o Recall: {'degree': 2, 'gamma': 0.001, 'coef0': -1}
Melhor Recall: 0.8526315789473684
```

Fonte: Autoria própria

Na Figura 97 temos o gráfico com a evolução das métricas, nela é possível ver que a revocação mais alta foi atingida logo no início das execuções.

Figura 97 - Evolução das métricas do SVM



Fonte: Autoria própria

Por fim, na Figura 98 temos o resultado da execução do modelo com o conjunto de métricas encontrado e na Figura 99 geramos a matriz de confusão a fim de observamos os quantitativos de acertos do nosso modelo.

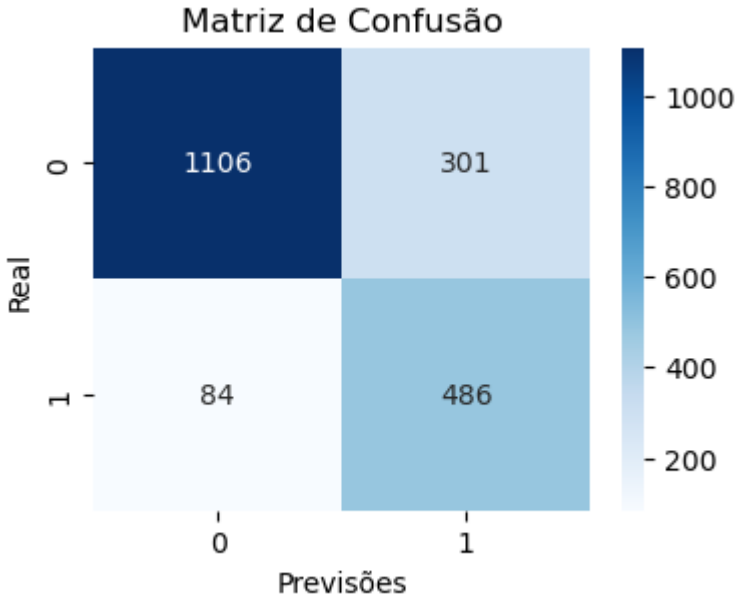
Figura 98 - Execução com o conjunto de parâmetros que foi encontrado

```
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.3, random_state=42)
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_treino, y_treino)
modelo_svm = SVC(random_state=42,
                 gamma=0.001,
                 degree=2,
                 coef0=-1
                )
modelo_svm.fit(X_resampled, y_resampled)
previsoes = modelo_svm.predict(X_teste)
print('Acurácia:', acuracia)
print(classification_report(y_teste, previsoes))
```

```
Acurácia: 0.8487607486090035
              precision    recall  f1-score   support

           0       0.93      0.79      0.85      1407
           1       0.62      0.85      0.72       570

    accuracy                           0.81      1977
   macro avg       0.77      0.82      0.78      1977
weighted avg       0.84      0.81      0.81      1977
```

Fonte: Autoria própria

Figura 99 - Matriz de confusão SVM



Fonte: Autoria própria

6 APRESENTAÇÃO DOS RESULTADOS

Durante as etapas de pré-processamentos e execuções dos nossos modelos, foi registrado em uma tabela os valores das métricas atingidas em cada etapa realizada. Com esses valores registrados podemos gerar gráficos que irão nos permitir uma melhor análise acerca dos resultados obtidos.

