

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

Silvio Rodrigues Finotti

**ENSAIOS SOBRE VIABILIDADE DE USO DE MODELOS DE APRENDIZADO DE
MÁQUINA PARA PREVISÃO DE RESULTADO EM ELEIÇÕES MUNICIPAIS**

Belo Horizonte
2021

Silvio Rodrigues Finotti

**ENSAIOS SOBRE VIABILIDADE DE USO DE MODELOS DE APRENDIZADO DE
MÁQUINA PARA PREVISÃO DE RESULTADO EM ELEIÇÕES MUNICIPAIS**

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.....	5
1.1. Contextualização.....	5
1.2. O problema proposto.....	5
2. Coleta de Dados.....	7
2.1. Dataset de informações de candidatos.....	7
2.2. Dataset de informações de <i>coligações entre os partidos</i>.....	11
2.3. Dataset de <i>bens informados pelos candidatos</i>.....	12
2.4. Dataset de <i>quantitativo de vagas</i>.....	13
2.5. Dataset de <i>receitas de campanha usadas pelos candidatos</i>.....	14
2.6. Dataset de <i>despesas de campanha assumidas pelos candidatos</i>.....	16
2.7. Dataset de resultados de eleições anteriores.....	18
2.8. Dataset de códigos de municípios padrão TSE.....	20
2.9. Dataset de códigos de municípios, distritos e subdistritos padrão IBGE...21	
3. Processamento/Tratamento de Dados.....	22
3.1. Leitura da base inicial de candidaturas para eleições de 2008.....	22
3.1.1. Análise inconsistências iniciais.....	24
3.1.2. Tratamento de <i>missing values</i>.....	26
3.2. Enriquecimento do dataset.....	33
3.2.1. Base de dados de <i>coligações partidárias</i>.....	34
3.2.2. Base de dados de <i>bens declarados por candidatos</i>.....	37
3.2.3. Base de dados de <i>quantidade de vagas</i>.....	38
3.2.4. Base de dados de <i>receitas do candidato</i>.....	40
3.2.5. Base de dados de <i>despesas do candidato</i>.....	43
3.2.6. Base de dados de resultados da eleição anterior.....	44
3.2.7. Tratamento de <i>Nan</i> nas colunas adicionadas ao dataset principal.....	45
3.3. Outros ajustes no dataset.....	46
3.3.1. Exclusão de colunas desnecessárias.....	46
3.3.2. Exclusão de registros de candidaturas não confirmadas.....	47
3.3.3. Criação da coluna “<i>target</i>”.....	48
3.3.4. Exclusão de colunas redundantes ou sem utilidade preditiva.....	49
4. Análise e Exploração dos Dados.....	50
4.1. Candidaturas por municípios.....	50

4.2. Candidatos por vagas.....	50
4.3. Candidaturas por partidos:.....	52
4.4. Ocupação dos candidatos.....	52
4.5. Distribuição da idade dos candidatos.....	53
4.6. Sexo dos candidatos.....	54
4.7. Grau de instrução dos candidatos.....	54
4.8. Nacionalidade, Estado e Município de nascimento.....	55
4.9. Coligações partidárias.....	55
4.10. Bens declarados pelos candidatos.....	57
4.11. Receitas e Despesas de campanha dos candidatos:.....	58
4.12. Correlação entre colunas preditoras.....	60
5. Criação de Modelos de Machine Learning.....	61
5.1. Arvore de Decisões para a base de todo o estado de SP.....	65
5.2. Arvore de Decisões para dataset do município de São Paulo.....	73
5.3. Outros algoritmos de Aprendizagem de Máquina para classificação.....	80
5.4. Testando os modelos em situação “de produção”.....	90
5.4.1. Interpretação dos Resultados.....	93
6. Apresentação dos Resultados.....	95
6.1. Observações finais e próximos passos.....	100
7. Links.....	102
REFERÊNCIAS.....	103

1. Introdução

1.1. Contextualização

Dada a possibilidade de escolha do tema para o trabalho de conclusão do curso de “Especialização em Ciência de Dados e Big Data”, o primeiro desafio foi encontrar um assunto que tivesse informações abertas, de livre uso, disponíveis e que atendessem aos requisitos e restrições técnicas exigidos pela PUC – Minas. Além disso, nos esforçamos para que o tema escolhido tivesse, também, algum grau de interesse geral e que trouxesse os desafios pertinentes ao que foi estudado no decorrer da Especialização. Assim, o assunto escolhido recaiu sobre eleições municipais, tendo como fonte principal os dados disponibilizados pelo Tribunal Superior Eleitoral – TSE.

A previsibilidade eleitoral é um assunto bastante complexo, polêmico e que tem, cada vez mais, despertado a atenção do público em geral. O modelo mais utilizado para estudo das previsibilidades em eleições é através de questionário aplicado a uma amostra da população – comumente conhecida como pesquisas de intenção de voto. Exemplos recentes de previsões não confirmadas – como o plebiscito para confirmar ou não a permanência da Inglaterra na União Europeia e também as eleições para presidência dos Estados Unidos, ambas em 2016, trouxeram ainda mais polêmica sobre a questão.

Assim, o objetivo principal do presente trabalho é coletar dados referentes a candidatos ao cargo de vereador e seus respectivos partidos e aplicar métodos de aprendizagem de máquina sobre estes dados para avaliar como eles se comportam na previsão de um candidato ser ou não eleito.

1.2. O problema proposto

A proposta deste trabalho é a coleta, tratamento e análise exploratória de dados referentes a candidatos a vereador nas eleições municipais. Uma vez coletados os dados, de fontes variáveis, e feitos os devidos tratamentos e análises

exploratórias, aplicaremos sobre eles modelos de aprendizado de máquina com o objetivo principal de tentar identificar os candidatos com maiores chances de serem eleitos, ou seja, usar os dados coletados para fins de previsão das chances do candidato ser eleito.

Para delimitar o escopo de nossa análise, utilizamos a técnica dos 5 W's:

(What?) O que iremos analisar?

O objetivo principal é verificar como se comportam e a viabilidade do uso de modelos de aprendizado de máquina supervisionado para fins de previsão de eleição de candidatos a vereador. Para chegar a este ponto, precisaremos montar uma base de candidatos com o máximo de informações possível, além de preparar, tratar e transformar estes dados para que possam ser utilizados para tentar identificar padrões que indiquem os candidatos com maiores chances de se elegerem nas eleições municipais para vereador

(Why?) Por que esse problema é importante?

Por se tratar de um assunto de interesse geral, entendemos que a coleta, junção e exploração de dados sobre candidatos e uma análise sobre como se comportam alguns modelos de aprendizado de máquina em previsões eleitorais parece bastante válido.

(Who?) De quem são os dados analisados?

A fonte principal dos dados utilizados neste trabalho foi o Tribunal Superior Eleitoral - TSE. Todos os dados aqui utilizados estão livremente disponibilizados no repositório de dados eleitorais no portal do TSE na internet. Utilizamos vários *datasets* do TSE para construirmos a base de informações de candidatos que posteriormente foi utilizada nos modelos de aprendizagem de máquina. De forma bastante sucinta, os principais *datasets* utilizados foram:

- base de dados de **candidaturas** (base principal)
- base de dados de **coligações partidárias**
- base de dados de **bens de candidatos**

- base de dados de ***quantitativo de vagas***
- base de dados de ***prestaçao de contas – receitas***
- base de dados de ***prestaçao de contas – despesas***
- base de dados de ***resultados – votação nominal por município e zona***
- base de dados de ***municípios do TSE***
- base de dados de ***municípios, distritos e subdistritos do IBGE***

(Where?) Trata dos aspectos geográficos e logísticos de sua análise.

De forma mais ampla, os ensaios aqui realizados podem se aplicar a todo o território brasileiro. Entretanto para fins específicos vamos tratar do estado de São Paulo como um todo e especificamente do município de São Paulo, durante as análises.

(When?) Qual o período está sendo analisado?

O foco será nos dados das eleições municipais do ano de 2008. Entretanto para nossa análise, acabamos usando também dados do período anterior (2004) para poder enriquecer os dados a serem analisados pelos algorítimos de aprendizagem de máquina. Também utilizamos dados de 2012, para verificar como se comportam os algoritmos treinados com dados de 2008, no ano de 2012.

2. Coleta de Dados

2.1. Dataset de informações de candidatos

A base de dados inicial deste trabalho é a que apresenta informações específicas para cada candidato que concorreu às eleições municipais no ano de 2008, no estado de São Paulo. Esta base foi obtida do repositório de dados do TSE¹, no tema *candidatos, base candidatos, em 08/06/2021*:

¹ https://cdn.tse.jus.br/estatistica/sead/odsele/consulta_cand/consulta_cand_2008.zip

The screenshot shows the 'Repositório de dados eleitorais' website. The main navigation menu on the left includes 'Candidatos', 'Comparecimento e Abstenção', 'Eleitorado', 'Partidos', 'Pesquisas eleitorais', 'Prestação de contas eleitorais', 'Prestação de contas partidárias', 'Procesual', and 'Resultados'. The 'Candidatos' option is highlighted. The right side of the page displays a grid of years from 1945 to 2018, with '2008' selected. Below the grid, a breadcrumb navigation shows 'página inicial > candidatos > 2008'. A sub-menu for the year 2008 lists 'Candidatos (formato ZIP)', 'Bens de candidatos (formato ZIP)', 'Legendas (formato ZIP)', and 'Vagas (formato ZIP)'. A red arrow points to the 'Candidatos (formato ZIP)' link.

Apresentamos, a seguir, uma breve descrição das informações originalmente constantes desta base de dados de candidatos:

Nome da coluna	Descrição da coluna	Tipo
DT_GERACAO	Data da extração dos dados para geração do arquivo.	object
HH_GERACAO	Hora da extração dos dados para geração do arquivo com base no horário de Brasília.	object
ANO_ELEICAO	Ano de referência da eleição para geração do arquivo. Observação: Para eleições suplementares o ano de referência da eleição é o da eleição ordinária correspondente.	int64
NR_TURNO	Número do turno da eleição.	int64
DS_ELEICAO	Descrição da eleição.	object
SG_UF	Sigla da Unidade da Federação em que ocorreu a eleição.	object
SG_UE	Sigla da Unidade Eleitoral em que o candidato concorre na eleição. A Unidade Eleitoral representa a Unidade da Federação ou o Município em que o candidato concorre na eleição e é relacionada à abrangência territorial desta candidatura. Em caso de abrangência Municipal (cargos de Prefeito, Vice-Prefeito e Vereador) é o código de identificação do município da candidatura.	int64
NM_UE	Nome da Unidade Eleitoral do candidato. Em caso de abrangência nacional é igual à 'Brasil'. Em caso de abrangência estadual é o nome da UF em que o candidato concorre. Em caso de abrangência municipal é o nome do município em que o candidato concorre.	object

CD_CARGO	Código do cargo ao qual o candidato concorre na eleição.	int64
DS_CARGO	Cargo ao qual o candidato concorre na eleição.	object
NM_CANDIDATO	Nome completo do candidato.	object
SQ_CANDIDATO	Número sequencial do candidato, gerado internamente pelos sistemas eleitorais para cada eleição. Observação: não é o número de campanha do candidato.	int64
NR_CANDIDATO	Número do candidato na urna.	int64
NR CPF_CANDIDATO	Número do CPF do candidato.	string
NM_URNA_CANDIDATO	Nome do candidato que aparece na urna.	object
CD_SITUACAO_CANDIDATURA	Código da situação do registro de candidatura do candidato.	int64
DS_SITUACAO_CANDIDATURA	Situação do registro da candidatura do candidato. Pode assumir os valores: Apto (candidato apto para ir para urna), Inapto (candidato inapto para ir para urna) e Cadastrado (registro de candidatura realizado, mas ainda não julgado). A situação inicial de uma candidatura é 'Cadastrado'. Após julgamento pela Justiça Eleitoral, a situação é alterada para 'Apto' ou 'Inapto' com relação ao encaminhamento da candidatura para a urna.	object
NR_PARTIDO	Número do partido de origem do candidato. Mesmo que o candidato participe de uma coligação, este número é o número do seu partido de origem.	int64
SG_PARTIDO	Sigla do partido de origem do candidato.	object
NM_PARTIDO	Nome do partido de origem do candidato.	object
SQ_LEGENDA	Sequencial da coligação da qual o candidato pertence, gerado pela Justiça Eleitoral.	int64
SG_LEGENDA	Sigla da coligação da qual o candidato pertence.	object
COMPOSICAO_LEGENDA	Composição da coligação da qual o candidato pertence. Observação: Coligação é a união de dois ou mais partidos a fim de disputarem eleições. A informação da coligação no arquivo está composta pela concatenação das siglas dos partidos intercaladas com o símbolo /.	object
NM_LEGENDA	Nome da coligação da qual o candidato pertence.	object
CD_OCUPACAO	Código da ocupação do candidato.	int64
DS_OCUPACAO	Ocupação do candidato.	object

DT_NASCIMENTO	Data de nascimento do candidato.	object
NR_TITULO_ELEITORAL_CANDIDATO	Número do título eleitoral do candidato.	int64
IDADE_DATA_ELEICAO	Idade do candidato na data da eleição. A idade é calculada com base na data da eleição para o cargo e unidade eleitoral constantes no arquivo de vagas.	int64
CD_GENERO	Código do gênero do candidato.	int64
DS_GENERO	Gênero do candidato.	object
CD_GRAU_INSTRUCAO	Código do grau de instrução do candidato.	int64
DS_GRAU_INSTRUCAO	Grau de instrução do candidato.	object
CD_ESTADO_CIVIL	Código do estado civil do candidato.	int64
DS_ESTADO_CIVIL	Estado civil do candidato.	object
CD_NACIONALIDADE	Código da nacionalidade do candidato.	int64
DS_NACIONALIDADE	Nacionalidade do candidato.	object
SG_UF_NASCIMENTO	Sigla da Unidade da Federação de nascimento do candidato.	object
CD_MUNICIPIO_NASCIMENTO	Código de identificação do município de nascimento do candidato.	int64
NM_MUNICIPIO_NASCIMENTO	Nome do município de nascimento do candidato.	object
VR_DESPESA_MAX_CAMPANHA	Valor máximo, em reais, de despesas de campanha declarada pelo partido para aquele candidato.	int64
CD_SIT_TOT_TURNO	Código da situação de totalização do candidato, naquele turno da eleição, após a totalização dos votos.	int64
DS_SIT_TOT_TURNO	Situação de totalização do candidato, naquele turno da eleição, após a totalização dos votos.	object

Com o objetivo de enriquecer o *dataset* inicial de candidatos, também utilizamos outras bases de informações. A seguir listamos estas bases com um resumo do conteúdo das mesmas e como elas se relacionam com a nossa base principal.

2.2. Dataset de informações de coligações entre os partidos²

Nesta base, coletada em 08/06/2021, estão detalhadas as diversas coligações feitas entre os partidos nas eleições do ano de 2008. Apesar de existirem colunas referentes a coligação / legendas na própria base de dados de candidatos, no ano de 2008 estas informações estão imprestáveis, como será visto no item 3 deste trabalho.

As informações desta base nos parecem muito relevantes, haja vista que o cálculo do coeficiente eleitoral nas eleições proporcionais para vereador é influenciado diretamente pelas coligações partidárias³ em 2008. Esta base contém 24.393 linhas e suas colunas estão descritas a seguir, estando destacadas em amarelo, aquelas que serviram para fazer o relacionamento com o dataset principal:

Nome da coluna	Descrição da coluna	Tipo
DT_GERACAO	Data da extração dos dados para geração do arquivo.	object
HH_GERACAO	Hora da extração dos dados para geração do arquivo com base no horário de Brasília.	object
ANO_ELEICAO	Ano de referência da eleição para geração do arquivo. Observação: Para eleições suplementares o ano de referência da eleição é o da eleição ordinária correspondente.	int64
NR_TURNO	Número do turno da eleição.	int64
DS_ELEICAO	Descrição da eleição.	object
SG_UF	Sigla da Unidade da Federação em que ocorreu a eleição.	object
SG_UE	Sigla da Unidade Eleitoral em que o candidato concorre na eleição. A Unidade Eleitoral representa a Unidade da Federação ou o Município em que o candidato concorre na eleição e é relacionada à abrangência territorial desta candidatura. Em caso de abrangência Municipal (cargos de Prefeito, Vice-Prefeito e Vereador) é o código de identificação do município da candidatura.	int64
DS_UE	Nome da Unidade Eleitoral do candidato. Em caso de abrangência nacional é igual à 'Brasil'. Em caso de abrangência estadual é o nome da UF em que o candidato concorre. Em caso de abrangência municipal é o nome do município em que o candidato concorre.	object
CD_CARGO	Código do cargo ao qual o candidato concorre na eleição.	int64

² Link para a base coligações: https://cdn.tse.jus.br/estatistica/sead/odsele/consulta_leendas/consulta_legendas_2008.zip

³ A partir das eleições de 2020 o forma de cálculo do coeficiente eleitoral foi alterado.

DS_CARGO	Cargo ao qual o candidato concorre na eleição.	object
TP_AGREMIACAO	Tipo de agremiação do partido. Pode assumir os valores PARTIDO ISOLADO ou COLIGACAO. Informa se o partido faz, ou não, parte de uma coligação.	object
NR_PARTIDO	Número do partido de origem do candidato. Mesmo que o candidato participe de uma coligação, este número é o número do seu partido de origem.	int64
SG_PARTIDO	Sigla do partido de origem do candidato.	object
NM_PARTIDO	Nome do partido de origem do candidato.	object
SG_COLIGACAO	Sigla da coligação da qual o candidato pertence.	object
NM_COLIGACAO	Nome da coligação da qual o candidato pertence.	object
COMP_COLIGACAO	Composição da coligação da qual o candidato pertence. Observação: Coligação é a união de dois ou mais partidos a fim de disputarem eleições. A informação da coligação no arquivo está composta pela concatenação das siglas dos partidos intercaladas com o símbolo /.	object
SQ_COLIGACAO	Sequencial da coligação da qual o candidato pertence, gerado pela Justiça Eleitoral.	int64

2.3. Dataset de bens informados pelos candidatos⁴

Trata-se de informação declarada pelo próprio candidato referente aos bens que o mesmo dispõe. Ainda que seja uma informação sem checagem posterior, nos parece ser útil para mensurar o patrimônio do candidato.

A base de bens de candidatos para o estado de São Paulo referente ao ano de 2008 contém, originalmente, 137.062 registros. A seguir, apresentamos resumo da descrição das colunas e respectivos tipos. As colunas que serviram de base para o relacionamento com o *dataset* principal estão destacadas em amarelo.

