

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**  
**NÚCLEO DE EDUCAÇÃO A DISTÂNCIA**  
**Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data**

**Yuri Aranha Kawagoe**

**Previsão da qualidade de serviço na resolução de falhas por fornecedores de  
EILD (Exploração Industrial de Linha Direta) utilizando algoritmos  
classificadores.**

Belo Horizonte

2021

**Yuri Aranha Kawagoe**

**PREVISÃO DA QUALIDADE DE SERVIÇO NA RESOLUÇÃO DE FALHAS POR  
FORNECEDORES DE EILD (EXPLORAÇÃO INDUSTRIAL DE LINHA DIRETA)  
UTILIZANDO ALGORITMOS CLASSIFICADORES.**

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

2021

## SUMÁRIO

1. Introdução .....	4
1.1. Contextualização .....	5
1.2. O problema proposto .....	7
2. Coleta de Dados .....	9
2.1. Dados do OSS – Gestão da Planta/Query Builder .....	9
2.2. Dados do Report .....	13
2.3. Dados da Confederação Nacional do Transporte .....	15
3. Processamento/Tratamento de Dados .....	16
3.1. Ferramentas utilizadas e metodologia .....	16
3.2. Processamento e tratamento das bases de dados do OSS – Gestão da Planta/Query Builder .....	17
3.3. Processamento e tratamento da base de dados do Report .....	19
3.4. Processamento e tratamento da base de dados do CNT .....	21
4. Análise e Exploração dos Dados .....	26
5. Criação de Modelos de Machine Learning .....	30
5.1. Modelo 1: “ <i>Gaussian Naive Bayes</i> ” .....	30
5.2. Modelo 2: “ <i>Decision Tree Classifier</i> ” .....	40
6. Apresentação dos Resultados .....	52
7. Links .....	61
REFERÊNCIAS .....	62
APÊNDICE .....	63

## 1. Introdução

A expansão dos serviços de telecomunicações no Brasil tem possibilitado a criação de provedores regionais e o atendimento de áreas com menores densidades populacionais a partir da terceirização de infraestruturas de redes de comunicação. A oferta de serviços de telecomunicações a clientes governamentais e privados ocorre em todo âmbito nacional, todavia as grandes empresas do setor não possuem rede própria para atendimento em todas as localidades, uma vez que os custos de construção e manutenção se tornam inviáveis para os negócios.

Atualmente, as grandes empresas optam por um serviço contratado denominado EILD, cuja sigla se refere a “Exploração Industrial de Linha Dedicada”. Definida como uma modalidade de Exploração Industrial em que uma Prestadora de Serviços de Telecomunicações fornece a outra Prestadora de Serviços de Telecomunicações, mediante remuneração preestabelecida, Linha Dedicada com características técnicas definidas para constituição da rede de serviços desta última, conforme Resolução número 590, de 15 de maio de 2012 da Anatel (Agência Nacional de Telecomunicações).

Os serviços fornecidos pelas empresas de telecomunicações estão sujeitos a falhas, por isso, conforme as boas práticas, as empresas empregam o Gerenciamento de Incidentes, que tem por objetivo restaurar a operação normal do serviço o mais rápido possível e garantir, desta forma, os melhores níveis de qualidade e disponibilidade do serviço (guia ITIL v3).

A qualidade do serviço é fator determinante para o negócio. As empresas do mercado de telecomunicações precisam responder de forma rápida e eficaz às exigências de seus consumidores devido ao ambiente competitivo onde estão inseridas. Parasuraman et al. (1988) afirmam que a qualidade percebida do serviço é um resultado da comparação das percepções com as expectativas do cliente. Portanto, cabe aos prestadores de serviço conhecer as expectativas dos seus clientes para buscar melhorias de desempenho que favoreçam uma percepção sempre boa.

No contexto de prestação de serviços por fornecedores de EILD, a qualidade do serviço depende de diversos fatores. As falhas podem ser resolvidas de maneira remota ou podem necessitar de deslocamento de técnicos para manutenções em

campo, seja no ambiente do cliente ou na estação do fornecedor. Nesses casos, a qualidade da infraestrutura rodoviária de cada estado pode afetar significativamente no tempo de deslocamento, interferindo tanto no tempo de recuperação da falha quanto na percepção de qualidade do serviço por parte do cliente.

### 1.1. Contextualização

A TELEBRAS, empresa pública vinculada ao ministério das comunicações fornece soluções de telecomunicações com atuação nacional e o propósito de levar conectividade em alta capacidade a todas as localidades do país, promovendo as políticas públicas de inclusão digital do Estado e atendendo às demandas de soluções em serviços de conexão para a Administração Pública. A TELEBRAS possui uma rede óptica de alta resiliência de mais de 28 mil km no território nacional, além de um Satélite Geoestacionário de Defesa e Comunicações Estratégicas (SGDC) que cobre 8.516.000 km<sup>2</sup>.

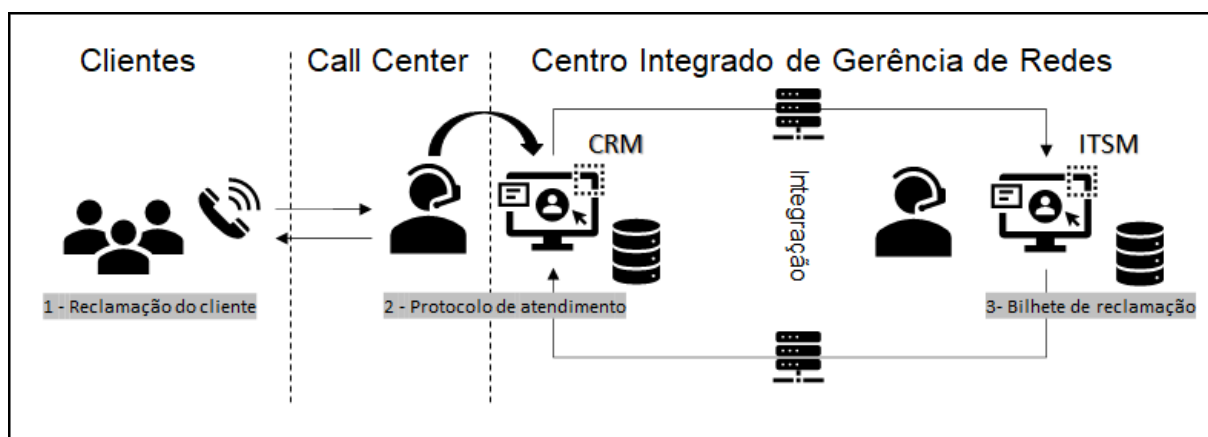
Apesar de sua alta capacidade em rede de *backbone*, a TELEBRAS não possui redes metropolitanas capazes de atender seus clientes em todas as localidades. Para suprir essa lacuna, optou-se pela contratação de fornecedores de redes de última milha ou EILD's (Exploração Industrial de Linha Dedicada). Essa opção reduz custos de investimentos na construção de redes próprias e permite maior flexibilização no atendimento dos contratos uma vez que muitos órgãos da Administração tem filiais distribuídas em vários estados Brasileiros.

Os contratos firmados entre a TELEBRAS e seus clientes possuem, em seus termos de referências, acordos de níveis de serviço e qualidade que precisam ser cumpridos para que o serviço seja faturado em sua totalidade e não haja prejuízos para ambas as partes.

A operação e manutenção dos serviços devem ser realizadas de forma eficiente e eficaz, de forma ininterrupta, salvo casos previstos nos contratos. Os serviços dos clientes são gerenciados e monitorados através dos sistemas de suporte a operação (em inglês OSS, *Operations Support System*) e todas as solicitações de clientes, incluindo reclamações de serviço, são recebidas através da central de relacionamento e registradas na plataforma de gestão de relacionamento com o cliente (em inglês CRM, *Customer Relationship Management*). O CRM possui integração com o Sistema de Gerenciamento de Serviços de Tecnologia da

Informação (em inglês, ITSM, *IT Service Management*), que é uma das aplicações que compõe o OSS. Essa integração permite que as solicitações de reparo se transformem em bilhetes de incidentes e possam ser tratados no fluxo de gestão de incidentes, sendo avaliados e resolvidos pelo Centro Integrado de Gerência de Redes da TELEBRAS (CIGR), conforme esquematizado na figura 1.

Figura 1 - Fluxo de solicitação de clientes.



Fonte: Autor

A ferramenta ITSM foi desenvolvida com base nos processos ITIL. Todavia foram implementadas diversas melhorias para melhor adequação a gestão de operações de telecomunicações.

O processo de gestão de incidentes de clientes da TELEBRAS apresenta as seguintes fases:

1. Abrir Chamado – A abertura de chamados é feita via *call center* pelos clientes ou de forma proativa pelos analistas do CIGR. Em ambos os casos é realizada a abertura de um protocolo no CRM.
2. Categorizar e Priorizar – A categorização e priorização são feitas pela integração CRM x ITSM. O ITSM gera um bilhete com prioridade alta a partir da abertura de um protocolo de reparo no CRM.
3. Reconhecer e Diagnosticar – O reconhecimento e diagnóstico são realizados pelos analistas do CIGR, que verificam todas as informações registradas no chamado a fim diagnosticar o incidente

de forma precisa. Nessa fase, o analista verifica se determinado cliente é atendido com rede própria ou através de fornecedor EILD.

4. Escalonar – O escalonamento é realizado para o segundo nível de suporte ou para os técnicos de campo, quando o analista de nível 1 não consegue resolver o incidente, nos casos de responsabilidade da TELEBRAS. Caso os clientes sejam atendidos através de EILD é realizada a abertura de chamado de reparo junto ao fornecedor.
5. Resolver – A resolução é feita pelos analistas do CIGR de primeiro nível, segundo nível ou técnicos de campo quando a rede é exclusiva da TELEBRAS. Nos casos de clientes atendidos por EILD, a resolução é feita por analistas ou técnicos de campo da empresa fornecedora com acompanhamento dos analistas do CIGR.
6. Fechar – O fechamento é feito pelos analistas do CIGR de primeiro nível, registrando a solução aplicada ao incidente e avaliando a qualidade da tratativa.

Ao longo do processo os chamados são atualizados com informações direcionadas aos clientes e informações internas a fim de manter um histórico preciso em cada chamado. Durante o processo múltiplos escalonamentos podem ser realizados para que a qualidade e velocidade no tratamento dos incidentes sejam garantidas.

## **1.2. O problema proposto**

Visando atender os clientes com o nível mais alto de qualidade, é essencial que todos os agentes envolvidos estejam alinhados no processo de gestão de incidentes, principalmente os fornecedores de EILD. Nesse sentido, é fundamental que os analistas da equipe de gestão de serviços saibam de forma antecipada que determinado incidente poderá não ser atendido com a qualidade esperada. Todavia, ao longo do tempo percebeu-se que a qualidade do serviço não era constante, ela sempre variava de acordo com o fornecedor, o dia da semana, a hora de abertura do chamado, o tipo de acesso fornecido, entre outros. Nesse contexto, observou-se a

necessidade de elaboração de um modelo preditivo capaz de classificar um bilhete em função a qualidade de atendimento. Caso o modelo seja capaz de classificar corretamente a qualidade de um bilhete aberto os analistas poderão atuar de maneira mais eficiente e precisa, dando maior atenção aos casos sinalizados como “Sem qualidade (Não)” e realizar escalonamentos mais tempestivos, com o objetivo de reduzir o tempo de SLA e melhorar o serviço prestado ao cliente final, além de subsidiar a área de contratação de fornecedores apontando casos recorrentes, nos quais não há uma boa prestação de serviço, sugerindo possíveis trocas de fornecedores.

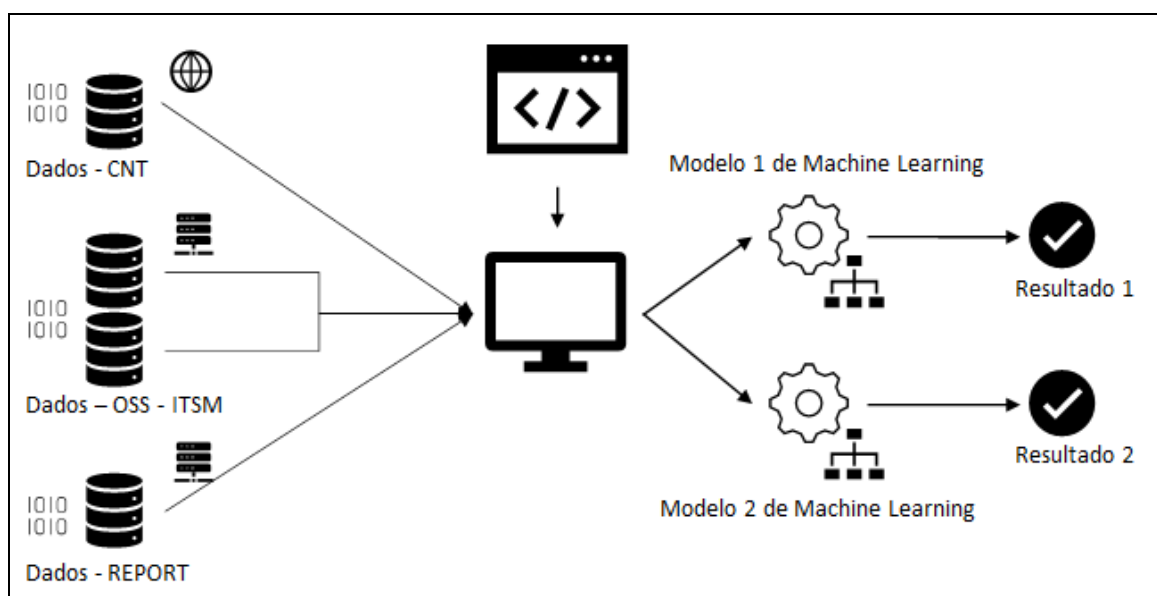
Para resolver o problema em questão optou-se pela aplicação de duas técnicas de aprendizagem supervisionada, na qual um programa de computador “toma decisões baseadas na experiência contida em exemplos solucionados com sucesso” (Weiss & Kulikowski, 1991). A primeira técnica escolhida foi o “Bayes Ingênuo” (em inglês, *Naive Bayes*), que consiste em um classificador probabilístico baseado no “Teorema de Bayes”. A segunda técnica escolhida foi a “Árvore de decisão”, que consiste em um classificador que particiona recursivamente um conjunto de dados até que cada subconjunto obtido contenha casos de uma única classe. Em seguida, serão avaliadas as métricas de cada modelo para verificar qual deles possui o melhor desempenho sobre os dados analisados.

Os dados analisados neste trabalho são dados técnicos operacionais de incidentes reais de clientes da TELEBRAS, ocorridos em todo território nacional, no primeiro semestre de 2020, que foram avaliados pela equipe de gestão de serviços do Centro Integrado de Gerência de Redes da TELEBRAS. A avaliação dos incidentes se deu com base na percepção de qualidade de atuação das empresas fornecedoras de EILD ao longo da vida útil de cada bilhete. A base de dados foi enriquecida com dados de cadastro de fornecedores de EILD da TELEBRAS e dados da pesquisa CNT (Confederação Nacional do Transporte) de Rodovias 2019, na qual seus objetivos principais são avaliar a qualidade das rodovias brasileiras e apontar suas principais deficiências e pontos críticos, sendo que a qualidade das rodovias Brasileiras se caracteriza como um atributo relevante para a percepção de qualidade na prestação do serviço, sendo frequentemente utilizada como justificativa pelos fornecedores para atrasos na recuperação do serviço.



Este trabalho busca, a partir da consolidação e limpeza da base de dados, seleção de atributos, criar modelos de aprendizado de máquina (em inglês, *machine learning*) capazes de classificar novos incidentes atendidos por fornecedores de EILD e comparar o desempenho dos modelos para selecionar o de melhor desempenho, conforme esquematizado na figura 2.

Figura 2 - Esquemático do Projeto.



Fonte: Autor

## 2. Coleta de Dados

Os dados utilizados neste trabalho foram coletados de dois sistemas internos da empresa: OSS – Gestão da Planta/*Query Builder* e *Report*. Após a coleta, foram exportados para o formato *Microsoft Excel*. Os dados externos foram coletados do relatório de pesquisa CNT de Rodovias referente ao ano de 2019. O relatório pode ser encontrado no sítio da Confederação Nacional de Transportes no *link* <https://pesquisarodovias.cnt.org.br/downloads/ultimaversao/gerencial.pdf>.

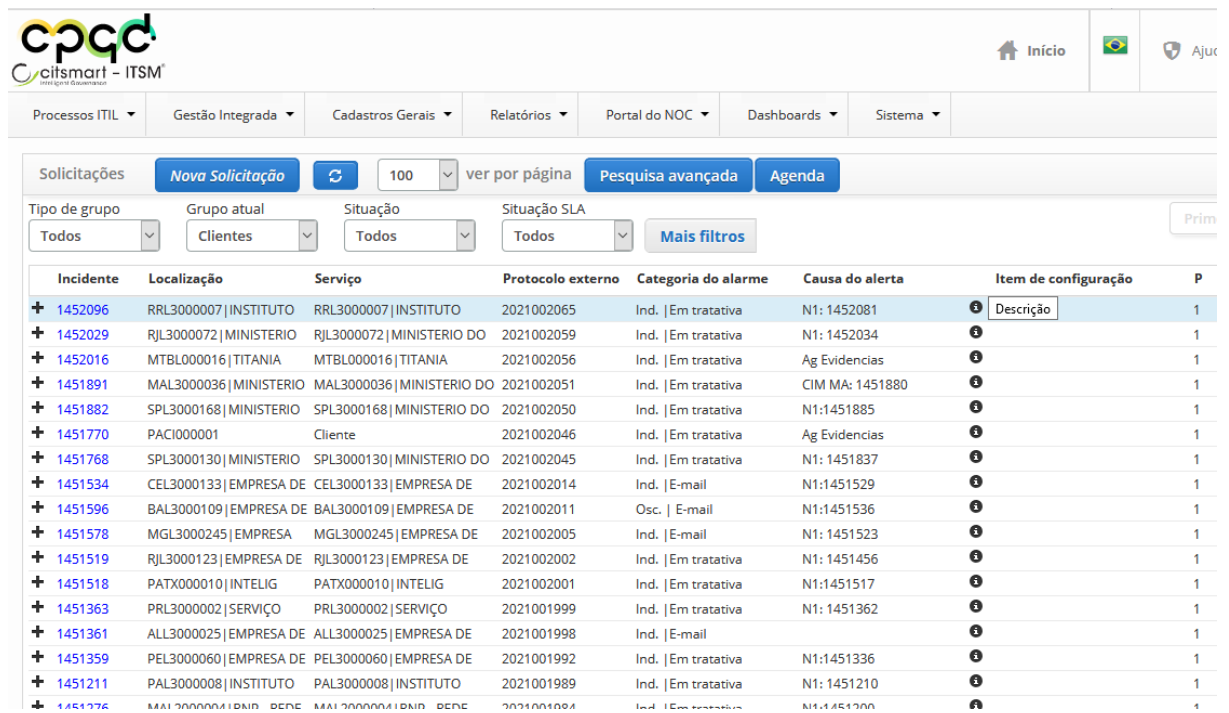
### 2.1. Dados do OSS – Gestão da Planta/*Query Builder*

Os dados coletados através do sistema OSS – Gestão da Planta/*Query Builder* foram coletados no dia 16 de dezembro de 2020. Os dados são referentes

aos bilhetes abertos no ITSM referentes a protocolos de reparo gerados pelo CRM e a avaliação de qualidade dada a cada bilhete pelo analista do CIGR que o tratou.