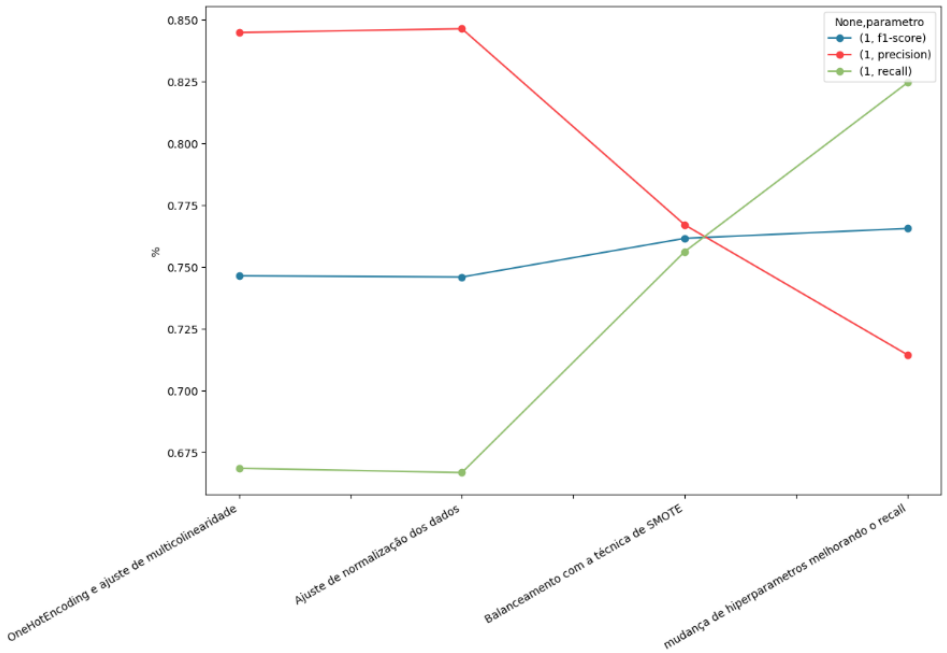
Para observarmos o efeito dos ajustes feitos até agora, iremos criar um gráfico com base na tabela de relatório em que estávamos salvando as métricas registradas. Na Figura 100 podemos ver o código utilizado e na Figura 101 o gráfico gerado.

Figura 100 - Código para geração de gráfico contendo evolução das métricas

```
val_ordem = ['OneHotEncoding e ajuste de multicolinearidade',
             'Ajuste de normalização dos dados',
             'Balanceamento com a técnica de SMOTE',
             'mudança de hiperparametros melhorando o recall']
df_relatorio_graf = df_relatorio[df_relatorio['obs'].isin(val_ordem)].reset_index(drop=True)
df_relatorio_graf = df_relatorio_graf[df_relatorio['modelo']=='Random Forest']
df_relatorio_graf['obs'] = pd.Categorical(df_relatorio_graf['obs'], categories=val_ordem, ordered=True)
df_relatorio_graf = df_relatorio_graf.sort_values(by='obs')
df_pivot = df_relatorio_graf.pivot(index='obs', columns='parametro', values=['1'])
ax = df_pivot.plot(kind='line', marker='o', figsize=(12, 8), color=['#277DA1','#F94144','#90BE6D'])
ax.set_ylabel('%')
ax.set_xlabel('Ajustes')
plt.xticks(rotation=30,ha='right')
plt.show()
```

Fonte: Autoria própria

Figura 101 - Gráfico com a evolução das métricas após cada ajuste realizado (*Random Forest*)

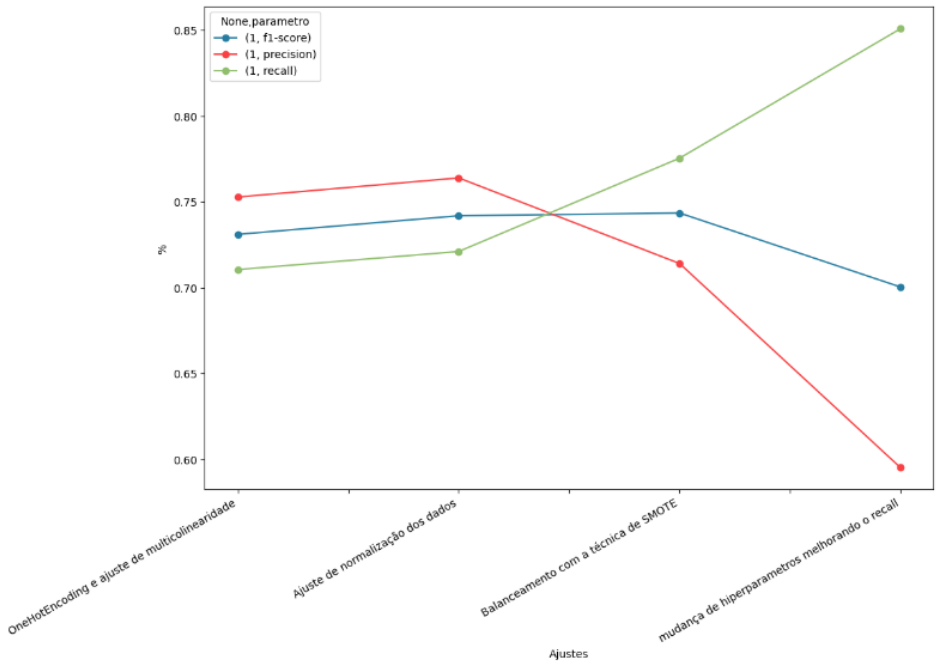


Fonte: Autoria própria

Dentre os pontos que podemos analisar, temos o fato de que praticamente não houve mudanças nos parâmetros após o ajuste de normalização dos dados, mostrando como o algoritmo de *Random Forest* se mostra eficiente mesmo com dados com escalas distintas. O cenário muda um pouco após realizarmos o balanceamento dos dados, mostrando que as métricas até então poderiam estar sendo tendenciosas em relação a classe majoritária que existia na base de dados. Por fim, na última etapa, otimizamos o modelo para retornar o resultado focando em uma melhor revocação.