Nome da coluna	Descrição da coluna	Tipo
DT_GERACAO	Data da extração dos dados para geração do arquivo.	object

⁴ A base bens dos candidatos foi coletado em 08/06/2021, no endereço : https://cdn.tse.-jus.br/estatistica/sead/odsele/bem_candidato/bem_candidato_2008.zip

HH_GERACAO	Hora da extração dos dados para geração do arquivo com base no horário de Brasília.	object
ANO_ELEICAO	Ano de referência da eleição para geração do arquivo. Observação: Para eleições suplementares o ano de referência da eleição é o da eleição ordinária correspondente.	string
DS_ELEICAO	Descrição da eleição.	object
SG_UF	Sigla da Unidade da Federação em que ocorreu a eleição.	object
SQ_CANDIDATO	Número sequencial do candidato, gerado internamente pelos sistemas eleitorais para cada eleição. Observação: não é o número de campanha do candidato.	int64
CD_TIPO_BEM_CANDIDATO	Código do tipo do bem do candidato	string
DS_TIPO_BEM_CANDIDATO	Descrição do tipo do bem do candidato	object
DS_BEM_CANDIDATO	Detalhe do bem do candidato	object
VR_BEM_CANDIDATO	Valor declarado do bem do candidatoa última atualização do registro do bem	float64
DT_ULTIMA_ATUALIZACAO	Data da última atualização do registro do bem	object
HH_ULTIMA_ATUALIZACAO	Hora da última atualização do registro do bem	object

2.4. Dataset de quantitativo de vagas⁵

Esta base tem o quantitativo de vagas disponíveis para cada cargo, por município, no ano de 2008, para o estado de São Paulo. Não é uma informação que será usada diretamente nos algoritmos de predição, porém servirá como limitador para a quantidade de alvos/ *targets* positivos.

Esta base tem 1.297 linhas. A descrição de suas colunas estão relacionadas a seguir, com destaque em amarelo para aquelas que foram usadas no relacionamento com o dataset principal.

Nome da coluna	Descrição da coluna	Tipo
DT_GERACAO	Data da extração dos dados para geração do arquivo.	object

⁵ Link para a base quantidade de vagas, coletado em 08/06/2021: https://cdn.tse.jus.br/estatistica/sead/odsele/consulta_vagas/consulta_vagas_2008.zip

HH_GERACAO	Hora da extração dos dados para geração do arquivo com base no horário de Brasília.	object
ANO_ELEICAO	Ano de referência da eleição para geração do arquivo. Observação: Para eleições supplementares o ano de referência da eleição é o da eleição ordinária correspondente.	int64
DS_ELEICAO	Descrição da eleição.	object
SG_UF	Sigla da Unidade da Federação em que ocorreu a eleição.	object
SG_UE	Sigla da Unidade Eleitoral em que o candidato concorre na eleição. A Unidade Eleitoral representa a Unidade da Federação ou o Município em que o candidato concorre na eleição e é relacionada à abrangência territorial desta candidatura. Em caso de abrangência Municipal (cargos de Prefeito, Vice-Prefeito e Vereador) é o código de identificação do município da candidatura.	int64
NM_UE	Nome da Unidade Eleitoral em que o candidato concorre na eleição.	object
CD_CARGO	Código do cargo ao qual o candidato concorre na eleição.	int64
DS_CARGO	Cargo ao qual o candidato concorre na eleição.	object
QT_VAGAS	Quantidade de vagas para aquele cargo naquela unidade eleitoral	int64

2.5. Dataset de receitas de campanha usadas pelos candidatos⁶

Do tema “prestaçao de contas” do TSE, usamos o *dataset* de receitas de campanha declaradas pelos candidatos. Esta base tem um total de 504.674 linhas referentes as receitas de candidatos as eleições de 2008 no estado de São Paulo.

Acreditamos que esta uma fonte de informações bastante relevante para identificar os candidatos que mais angariaram recursos para suas campanhas, bem como a origem desses recursos.

A descrição das colunas da base de receitas e a marcação (em amarelo) daquelas que foram usadas no relacionamento com o *dataset* principal são apresentados a seguir:

Nome da coluna	Descrição da coluna	Tipo
-----------------------	----------------------------	-------------

⁶ Link para a base de receitas, coletada em 08/06/2021: https://cdn.tse.jus.br/estatistica/sead/odsele/prestacao_contas/prestacao_contas_2008.zip

SQ_CANDIDATO	Número sequencial do candidato, gerado internamente pelos sistemas eleitorais para cada eleição. Observação: não é o número de campanha do candidato.	int64
NM_CANDIDATO	Nome completo do candidato.	object
SEXO	Descrição do gênero do candidato.	object
DS_CARGO	Cargo ao qual o candidato concorre na eleição.	object
CD_CARGO	Código do cargo ao qual o candidato concorre na eleição.	int64
NR_CANDIDATO	Número do candidato na urna.	int64
SG_UE_SUPERIOR	Sigla da Unidade da Federação a que o município está vinculado.	object
NM_UE	Nome da Unidade Eleitoral em que o candidato concorre na eleição.	object
SG_UE	Sigla da Unidade Eleitoral em que o candidato concorre na eleição. Em caso de abrangência Municipal (cargos de Prefeito, Vice-Prefeito e Vereador) é o código de identificação do município da candidatura.	string
DS_NR_TITULO_ELEITOR	Número do título eleitoral do candidato.	string
CD_NUM_CPF	Número do CPF	string
CD_NUM_CNPJ	Número do CNPJ	string
NR_PARTIDO	Número do partido de origem do candidato. Mesmo que o candidato participe de uma coligação, este número é o número do seu partido de origem.	int64
SG_PARTIDO	Sigla do partido de origem do candidato.	object
VR_RECEITA	Valor da receita	float64
DT_RECEITA	Data do recebimento da receita	object
DS_TITULO	Descrição da origem do recurso, podendo ser “Recursos Pró-prios”, “Recursos de Pessoas Físicas”, “Doações pela internet”, “Recursos de Outros Candidatos”, “Recursos de Partidos Políticos”, “Recursos de Origem Não Identificada”, “Comercialização de bens ou realização de eventos”, “Rendimentos de aplicações financeiras”	object
CD_ORIGEM_RECEITA	Código da origem do recurso	int64
DS_ESP_RECURSO	Espécie do recurso, podendo assumir os valores “Cheque”, “Em Espécie”, “Transferência Eletrônica”, “Depósito em Espécie”, “Cartão de Crédito”, “Cartão de Débito”, “Boleto de Cobrança”, “Estimado” e “Outros Títulos de Crédito”.	object

CD_ESP_RECURSO	Código da espécie do recurso	int64
NM_DOADOR	Nome do doador declarado à Justiça Eleitora	object
CD_CPF_CNPJ_DOADOR	Número do CPF/CNPJ do doador declarado à Justiça Eleitoral	string
SG_UE_SUPERIOR_1	Sigla da Unidade da Federação a qual o doador está vinculado	object
NM_UE_1	Nome da Unidade Eleitoral em que o doador está vinculado.	object
SG_UE_1	Sigla da Unidade Eleitoral (município) a qual o doador está vinculado	string
SITUACAOCADASTRAL	Situação cadastral do administrador dos recursos	object
NM_ADM	Nome do administrador dos recursos	object
CD_CPF_ADM	CPF do administrador dos recursos	string

2.6. Dataset de despesas de campanha assumidas pelos candidatos⁷

Do tema “prestação de contas”, usamos também o *dataset* de despesas de campanha declaradas pelos candidatos. Esta base tem um total de 945.412 linhas referentes a despesas de candidatos às eleições de 2008 no estado de São Paulo.

Acreditamos que esta uma fonte de informações bastante relevante para identificar os candidatos que mais “gastaram” na campanha.

A descrição de suas colunas e a marcação (em amarelo) daquelas que foram usadas no relacionamento com o *dataset* principal são apresentadas a seguir:

Nome da coluna	Descrição da coluna	Tipo
SQ_CANDIDATO	Número sequencial do candidato, gerado internamente pelos sistemas eleitorais para cada eleição. Observação: não é o número de campanha do candidato.	int64
NM_CANDIDATO	Nome completo do candidato.	object
DS_CARGO	Cargo ao qual o candidato concorre na eleição.	object
CD_CARGO	Código do cargo ao qual o candidato concorre na eleição.	int64

⁷ Link para a base de despesas, coletada em 08/06/2021: https://cdn.tse.jus.br/estatistica/sead/odsele/prestacao_contas/prestacao_contas_2008.zip

NR_CANDIDATO	Número do candidato na urna.	int64
SG_UE_SUPERIOR	Sigla da Unidade da Federação a que o município está vinculado.	object
NM_UE	Nome da Unidade Eleitoral em que o candidato concorre na eleição.	object
SG_UE	Sigla da Unidade Eleitoral em que o candidato concorre na eleição. Em caso de abrangência Municipal (cargos de Prefeito, Vice-Prefeito e Vereador) é o código de identificação do município da candidatura.	int64
CD_NUM_CNPJ		string
NR_PARTIDO	Número do partido de origem do candidato. Mesmo que o candidato participe de uma coligação, este número é o número do seu partido de origem.	int64
SG_PARTIDO	Sigla do partido de origem do candidato.	object
VR_DESPESA	Valor da despesa	float64
DT_DESPESA	Data da despesa	object
DS_TITULO	<p>Tipo da despesa, podendo ser "Despesas com pessoal", "Encargos sociais", "Impostos, contribuições e taxas", "Locação/cessão de bens imóveis", "Despesas com transporte ou deslocamento", "Publicidade por carros de som", "Locação/cessão de bens móveis (exceto veículos)", "Correspondências e despesas postais", "Materiais de expediente", "Combustíveis e lubrificantes", "Publicidade por placas, estandartes e faixas", "Serviços prestados por terceiros", "Serviços próprios prestados por terceiros", "Publicidade por jornais e revistas", "Publicidade por materiais impressos", "Alimentação", "Água", "Energia elétrica", "Comícios", "Pesquisas ou testes eleitorais", "Eventos de promoção da candidatura", "Encargos financeiros, taxas bancárias e/ou op. cartão de crédito", "Produção de programas de rádio, televisão ou vídeo", "Multas eleitorais", "Doações financeiras a outros candidatos/partidos", "Criação e inclusão de páginas na internet", "Diversas a especificar", "Aquisição/Doação de bens móveis ou imóveis", "Publicidade por telemarketing", "Telefone", "Produção de jingles, vinhetas e slogans", "Pré-instalação física de comitês de campanha", "Cessão ou locação de veículos", "Baixa de recursos estimáveis em dinheiro", "Atividades de militância e mobilização de rua"</p>	object
CD_TITULO	Código do tipo de despesa	int64
DS_ESP_RECURSO	Descrição da espécie do recurso	object

CD_ESP_RECURSO	Código da espécie do recurso	int64
DS_NR_DOCUMENTO	Número do documento de despesa	object
DS_TIPO_DOCUMENTO	Descrição do tipo do documento de despesa	object
CD_TIPO_DOCUMENTO	Código do tipo do documento de despesa	string
NM_FORNECEDOR	Nome do fornecedor	object
CD_CPF_CNPJ_FORNECEDOR	CPF/CNPJ do fornecedor	string
SG_UE_SUPERIOR_1	Sigla da Unidade da Federação a qual o fornecedor está vinculado	object
NM_UE_1	Nome da Unidade Eleitoral em que o fornecedor está vinculado.	object
SG_UE_1	Sigla da Unidade Eleitoral (município) a qual o fornecedor vinculado	string
NO_UE		object
SITUACAO_CADASTRAL	Situação cadastral do administrador dos recursos	object
NM_ADM	Nome do administrador dos recursos	object
CD_CPF_ADM	CPF do administrador dos recursos	string

2.7. Dataset de resultados de eleições anteriores⁸

Trata-se de informação referente à apuração dos resultados das eleições municipais anteriores. No caso específico, como nosso foco são as eleições de 2008, buscamos aqui o resultado referente às eleições municipais de 2004.

O objetivo é agregar a nossa base principal informações de como o partido se saiu nas eleições anteriores, informação que acreditamos ser de grande importância para enriquecer nosso *dataset*. A base que utilizamos foi a que apresenta o resultado da votação de cada candidato por município e zona eleitoral. Esta base tem um total de 132.802 linhas. A descrição de suas colunas e a marcação (em amarelo) daquelas que usamos para fazer o relacionamento com o *dataset* principal estão a seguir:

⁸ Link para a base de resultados da eleição anterior, coletada em 08/06/2021: https://cdn.tse.jus.br/estatistica/sead/odsele/votacao_candidato_munzona/votacao_candidato_munzona_2004.zip

Nome da coluna	Descrição da coluna	Tipo
DT_GERACAO	Data da extração dos dados para geração do arquivo.	object
HH_GERACAO	Hora da extração dos dados para geração do arquivo com base no horário de Brasília.	object
ANO_ELEICAO	Ano de referência da eleição para geração do arquivo.	int64
NR_TURNO	Número do turno da eleição.	int64
DS_ELEICAO	Descrição da eleição.	object
SG_UF	Sigla da Unidade da Federação em que ocorreu a eleição.	object
SG_UE	Sigla da Unidade Eleitoral em que o candidato concorre na eleição.	int64
CD_MUN	Código do município da zona eleitoral	int64
NM_UE	Nome da Unidade Eleitoral	object
NR_ZONA	Número da Zona Eleitoral	int64
CD_CARGO	Código do cargo ao qual o candidato concorre na eleição.	int64
NR_CAND	Número do candidato na urna.	int64
SQ_CANDIDATO	Número sequencial do candidato, gerado internamente pelos sistemas eleitorais para cada eleição. Observação: não é o número de campanha do candidato.	int64
NM_CAND	Nome completo do candidato.	object
NM_URNA_CAND	Nome do candidato que aparece na urna.	object
DS_CARGO	Descrição do cargo a que o candidato concorre	object
CD_SIT_CAND_SUP	Código da situação de totalização do candidato superior naquele turno. Esta variável deve ser considerada apenas quando o cargo do candidato for vice ou suplente.	int64
DS_SIT_CAND_SUP	Descrição da situação de totalização do candidato superior naquele turno.	object
CD_SIT_REG_CAND	Código da situação do registro de candidatura do candidato	float64
DS_SIT_REF_CAND	Descrição da situação de registro de candidatura do candidato	object
CD_SIT_T_CAND	Código da situação de totalização do candidato naquele turno	int64
DS_SIT_TOT_TURNO	Descrição da situação de totalização do candidato naquele turno	object

NR_PARTIDO	Número do partido	int64
SG_PARTIDO	Sigla do partido	object
NM_PARTIDO	Nome do partido	object
SQ_COLIGACAO	Código sequencial da legenda gerado pela Justiça Eleitoral	int64
NM_LEGENDA	Nome da legenda	object
COMP_LEG	Composição da legenda	object
TOT_VOTOS	Quantidade de votos nominais totalizados para aquele candidato naquele município e zona	int64

2.8. Dataset de códigos de municípios padrão TSE⁹

Base de dados com informações de nomes dos municípios e respectivos códigos tanto no formato usado pelo TSE quanto no formato usado pelo IBGE.

Esta informação será usada para levar os códigos do TSE para uma base mais completa do IBGE que contem nomes de distritos e subdistritos (item 2.9) e, posteriormente, para fazermos a tentativa de recuperação de códigos de municípios não informados no *dataset* principal. Ao todo, a base contem 5.569 linhas cujas colunas estão a seguir descritas, destacadas em amarelo as que foram usadas como relacionamento com outros *datasets*¹⁰.

Nome da coluna	Descrição da coluna	Tipo
codigo_tse	Código do município conforme padrão TSE	int64
uf	Unidade da Federação do município	object
nome_municipio	Nome do município	object
capital	Municipio é capital ? 1, se sim, 0 se não	int64
cd_mun_ibge	Código do município conforme padrão IBGE	int64

⁹ Link para a base de municípios TSE, coletada em 08/06/2021: <https://github.com/betafccc/Municipios-Brasileiros-TSE>

¹⁰ A coluna cd_mun_ibge foi usada para relacionamento com o *dataset* municípios, distritos e subdistritos do IBGE.

--	--	--

2.9. Dataset de códigos de municípios, distritos e subdistritos padrão IBGE¹¹

Base de dados com informações de nomes dos municípios, distritos dos municípios e subdistritos dos municípios e respectivos códigos no formato usado pelo IBGE.

Esta informação será usada para fazermos a tentativa de recuperação de códigos de municípios não informados no *dataset* principal. Ao todo a base contem 10.575 linhas cujas colunas estão a seguir descritas, destacadas em amarelo as que foram usadas como relacionamento com outros *datasets*¹²

nome coluna	descrição coluna	tipo
UF	Código da Unidade da Federação	int64
Nome_UF	Nome da Unidade da Federação	object
Mesorregião	Código da Mesorregião Geográfica	int64
Nome_Mesorregião	Nome da Mesorregião Geográfica	object
Microrregião	Código da Microrregião Geográfica	int64
Nome_Microrregião	Nome da Microrregião Geográfica	object
Município	Código do Município	int64
Nome_Município	Nome do Município	object
Distrito	Código do Distrito	int64
Nome_Distrito	Nome do Distrito	object
Subdistrito	Código do Subdistrito	int64
Nome_Subdistrito	Nome do Subdistrito	object

¹¹ Link para a base municípios, distritos e subdistritos do IBGE, coletada em 08/06/2021: <https://www.ibge.gov.br/geociencias/organizacao-do-territorio/estrutura-territorial/23701-divisao-territorial-brasileira.html?=&t=downloads>

¹² As colunas UF + Município juntas, formam o código de município completo que foi usado para relacionamento com o *dataset* de municípios do TSE. Já as colunas Nome_Município, Nome_Distrito e Nome_Subdistrito foram usadas na tentativa de encontrar nomes de livre digitação em coluna do *dataset* principal. Maiores detalhes estão descritos no item 3 deste trabalho.

--	--	--

3. Processamento/Tratamento de Dados

Todo o processo de leitura, tratamento, junções, imputações, transformações, treinamento, etc que executamos sobre as bases de dados foram feitos usando a linguagem Python[1], via Jupyter Notebook[2]. Diversas bibliotecas também foram usadas, sendo as principais: pandas[3], numpy[4], matplotlib[5], seaborn[6], difflib[7] e scikit-learn[8]. Os passos detalhados estão disponíveis no notebook jupyter de nome *tcc_eleicoes.ipynb*, no repositório deste trabalho no *github*¹³.

3.1. Leitura da base inicial de candidaturas para eleições de 2008

Conforme indicado nos arquivos de apoio (*leia.me*) do TSE, a codificação dos caracteres dos arquivos constantes no repositório de dados eleitoral é do tipo '*latin_1*' e os campos estão entre aspas "" (inclusive os campos numéricos) e são separados por ponto e vírgula.

O primeiro obstáculo que encontramos com as bases do TSE foi a falta de padronização dos nomes dos arquivos e das colunas ao longo dos anos. Para esta base de candidaturas, por exemplo, até o ano de 2010, o arquivo tem a extensão ".txt". Já a partir de 2012, a extensão passa a ser ".csv". No que concerne aos nomes das colunas, até o ano de 2012, não é disponibilizado, na primeira linha do arquivo, o nome das colunas. Somente a partir de 2012 é que os arquivos passam a apresentar a nomenclatura das colunas na sua primeira linha. Assim, buscamos os nomes das colunas para o ano de 2008 e 2012 a partir do arquivo "*leia.me*" disponibilizado junto do arquivo ".zip" originário. Além disso, o próprio nome das colunas ao longo dos anos sofre alterações nos arquivos "*leia.me*". Por exemplo, até 2012, a coluna que designa o sexo do candidato é referenciada como "sexo". A partir de 2012, passa a ser referenciada como "gênero".