A figura 3 exemplifica a fila de incidentes no ITSM. O sistema Gestão da Planta/*Query Builder* permite realizar consultas personalizadas na base de dados do ITSM e exportá-las, conforme evidenciado na figura 4.

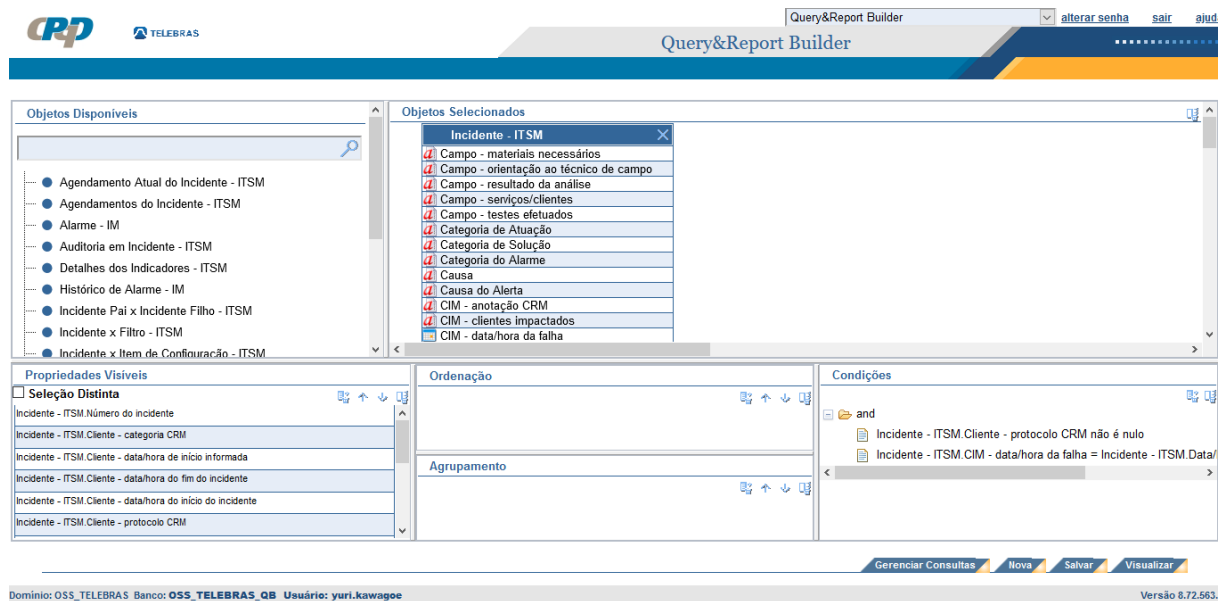
**Figura 3 – Fila de incidentes de clientes no ITSM.**



Incidente	Localização	Serviço	Protocolo externo	Categoria do alarme	Causa do alerta	Item de configuração	P
+ 1452096	RRL3000007 INSTITUTO	RRL3000007 INSTITUTO	2021002065	Ind.   Em tratativa	N1: 1452081	❶ Descrição	1
+ 1452029	RJL3000072 MINISTERIO	RJL3000072 MINISTERIO DO	2021002059	Ind.   Em tratativa	N1: 1452034	❶	1
+ 1452016	MTBL000016 TITANIA	MTBL000016 TITANIA	2021002056	Ind.   Em tratativa	Ag Evidencias	❶	1
+ 1451891	MAL3000036 MINISTERIO	MAL3000036 MINISTERIO DO	2021002051	Ind.   Em tratativa	CIM MA: 1451880	❶	1
+ 1451882	SPL3000168 MINISTERIO	SPL3000168 MINISTERIO DO	2021002050	Ind.   Em tratativa	N1:1451885	❶	1
+ 1451770	PACI000001	Cliente	2021002046	Ind.   Em tratativa	Ag Evidencias	❶	1
+ 1451768	SPL3000130 MINISTERIO	SPL3000130 MINISTERIO DO	2021002045	Ind.   Em tratativa	N1: 1451837	❶	1
+ 1451534	CEL3000133 EMPRESA DE	CEL3000133 EMPRESA DE	2021002014	Ind.   E-mail	N1:1451529	❶	1
+ 1451596	BAL3000109 EMPRESA DE	BAL3000109 EMPRESA DE	2021002011	Osc.   E-mail	N1:1451536	❶	1
+ 1451578	MGL3000245 EMPRESA	MGL3000245 EMPRESA DE	2021002005	Ind.   E-mail	N1: 1451523	❶	1
+ 1451519	RJL3000123 EMPRESA DE	RJL3000123 EMPRESA DE	2021002002	Ind.   Em tratativa	N1: 1451456	❶	1
+ 1451518	PATX000010 INTELIG	PATX000010 INTELIG	2021002001	Ind.   Em tratativa	N1:1451517	❶	1
+ 1451363	PRL3000002 SERVIÇO	PRL3000002 SERVIÇO	2021001999	Ind.   Em tratativa	N1: 1451362	❶	1
+ 1451361	ALL3000025 EMPRESA DE	ALL3000025 EMPRESA DE	2021001998	Ind.   E-mail		❶	1
+ 1451359	PEL3000060 EMPRESA DE	PEL3000060 EMPRESA DE	2021001992	Ind.   Em tratativa	N1:1451336	❶	1
+ 1451211	PAL3000008 INSTITUTO	PAL3000008 INSTITUTO	2021001989	Ind.   Em tratativa	N1: 1451210	❶	1
+ 1451276	MAL3000001 INSTITUTO	MAL3000001 INSTITUTO	2021001984	Ind.   Em tratativa	N1:1451200	❶	1

Fonte: Autor (OSS – ITSM)

**Figura 4 - Consulta e exportação de dados de incidentes no Query Builder.**



Fonte: Autor (OSS – Gestão da Planta/Query Builder)

A base de dados foi exportada no formato *Microsoft Excel* e salva com o nome de arquivo "Incidentes\_1sem", possuindo os seguintes atributos.

**Tabela 1 - Informações de "Incidentes\_1sem".**

Nome da coluna/campo	Descrição	Tipo
Incidente	Número do Incidente.	Numérico (6).
Categoria	Categoria do Incidente.	Texto.
Inicio_Informado	Data e hora de início do incidente informado pelos analistas.	Data e Hora.
Fim_Incidente	Data e hora de encerramento do incidente no sistema.	Data e Hora.
Inicio_Incidente	Data e hora de abertura do incidente no sistema.	Data e Hora.
Dia_Semana	Dia da semana da abertura do incidente no sistema.	Texto.

Hora_Inicio	Hora de início do incidente no sistema.	Numérico.
Protocolo	Protocolo gerado no CRM.	Numérico (10).
Circuito	Designação do circuito de cliente.	Alfanumérico, Normalizado (10). Exemplos: AML3000034, SPL3000181.
Cliente	Nome do cliente.	Texto, Normalizado. Exemplos: MINISTERIO DO TRABALHO – MTB, EMPRESA DE TECNOLOGIA E INFORMACOES DA PREVIDENCIA – DATAPREV.
Causa	Causa do incidente.	Texto, Normalizado. Exemplos: Energia, Falha de Configuração.
Causador	Causador do incidente.	Texto, Normalizado. Exemplos: Cliente, Telebras.
Solução	Solução aplicada para resolução do incidente.	Texto.
Prazo_SLA	Prazo para resolução do bilhete a partir da hora de início do incidente no sistema	Hora.
Tempo_Indisponibilidade	Tempo total de resolução do bilhete. Contabilizado a partir da subtração da data e hora de encerramento do incidente da data e hora de início do incidente no sistema.	Hora.

De maneira análoga, foram exportados os dados de avaliação do CIGR com relação a qualidade do serviço do fornecedor percebida pelos analistas durante o tempo de vida de cada bilhete da base de incidentes no mesmo período de 6 meses.

A base de dados foi exportada no formato *Microsoft Excel* e salva com o nome de arquivo "Avaliacao\_Incidentes", possuindo os seguintes atributos.

**Tabela 2 - Informações de "Avaliacao\_Incidentes".**

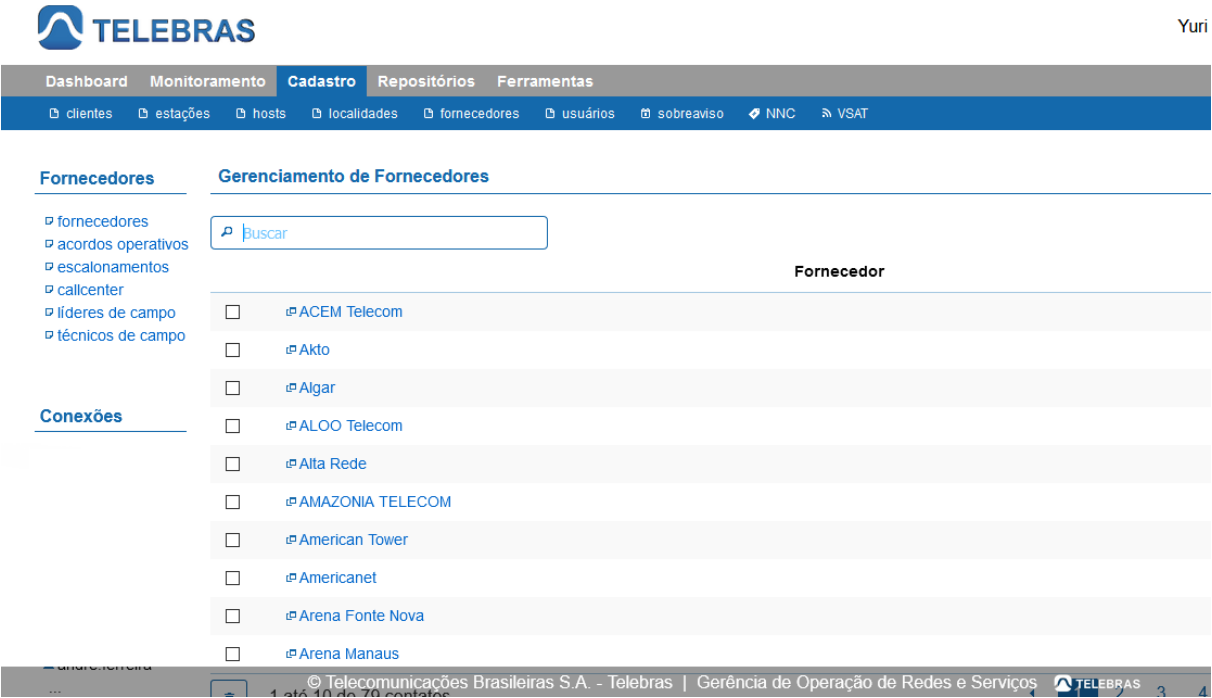
Nome da coluna/campo	Descrição	Tipo
Aval_CIGR	Avaliação positiva ou negativa fornecida pelos analistas com base na percepção de qualidade do serviço prestado pelos fornecedores de EILD.	Texto, Normalizado. Sim ou Não.

## **2.2. Dados do Report**

Os dados coletados através do sistema *Report* foram coletados no dia 18 de dezembro de 2020. Os dados são referentes ao cadastro de fornecedores de EILD.

A figura 5 exemplifica a tela do *Report* contendo a lista de fornecedores que pode ser exportada para o formato *Microsoft Excel*.

Figura 5 - Cadastro de Fornecedores EILD.



Fonte: Autor (Report)

A base de dados foi exportada no formato *Microsoft Excel* e salva com o nome de arquivo “Cadastro\_EILD”, possuindo os seguintes atributos.

Tabela 3 - Informações de "Cadastro\_EILD".

Nome da coluna/campo	Descrição	Tipo
Circuito	Designação do circuito de cliente.	Alfanumérico, Normalizado (10). Exemplos: AML3000034, SPL3000181.
Estado	Estado do Brasil no qual o circuito foi instalado.	Texto, Normalizado (2). Exemplos: DF, BH.
Produto	Tipo do produto /tecnologia do circuito.	Alfanumérico, Normalizado (2). Exemplos: L2, L3.
Acesso	Tipo de rede de acesso que atende a designação	Texto. Exemplos: Eild Satellite,

	de circuito.	Eild F-Optica.
Fornecedor	Nome do fornecedor EILD que atende a designação de circuito.	Texto. Exemplos: Oi, VIVO.

### 2.3. Dados da Confederação Nacional do Transporte

Os dados coletados da Confederação Nacional do Transporte são dados da página 161 do relatório de pesquisa CNT de rodovias 2019, desenvolvido pelo CNT/SEST SENAT. A tabela apresentada no relatório classifica o estado geral das rodovias Brasileiras por região e UF em: Ótimo, Bom, Regular, Ruim e Péssimo (Figura 6).

**Figura 6 - Classificação do Estado Geral em km - por Região e UF.**

REGIÃO E UF	ESTADO GERAL					TOTAL
	ÓTIMO	BOM	REGULAR	RUIM	PÉSSIMO	
<b>Brasil</b>	12.951	31.714	37.628	19.039	7.531	108.863
<b>Norte</b>	227	2.904	5.754	2.974	1.567	13.426
Rorônia	20	627	1.012	193	45	1.897
Acre	-	10	593	307	431	1.341
Amazonas	-	-	476	199	371	1.046
Roraima	70	617	388	44	1	1.120
Pará	82	698	1.473	1.399	314	3.966
Amapá	9	30	421	80	10	550
Tocantins	46	922	1.391	752	395	3.506
<b>Nordeste</b>	2.454	9.306	9.354	4.542	3.199	28.855
Maranhão	68	1.286	1.601	694	984	4.633
Piauí	308	1.056	1.085	672	302	3.423
Ceará	95	927	1.545	848	193	3.608
Rio Grande do Norte	84	561	732	224	275	1.876
Paraíba	230	664	396	251	169	1.710
Pernambuco	212	1.492	609	348	510	3.171
Alagoas	431	250	104	3	-	788
Sergipe	85	195	144	42	185	651
Bahia	941	2.875	3.138	1.460	581	8.995
<b>Sudeste</b>	7.753	8.131	8.351	5.034	964	30.233
Minas Gerais	832	3.684	5.730	4.373	744	15.363
Espírito Santo	164	496	600	109	31	1.400
Rio de Janeiro	971	835	437	243	107	2.593
São Paulo	5.786	3.116	1.584	309	82	10.877
<b>Sul</b>	1.556	6.013	6.835	3.466	605	18.475
Paraná	798	1.979	2.140	1.214	200	6.331
Santa Catarina	409	778	1.187	739	157	3.270
Rio Grande do Sul	349	3.256	3.508	1.513	248	8.874
<b>Centro-Oeste</b>	961	5.360	7.334	3.023	1.196	17.874
Mato Grosso do Sul	184	1.929	1.773	453	82	4.421
Mato Grosso	437	1.308	2.759	548	422	5.474
Goiás	237	1.965	2.625	1.992	687	7.506
Distrito Federal	103	158	177	30	5	473

Fonte: Pesquisa CNT de Rodovias 2019 (página 161).

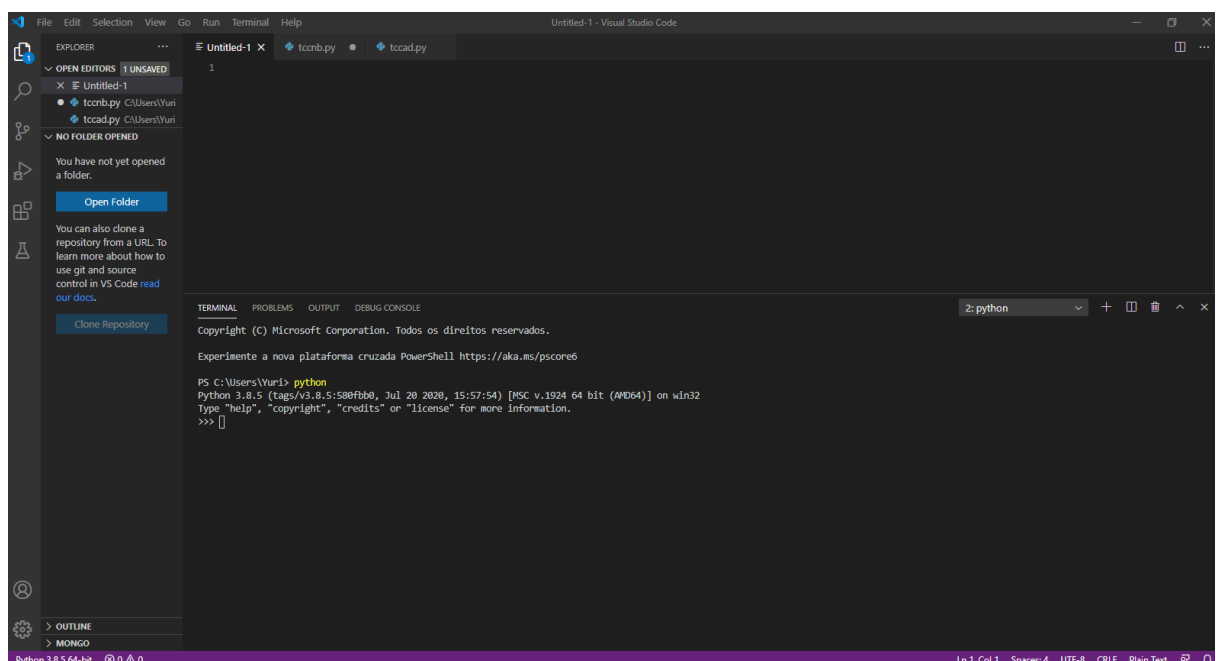
Os dados da tabela anterior foram extraídos do sítio <https://pesquisarodovias.cnt.org.br/downloads/ultimaversao/gerencial.pdf>. Utilizando a linguagem de programação Python, foi possível transformar a tabela no formato PDF (em inglês, *portable document format*) em um “*DataFrame*” da biblioteca “*Pandas*”, que consiste em uma estrutura tabular bidimensional, de tamanho variável, potencialmente heterogênea e com eixos rotulados, semelhante a uma planilha do *Microsoft Excel*. O processo de aquisição e tratamento dos dados será demonstrado na seção seguinte.

### 3. Processamento/Tratamento de Dados

#### 3.1. Ferramentas utilizadas

Como ferramenta para desenvolvimento dos scripts em *Python*, foi escolhido o *Microsoft Visual Studio Code* na versão 1.52.1 rodando o *Python* na versão 3.8.5 (Figura 7), disponível em <https://code.visualstudio.com/>.

Figura 7 - Screenshot do Visual Studio Code.



Fonte: Autor



### 3.2. Processamento e tratamento das bases de dados do OSS – Gestão da Planta/*Query Builder*

Primeiramente foi necessário importar a biblioteca “*Pandas*” (Figura 8), que é uma biblioteca de código aberto de licença BSD (em inglês, Berkeley Software Distribution) que fornece estruturas de dados de alto desempenho e fáceis de usar, além de ferramentas de análise de dados para a linguagem de programação Python.

Figura 8 - Importação da biblioteca pandas.

```
5  #Importação das bibliotecas
6  import pandas as pd
```

Em seguida, utilizou-se a função “*pandas.read\_excel*” para ler os dados das planilhas em *Excel* (“*Incidentes\_1sem.xlsx*” e “*Avaliacao\_Incidentes.xlsx*”) e transformá-los em duas variáveis (“*incidentes*” e “*avaliacao*”) do tipo “*DataFrame*” (Figura 9).