Em seguida, podemos gerar o gráfico referente as métricas da regressão logística (Figura 102).

Figura 102 - Gráfico com a evolução das métricas após cada ajuste realizado (Regressão Logística)

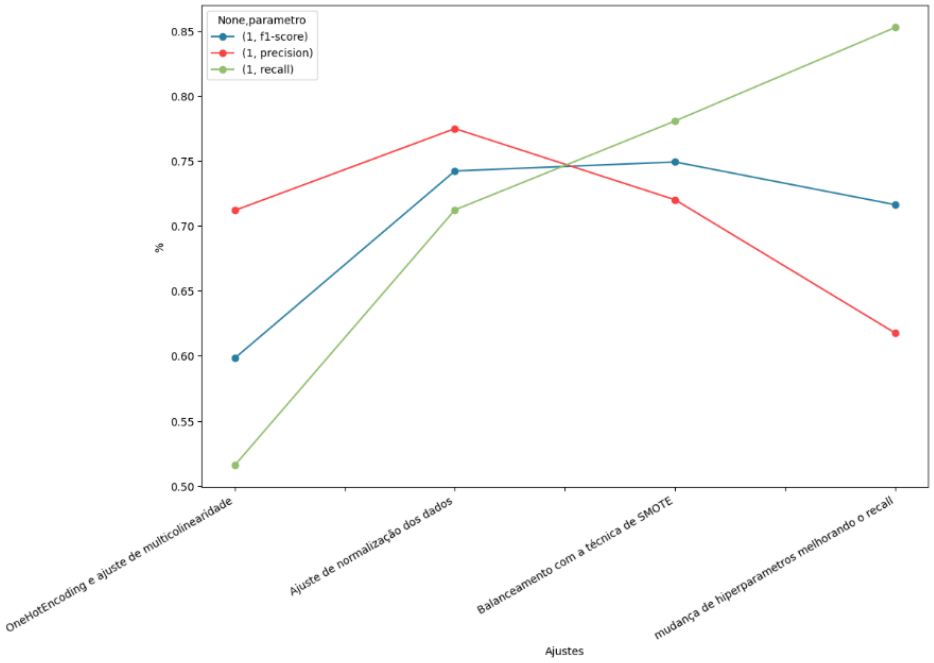


Fonte: Autoria própria

Neste gráfico podemos observar a capacidade do algoritmo em lidar com dados não normalizados, onde as métricas tiveram poucas alterações. No final, as alterações feitas priorizando o recall também resultaram em uma significativa redução da precisão.

Por fim iremos gerar também o gráfico referente ao algoritmo SVM (Figura 103).