Como tivemos que ler arquivos de anos diversos e com intuito de tentar deixar o processo de leitura das bases o menos complicado possível, criamos funções para

¹³ Link do repositório *github* deste trabalho:

leitura, de tal forma que toda essa “falta de padronização” nas nomenclaturas do TSE fosse tratada dentro das funções. Apresentamos a seguir a função para leitura do arquivo da “base de candidatos”:

```
# função para leitura de dados da base "candidatos" do TSE
def le_cand(ano, uf):
    # nomes das colunas até 2010
    c_10 = ['DT_GERACAO', 'HH_GERACAO', 'ANO_ELEICAO', 'NR_TURNO', 'DS_ELEICAO', 'SG_UF', 'SG_UE', 'NM_UE',
            'CD_CARGO', 'DS_CARGO', 'NM_CANDIDATO', 'SQ_CANDIDATO', 'NR_CANDIDATO', 'NR_CPF_CANDIDATO', 'NM_URNA_CANDIDATO',
            'CD_SITUACAO_CANDIDATURA', 'DS_SITUACAO_CANDIDATURA', 'NR_PARTIDO', 'SG_PARTIDO', 'NM_PARTIDO', 'SQ_LEGENDA',
            'SG_LEGENDA', 'COMPOSICAO_LEGENDA', 'NM_LEGENDA', 'CD_OCUPACAO', 'DS_OCUPACAO', 'DT_NASCIMENTO',
            'NR_TITULO_ELEITORAL_CANDIDATO', 'IDADE_DATA_ELEICAO', 'CD_GENERO', 'DS_GENERO', 'CD_GRAU_INSTRUCAO',
            'DS_GRAU_INSTRUCAO', 'CD_ESTADO_CIVIL', 'DS_ESTADO_CIVIL', 'CD_NACIONALIDADE', 'DS_NACIONALIDADE',
            'SG_UF_NASCIMENTO', 'CD_MUNICIPIO_NASCIMENTO', 'NM_MUNICIPIO_NASCIMENTO', 'VR_DESPESA_MAX_CAMPANHA',
            'CD_SIT_TOT_TURNO', 'DS_SIT_TOT_TURNO']

    # nomes das colunas em 2012
    c_12 = ['DT_GERACAO', 'HH_GERACAO', 'ANO_ELEICAO', 'NR_TURNO', 'DS_ELEICAO', 'SG_UF', 'SG_UE', 'NM_UE',
            'CD_CARGO', 'DS_CARGO', 'NM_CANDIDATO', 'SQ_CANDIDATO', 'NR_CANDIDATO', 'NR_CPF_CANDIDATO', 'NM_URNA_CANDIDATO',
            'CD_SITUACAO_CANDIDATURA', 'DS_SITUACAO_CANDIDATURA', 'NR_PARTIDO', 'SG_PARTIDO', 'NM_PARTIDO', 'SQ_LEGENDA',
            'SG_LEGENDA', 'COMPOSICAO_LEGENDA', 'NM_LEGENDA', 'CD_OCUPACAO', 'DS_OCUPACAO', 'DT_NASCIMENTO',
            'NR_TITULO_ELEITORAL_CANDIDATO', 'IDADE_DATA_ELEICAO', 'CD_GENERO', 'DS_GENERO', 'CD_GRAU_INSTRUCAO',
            'DS_GRAU_INSTRUCAO', 'CD_ESTADO_CIVIL', 'DS_ESTADO_CIVIL', 'CD_NACIONALIDADE', 'DS_NACIONALIDADE',
            'SG_UF_NASCIMENTO', 'CD_MUNICIPIO_NASCIMENTO', 'NM_MUNICIPIO_NASCIMENTO', 'VR_DESPESA_MAX_CAMPANHA',
            'CD_SIT_TOT_TURNO', 'DS_SIT_TOT_TURNO', 'NM_EMAIL']

    # nome do arquivo zipado, fornecido pelo TSE
    nome_zip = 'tse/candidatos/consulta_cand_' + str(ano) + '.zip'

    # define nome do arquivo dentro do arquivo zip, a depender do ano
    if ano > 2012:
        arq = 'consulta_cand_' + str(ano) + '_' + uf + '.csv'
        cols = None
    else:
        arq = 'consulta_cand_' + str(ano) + '_' + uf + '.txt'
        if ano == 2012:
            cols = c_12
        else:
            cols = c_10

    # abre arquivo zipado
    zf = ZipFile(nome_zip)

    # faz a leitura do arquivo da base de candidatos, usando read_csv
    df = pd.read_csv(zf.open(arq), sep=';', encoding='latin_1', names=cols, dtype={'NR_CPF_CANDIDATO': 'string'})
    return df
```

Assim, para fazer a leitura da base candidatos de qualquer ano entre 2004 a 2020, basta chamar a função **le_cand(ano, UF)** com os parâmetros **ano** da base a ser lida e **UF** (unidade da federação) que se pretende trabalhar.

```
In [4]: # lê dados de candidatos das eleições 2008
cd = le_cand(2008, 'SP')
```

Todo o “tratamento” da falta de padronização ao longo dos anos - alteração no padrão dos nomes dos arquivos, falta de linhas indicativas de nomes de colunas, alteração de nomes de colunas, são tratadas dentro da função.

A base principal que iremos trabalhar é a de *candidatos às eleições do ano de*

2008, para o estado de São Paulo. Feita a leitura da mesma, identificamos que ela apresenta 63.633 registros. Entretanto, nesta base estão TODOS os candidatos para as eleições do ano de 2008 que incluem, além de vereadores, prefeitos, vice-prefeitos e até mesmo registros referentes a plebiscito realizado no estado de São Paulo naquele ano, conforme pode ser visto a seguir nos “tipos de cargo” que aparecem no arquivo:

In [6]:	cols = ['CD_CARGO', 'DS_CARGO'] cd[cols].drop_duplicates()										
Out[6]:											
	<table border="1"> <thead> <tr> <th>CD_CARGO</th> <th>DS_CARGO</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>11</td> </tr> <tr> <td>3</td> <td>12</td> </tr> <tr> <td>40</td> <td>13</td> </tr> <tr> <td>63631</td> <td>VEREADOR 91 VOCÉ É A FAVOR DA ALTERAÇÃO DO NOME DA CID...</td> </tr> </tbody> </table>	CD_CARGO	DS_CARGO	0	11	3	12	40	13	63631	VEREADOR 91 VOCÉ É A FAVOR DA ALTERAÇÃO DO NOME DA CID...
CD_CARGO	DS_CARGO										
0	11										
3	12										
40	13										
63631	VEREADOR 91 VOCÉ É A FAVOR DA ALTERAÇÃO DO NOME DA CID...										
	4 rows × 2 columns										

Para o nosso estudo, interessa apenas os registros de candidatos ao cargo de Vereador. Feita a filtragem no `CD_CARGO == 13`, temos uma base inicialmente com 59.536 registros. Este é, portanto, nosso *dataset* “inicial” e no notebook *jupyter* é referenciado como ***dt08***.

3.1.1. Análise inconsistências iniciais

Em verificação inicial dos dados, pela saída de “comandos” *python* como ***info()***, ***nunique()***, ***duplicated()***, dentre outros¹⁴ identificamos as seguintes situações:

- **registros duplicados**

O nosso *dataset* de candidatos não apresenta nenhum registro duplicado.

- **colunas com conteúdo idêntico para todos os registros**

13 colunas do *dataset* tem o mesmo valor preenchido para todos os candidatos o que, em princípio, as tornam pouco úteis para qualquer tipo de análise. São elas:

¹⁴ Comandos e respectivas saídas estão detalhados no notebook *jupyter*, no github do trabalho.

```
In [23]: # colunas que contém um único valor em todos os registros e respectivo valor
for col in cd08.columns:
    if cd08[col].nunique() == 1:
        print(col, cd08[col].unique())

DT_GERACAO ['06/10/2020']
HH_GERACAO ['14:56:30']
ANO_ELEICAO [2008]
NR_TURNO [1]
DS_ELEICAO ['Eleições 2008']
SG_UF ['SP']
CD_CARGO [13]
DS_CARGO ['VEREADOR']
SG_LEGENDA ['#NE#']
COMPOSICAO_LEGENDA ['#NE#']
IDADE_DATA_ELEICAO [-1]
CD_MUNICIPIO_NASCIMENTO [-1]
VR_DESPESA_MAX_CAMPANHA [-1]
```

Destas, 8 colunas, marcadas em amarelo (DT_GERACAO, HH_GERACAO, ANO_ELEICAO, NR_TURNO, DS_ELEICAO, SG_UF, CD_CARGO e DS_CARGO), não apresentam inconsistências. Apesar de não terem utilidade para etapa de aprendizagem de máquina, algumas ainda serão úteis para fins de relacionamento com outras bases de dados.

As outras 5 coluna. marcadas em azul e vermelho na figura acima, são na verdade *missing values* (valores não informados), como será visto a seguir.

- ***missing values***

A análise inicial indica a aparente inexistência de registros sem dados preenchidos (*missing values*):

```
In [22]: # procura por missing values no dataframe
cd08.isnull().values.any()

Out[22]: False
```

Entretanto, os arquivos de apoio do TSE (leia.me) destacam que:

- campos preenchidos com “#NULO” significam que a informação está em branco no banco de dados;
- o correspondente para #NULO nos campos numéricos é “-1”;
- campos preenchidos com “#NE” significam que naquele ano a informação não era registrada em banco de dados pelos sistemas eleitorais;
- correspondente para #NE nos campos numéricos é “-3”

Assim, entendemos que campos preenchidos com os valores "#NULL", "#NE", "-1" e "-3" são, na verdade, *missing values*.

As colunas SG_LEGENDA e COMPOSICAO_LEGENDA têm apenas o valor '#NE#' informado para todos os candidatos (registros), portanto, contém APENAS *missing values*.

Já as colunas IDADE_DATA_ELECAO, CODIGO_MUNICIPIO_NASCIMENTO e VR_DESPESA_MAX_CAMPANHA têm apenas o valor '-1' informado para todos os candidatos (registros). Portanto, contêm APENAS *missing values*.

3.1.2. Tratamento de *missing values*

Como mencionado, identificamos 5 colunas que estão totalmente preenchidas com valores que representam *missing values*. Trataremos as mesmas da seguinte forma:

3.1.2.1 Colunas relacionadas a coligações

As colunas SG_LEGENDA e COMPOSICAO_LEGENDA estão relacionadas às coligações partidárias contêm apenas *missing values* e são inúteis desta forma. Como temos uma base de dados específica para coligações partidárias no repositório do TSE, iremos excluir da base de candidatos todas as colunas que tratem de coligações. Além dessas 2 colunas citadas que não têm valores úteis, vamos também excluir as colunas SQ_LEGENDA e NM_LEGENDA.

```
In [9]: # exclui colunas inconsistentes da base candidatos:  
remove_cols = ['SQ_LEGENDA', 'SG_LEGENDA', 'COMPOSICAO_LEGENDA', 'NM_LEGENDA']  
cd08.drop(remove_cols, axis=1, inplace=True)
```

3.1.2.2 Coluna VR_DESPESA_MAX_CAMPANHA

Não dispomos de fonte alternativa para buscar este valor. Entretanto, da forma como ela está (preenchida com "-1"), não tem nenhuma utilidade para nossa análise. Assim, decidimos também pela exclusão desta coluna.

3.1.2.3 Coluna IDADE_DATA_ELEICAO

Outra coluna totalmente preenchida com “-1”. Entretanto, neste caso específico, dispomos de uma coluna com a data de nascimento do candidato. Assim, vamos calcular a idade do candidato e imputar este valor calculado à coluna. Decidimos usar, para fins de padronização, a idade na data da posse e não na data da eleição, uma vez que a partir de 2012 o próprio TSE passou a adotar esta idade.

```
In [18]: # altera tipo do campo DT_NASCIMENTO para daytime
from datetime import date, timedelta
cd08['DT_NASCIMENTO'] = pd.to_datetime(cd08['DT_NASCIMENTO'], dayfirst=True)

In [19]: # calcula idade na data da posse (a posse é sempre no dia 1 de janeiro do ano seguinte ao das eleições)
cd08['IDADE_DATA_ELEICAO'] = (pd.to_datetime('2009-01-01') - cd08['DT_NASCIMENTO'])//timedelta(days=365.2425)

In [20]: # renomear coluna para idade_data_posse
cd08.rename(columns={'IDADE_DATA_ELEICAO':'IDADE_DATA_POSSE'}, inplace=True)
```

3.1.2.4 Coluna CODIGO_MUNICIPIO_NASCIMENTO

No dataset principal temos 3 colunas que se referem ao local de nascimento do candidato: sigla da UF de nascimento (SG_UF_NASCIMENTO), código do município de nascimento (CODIGO_MUNICIPIO_NASCIMENTO) e nome do município de nascimento (NM_MUNICIPIO_NASCIMENTO). Destas 3, a coluna nome do município é de livre preenchimento pelo candidato, aceitando qualquer valor digitado, sem nenhuma crítica. Portanto está sujeita a erros de acentuação, digitação e mesmo inserção de nomes de municípios inexistentes. Talvez por esta razão, o TSE tenha decidido preencher a coluna código do município com o valor “-1”, que é sinônimo de *NaN*, para todos os candidatos.

Para tentar deixar essas colunas mais úteis para a fase de aprendizado de máquina, vamos comparar os valores informados pelo candidato nas colunas UF e nome do município com os nomes oficiais de municípios, distritos de municípios, e subdistritos de municípios, conforme dados do IBGE e TSE. De forma sucinta o procedimento adotado foi o seguinte:

- Preparação das bases de dados a serem comparadas:
 - Da nossa base de candidatos, extraímos uma base mais enxuta, apenas com os campos: SQ_CANDIDATO, SG_UF_NASCIMENTO e NM_MUNICIPIO_NASCIMENTO.
 - Montagem base de referência para comparação:

- O TSE usa codificação de municípios própria, porém não tem informação de distritos/subdistritos dos municípios.
 - O IBGE tem codificação detalhada para cada município, distrito e subdistrito, dentre outros.
 - Juntamos a codificação de municípios do TSE à base do IBGE e criamos *dataframes* individualizados para municípios, distritos e subdistritos,
 - Estes três *dataframes* foram a nossa base final, que serviu de referência para as pesquisas.
- b) Normalização das bases para facilitar as comparações:
- deixar os nomes de municípios em minúsculas (tanto base IBGE/TSE quanto na base candidatos)
 - retirar a acentuação (na base IBGE/TSE e na base candidatos)

A seguir uma amostra dos *dataframes* e tabelas usados nas pesquisas de municípios.

- *dataframe* reduzido de candidatos:

```
uen.head(3)
```

	SQ_CANDIDATO	uf	nm_mun_dig	muni_norm
40	27238	SP	JUNQUEIROPOLIS	junqueiropolis
41	18841	SP	ADAMANTINA	adamantina
42	14043	SP	ADAMANTINA	adamantina

dataframe reduzido de candidatos, com coluna de nome município normalizada (*muni_norm*)

- tabela de municípios TSE:

```
muni_tse.head(2)
```

	cd_tse	uf	nm_municipio	capital	cd_mun_ibge
0	1120	AC	ACRELÂNDIA	0	1200013
1	1570	AC	ASSIS BRASIL	0	1200054

tabela “original” TSE, apenas com colunas de interesse

- tabela de municípios, distritos e subdistritos IBGE de 2008:

`tab_ibge.head(2)`

	uf	cd_mun_ibge	nm_mun	nm_dist	nm_subdist
0	RO	1100015	Alta Floresta D'Oeste	Rolim de Moura do Guaporé	NaN
1	RO	1100015	Alta Floresta D'Oeste	Izidolândia	NaN

tbla “original” IBGE, apenas com colunas de interesse

- dataframes* de referência para pesquisa, criados pela junção das tabelas IBGE e TSE

`tb_muni.head(2)`

`tb_dist.head(2)`

`tb_subd.head(2)`

	uf	cd_mun_ibge	nm_mun	cd_tse
0	AC	1200013	acrelândia	1120
1	AC	1200054	assis brasil	1570

	uf	cd_mun_ibge	nm_mun	cd_tse
0	AC	1200013	acrelândia	1120
1	AC	1200054	assis brasil	1570

	uf	cd_mun_ibge	nm_subdist	cd_tse
0	AL	2704302	primeira regiao	27855
1	AL	2704302	quarta regiao	27855

dataframes finais de referência para as pesquisas de municípios, já com nomes de municípios, distritos e subdistritos “normalizados”

c) Feita a normalização inicial, executamos comparações de semelhança entre entre a base de candidatos reduzida e os dataframes de pesquisa IBGE/TSE. A cada pesquisa executada separamos a base inicial em 2 outras: uma “positiva” na qual deixamos os registros que obtiveram resultado positivo na pesquisa e outra base “negativa”, para os registros que não obtiveram sucesso na pesquisa. À base “positiva” é acrescentado o código de município do TSE (recuperado da base de pesquisa) e deixamos ela separada para posterior atualização da base de candidatos completa. A base “negativa” passa a ser a base inicial para a próxima pesquisa. A sequencia de pesquisas que realizamos está listada a baixo:

- pesquisa 1: compara se o par (*uf, nome_município*) informados na base candidatos tem correspondência IDÊNTICA no par (*uf, nome_município*) da base do IBGE ou TSE.
- Pesquisa 2: verifica se o par (*uf, nome_município*) informados na base candidatos tem correlação IDÊNTICA no par (*uf, nome_distrito*) da base do IBGE.

- Pesquisa 3: verifica se o par (*uf, nome_município*) informados na base candidatos tem correlação IDÊNTICA no par (*uf, nome_subdistrito*) da base do IBGE.
- Pesquisa 4: verifica se o par (*uf, nome_município*) informados na base candidatos tem correlação IDÊNTICA em *uf* e SIMILAR em *nome_município* da base do IBGE.
- Pesquisa 5: verifica se o par (*uf, nome_município*) informados na base candidatos tem correlação IDÊNTICA em *uf* e SIMILAR em *nome_distrito* da base do IBGE.
- Pesquisa 6: verifica se o par (*uf, nome_município*) informados na base candidatos tem correlação IDÊNTICA em *uf* e SIMILAR em *nome_subdistrito* da base do IBGE.
- Pesquisa 7: A *uf ZZ*, indica localidade de nascimento fora do Brasil, conforme informado nos arquivos “leia.me” do TSE. Assim, vamos separar da base não identificada os registros que tem *uf* informada na base candidatos reduzida igual a ZZ. Vamos imputar o valor **-5** como código de município para esses registros e deixá-los a parte para posterior atualização da base de candidatos. Os registros ainda não identificados servirão para entrada da próxima pesquisa.
- Pesquisa 8: verifica se o *nome_município* informado na base candidatos tem correlação SIMILAR em *nome_município* da base do IBGE, independente do estado da federação.
- Imputação 9: Depois das 8 pesquisas anteriores, para os registros que ainda assim não foram identificados, imputamos o valor **-1** como código de município.

d) Juntamos os *dataframes* “positivos” das 8 pesquisas iniciais ao *dataframe* da imputação 9. Este novo *dataframe* foi usado para atualizar as colunas (uf, nm_municipio_nascimento e cd_mun_nascimento) da nossa base de candidatos.

e) Para realizar as pesquisas de similaridade utilizamos a função ***get_close_matches*** da biblioteca ***difflib*** do *python*. Esta biblioteca é baseada no algoritmo publicado por Ratcliff and Obershelp, no final dos anos 80, com o nome '*gestalt pattern matching*'¹⁵. Esta função retorna uma lista de nomes próximos ao que estamos comparando e tem um parâmetro de "corte" ***cutoff*** que varia de 0 a 1, sendo que:

- o valor 1 indica pesquisa IDÊNTICA,
- quanto mais próximo de 0 mais tolerável a erro,
- o padrão da biblioteca é 0.6
- após algumas observações, decidimos usar o valor 0.7, para pesquisas em municípios e 0.8 para pesquisas em distritos e subdistritos.