Figura 9 - Criação das variáveis “incidentes” e “avaliacao”.

```
20  #Bases obtidas através do sistema "GP-QueryBuilder"
21  incidentes = pd.read_excel('Incidentes_1sem.xlsx')
22  avaliacao = pd.read_excel('Avaliacao_Incidentes.xlsx')
```

Utilizando a função “*pandas.DataFrame.head*”, os métodos “*pandas.DataFrame.shape*”, “*pandas.DataFrame.isna*” junto com o método “*pandas.DataFrame.sum*” e “*pandas.DataFrame.duplicated*” nas variáveis, temos como resultados: os cinco primeiros registros de cada “*DataFrame*”, a quantidade de linhas e colunas de cada *DataFrame*, a quantidade de elementos vazios em cada coluna e a quantidade de elementos duplicados na coluna indicada dentro do parâmetro “*subset*” (*subset*=[‘Incidente’]) (Figuras 10, 11 e 12). É importante ressaltar que a coluna “Incidente” foi selecionada devido a necessidade de não haja incidentes repetidos para que as bases possam ser agrupadas de forma correta.

Figura 10 – Verificação e validação dos dados importados.

```

24 print(incidentes.head())
25 print()
26 print(incidentes.shape)
27 print()
28 print(incidentes.isna().sum())
29 print()
30 print(incidentes.duplicated(subset=['Incidente']).sum())
31 print()
32 print(avaliacao.head())
33 print()
34 print(avaliacao.shape)
35 print()
36 print(avaliacao.isna().sum())
37 print()
38 print(avaliacao.duplicated(subset=['Incidente']).sum())
39 print()

```

Figura 11 - Resultado da validação e verificação de "incidentes".

	Incidente	Categoria	Inicio_Informado	Fim_Incidente	...	Causador	Solução	Prazo_SLA	Tempo_Indisponibilidade
0	873748	Indisponibilidade	01/01/2020 19:00:00	01/01/2020 19:56:00	...	Telebras	[Detalhamento da Solução] - Categoria de atuaç...	120:00	00:25:00
1	873727	Indisponibilidade	01/01/2020 18:15:00	01/01/2020 19:08:00	...	Telebras	Normalizado sem intervenção técnica	120:00	00:53:00
2	873760	Indisponibilidade	01/01/2020 17:02:00	03/01/2020 10:26:00	...	Telebras	Sobressalente trocado e comunicação restabelec...	120:00	1900-01-01 17:24:00
3	873617	Indisponibilidade	01/01/2020 13:59:00	01/01/2020 14:37:00	...	Telebras	[Detalhamento da Solução] - Ação realizada: Ci...	120:00	00:38:00
4	873572	Indisponibilidade	01/01/2020 10:41:00	01/01/2020 12:49:00	...	Cliente	[Encerramento] - Ação realizada: Falha elétric...	120:00	02:08:00

[5 rows x 15 columns]

(13563, 15)

```

Incidente          0
Categoria          0
Inicio_Informado   0
Fim_Incidente      0
Inicio_Incidente   0
Dia_Semana         0
Hora_Inicio        0
Protocolo          0
Circuito           0
Cliente            0
Causa              0
Causador           0
Solução            0
Prazo_SLA          0
Tempo_Indisponibilidade 0
dtype: int64
0

```

Figura 12 - Resultado da verificação e validação de "avaliacao".

```

Incidente Aval_CIGR
0      873748      Sim
1      873727      Sim
2      873700      Não
3      873617      Sim
4      873572      Sim

(13563, 2)

Incidente      0
Aval_CIGR      0
dtype: int64
0

```

Conforme visualizado nas figuras 10, 11 e 12, foi possível verificar que o “*DataFrame*” “incidentes” possui 13563 linhas e 15 colunas, não possui elementos vazios em nenhuma das colunas e não possui elementos repetidos na coluna “Incidente”, enquanto o “*DataFrame*” “avaliacao” possui 13563 linhas e 2 colunas, não possui elementos vazios em nenhuma das colunas e não possui elementos repetidos na coluna “Incidente”. Dessa forma, concluiu-se que as bases poderiam ser relacionadas corretamente.

### 3.3. Processamento e tratamento da base de dados do *Report*

De maneira análoga à importação das bases anteriores, utilizou-se a função “*pandas.read\_excel*” para ler os dados da planilha (“Cadastro\_EILD.xlsx”), conforme figura 13.

Figura 13 - Criação da variável “eild”.

```

33 #Base obtida através de download do sistema de cadastro "Report"
34 eild = pd.read_excel('Cadastro_EILD.xlsx')

```

Utilizando a função “*pandas.DataFrame.head*”, os métodos “*pandas.DataFrame.shape*”, “*pandas.DataFrame.isna*” junto com o método “*pandas.DataFrame.sum*” e “*pandas.DataFrame.duplicated*” nas variáveis, temos como resultado: os 5 primeiros registros do “*DataFrame*”, a quantidade de linhas e

colunas do “*DataFrame*”, a quantidade de elementos vazios em cada coluna e a quantidade de elementos duplicados na coluna indicada dentro do parâmetro “*subset*” (*subset*=[‘Circuito’]) (Figuras 14 e 15). É importante ressaltar que a coluna “Circuito” foi selecionada devido a necessidade de não haja circuitos repetidos para que as bases sejam combinadas de forma correta posteriormente.

Figura 14 - Verificação e validação dos dados importados.

```
44 print(eild.head())
45 print()
46 print(eild.shape)
47 print()
48 print(eild.isna().sum())
49 print()
50 print(eild.duplicated(subset=[ 'Circuito' ]).sum())
51 print()
```

Figura 15 - Resultado da verificação e validação de "eild".

```

      Circuito Estado Produto      Acesso      Fornecedor
0  ACL3000003     AC      L3  Eild F-Optica  Staff Computer
1  ACL3000005     AC      L3  Eild F-Optica  Staff Computer
2  ACL3000006     AC      L3  Eild F-Optica  Staff Computer
3  ACL3000008     AC      L3  Eild Satelite      Hughes
4  ACL3000010     AC      L3  Eild F-Optica      Oi

(3303, 5)

Circuito      0
Estado        0
Produto        0
Acesso         0
Fornecedor     0
dtype: int64

0
```

Conforme visualizado nas figuras 14 e 15, foi possível verificar que o “*DataFrame*” “eild” possui 3303 linhas e 5 colunas, não possui elementos vazios em nenhuma coluna e não possui elementos repetidos na coluna “circuito”.

### 3.4. Processamento e tratamento da base de dados do CNT

Primeiramente foi necessário importar a classe “*Path*” da biblioteca “*Pathlib*” e as bibliotecas “*Requests*” e “*Tabula-py*” (Figura 16). A primeira consiste em uma biblioteca que oferece classes que representam caminhos de sistema de arquivos com semântica apropriada para diferentes sistemas. A segunda permite enviar solicitações *HTTP/1.1* (em inglês, *Hypertext Transfer Protocol*) de maneira facilitada. A terceira permite ler tabelas de arquivos PDF e convertê-las em “*DataFrames*” do “*Pandas*” e, também permite converter um arquivo PDF em arquivos CSV, TSV e JSON.

Figura 16 - Importação das bibliotecas “*pathlib*”, “*requests*” e “*tabula*”.

```
7 from pathlib import Path
8 import requests
9 import tabula
```

Em seguida, criou-se uma variável “arquivo” utilizando o nome do arquivo em PDF a ser salvo na pasta onde o *Python* está instalado, uma variável “url”, que recebeu o sítio da internet onde o arquivo PDF se encontra, uma variável “resposta”, utilizando a função “*requests.get*”, e em seguida utilizou-se a função “*write\_bytes*” na variável “arquivo” passando como parâmetro o conteúdo da variável resposta “*reposta.content*” (Figura 17).

Figura 17 - Baixando e salvando arquivo PDF da CNT.

```
53 #Baixando relatório gerencial de rodovias da CNT utilizando as bibliotecas pathlib e requests
54
55 arquivo = Path('relatorio.pdf')
56 url = 'https://pesquisarodovias.cnt.org.br/downloads/ultimaversao/gerencial.pdf'
57 resposta = requests.get(url)
58 arquivo.write_bytes(resposta.content)
```

O próximo passo foi converter a página de interesse ao projeto (página 161) do relatório da CNT para o formato CSV utilizando a função “*convert\_into*” do *Tabula*. Como parâmetros foram passados o nome do arquivo de origem em PDF, o nome do arquivo de destino em CSV, o número da página em que se encontra a

tabela, o parâmetro “*lattice*”, definido como Verdadeiro (em inglês, *True*), que separa cada célula da tabela no PDF de modo semelhante ao Excel e o parâmetro “*java\_options*”, que permite definir a codificação do arquivo baixado (Figura 18).

Figura 18 - Convertendo o arquivo PDF para CSV.

```
60 #Convertendo a página de interesse (pg 161) em formato .csv
61
62 tabula.convert_into("relatorio.pdf", "rod.csv", output_format="csv", pages=161, lattice=True, java_options="-Dfile.encoding=UTF8")
63
```

O passo seguinte consistiu em criar uma variável “*df*” e importar o arquivo CSV, transformando-o em “*DataFrame*” utilizando a biblioteca “*Pandas*” já importada anteriormente. Foi utilizado como parâmetro “*thousands*”, que permite que o “*DataFrame*” reconheça o caractere “.” Como separador de milhar (Figura 19).

Utilizando a função “*print(df)*” e o método “*pandas.DataFrame.dtypes*” foi possível verificar os tipos de dados de cada coluna no resultado (Figura 20).

Figura 19 - Criação da variável “*df*” e verificação dos tipos de dados.

```
66 df = pd.read_csv('rod.csv', thousands=',')
67 print(df)
68 print()
69 print(df.dtypes)
```

Figura 20 - Resultado da verificação dos dados.

	Unnamed: 0	ÓTIMO	BOM	REGULAR	RUIM	PÉSSIMO	Unnamed: 6
0	Brasil	12.951	31.714	37628	19039	7.531	108863
1	Norte	227	2.904	5754	2974	1.567	13426
2	Rondônia	20	627	1012	193	45	1897
3	Acre	-	10	593	307	431	1341
4	Amazonas	-	-	476	199	371	1046
5	Roraima	70	617	388	44	1	1120
6	Pará	82	698	1473	1399	314	3966
7	Amapá	9	30	421	80	10	550
8	Tocantins	46	922	1391	752	395	3506
9	Nordeste	2.454	9.306	9354	4542	3.199	28855
10	Maranhão	68	1.286	1601	694	984	4633
11	Piauí	308	1.056	1085	672	302	3423
12	Ceará	95	927	1545	848	193	3608
13	Rio Grande do Norte	84	561	732	224	275	1876
14	Paraíba	230	664	396	251	169	1710
15	Pernambuco	212	1.492	609	348	510	3171
16	Alagoas	431	250	104	3	-	788
17	Sergipe	85	195	144	42	185	651
18	Bahia	941	2.875	3138	1460	581	8995
19	Sudeste	7.753	8.131	8351	5034	964	30233
20	Minas Gerais	832	3.684	5730	4373	744	15363
21	Espírito Santo	164	496	600	109	31	1400
22	Rio de Janeiro	971	835	437	243	107	2593
23	São Paulo	5.786	3.116	1584	309	82	10877
24	Sul	1.556	6.013	6835	3466	605	18475
25	Paraná	798	1.979	2140	1214	200	6331
26	Santa Catarina	409	778	1187	739	157	3270
27	Rio Grande do Sul	349	3.256	3508	1513	248	8874
28	Centro-Oeste	961	5.360	7334	3023	1.196	17874
29	Mato Grosso do Sul	184	1.929	1773	453	82	4421
30	Mato Grosso	437	1.308	2759	548	422	5474
31	Goiás	237	1.965	2625	1992	687	7506
32	Distrito Federal	103	158	177	30	5	473
Unnamed: 0		object					
ÓTIMO		object					
BOM		object					
REGULAR		int64					
RUIM		int64					
PÉSSIMO		object					
Unnamed: 6		int64					
dtype:		object					

O próximo passo consistiu no tratamento do “*DataFrame*” para adequação ao projeto. O processo de tratamento seguiu os passos seguintes:

- 1) Substituir o caractere “-” por “0”. Método utilizado: `“pandas.DataFrame.replace”`;
- 2) Excluir as linhas (0, 1, 9, 19, 24, 28). Essas linhas correspondem as regiões do Brasil, dados que não interessam para o projeto. Método utilizado: `“pandas.DataFrame.drop”`.
- 3) Excluir as colunas (0 e 6). Essas colunas correspondem aos estados do Brasil e a somatória do número de estradas avaliadas, que não interessam para o projeto. Método utilizado: `“pandas.DataFrame.drop”`.
- 4) Criar uma variável “estado” com todos as siglas dos estados Brasileiros para estar de acordo com os dados do “*DataFrame*” “eild”, uma vez que as bases serão combinadas posteriormente.
- 5) Transformar os elementos decimais em inteiros e substituir os novos “NaN” criados por “0”, uma vez que o Python interpreta o caractere “.” Como separador decimal. Dessa forma, removeu-se o caractere “.” nas colunas “Ótimo”, “Bom” e “Péssimo”, que são do tipo “*Object*”. Métodos utilizados: `“pandas.Series.str.replace”` e `“pandas.DataFrame.fillna”`.
- 6) Transformar os tipos dos dados do “*DataFrame*” em dados numéricos. Método utilizado: `“pandas.DataFrame.to_numeric”`. Esse passo permitiu comparar os valores numéricos em cada coluna.
- 7) Criar a coluna “Estado”, utilizando a variável criada no passo 4.
- 8) Criar a coluna “Class\_Rodovias”, classificando cada estado de acordo com o maior número de rodovias classificadas como “Ótimo”, “Bom”, “Regular”, “Ruim” ou “Péssimo”. Exemplo: O estado de Rondônia tem mais estradas classificadas como Regulares, logo o estado recebeu a classificação “REGULAR” na coluna “Class\_Rodovias”. Método utilizado: `“pandas.DataFrame.idxmax”`.

As figuras 21 e 22 ilustram o código utilizado para o tratamento dos dados importados do arquivo PDF e o resultado, respectivamente.



Figura 21 - Tratamento dos dados da CNT.

```

71 df = df.replace(to_replace='-', value=0)
72 df=df.drop([0,1,9,19,24,28])
73 df=df.drop(df.columns[[0,6]], axis=1)
74 estado=[['RO','AC','AM','RR','PA','AP','TO','MA','PI','CE','RN','PB','PE','AL',
75 'SE','BA','MG','ES','RJ','SP','PR','SC','RS','MS','MT','GO','DF']]
76 df['ÓTIMO'] = df['ÓTIMO'].str.replace('\.', '')
77 df['ÓTIMO'] = df['ÓTIMO'].fillna(0)
78 df['BOM'] = df['BOM'].str.replace('\.', '')
79 df['BOM'] = df['BOM'].fillna(0)
80 df['PÉSSIMO'] = df['PÉSSIMO'].str.replace('\.', '')
81 df['PÉSSIMO'] = df['PÉSSIMO'].fillna(0)
82 df=df.apply(pd.to_numeric)
83 df['Estado'] = estado
84 df['Class_Rodovias'] = df[['ÓTIMO','BOM','REGULAR','RUIM','PÉSSIMO']].idxmax(axis=1)
85 print(df)
86 print(df.dtypes)
87 print()

```

Figura 22 – Resultado do tratamento da base de dados do CNT.

	ÓTIMO	BOM	REGULAR	RUIM	PÉSSIMO	Estado	Class_Rodovias
2	20	627	1012	193	45	RO	REGULAR
3	0	10	593	307	431	AC	REGULAR
4	0	0	476	199	371	AM	REGULAR
5	70	617	388	44	1	RR	BOM
6	82	698	1473	1399	314	PA	REGULAR
7	9	30	421	80	10	AP	REGULAR
8	46	922	1391	752	395	TO	REGULAR
10	68	1286	1601	694	984	MA	REGULAR
11	308	1056	1085	672	302	PI	REGULAR
12	95	927	1545	848	193	CE	REGULAR
13	84	561	732	224	275	RN	REGULAR
14	230	664	396	251	169	PB	BOM
15	212	1492	609	348	510	PE	BOM
16	431	250	104	3	0	AL	ÓTIMO
17	85	195	144	42	185	SE	BOM
18	941	2875	3138	1460	581	BA	REGULAR
20	832	3684	5730	4373	744	MG	REGULAR
21	164	496	600	109	31	ES	REGULAR
22	971	835	437	243	107	RJ	ÓTIMO
23	5786	3116	1584	309	82	SP	ÓTIMO
25	798	1979	2140	1214	200	PR	REGULAR
26	409	778	1187	739	157	SC	REGULAR
27	349	3256	3508	1513	248	RS	REGULAR
29	184	1929	1773	453	82	MS	BOM
30	437	1308	2759	548	422	MT	REGULAR
31	237	1965	2625	1992	687	GO	REGULAR
32	103	158	177	30	5	DF	REGULAR
ÓTIMO			int64				
BOM			int64				
REGULAR			int64				
RUIM			int64				
PÉSSIMO			int64				
Estado			object				
Class_Rodovias			object				

Com os resultados apresentados acima foi possível iniciar a fase de análise e exploração dos dados, que será demonstrada a seguir.

#### 4. Análise e Exploração dos Dados

A primeira etapa consistiu em combinar as bases de dados “eild” e “df” na base de incidentes para gerar a base final que será utilizada nos modelos. Para isso, utilizou-se o método “*pandas.DataFrame.merge*”, primeiramente combinando o “*DataFrame*” “avaliacao” em “incidentes” à esquerda (“*how=left*”), a partir da coluna “Incidente” (“*on=incidente*”) presente em ambas as bases. Depois, combinando o “*DataFrame*” “eild” em “incidentes” à esquerda, a partir da coluna “Circuito” presente em ambas as bases. Por fim, o “*DataFrame*” “df” em “incidentes” à esquerda, a partir da coluna “Estado” presente em ambas as bases.

A segunda etapa consistiu em verificar o resultado da combinação e validar a existência de elementos sem correspondências após a aplicação da função “*pandas.DataFrame.head*”, dos métodos “*pandas.DataFrame.shape*” e “*pandas.DataFrame.isna*” junto com o “*pandas.DataFrame.sum*” (Figuras 23 e 24).

Figura 23 - Combinando as bases e validando.

```

89  #Combinando as bases de dados para utilização nos modelos.
90
91  incidentes=pd.merge(incidentes, avaliacao, on='Incidente', how='left')
92
93  incidentes=pd.merge(incidentes, eild, on='Circuito', how='left')
94
95  incidentes=pd.merge(incidentes, df, on='Estado', how='left')
96
97  print(incidentes.head())
98  print()
99  print(incidentes.shape)
100 print()
101 print(incidentes.isna().sum())
102 print()

```

Figura 24 - Resultado das bases combinadas.