Figura 103 - Gráfico com a evolução das métricas após cada ajuste realizado (SVM)



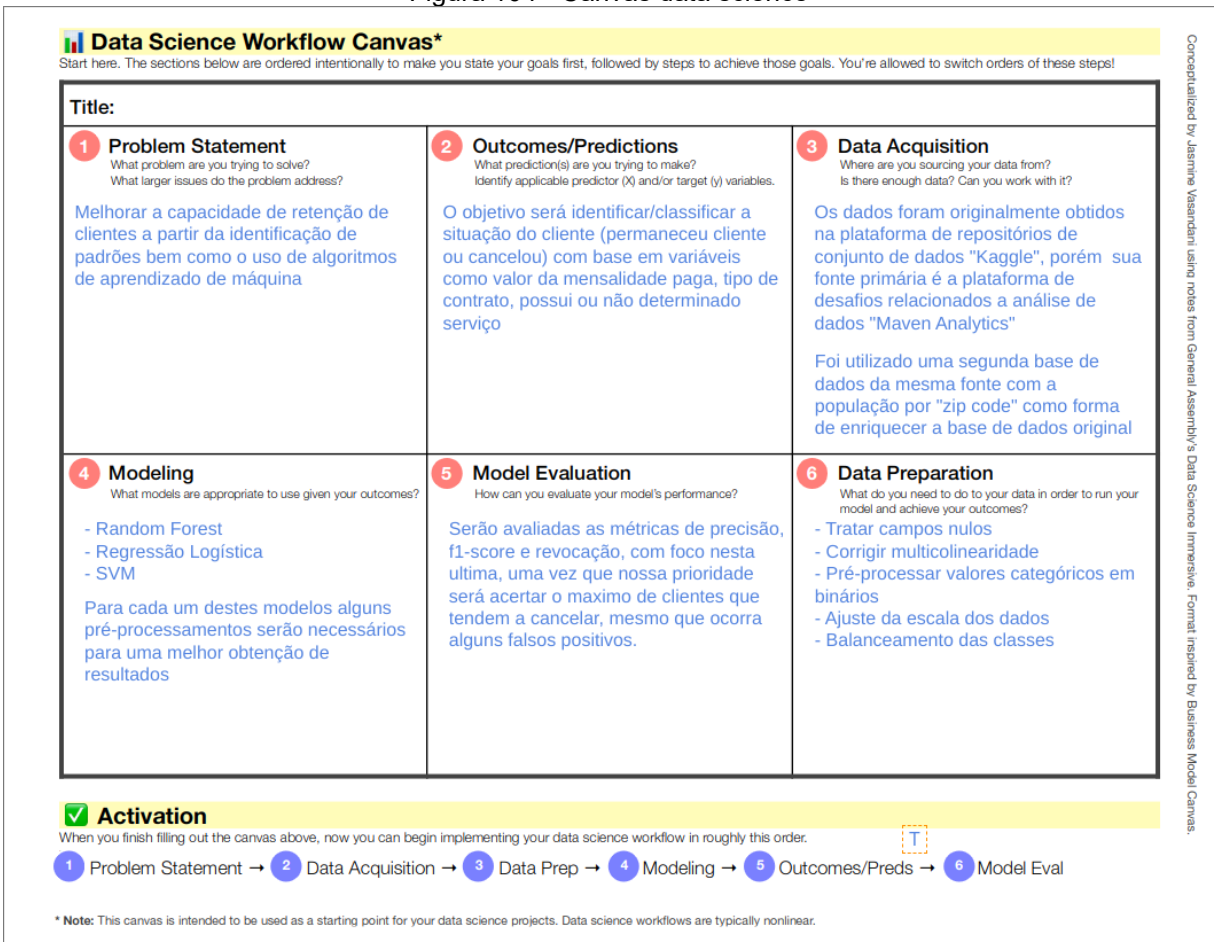
Fonte: Autoria própria

Neste gráfico, podemos observar como o algoritmo ganha performance ao realizarmos a normalização dos dados, mas assim como a regressão linear, também houve uma significativa redução da métrica de precisão.

Tendo isso em vista, em nosso cenário, recomendaríamos o uso do algoritmo Random Forest, pois apesar do mesmo também sofrer uma redução na métrica de precisão, está ainda permanece acima dos 70%.

Por fim, na Figura 104, temos o modelo do canvas proposto por Jasmine Vasandani (2019)

Figura 104 - Canvas data science



Data Science Workflow Canvas*

Start here. The sections below are ordered intentionally to make you state your goals first, followed by steps to achieve those goals. You're allowed to switch orders of these steps!

Title:

1 Problem Statement

What problem are you trying to solve?
What larger issues do the problem address?

Melhorar a capacidade de retenção de clientes a partir da identificação de padrões bem como o uso de algoritmos de aprendizado de máquina

2 Outcomes/Predictions

What prediction(s) are you trying to make?