Criamos uma função específica (***busca_semelhança***) para fazer estas pesquisas, tendo como parâmetros:

- o nome do *dataframe* a ser pesquisado,
- onde pesquisar (municípios ‘***m***’, distritos ‘***d***’ ou subdistritos ‘***s***’),
- grau de similaridade (***cutoff***), variando de 0 a 1 e
- pesquisa limitada a uf informada ou não (‘***uf***’ ou ‘***br***’)

Apenas para fins de ilustração, segue fluxo resumido da pesquisa 8:

uen.head(3)					tb_muni.head(2)			
SQ_CANDIDATO	uf	nm_mun_dig	muni_norm		uf	cd_mun_ibge	nm_mun	cd_tse
40	27238 SP	JUNQUEIROPOLIS	junqueiropolis		0 AC	1200013	acrelandia	1120
41	18841 SP	ADAMANTINA	adamantina		1 AC	1200054	assis brasil	1570
42	14043 SP	ADAMANTINA	adamantina					

¹⁵ https://en.wikipedia.org/wiki/Gestalt_Pattern_Matching

- **uen** é o nosso *dataframe* de candidatos reduzido e **tb_muni** é nossa tabela de referência de nomes de municípios.
- A pesquisa 8 busca pelo nome de município informado na tabela de candidatos depois de “normalizado” (**muni_norm**) SIMILAR a nome de município da tabela de referência depois de “normalizado” (**nm_mun**).
- A seguir, a codificação *python* para realizar esta pesquisa, comentada, e o retorno após a pesquisa:

```

: uen_t = uen.sample(10).copy()
# colunas de interesse para junção
cols_tse=['cd_tse', 'uf', 'nm_mun']
# função de busca
f = lambda x: next(iter(difflib.get_close_matches(x, uen_t['muni_norm'], cutoff=0.7, n=1)), np.nan)
# executa a busca para cada registro do dataframe
uen_t['resultado'] = (uen_t['muni_norm']).apply(f)
# agraga o código tse (do df tb_tse) as linhas que obtiveram resultado na busca
uen_t = uen_t.merge(tb_muni[cols_tse], how='left', left_on=['resultado'], right_on=['nm_mun'])
uen_t.drop(columns=['uf_y','nm_mun'], inplace=True)
uen_t.rename(columns={'uf_x':'uf'}, inplace=True)

# separa o resultado em 2 df: 1 que achou município e outro que não encontrou
uen_t.rename(columns={'muni_norm.x':'muni_norm'}, inplace=True)
df1 = uen_t[uen_t.cd_tse.notna()].copy()
cols1 = [0,1,2,4,5]
df1 = df1[df1.columns[cols1]]
cols = ['SQ_CANDIDATO','uf','nm_mun_dig','muni_norm']
df2 = uen_t[uen_t.cd_tse.isna()][cols]
print(f'{uen_t.shape[0]} municípios pesquisados')
print(f'{round(df1.shape[0] / uen_t.shape[0] *100,2)}% ({df1.shape[0]}) encontrados')
print(f'{round(df2.shape[0] / uen_t.shape[0] *100,2)}% ({df2.shape[0]}) não encontrados')

10 municípios pesquisados
80.0% (8) encontrados
20.0% (2) não encontrados

```

df1

	SQ_CANDIDATO	uf	nm_mun_dig	resultado	cd_tse
0	1867	SP	ITABERÁ	itabera	65,358.00
2	18530	SP	PIRACICABA	piracicaba	68,756.00
3	29072	SP	RIBEIRÃO PIRES	ribeirao pires	69,671.00
4	30635	SP	PATROCÍNIO PAULISTA	patrocinio paulista	68,276.00
5	54266	SP	MARILIA	marilia	66,818.00
7	52206	SP	TURIUBA	turiuba	72,052.00
8	59227	SP	GUAIMBÉ	guaimbe	64,475.00
9	67963	AL	TAQUARANA	taquarana	28,819.00

8 rows × 5 columns

df2

	SQ_CANDIDATO	uf	nm_mun_dig	muni_norm
1	445	SP	MOGI-MIRIM	mogi-mirim
6	22703	SP	TARUMÃ-SP	taruma-sp

2 rows × 4 columns

Como pode-se perceber, desta pesquisa “teste” (com 10 registros aleatórios do dataset *uen*) geramos 2 *dataframes*: df1, com registros que tiveram resultado positivo na busca e que ficarão guardados para posterior atualização da base de candidatos completa e df2, com registros que não tiveram resultado na busca.

Todo o processo de imputação, com todas as pesquisas realizadas e posterior atualização da base de candidatos está detalhado no notebook *jupyter* no *github*.

A seguir, um resumo dos resultados das buscas efetivas na nossa base de trabalho:

operação	qtd registros inicial	com resultado	sem resultado
Pesquisa 1	59.536	93,65%	6,34%
Pesquisa 2	3.779	0,29%	99,71%
Pesquisa 3	3.768	13,61%	86,62%
Pesquisa 4	3.264	76,75%	23,25%
Pesquisa 5	759	0,26%	99,74%
Pesquisa 6	757	7,66%	92,34%
Imputação 7	699	7,87%	92,13%
Pesquisa 8	644	82,27%	23,60%
Imputação 9	152		

Entendemos que a imputação foi um sucesso, tendo em vista que de um total de 59.536 registros conseguimos imputar valores com um grau de segurança muito bom em 99,74% deles. Apenas 152 registros ficaram com marcação *Nan* (-1) ao final do processo. Número este que poderia ser ainda melhor, caso tivéssemos tratado individualmente este 152 registros. Entretanto, entendemos que não era o caso.

3.2. Enriquecimento do dataset

Feitos os tratamentos iniciais de *missing values* e inconsistências, o nosso dataset ficou com 38 colunas de informações. Entretanto, uma simples leitura da descrição das mesmas, indica a existência de colunas sem nenhuma utilidade para os objetivos do nosso trabalho, bem como muitas outras com informações redundantes, como pode ser visto na figura a seguir - destacadas em amarelo são colunas sem nenhuma utilidade para a fase de aprendizagem de máquina e destacadas em roxo, colunas com informações redundantes:

```
: cd08.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 59536 entries, 0 to 59535
Data columns (total 38 columns):
 #   Column           Non-Null Count Dtype  
--- 
 0   DT_GERACAO       59536 non-null  object  
 1   HH_GERACAO       59536 non-null  object  
 2   ANO_ELEICAO      59536 non-null  int64   
 3   NR_TURNO         59536 non-null  int64   
 4   DS_ELEICAO       59536 non-null  object  
 5   SG_UF            59536 non-null  object  
 6   SG_UE            59536 non-null  int64   
 7   NM_UE            59536 non-null  object  
 8   CD_CARGO         59536 non-null  int64   
 9   DS_CARGO         59536 non-null  object  
 10  NM_CANDIDATO    59536 non-null  object  
 11  SQ_CANDIDATO    59536 non-null  int64   
 12  NR_CANDIDATO    59536 non-null  int64   
 13  NR_CPF_CANDIDATO 59536 non-null  string  
 14  NM_URNA_CANDIDATO 59536 non-null  object  
 15  CD_SITUACAO_CANDIDATURA 59536 non-null  int64   
 16  DS_SITUACAO_CANDIDATURA 59536 non-null  object  
 17  NR_PARTIDO        59536 non-null  int64   
 18  SG_PARTIDO         59536 non-null  object  
 19  NM_PARTIDO         59536 non-null  object  
 20  CD_OCUPLICAO     59536 non-null  int64   
 21  DS_OCUPLICAO     59536 non-null  object  
 22  DT_NASCIMENTO    59536 non-null  datetime64[ns]
 23  NR_TITULO_ELEITORAL_CANDIDATO 59536 non-null  int64   
 24  IDADE_DATA_POSSE 59536 non-null  int64   
 25  CD_GENERO          59536 non-null  int64   
 26  DS_GENERO          59536 non-null  object  
 27  CD_GRAU_INSTRUCAO 59536 non-null  int64   
 28  DS_GRAU_INSTRUCAO 59536 non-null  object  
 29  CD_ESTADO_CIVIL   59536 non-null  int64   
 30  DS_ESTADO_CIVIL   59536 non-null  object  
 31  CD_NACIONALIDADE 59536 non-null  int64   
 32  DS_NACIONALIDADE 59536 non-null  object  
 33  CD_SIT_TOT_TURNO 59536 non-null  int64   
 34  DS_SIT_TOT_TURNO 59536 non-null  object  
 35  SG_UF_NASCIMENTO 59536 non-null  object  
 36  NM_MUNICIPIO_NASCIMENTO 59536 non-null  object  
 37  CD_MUNICIPIO_NASCIMENTO 59536 non-null  int64   

dtypes: datetime64[ns](1), int64(17), object(19), string(1)
memory usage: 17.7+ MB
```

Assim, apesar do número de colunas, nos parece que as informações realmente relevantes são “escassas” e aparentemente “limitadas”. Logo, a necessidade de enriquecer a base com novas informações parece clara. No próprio repositório do TSE encontramos outras bases de informações que podem nos trazer informações úteis, para fins de enriquecimento da nossa base inicial. A seguir detalhamos cada uma das bases de informações adicionais que usamos para deixar nosso *dataset* principal mais robusto.

3.2.1. Base de dados de coligações partidárias

As coligações partidárias em determinados períodos (inclusive no ano de 2008 que estamos analisando) eram determinantes em cálculos de coeficiente eleitoral que, por sua vez, influenciavam diretamente na eleição ao não de um

candidato. Assim, é uma informação que entendemos bastante relevante para nosso estudo.

A figura abaixo mostra a relação de colunas originais da base de coligações partidárias. Destacamos em verde as colunas usadas para fazer a junção (*merge*) e em roxo as colunas que iremos agregar ao *dataset* principal.

#	Column	Non-Null Count	Dtype
0	DT_GERACAO	24394	non-null object
1	HH_GERACAO	24394	non-null object
2	ANO_ELEICAO	24394	non-null int64
3	NR_TURNO	24394	non-null int64
4	DS_ELEICAO	24394	non-null object
5	SG_UF	24394	non-null object
6	SG_UE	24394	non-null int64
7	DS_UE	24394	non-null object
8	CD_CARGO	24394	non-null int64
9	DS_CARGO	24394	non-null object
10	TP_AGREMIACAO	24394	non-null object
11	NR_PARTIDO	24394	non-null int64
12	SG_PARTIDO	24394	non-null object
13	NM_PARTIDO	24394	non-null object
14	SG_COLIGACAO	24394	non-null object
15	NM_COLIGACAO	24394	non-null object
16	COMP_COLIGACAO	24394	non-null object
17	SQ_COLIGACAO	24394	non-null int64

Assim, serão acrescentadas ao *dataset* principal as seguintes colunas:

- tipo de agremiação (partido isolado ou coligação),
- número sequencial da coligação (gerado pelo TSE),
- nome da coligação,
- composição da coligação (relação dos partidos que compõem a coligação, separados pelo caractere "/"),
- além disso, criamos uma nova coluna com a quantidade de partidos que compõem a coligação.

Ao final, foram estas as colunas agregadas ao nosso *dataset* principal:

SCIMENTO	CD_MUNICIPIO_NASCIMENTO	TP_AGREMIACAO	SQ_COLIGACAO	NM_COLIGACAO	COMP_COLIGACAO	QT_PART_COL
PLANALTO	68934	COLIGACAO	5069	DIREITOS IGUAIS	14-PTB / 15-PMDB	2
MARIA RODRIGUES ROXA	71897	COLIGACAO	3390	COLIGAÇÃO PMDB / PSC	15-PMDB / 20-PSC	2
CAPIVARI	63096	COLIGACAO	769	PDT-PSDB	12-PDT / 45-PSDB	2

A seguir a função para leitura do dataset de coligações, cômputo da coluna “QT_PART_COL” e definição das colunas de interesse para a fazer a junção com o dataset principal:

```
def le_coligacao(ano, uf):
    # definição dos nomes de colunas para os anos em o arquivo do TSE não tem os nomes na la linha
    c08 = ['DT_GERACAO', 'HH_GERACAO', 'ANO_ELEICAO', 'NR_TURNO', 'DS_ELEICAO', 'SG_UF', 'SG_UE', 'DS_UE', 'CD_CARGO',
           'DS_CARGO', 'TP_AGREMIACAO', 'NR_PARTIDO', 'SG_PARTIDO', 'NM_PARTIDO', 'SG_COLIGACAO', 'NM_COLIGACAO',
           'COMP_COLIGACAO', 'SQ_COLIGACAO']

    # definição tipos específicos na leitura ininal
    tipos = None

    # padroniza nome colunas que serão usadas na junção ou em algum cômputo
    rencols = {'DS_COMPOSICAO_COLIGACAO': 'COMP_COLIGACAO'}

    if ano > 2012:
        # nome do arquivo zipado baixado do site TSE
        nome_zip = 'tse/candidatos/consulta_coligacao_' + str(ano) + '.zip'
        # no do arquivo dentro do zip
        arq = 'consulta_coligacao_' + str(ano) + '_' + uf + '.csv'
        cols = None
    else:
        # nome do arquivo zipado baixado do site TSE
        nome_zip = 'tse/candidatos/consulta_legendas_' + str(ano) + '.zip'
        # no do arquivo dentro do zip
        arq = 'consulta_legendas_' + str(ano) + '_' + uf + '.txt'
        cols = c08

    zf = ZipFile(nome_zip)
    df = pd.read_csv(zf.open(arq), sep=';', encoding='latin_1', names=cols, dtype=tipos)

    # ajustes de nome de coluna que será usada como cômputo
    df.rename(columns=rencols, inplace=True)

    # padronização conteúdo coluna usada na agregação para ficar compativel com base de candidatos
    df['DS_ELEICAO'] = df['DS_ELEICAO'].str.title()

    # cria nova coluna com a quantidade de partidos na coligação
    df['QT_PART_COL'] = df['COMP_COLIGACAO'].str.count('/')+1

    # colunas finais de ineteresse para a agregação
    colsfim = ['ANO_ELEICAO', 'NR_TURNO', 'DS_ELEICAO', 'SG_UF', 'SG_UE', 'CD_CARGO', 'NR_PARTIDO',
               'TP_AGREMIACAO', 'SQ_COLIGACAO', 'NM_COLIGACAO', 'COMP_COLIGACAO', 'QT_PART_COL']
    return df[colsfim]
```

```
coli08 = le_coligacao(2008, 'SP')
coli08.head(2)
```

TURNO	DS_ELEICAO	SG_UF	SG_UE	CD_CARGO	NR_PARTIDO	TP_AGREMIACAO	SQ_COLIGACAO	NM_COLIGACAO	COMP_COLIGACAO	QT_PART_COL
1	Eleições 2008	SP	61018		11	12	COLIGACAO	1	NO CORAÇÃO DO Povo	12-PDT / 20-PSC
1	Eleições 2008	SP	61018		11	20	COLIGACAO	1	NO CORAÇÃO DO Povo	12-PDT / 20-PSC

A seguir a codificação para fazer a junção das colunas de interesse ao dataset principal:

```
# leva informação das coligações para a base candidatos
cd08 = cd08.merge(coli08, on=['ANO_ELEICAO', 'NR_TURNO', 'DS_ELEICAO', 'SG_UF',
                               'SG_UE', 'CD_CARGO', 'NR_PARTIDO'], how='left')
```

Nos partidos que não participam de coligações, duas das novas colunas são ‘NaN’, as quais trataremos da seguinte forma:

- qtd de partidos na coligação (QT_PART_COL): **imputar valor 1**

- numero sequencial da coligação: **imputar valor -1**

```
# ajusta os valores NaN que vieram da agregação (partidos que não participam de nenhuma coligação)
# substitui NaN em quantidadde de partidos na coligação para 1 (um único partido, ou seja, sem coligação)
cd08['QT_PART_COL'].fillna(value=1, inplace=True)
# substitui NaN por -1, para o sequencial da coligação.
cd08['SQ_COLIGACAO'].fillna(value=-1, inplace=True)
```

```
# ajustar tipo das novas colunas SQ_COLIGACAO e QT_PART_COL de float para int
cd08['QT_PART_COL'] = cd08['QT_PART_COL'].astype(int)
cd08['SQ_COLIGACAO'] = cd08['SQ_COLIGACAO'].astype(int)
```

3.2.2. Base de dados de bens declarados por candidatos

Outra base disponibilizada a parte pelo TSE é referente aos bens informados / declarados pelos candidatos quando da candidatura. Ainda que seja uma base informativa do candidato, sem checagem com alguma informação mais robusta (como Declaração de Bens do IRPF), entendemos que é uma informação de suma importância para nossa análise, pois trará uma visão geral sobre o patrimônio dos candidatos. Este *dataset* originalmente possui as seguintes colunas:

#	Column	Non-Null Count	Dtype
0	DT_GERACAO	137062	non-null object
1	HH_GERACAO	137062	non-null object
2	ANO_ELEICAO	137062	non-null string
3	DS_ELEICAO	137062	non-null object
4	SG_UF	137062	non-null object
5	SQ_CANDIDATO	137062	int64
6	CD_TIPO_BEM_CANDIDATO	137062	non-null string
7	DS_TIPO_BEM_CANDIDATO	137062	non-null object
8	DS_BEM_CANDIDATO	137062	non-null object
9	VR_BEM_CANDIDATO	137062	float64
10	DT_ULTIMA_ATUALIZACAO	137062	non-null object
11	HH_ULTIMA_ATUALIZACAO	137062	non-null object

Marcamos em amarelo a coluna usada na junção com a base principal e em azul as colunas referentes aos bens. Entendemos que para nossa análise não há interesse, neste momento, em classificar os tipos de bens. Entendemos que será mais útil termos uma visão geral do valor do patrimônio do candidato. Portanto, deste *dataset*, nos interessa o valor total dos bens de cada candidato e, como dado adicional, a quantidade de bens declarados. Para chegarmos a estas 2 colunas de interesse (quantidade de bens e valor total dos bens) fizemos uma agregação no *dataset* pelo sequencial do candidato.