	Incidente	Categoria	Inicio Informado	Fim_Incidente	Inicio Incidente	Dia_Semana	...	ÓTIMO	BOM	REGULAR	RUIM	PÉSSIMO	Class_Rodovias
0	873748	Indisponibilidade	01/01/2020 19:09:00	01/01/2020 19:56:00	01/01/2020 19:31:00	qua	...	0.0	0.0	476.0	199.0	371.0	REGULAR
1	873727	Indisponibilidade	01/01/2020 18:15:00	01/01/2020 19:08:00	01/01/2020 18:15:00	qua	...	832.0	3684.0	5730.0	4373.0	744.0	REGULAR
2	873700	Indisponibilidade	01/01/2020 17:02:00	03/01/2020 10:26:00	01/01/2020 17:02:00	qua	...	237.0	1965.0	2625.0	1992.0	687.0	REGULAR
3	873617	Indisponibilidade	01/01/2020 13:59:00	01/01/2020 14:37:00	01/01/2020 13:59:00	qua	...	164.0	496.0	600.0	109.0	31.0	REGULAR
4	873572	Indisponibilidade	01/01/2020 10:41:00	01/01/2020 12:49:00	01/01/2020 10:41:00	qua	...	70.0	617.0	388.0	44.0	1.0	BOM

[5 rows x 26 columns]

(13563, 26)

Incidente 0  
 Categoria 0  
 Inicio\_Informado 0  
 Fim\_Incidente 0  
 Inicio\_Incidente 0  
 Dia\_Semana 0  
 Hora\_Inicio 0  
 Protocolo 0  
 Circuito 0  
 Cliente 0  
 Causa 0  
 Causador 0  
 Solução 0  
 Prazo\_SLA 0  
 Tempo\_Indisponibilidade 0  
 Aval\_CIGR 0  
 Estado 2647  
 Produto 2647  
 Acesso 2647  
 Fornecedor 2647  
 ÓTIMO 2647  
 BOM 2647  
 REGULAR 2647  
 RUIM 2647  
 PÉSSIMO 2647  
 Class\_Rodovias 2647  
 dtype: int64

Com o resultado apresentado na figura 24, foi possível perceber que existem 13563 linhas e 26 colunas no “*DataFrame*”, sendo que as colunas combinadas das bases “eild” e “df” possuem 2647 elementos “NaN”, o que indica que os incidentes ocorridos em cada um desses 2647 circuitos não possuem um fornecedor de EILD cadastrado, sugerindo que esses incidentes são atendidos com rede própria da empresa. Por esse motivo, optou-se por removê-los do “*DataFrame*” porque não fazem parte do escopo do projeto, que tem como objetivo analisar a qualidade do serviço prestado apenas pelos fornecedores de EILD. Para a remoção, foi utilizado o método “*Pandas.DataFrame.dropna*”.

Além disso, foram removidas todas as colunas nas quais os atributos não interessam aos objetivos do projeto utilizando o método “*Pandas.DataFrame.drop*”, passando como parâmetro os nomes das colunas a serem removidas. Em seguida, as colunas do “*DataFrame*” “incidentes” foram reordenadas de modo que a classe (última coluna) seja a avaliação do CIGR com relação a qualidade (“Aval\_CIGR”). Utilizando novamente a função “*pandas.DataFrame.head*” e o método “*pandas.DataFrame.shape*” visualizou-se o resultado final da base de dados. (Figuras 25 e 26).

Figura 25 - Removendo “NaNs” e mantendo somente os atributos desejados.

```

104 incidentes=incidentes.dropna()
105
106 incidentes=incidentes.drop(columns=['Incidente','Inicio_Informado','Fim_Incidente','Inicio_Incidente','Protocolo','Causador',
107 'Solução','Prazo_SLA','Tempo_Indisponibilidade', 'ÓTIMO','BOM','REGULAR','RUIM', 'PÉSSIMO'])
108
109 incidentes=incidentes[['Categoria', 'Circuito', 'Cliente', 'Causa', 'Dia_Semana', 'Hora_Inicio', 'Estado', 'Produto', 'Acesso',
110 'Fornecedor', 'Class_Rodovias', 'Aval_CIGR']]
111
112 print(incidentes.head())
113 print()
114 print(incidentes.shape)
115

```

Figura 26 - Resultado final da base de dados.

	Categoria	Circuito	Cliente	Causa	Dia_Semana	...	Produto	Acesso	Fornecedor	Class_Rodovias	Aval_CIGR
0	Indisponibilidade	AML3000034	MINISTERIO DO TRABALHO - MTB	Outros	qua	...	L3	E1d Satellite	Hughes	REGULAR	Sim
1	Indisponibilidade	HGL3000347	AGÊNCIA NACIONAL DE TRANSPORTES TERRESTRES	Outros	qua	...	L3	E1d F-Optica	American Tower	REGULAR	Sim
2	Indisponibilidade	GOL3000010	AGÊNCIA NACIONAL DE TRANSPORTES TERRESTRES	Falha de Equipamento	qua	...	L3	E1d F-Optica	American Tower	REGULAR	Não
3	Indisponibilidade	ESL3000007	AGÊNCIA NACIONAL DE TRANSPORTES TERRESTRES	Falha de Equipamento	qua	...	L3	E1d F-Optica	GTI Telecom	REGULAR	Sim
4	Indisponibilidade	RRL3000019	MINISTERIO DO TRABALHO - MTB	Energia	qua	...	L3	E1d Satellite	Hughes	BOM	Sim

[5 rows x 12 columns]

(10916, 12)

O resultado final é uma base com 10916 linhas e 12 colunas, contendo os 11 atributos previsores selecionados (“Categoria”, “Circuito”, “Cliente”, “Causa”, “Dia\_Semana”, “Hora\_Inicio”, “Estado”, “Produto”, “Acesso”, “Fornecedor”) e a classe (“Aval\_CIGR”). Tais atributos foram selecionados devido ao fato de influenciarem diretamente e qualificarem o serviço prestado por um determinado fornecedor conforme explicado anteriormente.

O próximo passo consistiu em avaliar visualmente o balanceamento das classes (“Sim” e “Não”) na coluna “Aval\_CIGR”. Para isso foram importadas as bibliotecas “*matplotlib*”, que é uma biblioteca para criação de gráficos e visualizações de dados em geral, e “*seaborn*”, que é uma biblioteca de visualização de dados Python baseada em “*matplotlib.pyplot*”, fornecendo uma interface de alto nível para desenhar gráficos estatísticos atraentes e informativos (Figura 27). Além disso, foi utilizada a função “*Pandas.DataFrame.value\_counts*” para contabilizar os totais em cada classe (“Sim” e “Não”), a função “*seaborn.countplot*”, que mostra a distribuição de observações em cada classe utilizando um gráfico de barras e a função “*matplotlib.pyplot.show*” para visualizar o gráfico (Figura 28). Os resultados podem ser visualizados nas figuras 29 e 30.

Figura 27 - Importação das bibliotecas "seaborn" e "matplotlib.pyplot".

```
15 import seaborn as sb
16 import matplotlib.pyplot as plt
```

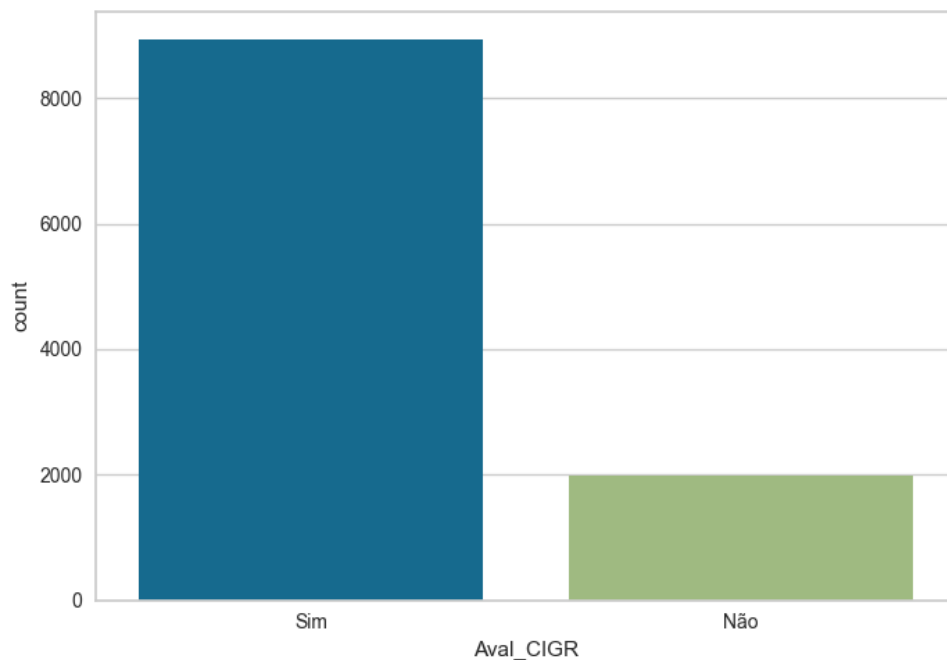
Figura 28 - Visualizando a distribuição das classes de "Aval\_CIGR".

```
121 #Visualizando a distribuição das classes de "Aval_CIGR"
122 print(incidentes['Aval_CIGR'].value_counts())
123 print()
124 ax = sb.countplot(x="Aval_CIGR", data=incidentes)
125 plt.show()
```

Figura 29 - Quantidade de valores "Sim" e "Não".

```
Sim      8930
Não      1986
Name: Aval_CIGR, dtype: int64
```

Figura 30 - Visualização gráfica da distribuição das classes



Com os resultados acima foi possível perceber que as classes estão desbalanceadas. A classe de maior interesse para o projeto (“Não”) é minoritária em uma proporção de 1:4,49. Nesse caso, a criação dos modelos de *machine learning* requerem um certo cuidado para que os resultados não sejam enviesados, uma vez que a probabilidade de ocorrência de um “Sim” será maior que a de ocorrência de um “Não” em termos de classificação.

## 5. Criação de Modelos de *Machine Learning*

Os modelos de *machine learning* deste trabalho foram desenvolvidos utilizando a biblioteca “*scikit-learn*”, que consiste em uma biblioteca robusta e com muitos recursos de *machine learning* para *Python*. Foram escolhidos dois modelos de aprendizado supervisionado da biblioteca: “*Gaussian Naive Bayes*” e “*Decision Tree Classifier*”. A implementação dos modelos será detalhada a seguir.

### 5.1. Modelo 1: “*Gaussian Naive Bayes*”

O “*Gaussian Naive Bayes*” é um algoritmo de classificação estatística baseado no Teorema de *Bayes*, na qual o classificador assume que o efeito de um determinado atributo em uma classe é independente dos outros atributos.

Primeiramente foi necessário importar as funções de alguns módulos da biblioteca “*scikit-learn*” que serão utilizados na criação do modelo 1. Além disso, foi importada a biblioteca “*yellowbrick*” para fins de visualização gráfica da matriz de confusão, que mostra cada combinação das classes verdadeiras e previstas para um conjunto de dados de teste em mapa de calor (Figura 31). A matriz criada utilizando a “*yellowbrick*” será apresentada no tópico Apresentação dos Resultados.

Figura 31 - Importação das funções e classes dos módulos das bibliotecas “*scikit-learn*” e “*yellowbrick*”.

```
10 from sklearn.model_selection import train_test_split
11 from sklearn.naive_bayes import GaussianNB
12 from sklearn.preprocessing import LabelEncoder
13 from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_score
14 from yellowbrick.classifier import ConfusionMatrix
```

Primeira importação: Função “*train\_test\_split*” do módulo “*sklearn.model\_selection*”, que divide vetores ou matrizes em subconjuntos de treino e teste.

Segunda importação: Classe “*GaussianNB*” do módulo “*sklearn.naive\_bayes*” para aplicação do algoritmo “*Gaussian Naive Bayes*”.

Terceira importação: Classe “*LabelEncoder*” do módulo “*sklearn.preprocessing*”, que permite codificar atributos categóricos em numéricos.

Quarta importação: Funções “*confusion\_matrix*”, “*classification\_report*”, “*roc\_curve*” e “*roc\_auc\_score*” do módulo “*sklearn.metrics*”, que permitem computar a matriz de confusão de uma classificação, exibir o relatório de métricas, plotar a curva ROC (em inglês, *Receiver Operating Characteristic*) e calcular o valor de AUC (em inglês, *Area Under the ROC Curve*).

Quinta importação: Classe “*ConfusionMatrix*” do módulo “*yellowbrick.classifier*”, que cria uma visualização de mapa de calor da função “*confusion\_matrix*”.

O próximo passo foi criar as variáveis “previsores” e “classe” a partir do “*DataFrame*” “*incidentes*” no formato de vetor. Para isso, utilizou-se o método “*pandas.DataFrame.iloc*” para todas as linhas e colunas de índices 0 a 10, no caso dos previsores, e índice 11 para todas as linhas e coluna de índice 11, no caso da classe (Figura 32).

Figura 32 - Criação das variáveis “previsores” e “classe”.

```
118   previsores = incidentes.iloc[:,0:11].values
119   classe = incidentes.iloc[:,11].values
```

O passo seguinte consistiu em criar uma variável “*labelencoder*” instanciando a classe “*LabelEncoder*” e, em seguida, criar um objeto para cada coluna categórica, utilizando o método “*fit\_transform*”, que aplica o codificador de rótulos (*label encoder*) e retorna rótulos codificados, ou seja, esse método permitiu transformar cada uma das colunas com atributos categóricos em colunas com atributos numéricos (Figura 33).

Figura 33 - Codificação dos atributos categóricos em numéricos.

```

121  labelencoder = LabelEncoder()
122
123  previsoires[:,0] = labelencoder.fit_transform(previsoires[:,0])
124  previsoires[:,1] = labelencoder.fit_transform(previsoires[:,1])
125  previsoires[:,2] = labelencoder.fit_transform(previsoires[:,2])
126  previsoires[:,3] = labelencoder.fit_transform(previsoires[:,3])
127  previsoires[:,4] = labelencoder.fit_transform(previsoires[:,4])
128  previsoires[:,6] = labelencoder.fit_transform(previsoires[:,6])
129  previsoires[:,7] = labelencoder.fit_transform(previsoires[:,7])
130  previsoires[:,8] = labelencoder.fit_transform(previsoires[:,8])
131  previsoires[:,9] = labelencoder.fit_transform(previsoires[:,9])
132  previsoires[:,10] = labelencoder.fit_transform(previsoires[:,10])
133

```

Em seguida, utilizou-se a função “*train\_test\_split*” para dividir a base de dados entre treinamento (variáveis “X\_treinamento” e “Y\_treinamento”) e teste (variáveis “X\_teste” e “Y\_teste”). Como parâmetros da função, foram utilizadas as variáveis “previsoires” e “classe” e definido um tamanho de 20% para teste e de 80% para treino, utilizando o parâmetro “*test\_size*” (0,2), considerando o tamanho da base de dados de, aproximadamente, 11000 linhas. Além disso, foi utilizado um número inteiro (1) no parâmetro “*random\_state*” para garantir a reprodução da mesma saída da função em cada vez que ela fosse executada (Figura 34).

Figura 34 - Divisão da base de dados entre teste e treinamento.

```

134  #Divisão da base de dados entre treinamento e teste (20% para teste e 80% para treinamento)
135  X_treinamento, X_teste, Y_treinamento, Y_teste = train_test_split(previsoires, classe, test_size = 0.2, random_state = 1)

```

O passo seguinte consistiu na criação e treinamento do modelo. Para isso foi criada a variável “*naive\_bayes*”, instanciando a classe “*GaussianNB*” e, em seguida, aplicado o método “*fit*” nas variáveis “X\_treinamento” e “Y\_treinamento” para treinamento do modelo (Figura 35).

Figura 35 - Criação e treinamento do modelo.

```

137  #Criação e treinamento do modelo
138  naive_bayes = GaussianNB()
139  naive_bayes.fit(X_treinamento, Y_treinamento)
140

```

Em seguida foi criada a variável “previsoes”, armazenando o método “predict” utilizado na variável “naive\_bayes” e passado como parâmetro a base de teste “X\_teste” (Figura 36). Como resultado temos a série mostrada na figura 37.



Figura 36 - Previsões utilizando a base de teste.

```

141 #Previsões utilizando os registros de teste
142 previsoes = naive_bayes.predict(X_teste)
143 print(previsoes)
144 print()

```

Figura 37 - Resultado das previsões.

```
['Sim' 'Sim' 'Sim' ... 'Sim' 'Sim' 'Sim']
```

Em seguida, foi gerada a matriz de confusão e feito o cálculo das métricas do modelo. Para a matriz de confusão, foi criada a variável “confusao”, armazenando a função “*confusion\_matrix*”, passando como parâmetros as variáveis “Y\_teste” e “previsoes”. Para as métricas, foi criada a variável “report”, armazenando a função “*classification\_report*”, passando como parâmetros as variáveis “Y\_teste” e “previsoes” (Figura 38). Os resultados são demonstrados na figura 39.

Figura 38 - Geração da matriz de confusão e relatório das métricas.

```

168 #Geração da matriz de confusão e cálculo das métricas
169 confusao = confusion_matrix(Y_teste, previsoes)
170 print(confusao)
171 print()
172
173 report = classification_report(Y_teste, previsoes)
174 print(report)
175 print()

```

Figura 39 - Resultado da matriz de confusão e das métricas e do modelo “*Gaussian Naive Bayes*” criado.

```

[[ 20 393]
 [ 28 1743]]

```

	precision	recall	f1-score	support
Não	0.42	0.05	0.09	413
Sim	0.82	0.98	0.89	1771
accuracy			0.81	2184
macro avg	0.62	0.52	0.49	2184
weighted avg	0.74	0.81	0.74	2184

As métricas de interesse ao projeto apresentadas são a Precisão (em inglês, *Precision*), a revocação (em inglês, *Recall*), a pontuação-f1 (em inglês, *f1-score*) e a acurácia (em inglês, *accuracy*).

1. A precisão é a proporção  $[\text{vp} / (\text{vp} + \text{fp})]$  em que **vp** é o número de verdadeiros positivos e **fp** o número de falsos positivos. A precisão é intuitivamente a capacidade do classificador de não rotular como positiva uma amostra negativa.
2. A revocação é a razão  $[\text{vp} / (\text{vp} + \text{fn})]$  onde **vp** é o número de verdadeiros positivos e **fn** o número de falsos negativos. O recall é intuitivamente a capacidade do classificador de encontrar todas as amostras positivas.
3. A pontuação F1 pode ser interpretada como uma média ponderada da precisão e revocação  $[2 * ((\text{precisão} * \text{revocação}) / (\text{precisão} + \text{revocação}))]$ , onde uma pontuação F1 atinge seu melhor valor em 1 e a pior pontuação em 0.
4. A acurácia contabiliza a proporção das previsões corretas  $[(\text{vp} + \text{vn}) / (\text{vp} + \text{vn} + \text{fp} + \text{fn})]$ .

Por fim, foi gerada a curva ROC, que é uma representação gráfica que ilustra o desempenho de um sistema classificador binário à medida que o seu limiar de discriminação varia.

A curva ROC é também conhecida como curva característica de operação relativa, porque o seu critério de mudança é resultado da operação de duas características (Verdadeiros positivos e Falsos Positivos). A curva ROC é obtida pela representação da razão **TVP = Verdadeiros Positivos / Positivos Totais (Verdadeiros Positivos + Falsos Negativos)** versus a razão **TFP = Falsos Positivos / Negativos Totais (Verdadeiros Negativos + Falsos Positivos)**, para vários valores do limiar de classificação.

O valor de AUC fornece uma medida agregada de desempenho em todos os limites de classificação possíveis. O valor de AUC representa a probabilidade de que o modelo classifique um caso positivo aleatório melhor do que um caso negativo aleatório.