Identify applicable predictor (X) and/or target (y) variables.

O objetivo será identificar/classificar a situação do cliente (permaneceu cliente ou cancelou) com base em variáveis como valor da mensalidade paga, tipo de contrato, possui ou não determinado serviço

3 Data Acquisition

Where are you sourcing your data from?
Is there enough data? Can you work with it?

Os dados foram originalmente obtidos na plataforma de repositórios de conjunto de dados "Kaggle", porém sua fonte primária é a plataforma de desafios relacionados a análise de dados "Maven Analytics"

Foi utilizado uma segunda base de dados da mesma fonte com a população por "zip code" como forma de enriquecer a base de dados original

4 Modeling

What models are appropriate to use given your outcomes?

- Random Forest
- Regressão Logística
- SVM

Para cada um destes modelos alguns pré-processamentos serão necessários para uma melhor obtenção de resultados

5 Model Evaluation

How can you evaluate your model's performance?

Serão avaliadas as métricas de precisão, f1-score e revocação, com foco nesta ultima, uma vez que nossa prioridade será acertar o maximo de clientes que tendem a cancelar, mesmo que ocorra alguns falsos positivos.

6 Data Preparation

What do you need to do to your data in order to run your model and achieve your outcomes?

- Tratar campos nulos
- Corrigir multicolinearidade
- Pré-processar valores categóricos em binários
- Ajuste da escala dos dados
- Balanceamento das classes

✅ Activation

When you finish filling out the canvas above, now you can begin implementing your data science workflow in roughly this order.

1 Problem Statement → 2 Data Acquisition → 3 Data Prep → 4 Modeling → 5 Outcomes/Preds → 6 Model Eval

* **Note:** This canvas is intended to be used as a starting point for your data science projects. Data science workflows are typically nonlinear.

Conceptualized by Jasmine Vasandani using notes from General Assembly's Data Science Immersive. Format inspired by Business Model Canvas.

Fonte: LAOYAN, Sarah. (2019)

7 LINKS

Link do repositório no GitHub: https://github.com/JuniorNS96/TCC.

Link do repositório no Google Drive: https://drive.google.com/drive/folders/14N708atFiMDrk5ej3rQbQAw1HZzUp6tJ?usp=sharing

REFERÊNCIAS

MAVEN ANALYTICS. **Telecom Customer Churn**, 2022. Disponível em: https://www.mavenanalytics.io/data-playground?search=telecom. Acesso em: 08 de out. de 2023.

DATABRICKS. **O que é um dataframe?**. Disponível em: https://www.databricks.com/br/glossary/what-are-dataframes. Acesso em: 08 de out. de 2023.

PANDAS. **Api Reference**. Disponível em: https://pandas.pydata.org/docs/reference/index.html. Acesso em: 08 de out. de 2023.

LAOYAN, Sarah. **Entendendo o princípio de Pareto (a regra 80/20)**, 2022. Disponível em: https://asana.com/pt/resources/pareto-principle-80-20-rule. Acesso em: 17 de nov. de 2023

SCIKIT-LEARN. **Machine Learning in Python**. Disponível em: https://scikit-learn.org/stable/. Acesso em: 17 de nov. de 2023

APRENDA DATA SCIENCE. **Introdução à Correlação**. Disponível em: https://www.aprendadatascience.com/blog/introdu%C3%A7%C3%A3o-a-correla%C3%A7%C3%A3o. Acesso em: 22 de nov. de 2023

MINITAB. **Basta! Lidando com a multicolinearidade na análise de regressão**, 2019. Disponível em: https://blog.minitab.com/pt/basta-lidando-com-a-multicolinearidade-na-analise-de-regressao#:~:text=A%20multicolinearidade%20ocorre%20quando%20o,certa%20forma%2C%20um%20pouco%20redundantes. Acesso em: 22 de nov. de 2023

VASANDANI, Jasmine. **A Data Science Workflow Canvas to Kickstart Your Projects**, 2019. Disponível em: https://towardsdatascience.com/a-data-science-workflow-canvas-to-kickstart-your-projects-db62556be4d0. Acesso em: 26 de out. de 2023.