A função que usamos para fazer a leitura do *dataset*, agregação por candidato e com soma do valor dos bens e contagem da quantidade, está descrita a seguir, com os devidos comentários:

```
def le_bens(ano, uf):
    c08 = ['DT_GERACAO', 'HH_GERACAO', 'ANO_ELEICAO', 'DS_ELEICAO', 'SG_UF', 'SQ_CANDIDATO',
           'CD_TIPO_BEM_CANDIDATO', 'DS_TIPO_BEM_CANDIDATO', 'DS_BEM_CANDIDATO',
           'VR_BEM_CANDIDATO', 'DT_ULTIMA_ATUALIZACAO', 'HH_ULTIMA_ATUALIZACAO']

    nome_zip = 'tse/candidatos/bem_candidato_' + str(ano) + '.zip'
    tipos = {'ANO_ELEICAO': 'string', 'CD_TIPO_BEM_CANDIDATO': 'string'}
    if ano > 2012:
        arq = 'bem_candidato_' + str(ano) + '_' + uf + '.csv'
        cols = None
    else:
        arq = 'bem_candidato_' + str(ano) + '_' + uf + '.txt'
        cols = c08

    zf = ZipFile(nome_zip)
    df = pd.read_csv(zf.open(arq), sep=';', encoding='latin_1', names=cols, dtype=tipos)

    # altera coluna valor bem caso ela tenha sido lida como str pelo read_csv
    if type(df['VR_BEM_CANDIDATO'][0]) == str:
        df['VR_BEM_CANDIDATO'] = df['VR_BEM_CANDIDATO'].str.replace(',', '.').astype(float)

    # cria o df de interesse para agregação, com contagem do total de bens declarados e soma dos valores dos bens
    df1 = df.groupby('SQ_CANDIDATO', as_index=False).agg({'VR_BEM_CANDIDATO': ['count', 'sum']})

    # renomeia colunas finais do df de interesse
    df1.columns = ['SQ_CANDIDATO', 'QT_BENS', 'VL_TOT_BENS']
    return df1
```

Após a leitura dos dados e tratamentos dado na função, a saída do *dataset* fica assim:

```
In [101]: bens08 = le_bens(2008, 'SP')

In [111]: bens08.sample(3)

Out[111]:
   SQ_CANDIDATO  QT_BENS  VL_TOT_BENS
0  35920        59425          7  237,575.84
1  29088        48507          3   17,500.00
2  13229        22452          2   87,000.00
```

3 rows × 3 columns

Marcamos em azul as colunas que foram agregadas ao *dataset* principal, usando como chave para junção a coluna **SQ_CANDIDATO**.

```
# agregando informações de bens (quantidade e valor total) para a base de candidatos:
cd08 = cd08.merge(bens08, on=['SQ_CANDIDATO'], how='left')
```

3.2.3. Base de dados de quantidade de vagas

Esta base de dados apresenta a quantidade de vagas disponíveis, por cargo e por município. Servirá como um “limitador” para a quantidade de positivos da nossa coluna alvo (*target*).

No formato original, o *dataset* tem as colunas a seguir relacionadas. Marcamos em amarelo as que servirão de junção e em roxo a coluna que levaremos para o *dataset* principal:

#	Column	Non-Null Count	Dtype
0	DT_GERACAO	1297	non-null
1	HH_GERACAO	1297	non-null
2	ANO_ELEICAO	1297	non-null
3	DS_ELEICAO	1297	non-null
4	SG_UF	1297	non-null
5	SG_UE	1297	non-null
6	NM_UE	1297	non-null
7	CD_CARGO	1297	non-null
8	DS_CARGO	1297	non-null
9	QT_VAGAS	1297	non-null

A função que faz a leitura do *dataset* e separa as colunas de interesse está detalhada a seguir, bem como amostra da base de dados após leitura com destaque para a coluna que levaremos para o *dataset* principal:

```
def le_vagas(ano, uf):
    c08 = ['DT_GERACAO', 'HH_GERACAO', 'ANO_ELEICAO', 'DS_ELEICAO', 'SG_UF', 'SG_UE', 'NM_UE',
           'CD_CARGO', 'DS_CARGO', 'QT_VAGAS']

    nome_zip = 'tse/candidatos/consulta_vagas_' + str(ano) + '.zip'
    tipos = {'ANO_ELEICAO': 'string'}
    if ano > 2012:
        arq = 'consulta_vagas_' + str(ano) + '_' + uf + '.csv'
        cols = None
    else:
        arq = 'consulta_vagas_' + str(ano) + '_' + uf + '.txt'
        cols = c08

    zf = ZipFile(nome_zip)
    df = pd.read_csv(zf.open(arq), sep=';', encoding='latin_1', names=cols, dtype=tipos)
    df['DS_ELEICAO'] = df['DS_ELEICAO'].str.title()
    # colunas finais de interesse para a junção na base de candidatos
    colsfim = ['ANO_ELEICAO', 'DS_ELEICAO', 'SG_UF', 'SG_UE', 'CD_CARGO', 'QT_VAGAS']
    return df[colsfim]
```

```
vagas08 = le_vagas(2008, 'SP')
```

```
vagas08.head(3)
```

	ANO_ELEICAO	DS_ELEICAO	SG_UF	SG_UE	CD_CARGO	QT_VAGAS
0	2008	Eleições 2008	SP	61018	11	1
1	2008	Eleições 2008	SP	61018	13	9
2	2008	Eleições 2008	SP	61034	11	1

A coluna marcada em roxo na figura acima é a que levaremos para o *dataset* principal após o comando “*merge*” a seguir:

```
# agregando informações de qtd vagas para a base de candidatos:
cd08 = cd08.merge(vagas08, on=['ANO_ELEICAO','DS_ELEICAO','SG_UF','SG_UE','CD_CARGO'], how='left')
```

3.2.4. Base de dados de receitas do candidato

Do tema “prestação de contas”, do repositório do TSE, usaremos a base de dados de receitas recebidas pelos candidato para uso na campanha. As colunas originais do dataset estão relacionadas abaixo:

#	Column	Non-Null Count	Dtype
0	SQ_CANDIDATO	504674	non-null int64
1	NM_CANDIDATO	504674	non-null object
2	SEXO	504674	non-null object
3	DS_CARGO	504674	non-null object
4	CD_CARGO	504674	non-null int64
5	NR_CANDIDATO	504674	non-null int64
6	SG_UE_SUPERIOR	504674	non-null object
7	NM_UE	504674	non-null object
8	SG_UE	504674	non-null string
9	DS_NR_TITULO_ELEITOR	504674	non-null string
10	CD_NUM_CPF	504674	non-null string
11	CD_NUM_CNPJ	503759	non-null string
12	NR_PARTIDO	504674	non-null int64
13	SG_PARTIDO	504674	non-null object
14	VR_RECEITA	504674	non-null float64
15	DT_RECEITA	504674	non-null object
16	DS_TITULO	504674	non-null object
17	CD_ORIGEM RECEITA	504674	non-null int64
18	DS_ESP_RECURSO	504674	non-null object
19	CD_ESP_RECURSO	504674	non-null int64
20	NM_DOADOR	504674	non-null object
21	CD_CPF_CNPJ_DOADOR	494009	non-null string
22	SG_UE_SUPERIOR_1	207351	non-null object
23	NM_UE_1	207351	non-null object
24	SG_UE_1	207351	non-null string
25	SITUACAO_CADASTRAL	504674	non-null object
26	NM ADM	267486	non-null object
27	CD_CPF ADM	267486	non-null string

Marcamos em amarelo a coluna usada para junção com o *dataset* principal e em azul as colunas que nos interessam. Os “tipos” de receitas existentes no *dataset*, em 2008 são estes:

```
# relação de códigos e descrições de receitas para o ano 2008
rec08[['CD_ORIGEM RECEITA','DS_TITULO']].drop_duplicates()
```

CD_ORIGEM RECEITA	DS_TITULO
241921	10010200 Recursos de pessoas físicas
241922	10010100 Recursos próprios
241931	10040000 RECURSOS DE OUTROS CANDIDATOS/COMITÉS
241957	10010300 Recursos de pessoas jurídicas
242212	10020000 RECURSOS DE PARTIDO POLÍTICO
244403	10030300 Recursos de origens não identificadas
246527	10030200 Rendimentos de aplicações financeiras
274567	10050000 Descrição das doações relativas à comercialização

Entendemos que talvez seja válido trazer para o *dataset* principal as receitas totalizadas por candidato e por origem. Assim, definimos algumas “classes” de receitas para levar ao *dataframe* principal, cuja composição foi assim definida:

- **Receitas próprias**, cód. receita (10010100)
- **Receitas de pessoas físicas**, cód. receita (10010200)
- **Receitas de pessoas jurídicas**, cód. receita (10010300)
- **Receitas de partidos ou de candidatos**, cód. rec. (10020000,10040000)
- **Receitas outras**, cód. receita (10030200,10030300,10050000)

Para trazermos estas 5 novas colunas para o nosso *dataset* principal, inicialmente fizemos a leitura da base de receitas.

```
# função para leitura do arquivo de receitas do candidato
def le_receitas(ano, uf):
    # nome do arquivo zip do TSE a ser lido
    nome_zip = 'tse/contas/prestacao_contas_{}{}'.format(str(ano), '.zip')
    tipos = {'SG_UE':'string', 'CD_NUM_CPF':'string', 'CD_NUM_CNPJ':'string', 'DS_NR_TITULO_ELEITOR':'string',
             'CD_CPF_CNPJ_DODADOR':'string', 'SG_UE_1':'string', 'CD_CPF_ADM':'string', 'CPF/CNPJ do doador':'string'}
    zf = ZipFile(nome_zip)
    # padroniza nome colunas que serão usadas na junção
    rencols = {'SEQUENCIAL_CANDIDATO':'SQ_CANDIDATO', 'Sequencial_Candidato':'SQ_CANDIDATO',
               'Valor_receita':'VR_RECEITA', 'CD_TITULO':'CD_ORIGEM_RECEITA'}

    # nome do arquivo dentro do zip
    arq = 'receitas_candidatos_{}{}'.format(str(ano), '_brasil.csv')

    # leitura do csv por partes devido ao seu tamanho, já filtrando apenas o estado de nosso interesse
    iter_csv = pd.read_csv(zf.open(arq), sep=';', encoding='latin_1', dtype=tipos, iterator=True, chunksize=2000)
    df = pd.concat([chunk[chunk['SG_UE_SUPERIOR'] == uf] for chunk in iter_csv])

    df.rename(columns=rencols, inplace=True)
    # substitui o marcador de centavos (vírgula por ponto) na coluna "valor da receita"
    df['VR_RECEITA'] = df['VR_RECEITA'].str.replace(',', '.')
    # altera tipo para float
    df['VR_RECEITA'] = df['VR_RECEITA'].astype(float)
    return df
```

Aqui cabe um comentário adicional: o arquivo de receitas para o ano de 2008 não está separado por estado. Portanto, estão nele todas as receitas de todos os candidatos do Brasil no ano de 2008 – um total de 700Mb de arquivo csv. Assim, fizemos a leitura do arquivo .csv por partes (*chunks*) e já filtrando pelo estado de nosso interesse. O próprio comando *pd.read_csv* já tem a opção de fazer a leitura por partes, bastando para isso definir “*iterator = True*” e especificar o tamanho do “*chunksize*”, como pode ser visto na nossa função de leitura acima. A seguir o código para leitura da base de receitas, usando a função, bem como a dimensão da base de receitas (linhas x colunas):

In [118]: `rec08 = le_receitas(2008, 'SP')`

In [119]: `rec08.shape`

Out[119]: `(504674, 28)`

O passo seguinte foi agregar o *dataframe* por candidato e código de receita e juntar os códigos de receita para gerar as classes de receita que definimos anteriormente e resumidas na tabela a seguir:

Nome da classe de receitas	Códigos de receita que compõem a classe
Receitas próprias	10010100
Receitas de pessoas físicas	10010200
Receitas de pessoas jurídica	10010300
Receitas de partidos, comitês ou candidatos	10020000 e 10040000
Outras receitas	10030200, 10030300 e 10050000

Para isto criamos a função a seguir, devidamente comentada

```
def cria_receita(nome_col,lst_cod_rec):
    # colunas de interesse no dataset de receitas
    cols = ['SQ_CANDIDATO', 'VR_RECEITA']

    # agrupa (soma) as receitas por candidato e código de receita
    temp = rec08.groupby(['SQ_CANDIDATO','CD_ORIGEM_RECEITA'], as_index=False).agg({'VR_RECEITA':'sum'})

    # filtra apenas os códigos de receita escolhidos(no parametro da função) e agrupa (soma) por candidato
    df = temp.query('CD_ORIGEM_RECEITA in @lst_cod_rec').groupby('SQ_CANDIDATO', as_index=False).sum('VR_RECEITA')[[

        # renomeia a coluna valor da receita para o nome definido na função (parametro)
        df.rename(columns = {'VR_RECEITA':nome_col}, inplace=True)
    return df
```

E então, chamando a função, criamos um *dataframe* para cada classe de receitas que havíamos definido:

```
r_prop = cria_receita('REC_PROP',(10010100,))
r_pf = cria_receita('REC_PF',(10010200,))
r_pj = cria_receita('REC_PJ',(10010300,))
r_partcand = cria_receita('REC_PARTCAND',(10020000,10040000))
r_out = cria_receita('REC_OUTRAS',(10030200,10030300,10050000))
```

A título de exemplo, segue amostra de conteúdo de um desses *dataframes* (o de receitas próprias):

```
In [134]: r_prop.sample(3)
```

```
Out[134]:
      SQ_CANDIDATO  REC_PROP
24416          48812    900.58
23252          46341   1,020.80
24227          48464    600.00
```

3 rows × 2 columns

E por fim, agregamos as novas colunas ao dataset principal de candidatos:

```
# agregando as novas colunas a base de dados de candidatos
cd08 = cd08.merge(r_prop, on=['SQ_CANDIDATO'], how='left')
cd08 = cd08.merge(r_pf, on=['SQ_CANDIDATO'], how='left')
cd08 = cd08.merge(r_pj, on=['SQ_CANDIDATO'], how='left')
cd08 = cd08.merge(r_partcand, on=['SQ_CANDIDATO'], how='left')
cd08 = cd08.merge(r_out, on=['SQ_CANDIDATO'], how='left')
```

3.2.5. Base de dados de despesas do candidato

Também do tema “prestação de contas”, do repositório do TSE, usaremos a base de dados de despesas do candidato na campanha.

O processo de leitura, assim como nas receitas, foi também feito em partes e filtrando o estado de nosso interesse, em virtude do tamanho do arquivo de origem: 1,4Gb. A função para leitura dos dados e o tamanho (linhas x colunas) após a leitura, encontra-se a seguir:

```
# função para leitura dos arquivos de despesas dos candidatos
def le_despesas(ano, uf):
    nome_zip = 'tse/contas/prestacao_contas_' + str(ano) + '.zip'
    tipos = {'ANO_ELEICAO': 'string', 'CD_NUM_CNPJ': 'string', 'CD_TIPO_DOCUMENTO': 'string',
             'CD_CPF_CNPJ_FORNECEDOR': 'string', 'SG_UE_1': 'string', 'CD_CPF ADM': 'string',
             'Número do documento': 'string', 'CPF/CNPJ do fornecedor': 'string'}
    # padroniza nome colunas que serão usadas na junção
    rencols = {'SEQUENCIAL_CANDIDATO': 'SQ_CANDIDATO', 'Sequencial Candidato': 'SQ_CANDIDATO',
               'Valor despesa': 'VR_DESPESA', 'VR_DESPESA CONTRATADA': 'VR_DESPESA'}
    zf = ZipFile(nome_zip)

    arq = 'despesas_candidatos_' + str(ano) + '_brasil.csv'
    iter_csv = pd.read_csv(zf.open(arq), sep=';', encoding='latin 1', dtype=tipos, iterator=True, chunksize=2000)
    df = pd.concat([chunk[chunk['SG_UE_SUPERIOR'] == uf] for chunk in iter_csv])
    df.rename(columns=rencols, inplace=True)
    # ajusta coluna de valores de str para float
    df['VR_DESPESA'] = df['VR_DESPESA'].str.replace(',', '.')
    df['VR_DESPESA'] = df['VR_DESPESA'].astype(float)
    return df

: desp08 = le_despesas(2008, 'SP')

: desp08.shape
: (945412, 29)
```

Deste *dataset*, resolvemos trazer apenas o valor total de despesas por candidato e, para tal, bastou uma agregação com soma no código sequencial do candidato e posterior agregação da coluna ao *dataset* principal:

```
# despesas totais
cols = ['SQ_CANDIDATO', 'VR_DESPESA']
d08_tot = desp08.groupby('SQ_CANDIDATO', as_index=False).sum('VR_DESPESA')[cols]

# agregando as novas colunas a base de dados de candidatos
cd08 = cd08.merge(d08_tot, on=['SQ_CANDIDATO'], how='left')
```

3.2.6. Base de dados de resultados da eleição anterior

Outra base disponibilizada pelo TSE, no tema “resultados”, e que usamos em nosso trabalho, foi a base de resultados das eleições anteriores – “Votação nominal por município e zona” do ano 2004.

Nosso objetivo, ao usar esta base, foi trazer informações de como cada partido se comportou nas eleições anteriores para vereador. Assim, as informações que iremos extrair do *dataset* são:

- quantidade de votos para vereador, por município, que o partido recebeu em 2004 e
- número de vereadores eleitos por município do partido.