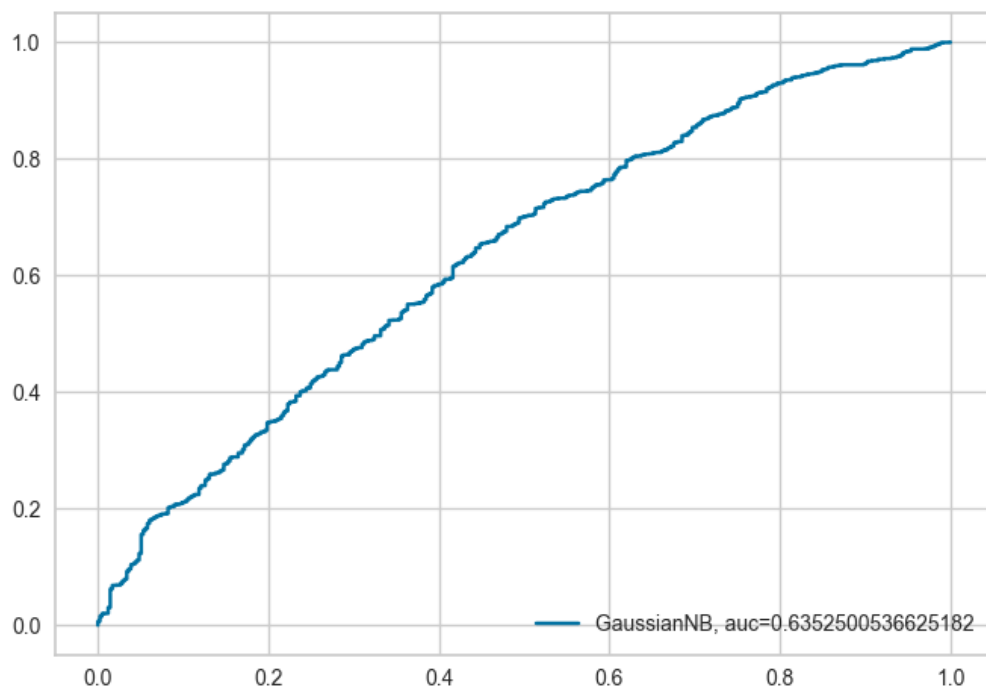
Para geração da curva ROC com o valor AUC referente ao modelo “*Gaussian Naïve Bayes*” foi criada uma variável “prob” utilizando a função “*predict\_proba*” na variável “X\_teste”, em seguida foi criada a curva utilizando a função “*sklearn.metrics.roc\_curve*”, passando como parâmetros as variáveis “Y\_teste”, “prob” e ‘sim’ como rótulo da classe positiva (pos\_label=’Sim’). Em seguida, foi calculado o valor AUC utilizando a função “*sklearn.metrics.roc\_auc\_score*”, passando como parâmetros “Y\_teste” e “prob”. Por fim, utilizou-se a função “*plot*” do “*matplotlib.pyplot*”

para geração e visualização do gráfico (Figura 40). O resultado pode ser visualizado na figura 41.

Figura 40 – Geração da curva ROC e AUC.

```
183 #Curva ROC e AUC
184 prob = naive_bayes.predict_proba(X_teste)[:,-1]
185 tfp, tvp, thresholds = roc_curve(Y_teste, prob, pos_label='Sim')
186 auc = roc_auc_score(Y_teste, prob)
187 plt.plot(tfp, tvp, label="GaussianNB, auc="+str(auc))
188 plt.legend(loc=4)
189 plt.show()
```

Figura 41 - ROC e AUC para o modelo "Gaussian Naive Bayes".



Analisando a matriz de confusão observou-se poucos valores na classe “Não”, que é um indicativo de que o modelo conseguiu melhor classificação na classe “Sim”. Além disso, é possível observar que a acurácia do modelo é alta (81%), todavia não é suficiente para indicar um bom desempenho do modelo, uma vez que o valor de vp é muito alto. Também é possível confirmar o enviesamento do modelo devido através das métricas, apresentando um baixo recall de 5% para a classe

“Não”, que consiste na porcentagem de dados classificados como negativos comparado a quantidade real de negativos que existem na amostra para essa classe.

Analisando a curva ROC, percebeu-se que o modelo possui capacidade de classificação. No entanto, a interpretação visual da curva ROC, no contexto de conjuntos de dados desbalanceados, pode ser enganosa no que diz respeito às conclusões sobre a confiabilidade do desempenho de classificação, devido a uma interpretação intuitiva, mas errada, da sensibilidade (proporção de casos positivos que foram identificados corretamente). Para casos desbalanceados a curva Precisão x Revocação na classe minoritária seria mais adequada para avaliar o desempenho do modelo. Dessa forma, optou-se por utilizar técnicas para lidar com o desbalanceamento das classes.

Como solução para o desbalanceamento decidiu-se utilizar duas técnicas de sobreamostragem: “SMOTE” e “ADASYN”.

O “SMOTE” (em inglês, *Synthetic Minority Oversampling Technique*) é uma técnica proposta por Chawla et al. (2002) que gera dados artificiais por meio de interpolação, gerando casos sintéticos (artificiais) para a classe de interesse a partir dos casos já existentes. Os novos casos são gerados na vizinhança de cada caso da classe minoritária com o intuito de se crescer o espaço de decisão desta classe aumentar o poder de generalização dos classificadores obtidos. Os novos casos sintéticos são interpolados aleatoriamente ao longo do segmento de reta que une cada caso da classe minoritária a um de seus k-vizinhos mais próximos, escolhidos de forma aleatória.

O “ADASYN” (em inglês, *Adaptive Synthetic*) é similar ao SMOTE, porém usa distribuição local de densidade da classe minoritária como um critério para decidir automaticamente o número de amostras sintéticas que devem ser geradas para cada amostra minoritária, alterando de maneira adaptativa os pesos das diferentes amostras minoritárias para compensar a distorção das distribuições.

Primeiramente foi necessário importar as classes “SMOTE” e “ADASYN” da biblioteca “*imbalanced-learn*” (Figura 42).

Figura 42 - Importação das classes “SMOTE” e “ADASYN”.

```
17 from imblearn.over_sampling import SMOTE, ADASYN
```

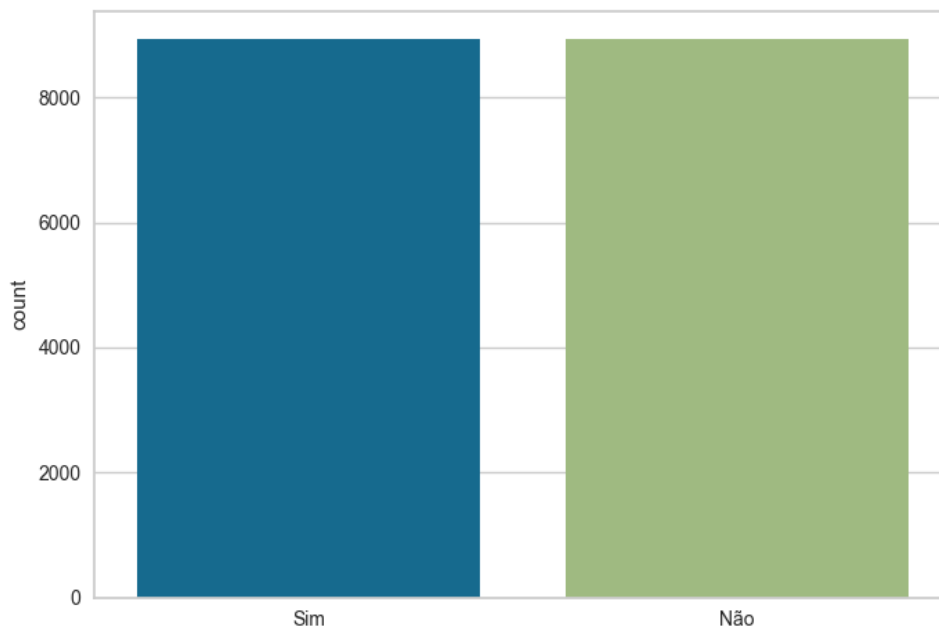
Em seguida, no mesmo modelo de *machine learning* criado originalmente, adicionou-se as linhas de código apresentadas na figura 43 logo após a codificação dos atributos categóricos e antes do treinamento do modelo. Primeiramente, instanciou-se a classe “SMOTE”, com o parâmetro “*random\_state*” igual a 1 para controlar

a randomização do algoritmo e, em seguida, aplicou-se o método “*fit\_sample*” nas variáveis “previsores” e “classe” a fim de balancear as classes utilizando a técnica “*SMOTE*”. Depois, criou-se uma variável “balanceado” na qual foi armazenada a função “*seaborn.countplot*” para representar graficamente a distribuição de classes em um gráfico de barras. O resultado pode ser verificado na figura 44.

Figura 43 - Sobreamostragem utilizando “*SMOTE*”.

```
145 #Reamostragem para lidar com o problema do desbalanceamento de classes
146 smote = SMOTE(random_state=1)
147 previsores, classe = smote.fit_sample(previsores, classe)
148 balanceado = sb.countplot(x=classe)
149 plt.show()
```

Figura 44 - Classes balanceadas utilizando “*SMOTE*”.

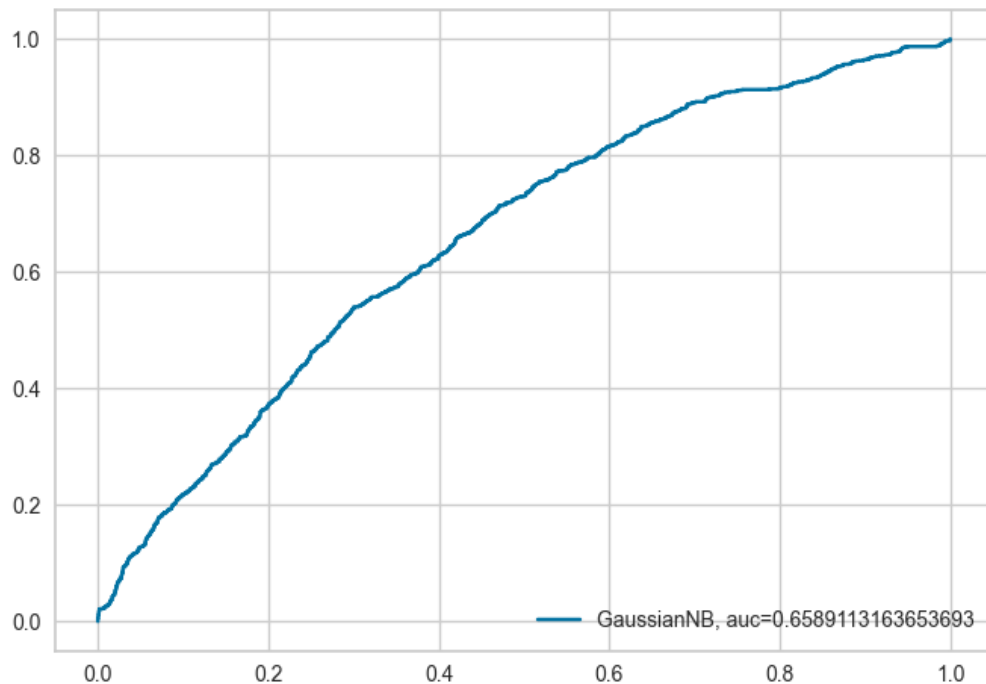


Como resultado do novo modelo utilizando “*SMOTE*”, obteve-se a matriz de confusão, as métricas e a curva ROC AUC mostradas na figura 45 e 46.

Figura 45 - Resultado da matriz de confusão e das métricas e do modelo “*Gaussian Naive Bayes*” sobreamostrado com “SMOTE”.

[[1361 441]				
[ 982 788]]				
	precision	recall	f1-score	support
Não	0.58	0.76	0.66	1802
Sim	0.64	0.45	0.53	1770
accuracy			0.60	3572
macro avg	0.61	0.60	0.59	3572
weighted avg	0.61	0.60	0.59	3572

Figura 46 - Curva ROC AUC “*Gaussian Naive Bayes*” utilizando “SMOTE”.



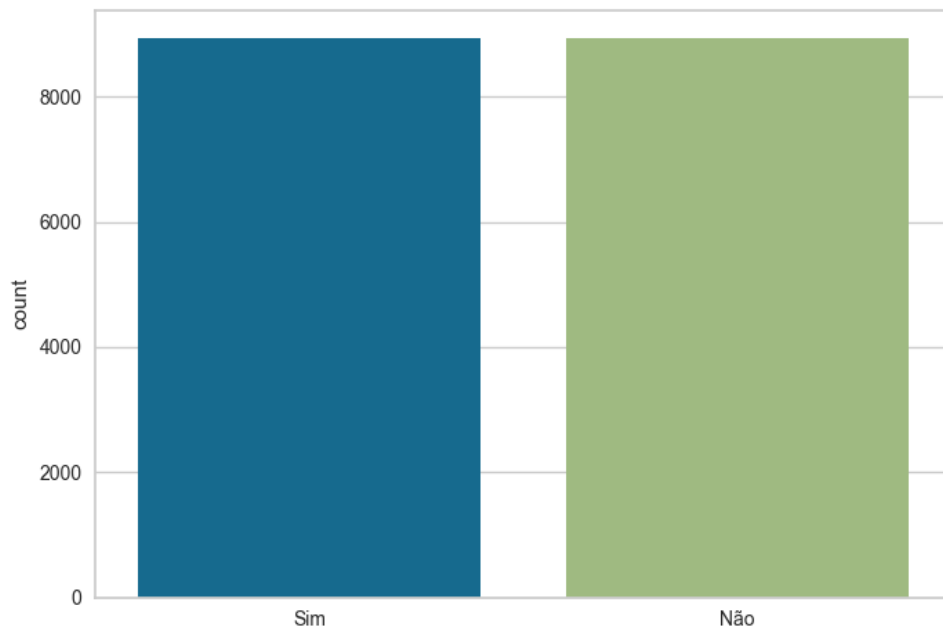
Para teste o “ADASYN”, no mesmo modelo de *machine learning* criado originalmente, adicionou-se as linhas de código apresentadas na figura 47 no lugar das linhas de código criadas para o “SMOTE”. Primeiramente, instanciou-se a classe “ADASYN”, utilizando o parâmetro “*random\_state*” igual a 1, e em seguida aplicou-se o método “*fit\_sample*” nas variáveis “*previsores*” e “*classe*” a fim de balancear as classes utilizando a técnica “ADASYN”. Depois, criou-se uma variável “balanceado” na qual foi armazenada a função “*seaborn.countplot*” para representar graficamente

a distribuição de classes em um gráfico de barras. O resultado pode ser verificado na figura 48.

Figura 47 - Sobreamostragem utilizando "ADASYN".

```
151 adasyn = ADASYN(random_state=1)
152 previsores, classe = adasyn.fit_sample(previsores, classe)
153 balanceado = sb.countplot(x=classe)
154 plt.show()
```

Figura 48 - Classes balanceadas utilizando "ADASYN".



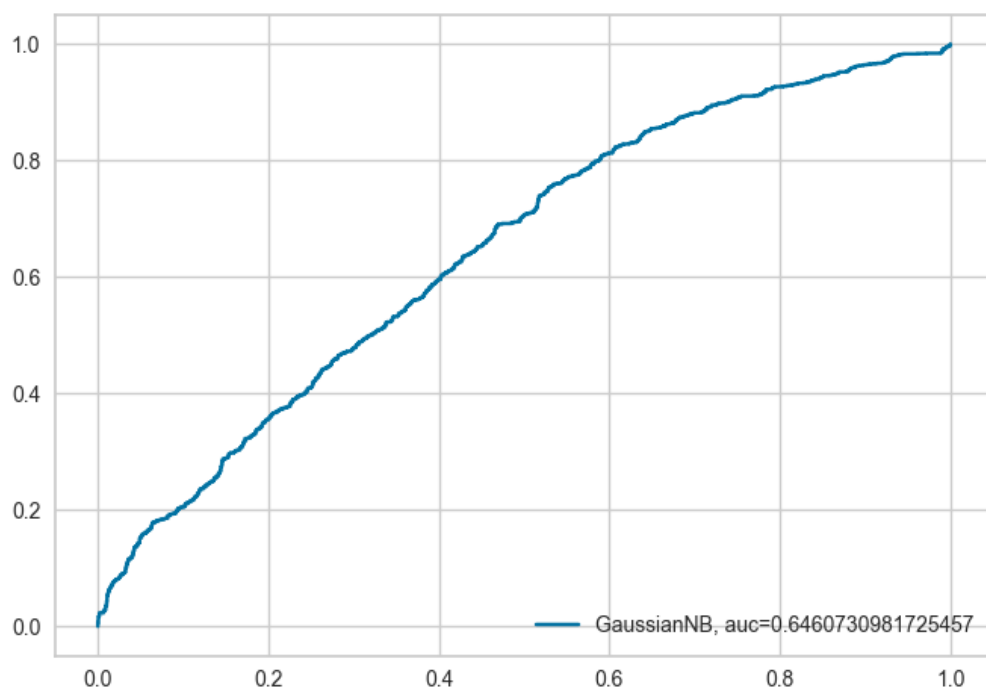
Como resultado do novo modelo utilizando "ADASYN", obteve-se a matriz de confusão, as métricas e a curva ROC AUC mostradas na figura 49 e 50.

Figura 49 - Resultado da matriz de confusão e das métricas e do modelo “*Gaussian Naive Bayes*” superamos-trado com “*ADASYN*”.

```
[[1087  668]
 [ 788 1022]]
```

	precision	recall	f1-score	support
Não	0.58	0.62	0.60	1755
Sim	0.60	0.56	0.58	1810
accuracy			0.59	3565
macro avg	0.59	0.59	0.59	3565
weighted avg	0.59	0.59	0.59	3565

Figura 50 - Curva ROC AUC “*Gaussian Naive Bayes*” utilizando ADASYN.



## 5.2. Modelo 2: “*Decision Tree Classifier*”

O “*Decision Tree Classifier*” (em inglês, Classificador de Arvore de Decisão) é um algoritmo de classificação baseado em árvores de decisão, na qual cada nó interno denota um teste em um atributo, cada ramo representa um resultado do teste e os nós folha representam classes ou distribuições de classes. O nó inicial em uma



árvore de decisão é conhecido como nó raiz, no qual ocorre o particionamento recursivo até os nós folhas alcançarem o nível máximo de pureza.

Primeiramente foi necessário importar as funções de alguns módulos da biblioteca “scikit-learn” que serão utilizadas na criação e visualização do modelo de machine learning. Além disso, foi importada a biblioteca “yellowbrick” para fins de visualização gráfica da matriz de confusão, que mostra cada combinação das classes verdadeiras e previstas para um conjunto de dados de teste e a biblioteca “matplotlib”, que consiste em uma biblioteca abrangente para a criação de visualizações estáticas, animadas ou interativas em Python (Figura 51).

**Figura 51 - Importação das funções e classes dos módulos das bibliotecas “scikit-learn”, “yellowbrick” e “matplotlib”.**

```
10 from sklearn.model_selection import train_test_split
11 from sklearn.preprocessing import LabelEncoder
12 from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_score
13 from sklearn.tree import DecisionTreeClassifier
14 from sklearn.tree import plot_tree
15 from yellowbrick.classifier import ConfusionMatrix
16 import seaborn as sb
17 import matplotlib.pyplot as plt
18 from matplotlib import figure
```

Primeira importação: Função “*train\_test\_split*” do módulo “*sklearn.model\_selection*”, que divide vetores ou matrizes em subconjuntos de treino e teste randômicos.