Assim como nos outros *datasets*, criamos uma função para leitura do mesmo. Nesta função já separamos apenas as colunas que nos interessam e já criamos uma nova coluna com a informação se o candidato foi eleito ou não:

```
# função para leitura dos resultados da eleição anterior
def le_eleicaoanterior(ano, uf):
    # definição dos nomes de colunas para os anos em o arquivo do TSE não os nomes na 1a linha
    c08 = ['DT_GERACAO', 'HH_GERACAO', 'ANO_ELEICAO', 'NR_TURNO', 'DS_ELEICAO', 'SG_UF', 'SG_UE', 'CD_MUN',
           'NM_UE', 'NR_ZONA', 'CD_CARGO', 'NR_CAND', 'SQ_CANDIDATO', 'NM_CAND', 'NM_URNA_CAND', 'DS_CARGO',
           'CD_SIT_CAND_SUP', 'DS_SIT_CAND_SUP', 'CD_SIT_REG_CAND', 'DS_SIT_REF_CAND', 'CD_SIT_T_CAND',
           'DS_SIT_T_CAND', 'NR_PARTIDO', 'SG_PARTIDO', 'NM_PARTIDO', 'SQ_LEGENDA', 'NM_LEGENDA', 'COMP_LEG', 'TOT_VOTOS']
    # nome do arquivo zipado baixado do site TSE
    nome_zip = 'tse/resultados/votacao_candidato_munzona_' + str(ano) + '.zip'
    zf = ZipFile(nome_zip)
    # nome do arquivo final dentro do zip
    arq = 'votacao_candidato_munzona' + str(ano) + ' ' + uf + '.txt'
    # padroniza nome colunas que serão usadas na junção ou em algum cômputo
    rencols = {'DS_SIT_T_CAND': 'DS_SIT_TOT_TURNO', 'SQ_LEGENDA': 'SQ_COLIGACAO', 'QT_VOTOS_NOMINAIS': 'TOT_VOTOS'}
    # definição das colunas agregadoras, conforme ano do arquivo
    ag08 = ['ANO_ELEICAO', 'NR_TURNO', 'DS_ELEICAO', 'SG_UF', 'SG_UE', 'NM_UE', 'CD_CARGO',
            'NR_PARTIDO', 'SQ_CANDIDATO', 'SQ_COLIGACAO', 'ELEITO']
    # faz a leitura do arquivo para dataframe
    df = pd.read_csv(zf.open(arq), sep=';', encoding='latin_1', dtype=tipos, names=c08)

    # ajustes conteúdo de coluna que será usada como agregador
    df['DS_ELEICAO'] = df['DS_ELEICAO'].str.title()

    # ajustes de nome de coluna que será usada como cômputo
    df.rename(columns=rencols, inplace=True)

    # cria coluna 'ELEITO' valores 1 para sim e 0 para não
    df['ELEITO'] = np.where((df['DS_SIT_TOT_TURNO'] == "ELEITO") |
                           (df['DS_SIT_TOT_TURNO'] == "MÉDIA") |
                           (df['DS_SIT_TOT_TURNO'] == "ELEITO POR QP") |
                           (df['DS_SIT_TOT_TURNO'] == "ELEITO POR MEDIA"), 1, 0)

    # tabela de agregação temporária
    df1 = df.groupby(ag08, as_index=False).agg({'TOT_VOTOS': 'sum'})

    # colunas para agregação final
    agfim = ['ANO_ELEICAO', 'NR_TURNO', 'DS_ELEICAO', 'SG_UF', 'SG_UE', 'CD_CARGO', 'NR_PARTIDO']

    #colsfim = ['NR_TURNO', 'SG_UF', 'SG_UE', 'CD_CARGO', 'NR_PARTIDO', 'TOT_VOTOS', 'ELEITO']
    df2 = df1.groupby(agfim, as_index=False).agg({'TOT_VOTOS': 'sum', 'ELEITO': 'sum'})[colsfim]
    df2.rename(columns={'TOT_VOTOS': 'VOTOS_PART_ELEC_ANT', 'ELEITO': 'QT_ELEITOS_ELEC_ANT'}, inplace=True)
    return df2
```

A seguir uma amostra da base de dados após a leitura, marcado em amarelo as colunas usadas na junção e em azul as que levaremos para o dataset principal:

```
: leg04 = le_eleicaoanterior(2004, 'SP')

: leg04.sample(5)
```

NR_TURNO	SG_UF	SG_UE	CD_CARGO	NR_PARTIDO	VOTOS_PART_ELEC_ANT	QT_ELEITOS_ELEC_ANT
3844	1	SP	64777	13	19	21773
3665	1	SP	64572	13	22	664
1267	1	SP	61905	13	13	455
2169	1	SP	62693	13	27	415
9293	1	SP	71650	13	12	209

E a junção com o *dataset* principal:

```
: # agraga dados de quantidades de votos do partido e eleitos pelo partido na legislatura atual
cd08 = cd08.merge(leg04, on=['NR_TURNO', 'SG_UF', 'SG_UE', 'CD_CARGO', 'NR_PARTIDO'], how='left')
```

3.2.7. Tratamento de *Nan* nas colunas adicionadas ao *dataset* principal

A seguir, as colunas que foram acrescentadas ao nosso *dataset*. Já havíamos tratado, de forma específica os *Nan* das colunas referentes as coligações e a coluna quantidade de vagas, por óbvio, não poderia ter *Nan*. Assim, resta ainda tratar os valores omissos nas colunas destacadas em amarelo:

38	TP_AGREMIACAO	59536	non-null	object
39	SQ_COLIGACAO	59536	non-null	int64
40	NM_COLIGACAO	59536	non-null	object
41	COMP_COLIGACAO	59536	non-null	object
42	QT_PART_COL	59536	non-null	int64
43	QT_BENS	37914	non-null	float64
44	VL_TOT_BENS	37914	non-null	float64
45	QT_VAGAS	59536	non-null	int64
46	REC_PROP	32816	non-null	float64
47	REC_PF	24996	non-null	float64
48	REC_PJ	8508	non-null	float64
49	REC_PARTCAND	44839	non-null	float64
50	REC_OUTRAS	528	non-null	float64
51	VR_DESPESA	51921	non-null	float64
52	VOTOS_PART_ELEC_ANT	51816	non-null	float64
53	QT_ELEITOS_ELEC_ANT	51816	non-null	float64

Trataremos assim os *Nan*:

- Colunas referentes aos bens de candidatos (QT_BENS e VL_TOT_BENS), colunas referentes a valores de receitas de campanha (REC_PROP, REC_PF, REC_PJ, REC_PARTCAND e REC_OUTRAS) e valores de

despesas de campanha (VR_DESPESA): considerar *NaN* como ZERO, sem prejuízo para a análise pois trata de valores informados pelos candidatos e os campos *NaN* indicam que o candidato não informou valores, logo, seria de fato zero.

- Colunas referentes a votação do partido nas eleições anteriores (VOTOS_PART_ELEC_ANT e QT_ELEITOS_ELEC_ANT): em tese são, de fato, partidos que não tiveram votos e nem candidatos eleitos na eleição anterior. Portanto, imputaremos valor ZERO.

Em resumo, todas as colunas que ainda constem *NaN* podem ser preenchidas com ZERO. A seguir código para tratar *NaN* e ajustar colunas *integer*:

```
: # substituindo NaN por ZERO no dataframe
cd08.fillna(0, inplace=True)

: # ajustando tipo das colunas VOTOS_PART_CARGO, QT_ELEITOS_CARGO e qd_bens para integer
cd08['VOTOS_PART_ELEC_ANT'] = cd08['VOTOS_PART_ELEC_ANT'].astype(int)
cd08['QT_ELEITOS_ELEC_ANT'] = cd08['QT_ELEITOS_ELEC_ANT'].astype(int)
cd08['QT_BENS'] = cd08['QT_BENS'].astype(int)
```

3.3. Outros ajustes no dataset

3.3.1. Exclusão de colunas desnecessárias

Passada a etapa de enriquecimento da base de dados, vamos excluir colunas que estão preenchidas com apenas um único valor para todas as linhas e que não tem mais nenhuma serventia para nossa análise. A seguir, as colunas nesta situação:

```
In [165]: for col in cd08.columns:
    if cd08[col].nunique() == 1:
        print(col, cd08[col].unique())

DT_GERACAO ['06/10/2020']
HH_GERACAO ['14:56:30']
ANO_ELEICAO [2008]
NR_TURNO [1]
DS_ELEICAO ['Eleições 2008']
SG_UF ['SP']
CD_CARGO [13]
DS_CARGO ['VEREADOR']
```

Fizemos a exclusão pelo comando ***drop***:

```
: # exclusão colunas desnecessárias
rv08 = cd08.drop(columns=['DT_GERACAO', 'HH_GERACAO', 'ANO_ELEICAO', 'NR_TURNO',
                         'DS_ELEICAO', 'SG_UF', 'CD_CARGO', 'DS_CARGO'])
```

3.3.2. Exclusão de registros de candidaturas não confirmadas

Para nossa análise, não interessa registros de candidaturas inválidos. Na base de dados são 2 as colunas que trazem esta situação:

- DS_SITUACAO_CANDIDATURA e
- DS_SIT_TOT_TURNO

Inicialmente, identificamos quais os tipos de situação de candidatura presentes no *dataset* e fizemos a primeira “limpeza” selecionando apenas as candidaturas em situação “***deferido***”. A seguir, as linhas de código que fazem esta primeira filtragem:

```
In [213]: # tipos de "situação de candidatura"
rv08['DS_SITUACAO_CANDIDATURA'].unique()

Out[213]: array(['DEFERIDO', 'RENÚNCIA', 'INDEFERIDO', 'INELEGÍVEL',
                 'DEFERIDO COM RECURSO', 'INDEFERIDO COM RECURSO', 'FALECIDO',
                 'CANCELADO', 'CASSADO', 'NÃO CONHECIMENTO DO PEDIDO'], dtype=object)
```

```
In [214]: # quantidade de registros e colunas do dataset
rv08.shape

Out[214]: (59536, 46)
```

```
In [215]: # nos interessa apenas os registros deferidos ("DEFERIDO" e "DEFERIDO COM RECURSO")
rv08 = rv08.query('DS_SITUACAO_CANDIDATURA in ("DEFERIDO COM RECURSO", "DEFERIDO")')
```

```
In [216]: # quantidade de registros e colunas do dataset após exclusão das candidaturas indeferidas
rv08.shape

Out[216]: (56503, 46)
```

Em seguida, verificamos ainda os tipos de situação que aparecem na coluna DS_SIT_TOT_TURNO:

```
In [219]: # tipos de "situação de totalização ao final do turno"
rv08['DS_SIT_TOT_TURNO'].unique()

Out[219]: array(['NÃO ELEITO', 'SUPLENTE', 'ELEITO', 'MÉDIA',
                 'INDEFERIDO COM RECURSO'], dtype=object)
```

Mesmo após a limpeza inicial ainda aparecem registros com situação “indeferida”, segundo a informação desta coluna.

São apenas 3 registros, mas de toda forma, entendemos que os mesmos devem também ser excluídos de nossa base, pois, s.m.j., tratam de registros indeferidos. A seguir, a codificação que identifica estes registros e que faz a exclusão dos mesmos de nossa base:

```
In [220]: # nos interessa apenas as situações de candidaturas válidas
# devemos excluir também as situações de 'INDEFERIDO COM RECURSO'
rv08.query('DS_SIT_TOT_TURNO == "INDEFERIDO COM RECURSO"')

Out[220]:
SG_UE NM_UE NM_CANDIDATO SQ_CANDIDATO NR_CANDIDATO NR_CPF_CANDIDATO NM_URNACAO_CANDIDATO CD_SITUACAO_CANDIDATURA
14899 64017 EMBU ANTONIO BIZERRA NETO 22180 25619 01401158897 TONHÃO DA PIZZARIA 2
14960 64017 EMBU GENILDO RODRIGUES DA SILVA 36932 40602 40176061487 PÉ DE BOI 2
30040 66753 MARABÁ DOMINGOS PAULISTA PEROSO NETO 54535 45111 72555599800 DOMINGOS PEROSO 2

3 rows × 46 columns
```

```
In [223]: rv08 = rv08.query('DS_SIT_TOT_TURNO in ("NÃO ELEITO", "SUPLENTE", "ELEITO", "MÉDIA")')

In [224]: # quantidade de registros e colunas do dataset após exclusão das candidaturas indeferidas
rv08.shape
```

```
Out[224]: (56500, 46)
```

3.3.3. Criação da coluna “target”

Um dos objetivos finais de nossa análise é avaliar como se comportam algoritmos de aprendizado de máquina com intuito de identificar candidatos com maiores chances de serem eleitos. Trata-se, portanto, de aprendizado supervisionado e, para isto, precisamos ter uma coluna de resultado (ou *target*). No nosso caso, esta coluna deve dizer se candidato foi eleito ou não. A situação de candidato eleito em 2008 pode aparecer como “ELEITO” ou “MÉDIA”. Em outros, anos pode também aparecer como “ELEITO POR MÉDIA” ou “ELEITO POR QP”. Segue código para gerar coluna resultado (*target*), imputando 1 para candidato “eleito” e 0 para “não eleito”, independente do ano da eleição.

```
: # cria coluna alvo: 'RESULTADO' com valores 1 para ELEITO (sim) e 0 para NÃO ELEITO (não)
rv08['RESULTADO'] = np.where((rv08['DS_SIT_TOT_TURNO'] == "ELEITO") |
                             (rv08['DS_SIT_TOT_TURNO'] == "MÉDIA") |
                             (rv08['DS_SIT_TOT_TURNO'] == "ELEITO POR QP") |
                             (rv08['DS_SIT_TOT_TURNO'] == "ELEITO POR MÉDIA"), 1, 0)
```

3.3.4. Exclusão de colunas redundantes ou sem utilidade preditiva

Ainda temos em nosso *dataset* diversas colunas redundantes e colunas sem nenhuma finalidade preditiva. Por exemplo, temos diversas colunas que apenas tem função de identificação do candidato: numero sequencial do candidato, número eleitoral do candidato, nome do candidato, nome de urna, numero cpf, número eleitor. Outro exemplo são diversos “pares de colunas” redundantes, uma com o código e outra com a descrição da característica. Especificamente nestas colunas, mantivemos, para fins de análise exploratória, as colunas que contêm a descrição. Já para a etapa de aprendizado de máquina, mantivemos as colunas com os códigos. A seguir resumimos as colunas existentes em nossa base de dados e destacamos em amarelo as que mantivemos para a análise exploratória:

#	Coluna	tipo	observação
0	SG_UE	categorica	ok
1	NM_UE		redundante
2	NM_CANDIDATO		redundante
3	SQ_CANDIDATO	chave identificadora	ok
4	NR_CANDIDATO		redundante
5	NR_CPF_CANDIDATO		redundante
6	NM_URNA_CANDIDATO		redundante
7	CD_SITUACAO_CANDIDATURA		redundante
8	DS_SITUACAO_CANDIDATURA	categorica	ok
9	NR_PARTIDO		redundante
10	SG_PARTIDO	categorica	ok
11	NM_PARTIDO		redundante
12	CD_OCUPACAO		redundante
13	DS_OCUPACAO	categorica	ok
14	DT_NASCIMENTO		redundante
15	NR_TITULO_ELEITORAL_CANDIDATO		redundante
16	IDADE_DATA_POSSE	num_int	ok
17	CD_GENERO		redundante
18	DS_GENERO	categorica	ok
19	CD_GRAU_INSTRUCAO		redundante
20	DS_GRAU_INSTRUCAO	categorica ordinal	ok
21	CD_ESTADO_CIVIL		redundante
22	DS_ESTADO_CIVIL	categorica	ok
23	CD_NACIONALIDADE		redundante
24	DS_NACIONALIDADE	categorica	ok
25	CD_SIT_TOT_TURNO		redundante
26	DS_SIT_TOT_TURNO	categorica	ok
27	SG_UF_NASCIMENTO	categorica	ok
28	NM_MUNICIPIO_NASCIMENTO		redundante
29	CD_MUNICIPIO_NASCIMENTO	categorica	ok
30	TP_AGREMIACAO	categorica	ok
31	SQ_COLIGACAO		redundante
32	NM_COLIGACAO		redundante
33	COMP_COLIGACAO		ok
34	QT_PART_COL	num_int	ok
35	QT_BENS	num_int	ok
36	VL_TOT_BENS	num_float	ok
37	QT_VAGAS	num_int	ok
38	REC_PROP	num_float	ok
39	REC_PF	num_float	ok
40	REC_PJ	num_float	ok
41	REC_PARTCAND	num_float	ok
42	REC_OUTRAS	num_float	ok
43	VR_DESPESA	num_float	ok
44	VOTOS_PART_ELEC_ANT	num_int	ok
45	QT_ELEITOS_ELEC_ANT	num_int	ok
46	RESULTADO	target	ok

4. Análise e Exploração dos Dados

Na análise exploratória utilizamos, além de gráficos gerados “manualmente” no jupyter notebook, a biblioteca SweetViz que gera uma análise geral do dataset com opção de saída em html (arquivo html está no github do projeto).

```
# gera arquivo html com análise exploratória dos dados deste dataset, usando biblioteca sweetviz
testeviz = sv.analyze(base)
testeviz.show_html('base.html')
```

Done! Use 'show' commands to display/save.

[100%] 00:01 -> (00:00 left)

Report base.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop up, regardless, the report IS saved in your notebook/colab files.

A seguir, as informações mais relevantes obtidas durante esta etapa:

4.1. Candidaturas por municípios

Nosso dataset trata informações de candidaturas de todos os municípios do estado de São Paulo. Ao todo foram 56.600 candidaturas válidas (passíveis de serem votadas) em 645 municípios, sendo que só o município de São Paulo (capital) teve 1.060 candidatos ao cargo de vereador em 2008 (informação gerada pelo pacote *DataViz*):



4.2. Candidatos por vagas

A relação “quantidade de candidatos” por “vaga” varia bastante dentre os municípios. O município de Guarulhos foi o que teve mais candidatos por vagas em 2008 (24,74), enquanto o município de Itaju, foi o que teve menos (1,56 candidatos por vaga). Vide código e saída a seguir:

```
# agrupa por UE e retorna a qt. de candidaturas e de vagas por UE
tt = base.groupby('NM_UE', as_index=False).agg({'QT_VAGAS':[ 'count', 'max']})
# exclui o nível 0 das colunas (groupby cria colunas multiindexadas)
tt.columns = tt.columns.droplevel(0)
# renomeia colunas
tt.columns = ['municipio', 'qt_candidatos', 'qt_vagas']
# cria coluna com a relação candidatos por vagas
tt['cand_por_vagas'] = tt['qt_candidatos'] / tt['qt_vagas']
# ordena lista por 'cand_por_vagas'
tt.sort_values('cand_por_vagas', ascending=False)
```

	municipio	qt_candidatos	qt_vagas	cand_por_vagas
204	GUARULHOS	841	34	24.74
633	VOTORANTIM	240	11	21.82
554	SÃO BERNARDO DO CAMPO	456	21	21.71
381	OSASCO	448	21	21.33
629	VINHEDO	211	10	21.10
...
331	MIRA ESTRELA	18	9	2.00
172	FERNANDO PRESTES	16	9	1.78
183	GABRIEL MONTEIRO	16	9	1.78
632	VITÓRIA BRASIL	15	9	1.67
245	ITAJU	14	9	1.56

645 rows × 4 columns

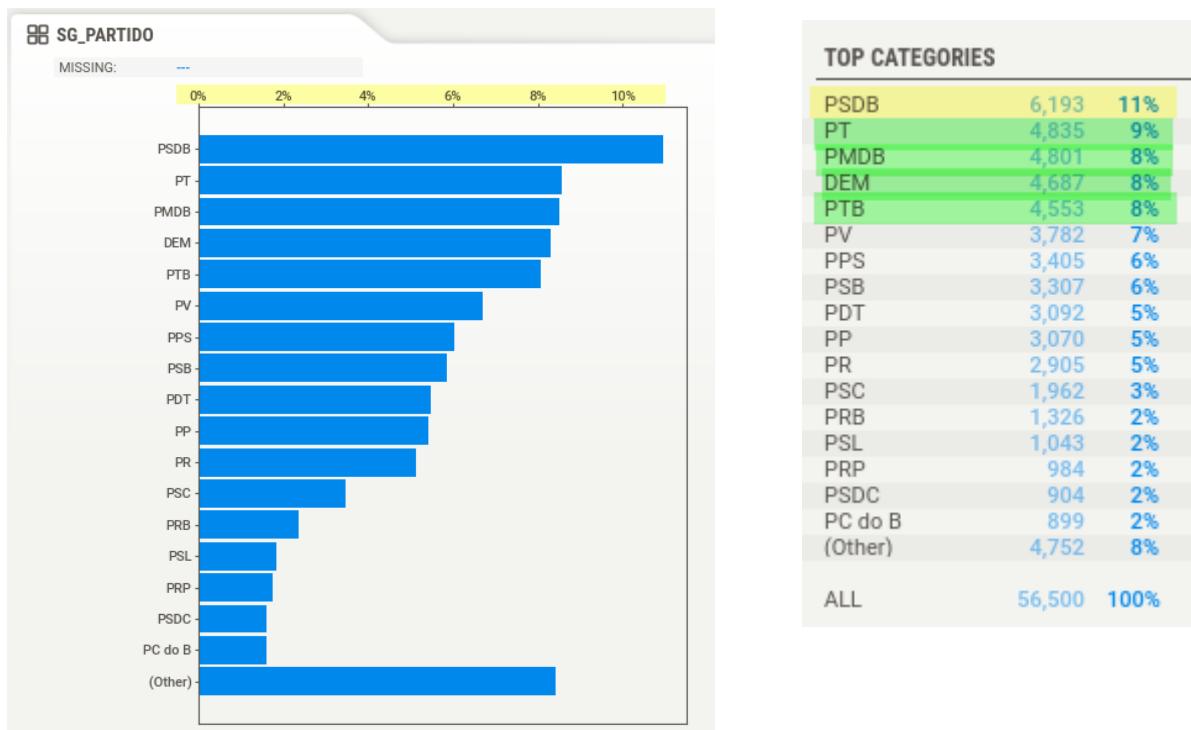
As estatísticas básicas da relação candidato / vaga mostram uma média de 8,34 candidatos por vagas e desvio padrão de 4,46.

```
# estatísticas básicas de candidatos e vagas por município
tt.describe()
```

	qt_candidatos	qt_vagas	cand_por_vagas
count	645.00	645.00	645.00
mean	87.60	9.75	8.34
std	82.41	2.82	4.46
min	14.00	9.00	1.56
25%	42.00	9.00	4.67
50%	67.00	9.00	7.44
75%	100.00	9.00	10.89
max	1,060.00	55.00	24.74

4.3. Candidaturas por partidos:

Ao todo, 27 partidos lançaram candidatos a vereador no estado de São Paulo em 2008, sendo PSBD aquele que mais lançou candidatos (mais de 6 mil). PT, PMDB, DEM e PTB estão logo em seguida com números bem próximos entre si.



4.4. Ocupação dos candidatos

Ao todo, 277 ocupações distintas foram informadas pelos 56.500 candidatos do nosso dataset. A ocupação que mais aparece é “outros”, com 15% do total. Logo em seguida, “comerciante” com 11% e “Servidor Público Municipal” com 9%:

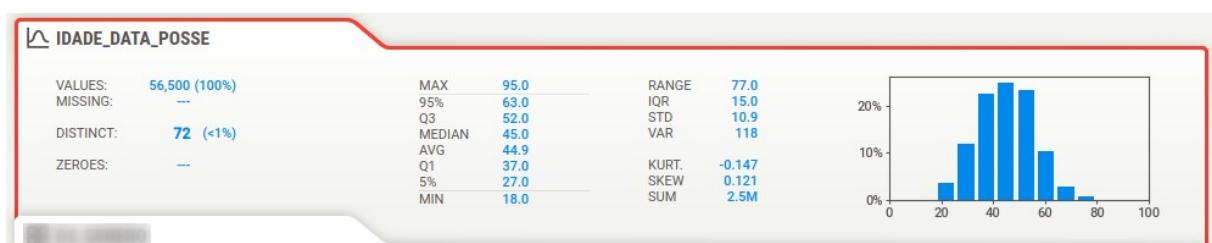


Somente a partir de 2014 que o TSE incluiu uma coluna específica para o candidato informar se era “candidato a reeleição”. Como nossa base de dados é referente ao ano de 2008, não temos esta informação de forma direta. Porém, da análise das ocupações conseguimos identificá-la de forma indireta, já que existe a ocupação vereador dentre as opções disponibilizadas pelo TSE. No nosso *dataset*, 3% dos candidatos informaram ocupação “vereador”, o que nos leva a crer que são candidatos à reeleição:

DS_OCUPACAO		
MISSING:	---	
8,203	15%	OUTROS
6,118	11%	COMERCIANTE
4,869	9%	SERVIDOR PÚBLICO MUNICIPAL
2,281	4%	APOSENTADO (EXCETO SERVIDOR PÚBLICO)
2,141	4%	EMPRESÁRIO
1,877	3%	DONA DE CASA
1,655	3%	PROFESSOR DE ENSINO MÉDIO
1,549	3%	ADVOGADO
1,548	3%	AGRICULTOR
1,467	3%	VEREADOR
1,191	2%	SERVIDOR PÚBLICO ESTADUAL
1,178	2%	MOTORISTA DE VEÍCULOS DE TRANSPORTE COLETIVO DE PASSAGEIROS
1,031	2%	PROFESSOR DE ENSINO FUNDAMENTAL
932	2%	TRABALHADOR RURAL
872	2%	MOTORISTA DE VEÍCULOS DE TRANSPORTE DE CARGA
731	1%	VENDEDOR DE COMÉRCIO VAREJISTA E ATACADISTA
658	1%	AUXILIAR DE ESCRITÓRIO E ASSEMELHADOS
653	1%	CABELEIREIRO E BARBEIRO
635	1%	ADMINISTRADOR
632	1%	MÉDICO
597	1%	TRABALHADOR DE CONSTRUÇÃO CIVIL
559	<1%	COMERCIÁRIO

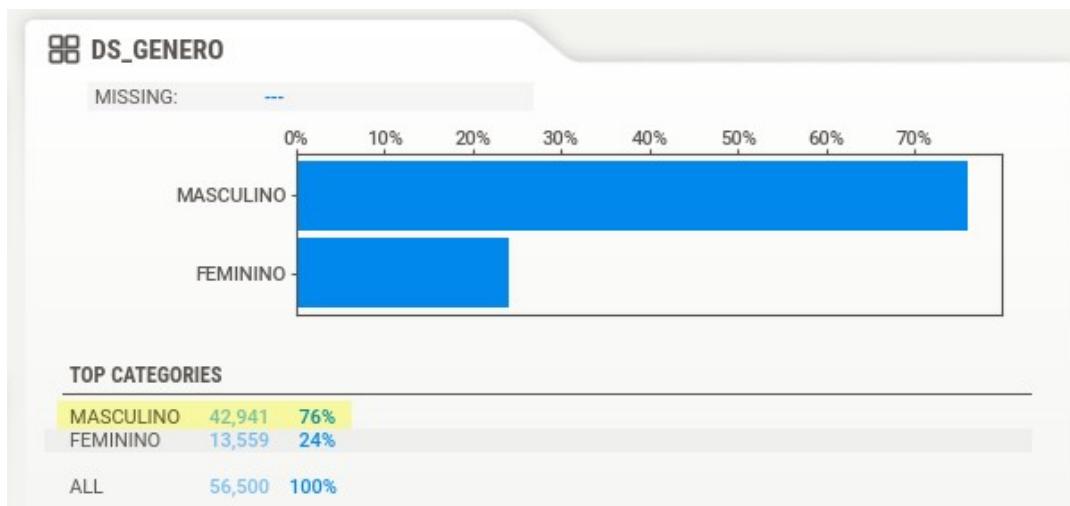
4.5. Distribuição da idade dos candidatos

A idade dos candidatos na data da posse tem uma distribuição que se aproxima da curva “normal”. A média de idade é de 44,9 anos e o desvio padrão 10,9 anos. Os valores máximos e mínimos são 95 e 18 anos e 90% dos candidatos tem idade entre 27 e 63.



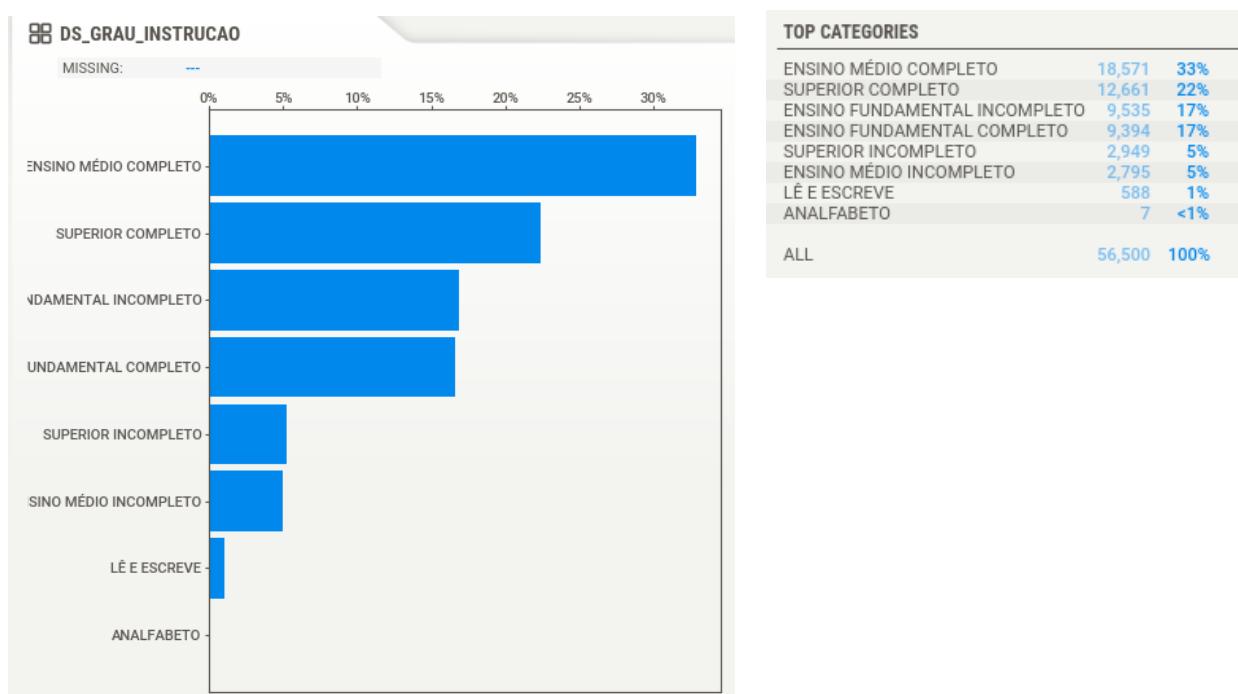
4.6. Sexo dos candidatos

A grande maioria (mais de ¾) dos candidatos a vereador, no estado de São Paulo em 2008, era do sexo masculino, mostrando reflexos históricos de nossa sociedade patriarcal.



4.7. Grau de instrução dos candidatos

Trinta e três por cento (33%) dos candidatos informaram ter segundo grau completo e 22% informaram curso superior completo. Portanto, mais de 50% dos candidatos estão nessas duas classificações. Pouco mais de 1% dos candidatos se declarou “analfabeto” ou analfabeto funcional (lê e escreve).



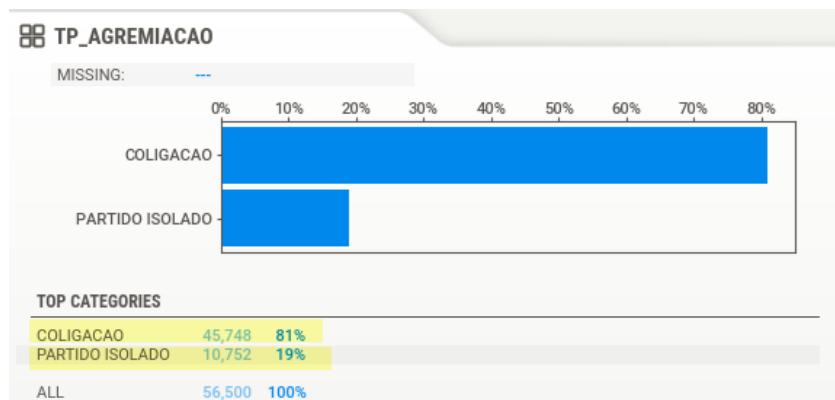
4.8. Nacionalidade, Estado e Município de nascimento

A grande maioria (99%) dos candidatos são Brasileiros natos, quase 80% são Paulistas e 8% são Paulistanos.

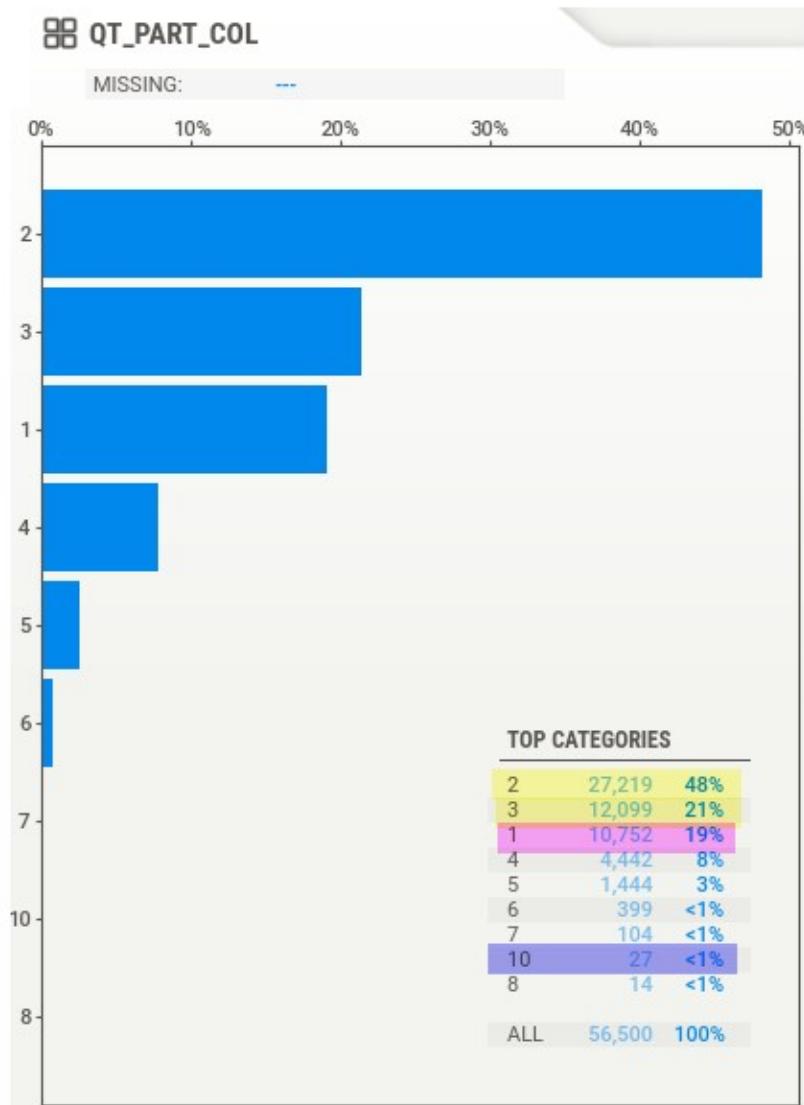


4.9. Coligações partidárias

As três colunas de nosso *dataset* que tratam de coligações partidárias trazem informações bem peculiares. Apenas 19% das candidaturas não fazem parte de nenhuma coligação (81% fazem parte de coligações).



A maioria das candidaturas são de coligações com 2 partidos (48%) ou com 3 partidos (21%). Dezenove por cento (19%) das candidaturas são de partidos isolados (“coligação” de apenas 1 partido, sinônimo de partido sem coligação) e dez (10) é o número máximo de partidos em uma coligação.



Ao todo são mais de 1.300 coligações distintas nos municípios de SP. Este número absurdo de coligações reflete um pouco do nosso “modelo eleitoral”, no qual é permitido que um partido faça coligações variadas em cada um dos municípios.

COMP_COLIGACAO			
VALUES:	56,500	(100%)	
MISSING:	---		
DISTINCT:	1,381	(2%)	
	10,752	19%	#NULO#
	416	<1%	45-PSDB / 25-DEM
	385	<1%	15-PMDB / 25-DEM
	375	<1%	45-PSDB / 23-PPS
	351	<1%	13-PT / 40-PSB
	340	<1%	45-PSDB / 15-PMDB
	323	<1%	11-PP / 25-DEM
	43,558	77%	(Other)

4.10. Bens declarados pelos candidatos

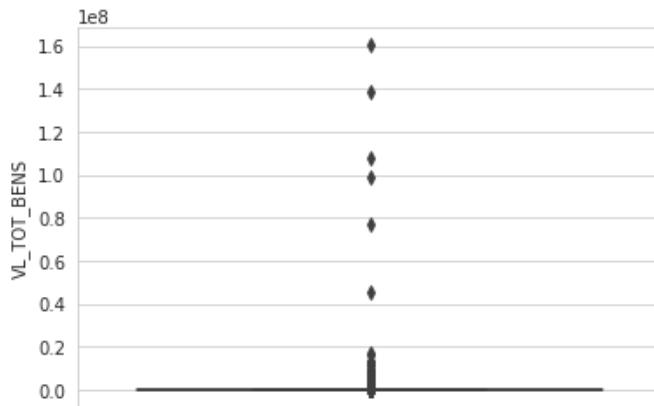
Esta informação merece uma reflexão especial em virtude de ter uma dispersão bastante significativa. As estatísticas básicas aplicadas sobre a coluna “valor total dos bens” nos dão uma visão geral da situação:

```
# estatísticas sobre as colunas dos bens declarados
cols = ['QT_BENS', 'VL_TOT_BENS']
base[cols].describe()
```

	QT_BENS	VL_TOT_BENS
count	56,500.00	56,500.00
mean	1.93	89,108.53
std	3.01	1,178,618.82
min	0.00	0.00
25%	0.00	0.00
50%	1.00	17,800.00
75%	2.00	72,850.10
max	73.00	160,302,000.00

O maior patrimônio informado ao TSE, dentre os candidatos a vereador soma 160 milhões de reais. Por outro lado, pelo menos 25% (vide dado do 1º quartil) não informaram bens, ou seja, não teriam patrimônio e 75% dos candidatos tem patrimônio inferior a 73 mil reais (vide 3º quartil). A média de patrimônio dos 56.500 candidatos é de 89 mil reais. O *boxplot* (ou “gráfico de caixa”) do valor total dos bens nos dá a dimensão do quanto dispersa é a distribuição desta variável:

```
sns.boxplot(y= base['VL_TOT_BENS'])
<AxesSubplot:ylabel='VL_TOT_BENS'>
```

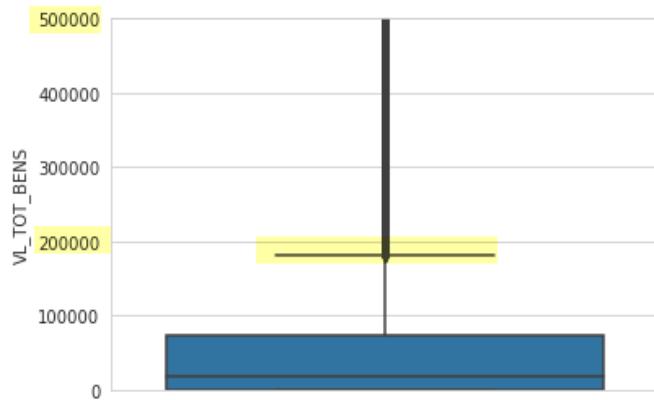


A imensa diferença entre o valor máximo e grande maioria das observações inviabiliza a visualização, no *boxplot*, dos valores dos quartis, média e limites de definição de *outliers*. Todos estes pontos aparecem como uma única linha próxima a

zero. Apenas para fins de melhor visualização, limitamos o valor máximo do eixo "y" em 500 mil reais. Assim conseguimos ver melhor o *boxplot* e identificar que qualquer patrimônio acima de 200 mil reais já é considerado *outlier*:

```
import matplotlib.pyplot as plt
plt.ylim(0, 500000)
sns.boxplot(y = base['VL_TOT_BENS'])

<AxesSubplot:ylabel='VL_TOT_BENS'>
```



```
# relação dos 10 maiores patrimônios
base[cols].sort_values('VL_TOT_BENS',
ascending = False).head(10)
```

	QT_BENS	VL_TOT_BENS
39721	9	160,302,000.00
18955	18	138,571,000.00
1833	6	108,135,716.00
42512	4	99,208,050.00
51675	43	76,613,063.38
52042	2	45,045,000.00
1830	1	16,803,000.00
35160	2	15,867,654.76
40199	16	13,134,500.00
33593	9	12,951,654.32

Isso mostra claramente como a grande “massa” de registros é composta por valores de patrimônio baixos. Entretanto, no caso específico, parece que os *outliers* de patrimônio são situações plenamente possíveis e não erros de coleta nos dados. Parece, também, bastante razoável imaginar a existência de candidatos com patrimônio acima de 200 mil reais. Mesmo nas situações mais “extremas”, em que o patrimônio chega a 160 milhões de reais (vide lista dos 10 maiores patrimônios acima), nos parece uma situação plausível.