Segunda importação: Classe “*LabelEncoder*” do módulo “*sklearn.preprocessing*”, que permite codificar atributos categóricos em numéricos.

Terceira importação: Funções “*confusion\_matrix*”, “*classification\_report*”, “*roc\_curve*” e “*roc\_auc\_score*” do módulo “*sklearn.metrics*”, que permitem computar a matriz de confusão de uma classificação, exibir o relatório de métricas, plotar a curva ROC (em inglês, *Receiver Operating Characteristic*) e calcular o valor de AUC (em inglês, *Area Under the ROC Curve*).

Quarta importação: Classe “*DecisionTreeClassifier*” do módulo “*sklearn.tree*” para aplicação do algoritmo “*Decision Tree Classifier*”.

Quinta importação: Função “*plot\_tree*” do módulo “*sklearn.tree*”, que permite plotar a árvore de decisão.

Sexta importação: Classe “*ConfusionMatrix*” do módulo “*yellowbrick.classifier*”, que cria uma visualização de mapa de calor da função “*confusion\_matrix*”.

Sétima importação: Biblioteca “*matplotlib.pyplot*”, que se destina principalmente a plotagens interativas e casos simples de geração de plotagem da programação.

Oitava importação: Função “figure” do “matplotlib”, que permite criar figuras.

O próximo passo criar as variáveis “previsores” e “classe” a partir do “DataFrame” “incidentes”. Para isso, utilizou-se o método “pandas.DataFrame.iloc” para todas as linhas e colunas de índices 0 a 10, no caso dos previsores, e índice 11 para todas as linhas e coluna de índice 11, no caso da classe (Figura 52).

Figura 52 - Criação das variáveis “previsores” e “classe”.

```
121  previsores = incidentes.iloc[:,0:11].values
122  classe = incidentes.iloc[:,11].values
```

O passo seguinte consistiu em criar uma variável “labelencoder”, instanciando a classe “LabelEncoder” e, em seguida, criar um objeto para cada coluna categórica, utilizando o método “fit\_transform”, que aplica o codificador de rótulos (“label encoder”) e retorna rótulos codificados, ou seja, esse método permite transformar cada uma das colunas com atributos categóricos em colunas com atributos numéricos (Figura 53).

Figura 53 - Codificação dos atributos categóricos em atributos numéricos.

```
124  labelencoder = LabelEncoder()
125
126  previsores[:,0] = labelencoder.fit_transform(previsores[:,0])
127  previsores[:,1] = labelencoder.fit_transform(previsores[:,1])
128  previsores[:,2] = labelencoder.fit_transform(previsores[:,2])
129  previsores[:,3] = labelencoder.fit_transform(previsores[:,3])
130  previsores[:,4] = labelencoder.fit_transform(previsores[:,4])
131  previsores[:,6] = labelencoder.fit_transform(previsores[:,6])
132  previsores[:,7] = labelencoder.fit_transform(previsores[:,7])
133  previsores[:,8] = labelencoder.fit_transform(previsores[:,8])
134  previsores[:,9] = labelencoder.fit_transform(previsores[:,9])
135  previsores[:,10] = labelencoder.fit_transform(previsores[:,10])
```

Em seguida, utilizou-se a função “train\_test\_split” para dividir a base de dados entre treinamento (variáveis “X\_treinamento” e “Y\_treinamento”) e teste (variáveis “X\_teste” e “Y\_teste”). Como parâmetros da função foram utilizadas as variáveis “previsores” e “classe”. Foi definido um tamanho 20% para teste e 80% para treino, utilizando o parâmetro “test\_size” (0,2), considerando o tamanho da base de dados de, aproximadamente, 11000 linhas. Além disso, foi utilizado um número inteiro (1) no parâmetro “random\_state” para garantir a reprodução da mesma saída da função em cada vez que ela fosse executada (Figura 54).

Figura 54 - Divisão da base de dados entre teste e treinamento.

```
137  #Divisão da base de dados entre treinamento e teste (20% para teste e 80% para treinamento)
138  X_treinamento, X_teste, Y_treinamento, Y_teste = train_test_split(previsores, classe, test_size = 0.2, random_state = 1)
```

O passo seguinte consistiu na criação e treinamento do modelo. Para isso foi criada a variável “arvore”, instanciando a classe “DecisionTreeClassifier” e, em seguida, aplicado o método “fit” nas variáveis “X\_treinamento” e “Y\_treinamento”, para treinamento do modelo. Novamente foi passado como parâmetro de estado aleatório (“random\_state”) um número inteiro (1), garantindo o comportamento determinístico durante o treinamento do modelo. Em seguida, foi criada uma figura com a representação gráfica da árvore de decisão utilizando a função “matplotlib.pyplot.figure”, uma variável “a” que recebeu a função “plot\_tree”, passando como parâmetros a variável “arvore”, a profundidade (“max\_depth=3”), preenchimento dos nós (“filled=True”) e arredondamento das bordas dos nós (“rounded=True”) (Figura 55). O resultado é apresentado na figura 56.

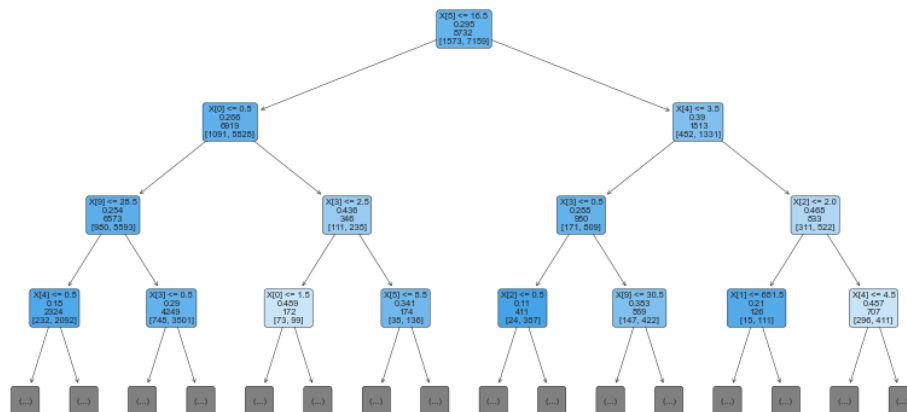
Figura 55 - Criação, treinamento do modelo e visualização da árvore de decisão.

```

140 #Criação e treinamento do modelo
141 arvore = DecisionTreeClassifier(random_state = 1)
142 arvore.fit(X_treinamento,Y_treinamento)
143
144 #Visualização da árvore de decisão
145 plt.figure(figsize=(10,5))
146 a=plot_tree(arvore, max_depth=3, filled=True, rounded=True)
147 plt.show()

```

Figura 56 - Árvore de decisão do modelo.



Em seguida, foi criada a variável “previsoes” armazenando o método “predict” utilizado na variável “arvore” e passado como parâmetro a base de teste “X\_teste” (Figura 57). Como resultado temos a série mostrada na figura 58.

Figura 57 - Previsões utilizando a base de teste.

```

149 #Previsões utilizando os registros de teste
150 previsoes = arvore.predict(X_teste)
151 print(previsoes)
152 print()

```

Figura 58 - Resultado das previsões.

```
['Sim' 'Não' 'Sim' ... 'Sim' 'Sim' 'Não']
```

Por fim, foi gerada a matriz de confusão e feito o cálculo das métricas do modelo. Para a matriz de confusão, foi criada a variável “confusao”, armazenando a função “*confusion\_matrix*”, passando como parâmetros as variáveis “*Y\_teste*” e “*previsoes*”. Para as métricas, foi criada a variável “report” armazenando a função “*classification\_report*”, passando como parâmetros as variáveis “*Y\_teste*” e “*previsoes*” (Figura 59). Os resultados são demonstrados na figura 60.

Figura 59 - Geração da matriz de confusão e relatório das métricas.

```

172 #Geração da matriz de confusão e cálculo da acurácia
173 confusao = confusion_matrix(Y_teste, previsoes)
174 print(confusao)
175 print()
176
177 report = classification_report(Y_teste, previsoes)
178 print(report)
179 print()

```

Figura 60 - Resultado da matriz de confusão e das métricas e do modelo “*Decision Tree Classifier*” criado.

```

[[ 201  212]
 [ 232 1539]]

```

	precision	recall	f1-score	support
Não	0.46	0.49	0.48	413
Sim	0.88	0.87	0.87	1771
accuracy			0.80	2184
macro avg	0.67	0.68	0.67	2184
weighted avg	0.80	0.80	0.80	2184

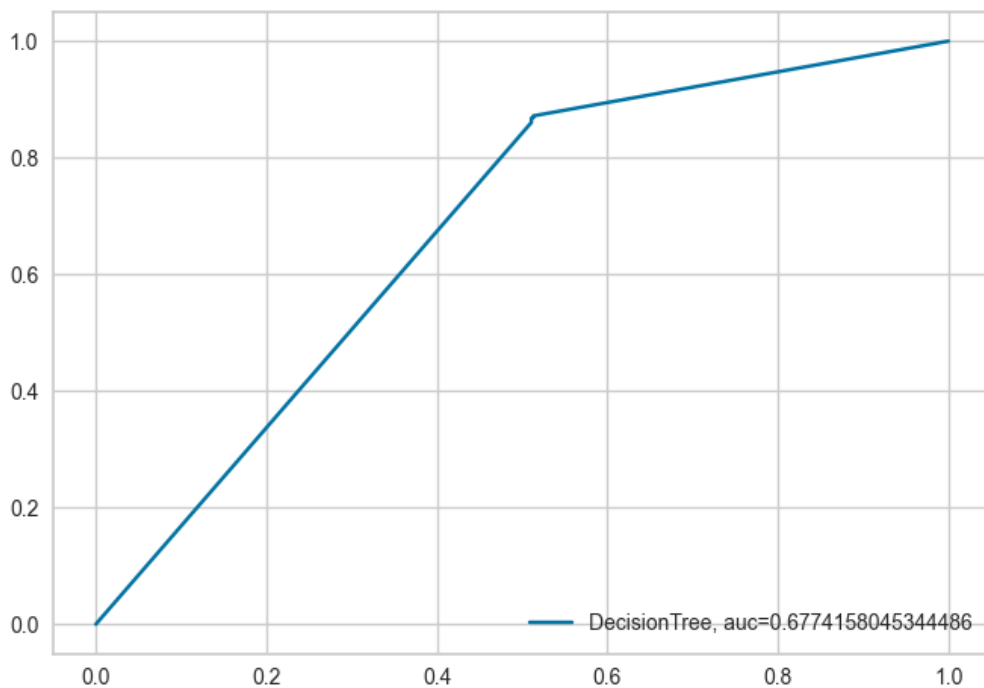
Para geração da curva ROC com o valor AUC referente ao modelo “*Decision Tree Classifier*” foi criada uma variável “*prob*” utilizando a função “*predict\_proba*” na variável “*X\_teste*”, em seguida foi criada a curva utilizando a função “*sklearn.metrics.roc\_curve*”, passando como parâmetros a variável “*Y\_teste*”, “*prob*” e ‘sim’ como rótulo da classe positiva (*pos\_label*='Sim'). Em seguida, foi calculado o

valor AUC utilizando a função “*sklearn.metrics.roc\_auc\_score*”, passando como parâmetros “*Y\_teste*” e “*prob*”. Por fim, utilizou-se a função “*plot*” do “*matplotlib.pyplot*” para geração e visualização do gráfico (Figura 61). O resultado pode ser visualizado na figura 62.

Figura 61 - Geração e visualização da curva ROC AUC do modelo “*Decision Tree Classifier*”.

```
187 #Curva ROC e AUC
188 prob = arvore.predict_proba(X_teste)[:,-1]
189 tfp, tvp, thresholds = roc_curve(Y_teste, prob, pos_label='Sim')
190 auc = roc_auc_score(Y_teste, prob)
191 plt.plot(tfp, tvp, label="DecisionTree, auc="+str(auc))
192 plt.legend(loc=4)
193 plt.show()
```

Figura 62 - ROC e AUC para o modelo “*Decision Tree Classifier*”.



Analisando a matriz de confusão percebe-se poucos valores na classe “Não”, que é um indicativo de que o modelo conseguiu melhor classificação na classe “Sim”. Além disso, é possível observar que a acurácia do modelo é alta (80%), novamente não sendo um bom indicador de desempenho do modelo. Observando as métricas, todavia é possível confirmar o enviesamento do modelo pela diferença no recall, que apresentou 49% para a classe “Não” e 87% para a classe “Sim”.

Analisando a curva ROC, percebeu-se que o modelo também possui capacidade de classificação. No entanto, a interpretação visual da curva ROC, no contexto

de conjuntos de dados desbalanceados, pode ser enganosa no que diz respeito às conclusões sobre a confiabilidade do desempenho de classificação, conforme explicado no primeiro modelo “*Gaussian Naive Bayes*”. Dessa forma, também, optou-se por utilizar técnicas para lidar com o desbalanceamento de classes.

Primeiramente foi necessário importar as classes “*SMOTE*” e “*ADASYN*” da biblioteca “*imbalanced-learn*” (Figura 63).

Figura 63 - Importação das classes “*SMOTE*” e “*ADASYN*”.

```
19 from imblearn.over_sampling import SMOTE, ADASYN
```

Em seguida, no mesmo modelo de *machine learning* criado originalmente de árvore de decisão, adicionou-se as linhas de código apresentadas na figura 64, logo após a codificação dos atributos categóricos, e antes do treinamento do modelo. Primeiramente, instanciou-se a classe “*SMOTE*”, em seguida aplicou-se o método “*fit\_sample*” nas variáveis “previsores” e “classe” a fim de balancear as classes utilizando a técnica “*SMOTE*”. Depois, criou-se uma variável “balanceado” na qual foi armazenada a função “*seaborn.countplot*” para representar graficamente a distribuição de classes em um gráfico de barras. O resultado da nova distribuição e a árvore de decisão podem ser verificados nas figuras 65 e 66.

Figura 64 - Superamostragem utilizando “*SMOTE*”.

```
144 #Reamostragem para lidar com o problema do desbalanceamento de classes
145 smote = SMOTE(random_state=1)
146 previsores, classe = smote.fit_sample(previsores, classe)
147 balanceado = sb.countplot(x=classe)
148 plt.show()
```

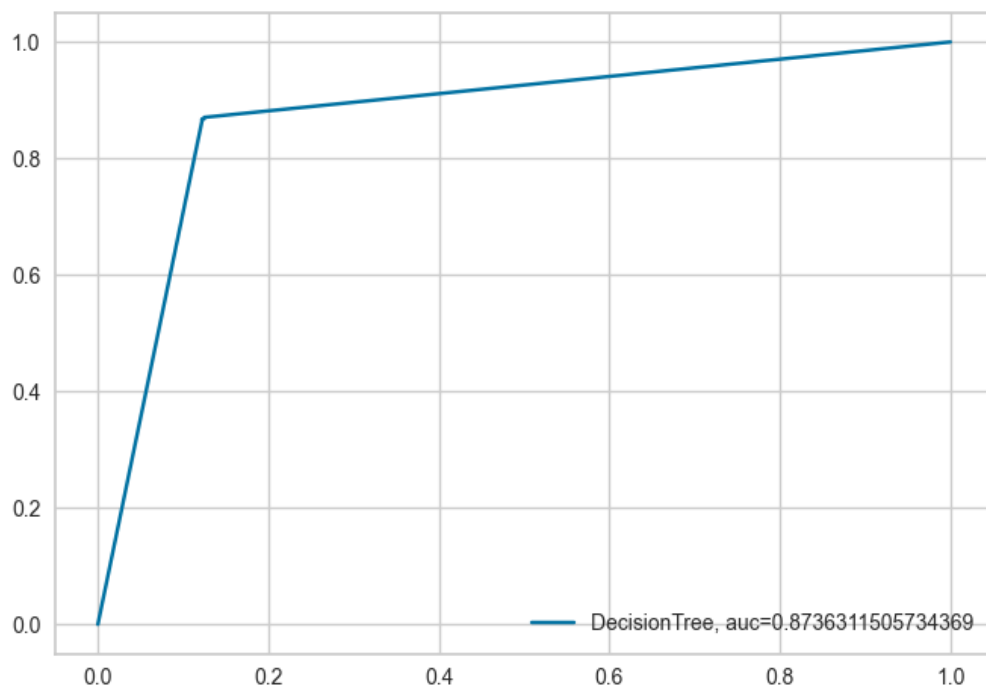


Figura 67 - Resultado da matriz de confusão e das métricas e do modelo “*Decision Tree Classifier*” superamostrado com “*SMOTE*”.

```
[[1578  224]
 [ 234 1536]]
```

	precision	recall	f1-score	support
Não	0.87	0.88	0.87	1802
Sim	0.87	0.87	0.87	1770
accuracy			0.87	3572
macro avg	0.87	0.87	0.87	3572
weighted avg	0.87	0.87	0.87	3572

Figura 68 - ROC e AUC para o modelo “*Decision Tree Classifier*” utilizando “*SMOTE*”.



Para o teste com “*ADASYN*”, no mesmo modelo de *machine learning* criado originalmente, adicionou-se as linhas de código apresentadas na figura 69 no lugar das linhas de código criadas para o “*SMOTE*”. Primeiramente, instanciou-se a classe “*ADASYN*”, utilizando o parâmetro “*random\_state*” igual a 1, e em seguida aplicou-se o método “*fit\_sample*” nas variáveis “previsores” e “classe” a fim de balancear as classes utilizando a técnica “*ADASYN*”. Depois, criou-se uma variável “balanceado” na qual foi armazenada a função “*seaborn.countplot*” para representar graficamente



a distribuição de classes em um gráfico de barras. O resultado da nova distribuição e a árvore de decisão podem ser verificados nas figuras 70 e 71, respectivamente.

Figura 69 - Superamostragem utilizando "ADASYN".

```
150 adasyn = ADASYN(random_state=1)
151 previsores, classe = adasyn.fit_sample(previsores, classe)
152 balanceado = sb.countplot(x=classe)
153 plt.show()
```

Figura 70 - Classes balanceadas utilizando "ADASYN".

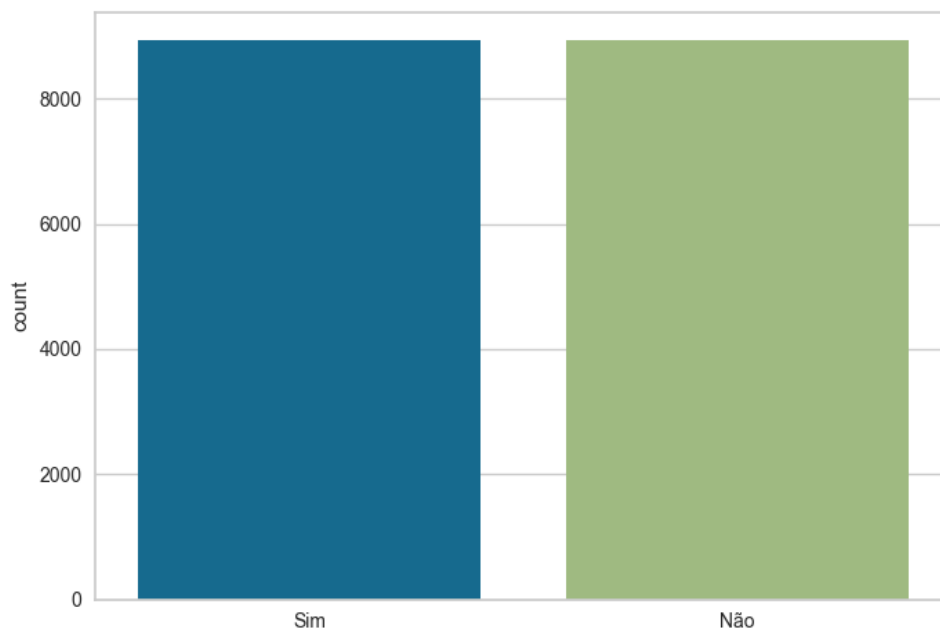
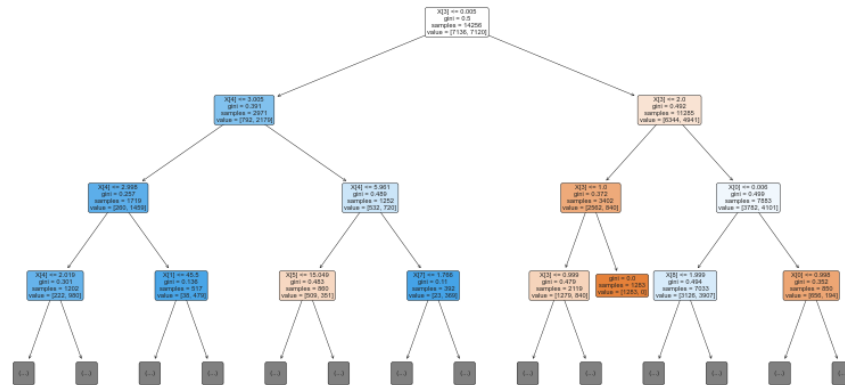


Figura 71 - Árvore de decisão utilizando "ADASYN".

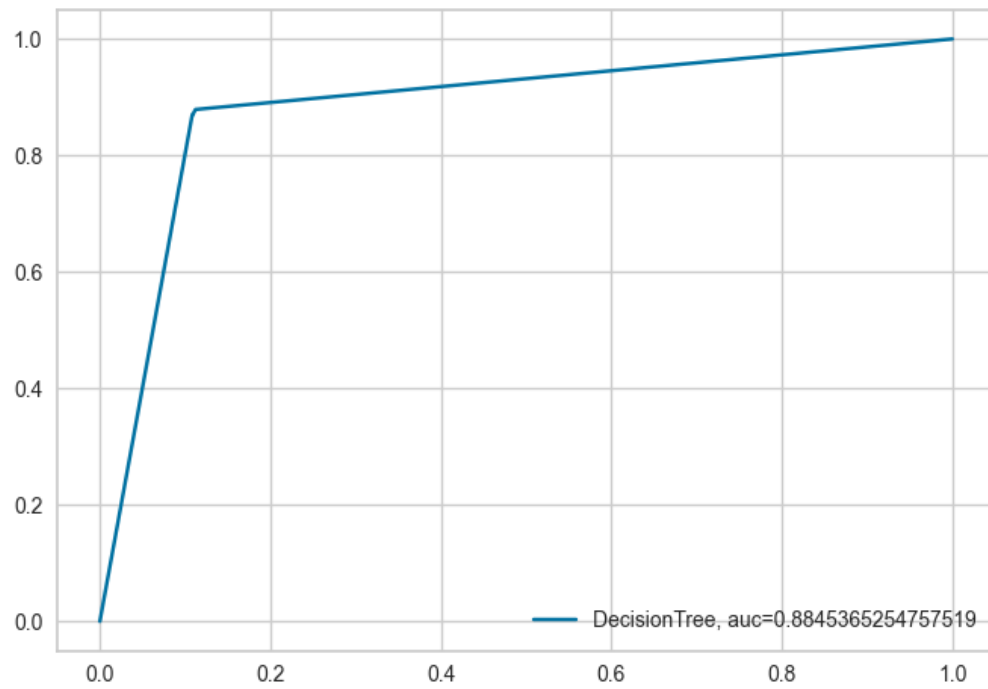


Como resultado do novo modelo utilizando "ADASYN", obteve-se a matriz de confusão, as métricas e a curva ROC AUC mostradas nas figuras 72 e 73.

Figura 72 - Resultado da matriz de confusão e das métricas e do modelo "Decision Tree Classifier" supermostrado com "ADASYN".

[[1562 193]					
[ 230 1580]]					
	precision	recall	f1-score	support	
Não	0.87	0.89	0.88	1755	
Sim	0.89	0.87	0.88	1810	
accuracy			0.88	3565	
macro avg	0.88	0.88	0.88	3565	
weighted avg	0.88	0.88	0.88	3565	

Figura 73 - ROC e AUC para o modelo "*Decision Tree Classifier*" utilizando "*ADASYN*".



## 6. Apresentação dos Resultados

Os resultados do projeto foram obtidos a partir do estabelecimento das metas mapeadas no modelo Canvas de Vasandani conforme apresentado na figura 74.

Figura 74 – Modelo Canvas.

**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:		
<b>1 Problem Statement</b> What problem are you trying to solve? What larger issues do the problem address?  Melhorar a eficiência operacional na tratativa de falhas de clientes atendi- dos por fornecedores EILD, a partir da previsão de classificação positiva ou negativa da qualidade no serviço prestado.	<b>2 Outcomes/Predictions</b> What prediction(s) are you trying to make? Identify applicable predictor (x) and/or target (y) variables.  Prever a classe a partir das variáveis previsoras.  Variáveis previsoras: Atributos dos bilhetes, dados de cadastro de forne- cedores EILD e classificação de rodo- vias.  Classe: Aval_CIGR.	<b>3 Data Acquisition</b> Where are you sourcing your data from? Is there enough data? Can you work with it?  Sistema de gestão de incidentes da TELEBRAS.  Sistema de cadastro de fornecedores da TELEBRAS.  Dados de qualidade de rodovias da confederação nacional do transporte.
<b>4 Modeling</b> What models are appropriate to use given your outcomes?  Algoritmos de <i>machine learning</i> de aprendizado supervisionada.  Gaussian Naive Bayes  Decision Tree Classifier	<b>5 Model Evaluation</b> How can you evaluate your model's performance?  Matriz de confusão.  Métricas: Precision, Recall, F1-Score.  Curva ROC/AUC.	<b>6 Data Preparation</b> What do you need to do to your data in order to run your model and achieve your outcomes?  Tratar e limpar as bases adquiridas.  Agrupar as bases.  Encontrar os atributos relevantes para realizar as previsões.

**✓ 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.

Conforme observado no decorrer do trabalho, foram implementados dois mo-  
delos de *machine learning* para classificação de qualidade no serviço prestado em  
bilhetes de falha para casos atendidos por fornecedores de EILD, com o intuito de  
prever se um bilhete será atendido de maneira satisfatória ou não, a partir dos atri-  
butos previsoires selecionados.

Foram apresentadas duas propostas para resolução do problema estabelecido, porém observou-se o problema do desbalanceamento de classes nas duas, o que gerou interpretações erradas sobre o desempenho dos modelos inicialmente.

A princípio, os dois modelos foram criados sem que as classes da variável alvo (“Aval\_CIGR”) estivessem balanceadas, gerando uma acurácia para o modelo “*Gaussian Naive Bayes*” de 81% e para o modelo “*Decision Tree Classifier*” de 91%. Todavia, a acurácia, por si só, não é uma métrica confiável conforme visto no decorrer do trabalho, sendo necessário avaliar outras métricas como precisão, revocação e pontuação-f1. Além disso, as curvas ROC também não são confiáveis para garantir um bom desempenho dos modelos quando as classes estão desbalanceadas, uma vez que são baseadas em verdadeiros positivos e falsos positivos, estando na mesma coluna na matriz de confusão quando a taxa de positivos é muito alta. Um vez que a base não possui mais classes do tipo “Não”, optou-se então pela utilização de técnicas de sobreamostragem (“*SMOTE*” e “*ADASYN*”) para balancear as classes com valores sintéticos para essa classe (minoritária) e assim decidir qual seria o melhor modelo e técnica de melhor desempenho.

A sobreamostragem da classe minoritária (“Não”) corrigiu o problema do desbalanceamento e permitiu com que os modelos apresentassem uma maior fidelidade no desempenho com relação aos modelos originais.

Os resultados das métricas dos modelos são apresentados na tabela 4.

**Tabela 4 - Resultado das métricas dos modelos**

Gaussian Naive Bayes						
Amostragem	Classes	Precisão	Recall	F1-Score	Acurácia	AUC
Classe com amostras originais	Não	0,42	0,05	0,09	0,81	0,635
	Sim	0,82	0,98	0,89		
Classes com over-sampling (SMOTE)	Não	0,58	0,76	0,66	0,6	0,658
	Sim	0,64	0,45	0,53		
Classes com over-sampling (ADASYN)	Não	0,58	0,62	0,6	0,59	0,646
	Sim	0,6	0,56	0,58		
Decision Tree Classifier						
Amostragem	Classes	Precisão	Recall	F1-Score	Acurácia	AUC
Classe com amostras originais	Não	0,46	0,49	0,48	0,8	0,677
	Sim	0,88	0,87	0,87		
Classes com over-sampling (SMOTE)	Não	0,87	0,88	0,87	0,87	0,873
	Sim	0,87	0,87	0,87		
Classes com over-sampling (ADASYN)	Não	0,87	0,89	0,88	0,88	0,884
	Sim	0,89	0,87	0,88		

As matrizes de confusão do modelo “*Gaussian Naive Bayes*” são apresentadas nas figuras 75, 76, 77.

Figura 75 - Matriz de confusão “*Gaussian Naive Bayes*” (desbalanceada).

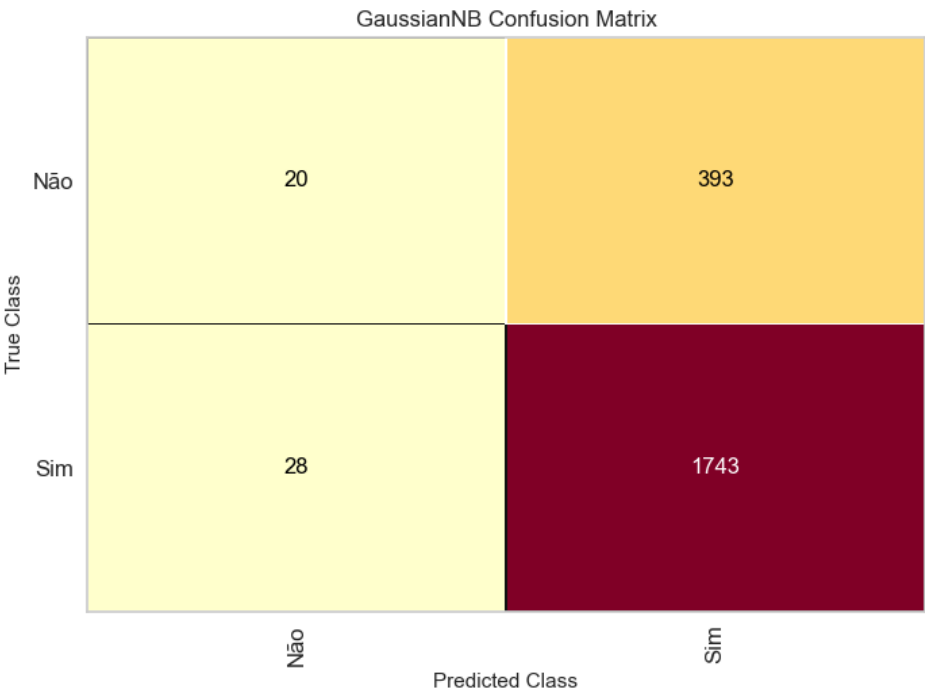


Figura 76 - Matriz de confusão “*Gaussian Naive Bayes*” (“SMOTE”).

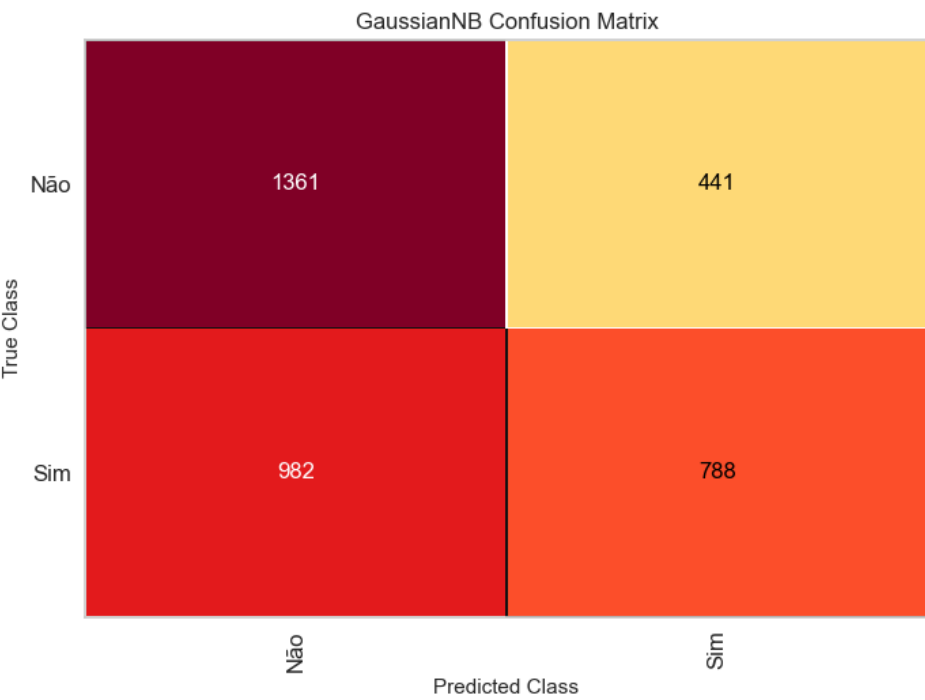
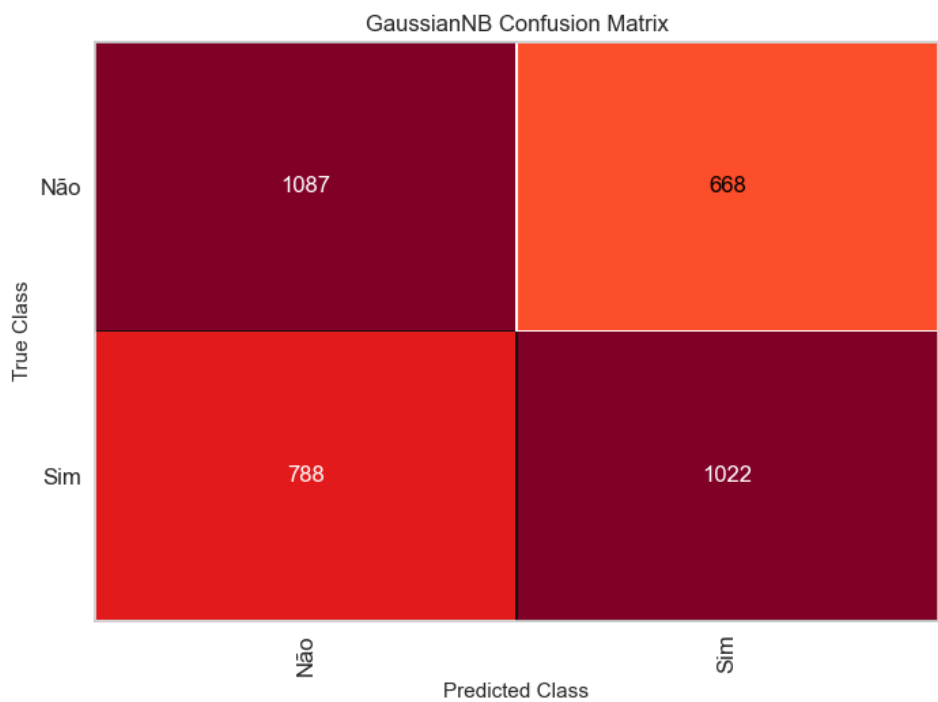


Figura 77 - Matriz de confusão “Gaussian Naive Bayes” (“ADASYN”).



As matrizes de confusão do modelo “*Decision Tree Classifier*” são apresentadas nas figuras 78, 79, 80.

Figura 78 - Matriz de confusão “Decision Tree Classifier”(desbalanceada).

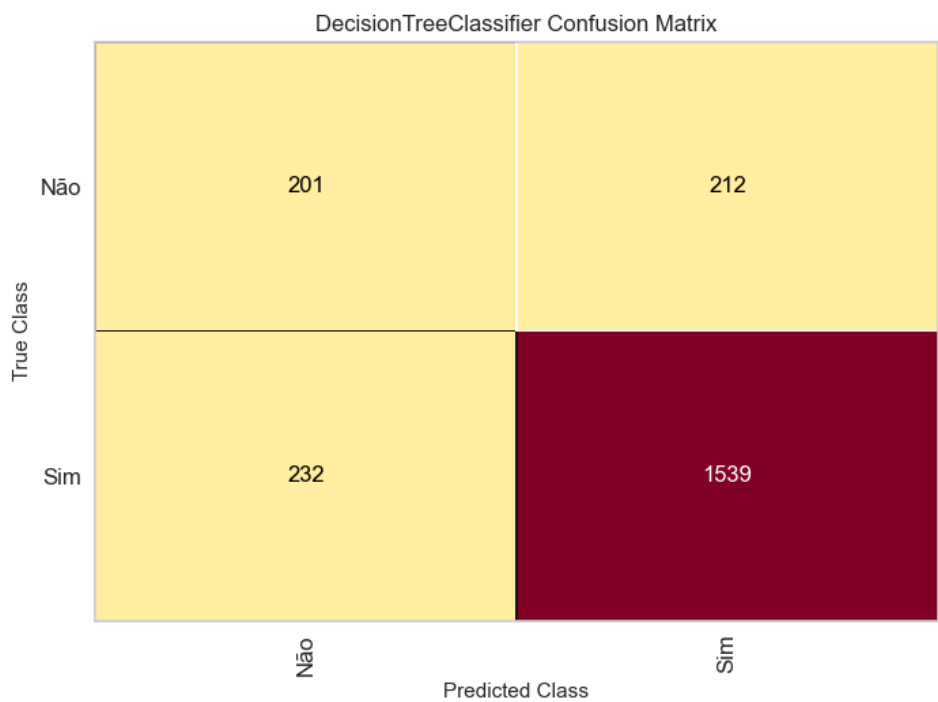


Figura 79 - Matriz de confusão “Decision Tree Classifier (“SMOTE”)”.

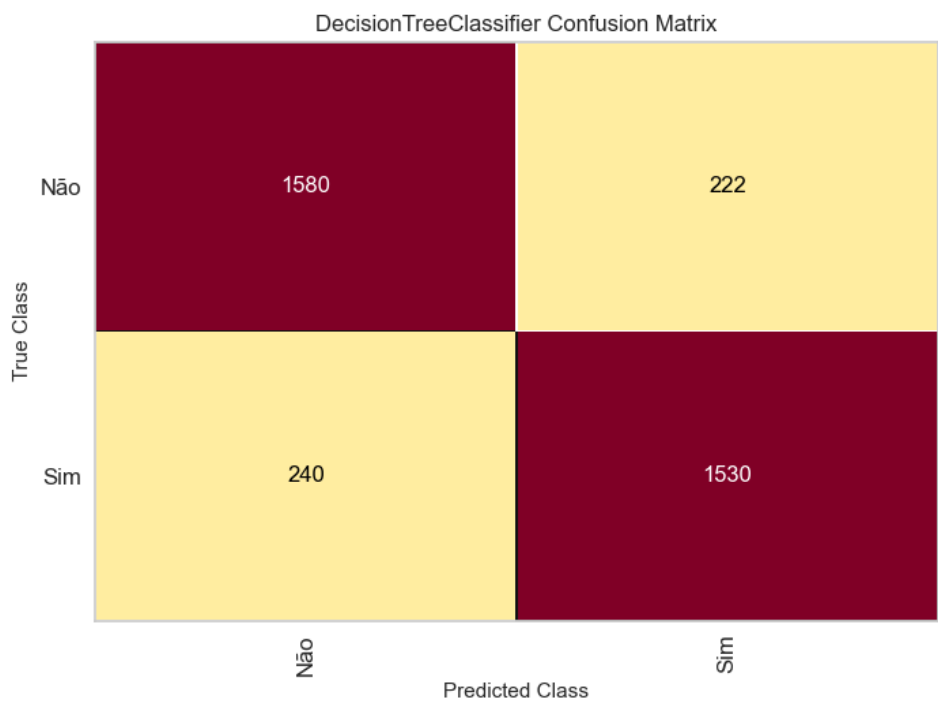
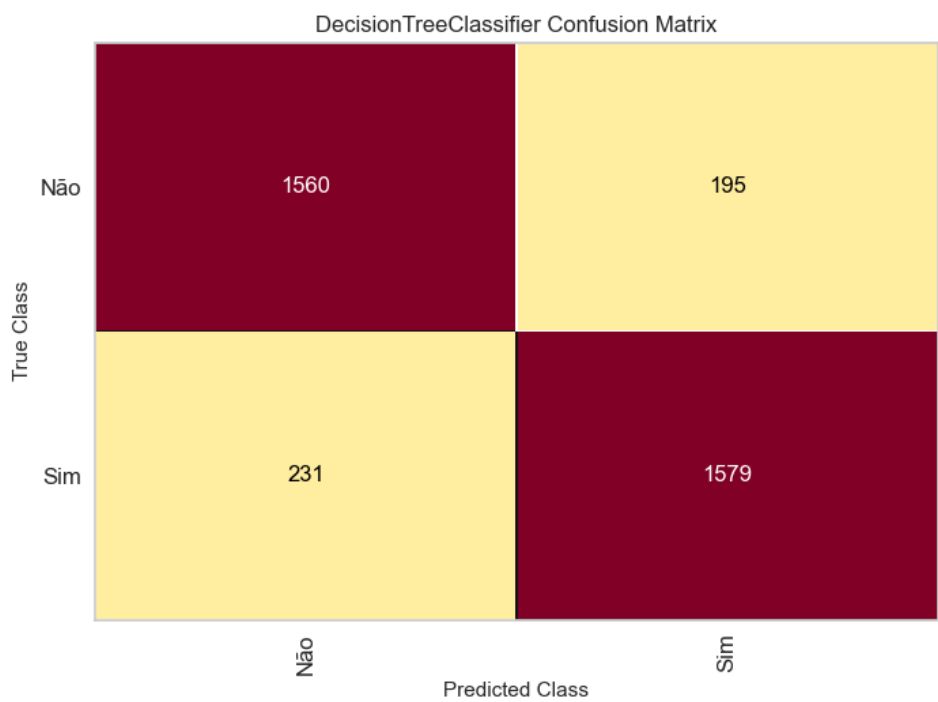


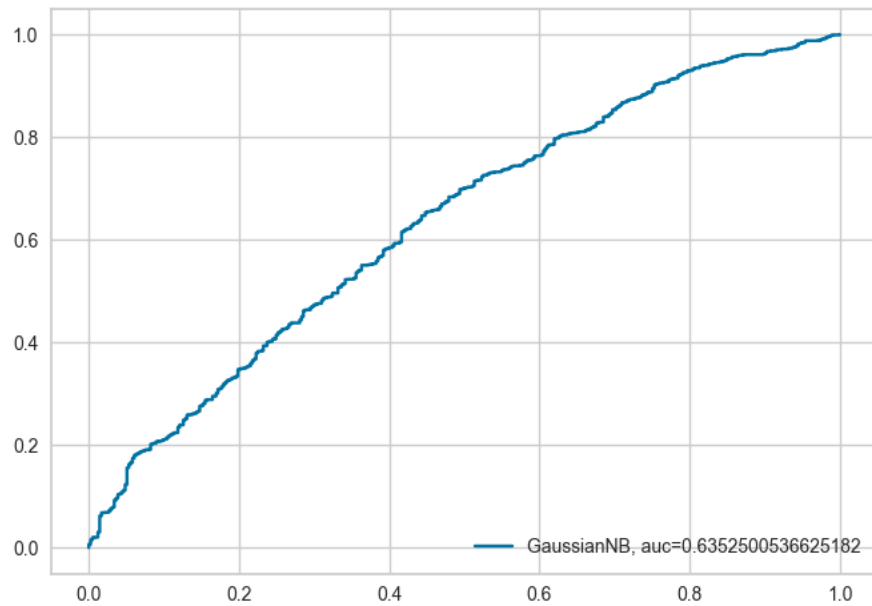
Figura 80 - Matriz de confusão “Decision Tree Classifier (“ADASYN”)”.



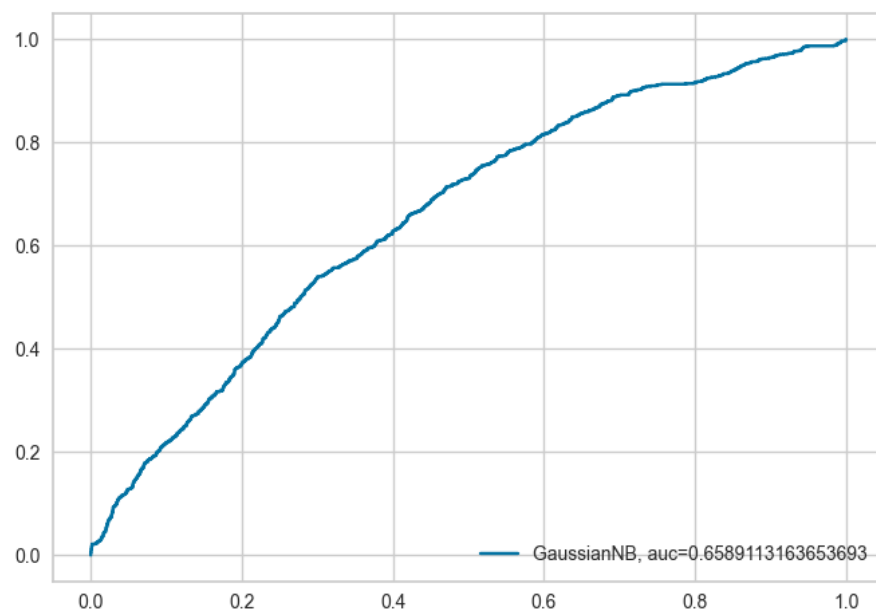


As curvas ROC e AUC do modelo "*Gaussian Naive Bayes*" são apresentadas nas figuras 81, 82, 83.

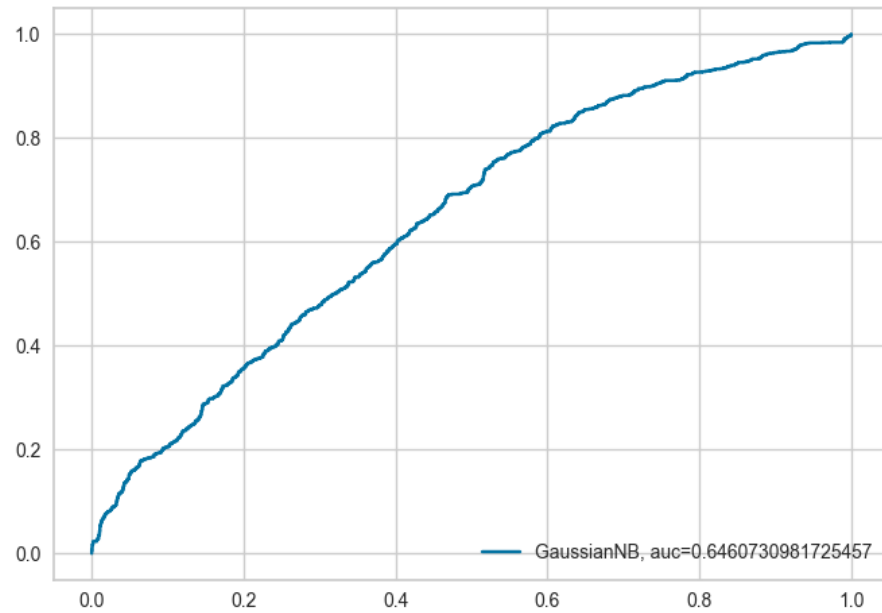
**Figura 81 - Curva ROC e AUC do modelo "*Gaussian Naive Bayes*"(desbalanceado).**



**Figura 82 - Curva ROC e AUC do modelo "*Gaussian Naive Bayes*"("SMOTE").**



**Figura 83 - Curva ROC e AUC do modelo "*Gaussian Naive Bayes*"("ADASYN").**



As curvas ROC e AUC do modelo "*Decision Tree Classifier*" são apresentadas nas figuras 84, 85, 86

**Figura 84 - Curva ROC e AUC do modelo "*Decision Tree Classifier*"(desbalanceado).**

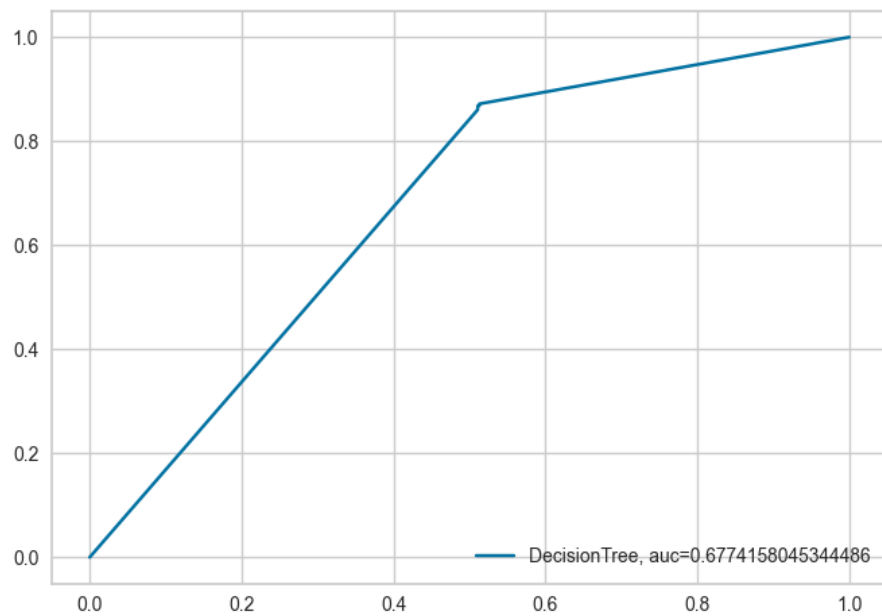


Figura 85 - Curva ROC e AUC do modelo "*Decision Tree Classifier*" ("*SMOTE*").

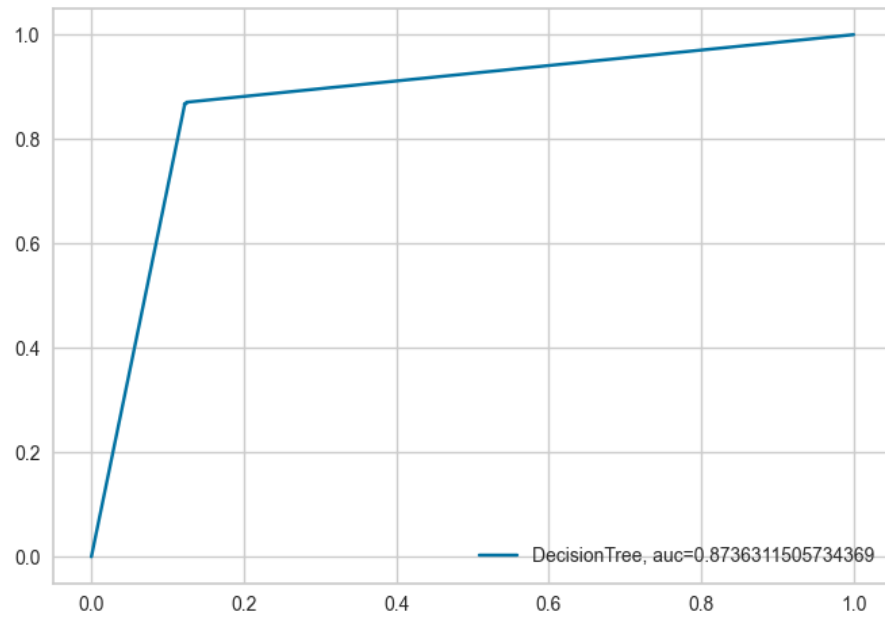
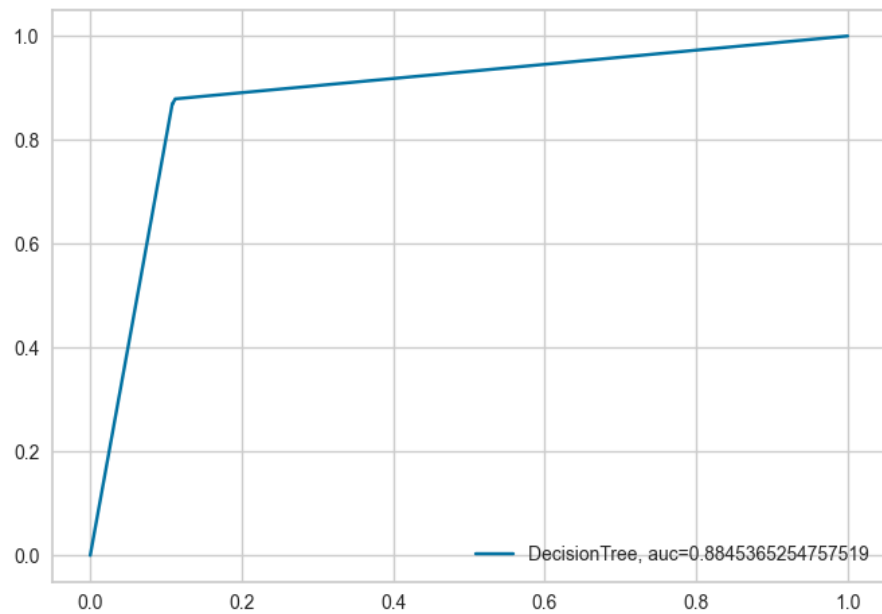


Figura 86 - Curva ROC e AUC do modelo "*Decision Tree Classifier*" ("*ADASYN*").



A partir dos resultados apresentados é possível perceber que o balanceamento de classes dos modelos foi fundamental para melhorar o desempenho de cada um deles. Os modelos sem balanceamento apresentam um enviesamento devido a classe “Sim” ocorrer mais frequentemente.

Comparativamente, o modelo “*Gaussian Naive Bayes*” teve melhor desempenho quando sobreamostrado utilizando a técnica “*SMOTE*”, enquanto o modelo “*Decision Tree Classifier*” teve melhor desempenho quando sobreamostrado utilizando a técnica “*ADASYN*”. No geral, o modelo “*Decision Tree Classifier*” demonstrou um melhor desempenho do que o modelo “*Gaussian Naive Bayes*” no conjunto de dados escolhido. As diferenças de cada modelo amostrado puderam ser verificados nas métricas, nas matrizes de confusão e nas curvas ROC com o valor de AUC.

Alguns dos objetivos da ciência de dados são a melhoria da eficiência operacional e a redução de custos. Na visão da gestão de incidentes, é muito importante que se saiba quando um incidente não será atendido com a qualidade esperada, possibilitando ações antecipadas para garantir mais qualidade de serviço ao cliente final. A classe “Não” possui maior importância para o negócio, uma vez que os casos verdadeiros ou falsos preditos como “Não” necessitarão de atenção e dispendirão um gasto maior de recursos. Falsos positivos farão com que recursos desnecessários sejam despendidos, enquanto falso negativos farão com que esses casos sejam negligenciados. É essencial analisar as características e contrapartidas na utilização de um modelo ou em dados reais.

Por fim, concluiu-se que o modelo “*Decision Tree Classifier*” sobreamostrado com “*ADASYN*” apresentou melhor capacidade de classificação, obtendo o melhor resultado nas métricas para a classe “Não” – precisão (0,87), revocação (0,89) e Pontuação – F1 (0,88). Cabe agora testar o modelo em conjuntos de dados reais para verificar se o modelo está corretamente ajustado sendo capaz de atender aos objetivos do negócio.

## 7. Links

Link para o vídeo: <https://youtu.be/8bN2D-M-OTw>

Link para o repositório:

<https://www.dropbox.com/sh/z8785xy4u8q1oh4/AABsbYW-9Xx7eBeDWID1TewLa?dl=0>

## REFERÊNCIAS

- Agência Nacional de Telecomunicações. **Resolução nº 590**, 2012. Disponível em: <<https://www.anatel.gov.br/legislacao/resolucoes/2012/332-resolucao-590>>.
- Cartlidge, Alison (2007). ***An Introductory Overview of ITIL V3*** 1 ed. [S.l.]: The UK Chapter of the itSMF.
- Parasuraman, A., Zeithaml, V. A., & Berry, L. L. (1988). SERVQUAL: ***A multiple-item scale for measuring consumer perceptions of service quality***.
- David Clifford; Jan van Bon (2008). ***Implementing ISO/IEC 20000 Certification: The Roadmap. ITSM Library***. Van Haren Publishing.
- Telecomunicações Brasileiras S.A., **Institucional**. Disponível em: <<https://www.telebras.com.br/aceso-a-informacao/institucional/>>.
- Weiss, S., M., Kulikowski, C., A., ***Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Networks, Machine Learning and Expert Systems***, Morgan Kaufmann, San Mateo, 1991.
- Confederação Nacional do Transporte. **Pesquisa CNT de Rodovias**, 2019. Disponível em: <<https://pesquisarodovias.cnt.org.br/downloads/ultimaversao/gerencial.pdf>>.
- Saito T, Rehmsmeier M (2015) ***The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets***. PLoS ONE 10(3): e0118432.
- Powers, David M. W. (2011). ***Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation***. Journal of Machine Learning Technologies
- J. Han and M. Kamber, **Data Mining: Concepts and Techniques**. Morgan-Kaufmann Publishers, San Francisco, 2001.
- Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. ***Smote: synthetic minority over-sampling technique***. *Journal of artificial intelligence research*, 16:321–357, 2002.
- He, Haibo, Yang Bai, Edwardo A. Garcia, and Shutao Li. “**ADASYN: Adaptive synthetic sampling approach for imbalanced learning**,” In IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), pp. 1322-1328, 2008.
- KOHAVI, Ron; PROVOST, Foster. ***Glossary of terms. Machine Learning***, v. 30, 1998.

## APÊNDICE

### **Programação/Scripts**

\*Os scripts podem ser encontrados no link do Dropbox do tópico 7.

### **Tabelas**

\*As tabelas podem ser encontradas no link do Dropbox do tópico 7.