4.11. Receitas e Despesas de campanha dos candidatos:

Assim como na análise dos bens dos candidatos, nas informações de receitas e despesas também nos deparamos com uma dispersão significativa, mas em menor escala. A seguir, as estatísticas básicas aplicadas sobre estas colunas:

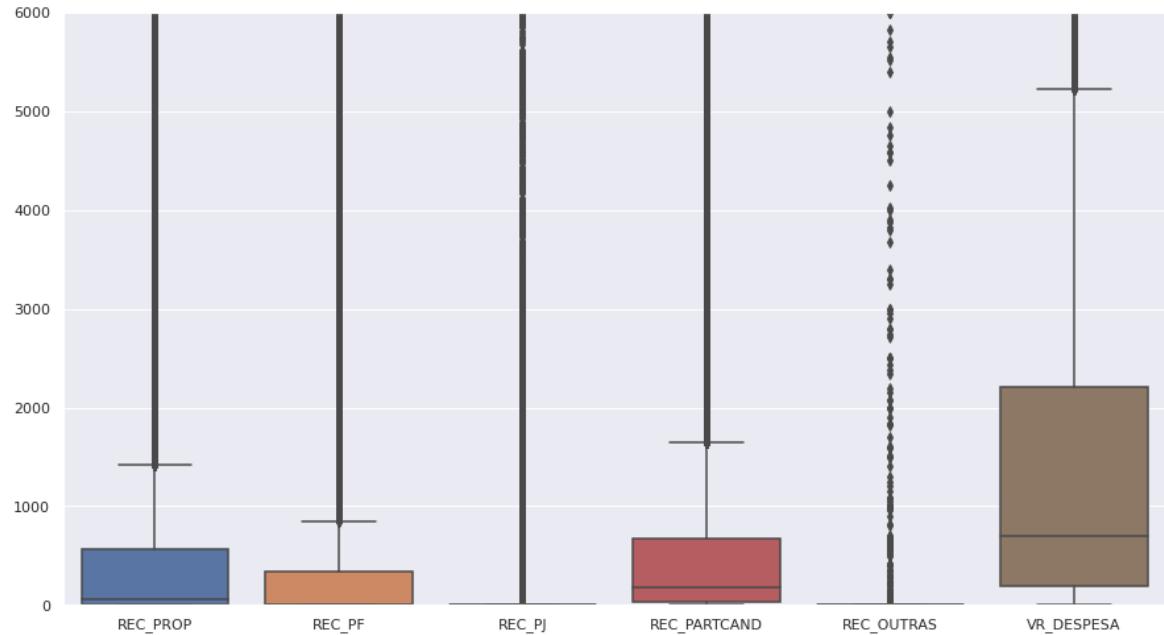
```
cols = ['REC_PROP', 'REC_PF', 'REC_PJ', 'REC_PARTCAND', 'REC_OUTRAS', 'VR_DESPESA']
base[cols].describe()
```

	REC_PROP	REC_PF	REC_PJ	REC_PARTCAND	REC_OUTRAS	VR_DESPESA
count	56,500.00	56,500.00	56,500.00	56,500.00	56,500.00	56,500.00
mean	909.83	1,134.65	681.41	1,216.23	56.75	3,988.81
std	4,892.78	6,089.37	10,462.07	11,457.09	2,334.30	25,940.81
min	0.00	0.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	32.98	0.00	190.00
50%	58.50	0.00	0.00	185.00	0.00	698.85
75%	567.00	340.00	0.00	676.15	0.00	2,201.68
max	421,989.80	328,838.92	926,000.00	1,081,211.85	329,322.33	1,799,320.54

A seguir, o *boxplot* destas colunas:

```
plt.ylim(0, 6000)
sns.set(rc={'figure.figsize':(15,8.27)})
sns.boxplot(data = base[cols])
```

<AxesSubplot:>



Mais uma vez, como a grande maioria dos registros apresentam valores de receitas e despesa muito baixos, os limites de definição de *outliers* ($1,5 \times$ diferença interquartílica) ficam também muito baixos. Pelo gráfico podemos constatar que, em qualquer grupo de receita, valores acima de 2 mil reais são considerados atípicos (*outlier*). Para despesas, valores pouco acima de 5 mil reais também são considerados “fora da curva”.

Assim como na análise dos valores de bens dos candidatos, entendemos também, que os valores de receitas e despesas constantes da base nos parecem plenamente plausíveis. Não nos parece haver erro grotesco nos valores de receita/despesas, portanto entendemos não ser o caso de exclusão dos mesmos.

Poderíamos fazer transformações de simples implementação nestas colunas que tentam tratar estes pontos fora da curva. Como por exemplo, a transformação logarítmica ou *robust scaler* (dentre outras). Entretanto, num primeiro momento, optamos por não fazer tratamento dos *outliers* encontrados nas colunas de valores de bens, receitas e despesas por entender que se tratam de *outliers* “naturais” e não de erros.

Merece também destaque o fato de estarmos tratando, conjuntamente, dados de candidatos de todos os municípios do estado de São Paulo, cada um com sua realidade cultural, econômica e social distinta. É possível que tal fato também tenha sua parcela de contribuição na grande dispersão das colunas de valores monetários.

4.12. Correlação entre colunas preditoras

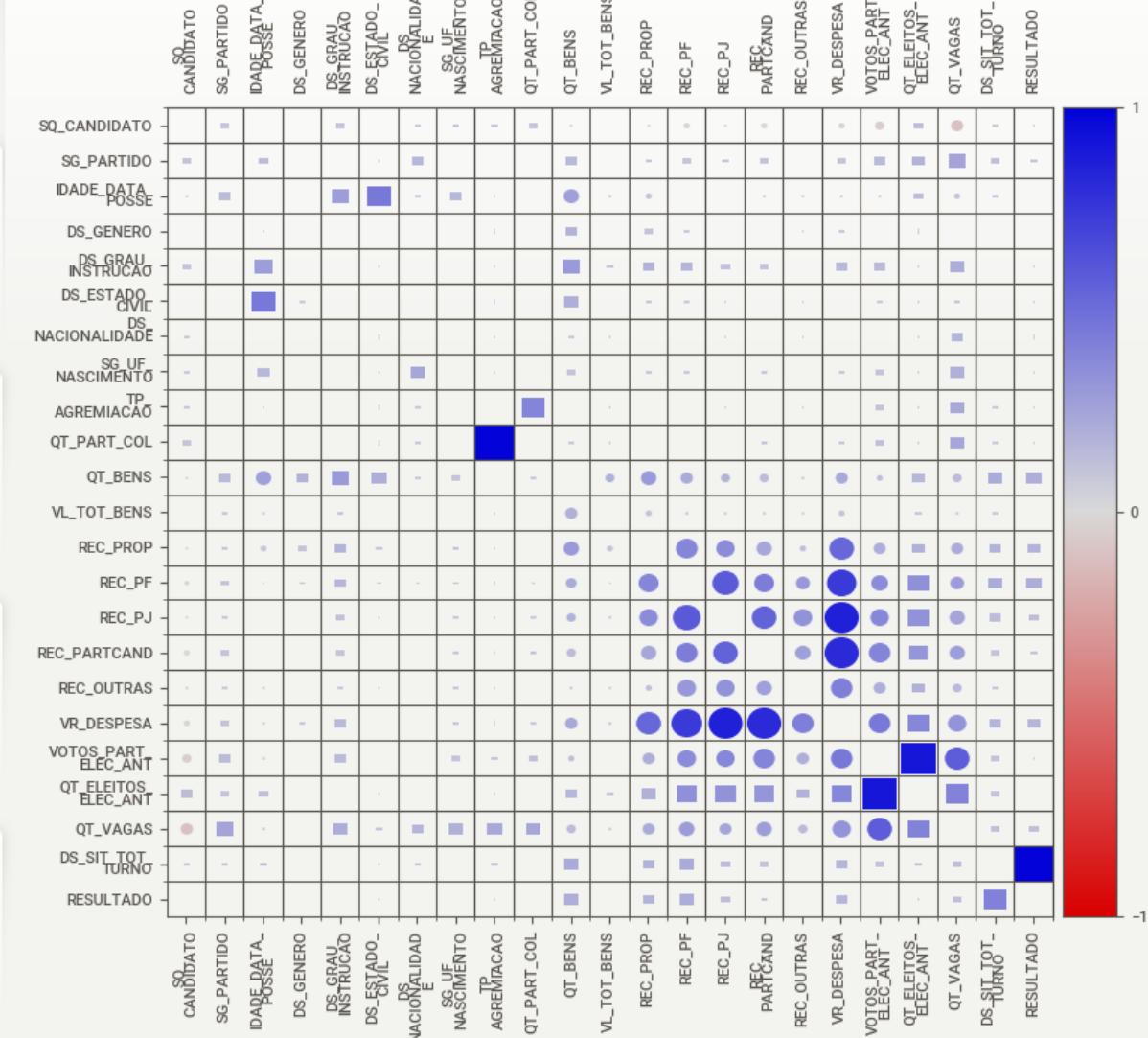
A seguir, apresentamos um gráfico de correlação entre as diversas colunas de nosso *dataset*. O gráfico em questão foi gerado pela biblioteca *SweetViz* do *python* e o arquivo html completo gerado por esta biblioteca encontra-se no *github* deste projeto. De forma sucinta, o gráfico de correlação mostra o quanto as colunas do *dataset* estão relacionadas entre si, ou seja, o quanto uma determinada coluna “explica” cada uma das demais. No nosso *dataset*, percebe-se que as colunas “qtd partidos na coligação” e “tipo de agremiação” tem correlação de 1, ou seja, uma coluna explica totalmente o comportamento da outra. Neste caso específico, a coluna tipo de agremiação pode assumir apenas 2 valores: “coligação” ou “partido isolado” e toda vez que ela for partido isolado a coluna “qtd partidos na coligação” assume valor “1”. Assim, de fato, uma coluna explica 100% o funcionamento da outra. Mesma situação ocorre com as colunas “resultado” (nossa *target*) e *ds_sit_tot_turno*, pois criamos aquela a partir desta.

Associations

[Only including dataset "DataFrame"]

■ Squares are categorical associations (uncertainty coefficient & correlation ratio) from 0 to 1. The uncertainty coefficient is **assymmetrical**, (i.e. ROW LABEL values indicate how much they PROVIDE INFORMATION to each LABEL at the TOP).

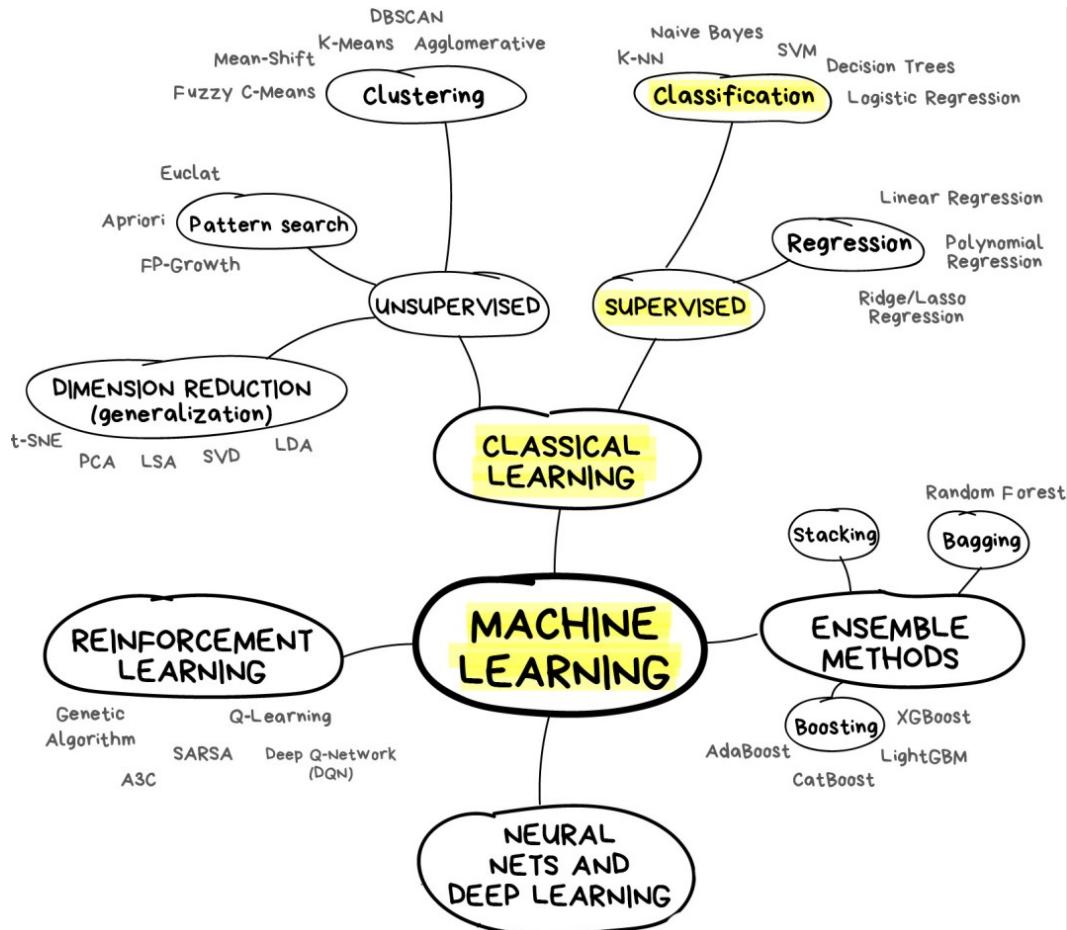
• Circles are the symmetrical numerical correlations (Pearson's) from -1 to 1. The **trivial diagonal** is intentionally left blank for clarity.



5. Criação de Modelos de *Machine Learning*

O nosso *dataset* é composto por dados de todas as candidaturas válidas ao cargo de vereador, no ano de 2008, no estado de São Paulo. Cada candidatura é um registro (linha) e as preditoras (colunas) contém diversos tipos de informações que qualificam as candidaturas. A coluna de resultados (*target*) é a que define, ao final, se o candidato foi eleito (1) ou não eleito (0). Portanto, o nosso estudo, que tem uma coluna “*target*”, é o caso típico de classificação, candidato ao uso de modelos de aprendizagem de máquina supervisionados. A seguir, um gráfico com os diversos

tipos de modelos de aprendizado de máquina¹⁶. Destacamos em amarelo, onde o nosso problema se encontra.



Source: Machine Learning for Everyone

O primeiro modelo que usaremos será “Árvore de Decisões¹⁷” (*Decision Tree Classifier*), por ser um modelo facilmente explicável e, portanto, bastante didático. De forma bastante simplista, um modelo de árvore de decisão transforma as colunas preditoras em perguntas do tipo *sim* ou *não*, que vão sendo encadeadas até chegar a resposta final do “target”.

No artigo “*Induction of Decision Trees*”, de 2003, Wagacha^[9] descreveu¹⁸ Árvore de Decisão como sendo um classificador em forma de árvore, onde cada nó pode ser:

- um “**nó decisão**”, com uma pergunta sobre um único atributo/preditor e que vai gerar um galho e continuação da árvore para cada resposta possível ou

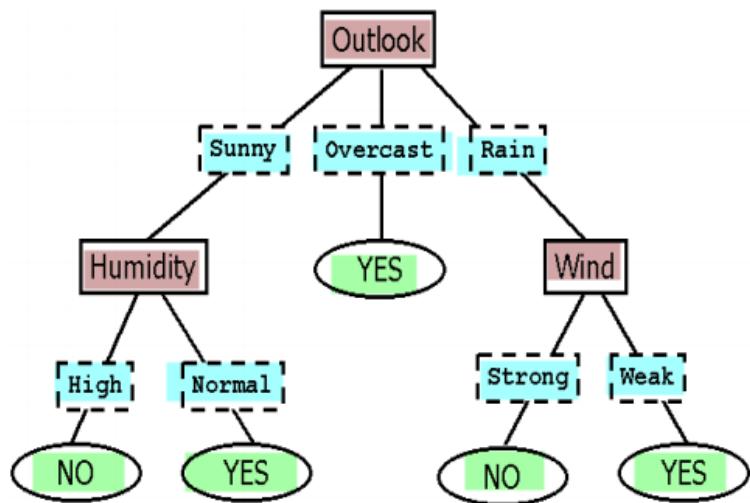
¹⁶ https://vas3k.com/blog/machine_learning/.

¹⁷ Mais sobre árvore de decisão em https://en.wikipedia.org/wiki/Decision_tree_learning.

¹⁸ Tradução livre nossa.

- um “**nó folha**”, com o valor final da classificação do “*target*”.

Uma árvore de decisão pode ser usada para classificar uma observação, iniciando pela raiz da árvore e movendo pelos “nó decisão” e galhos até chegar em um “nó folha”, com a classificação final da observação. Vide figura abaixo com exemplo sobre decisão se uma pessoa “joga” ou “não joga” tênis, a depender das condições climáticas, com “nó decisões” destacado em marrom, “galhos” em azul e “nó folhas” em verde:



Antes de iniciar a modelagem do nosso problema, é importante destacar que estamos diante de uma massa de dados muito heterogênea. Afinal, estamos tratando candidaturas de 645 municípios diferentes no estado de São Paulo. Se levarmos em conta que cada município tem suas características próprias culturais, econômicas, históricas, religiosas, dentre outras, é de se esperar que uma modelagem feita sobre todos eles em conjunto não seja tão precisa como uma modelagem individualizada. É o que veremos a seguir.

As colunas de nosso *dataset* inicial são basicamente aquelas que fizemos o quadro de correlação no item anterior. A seguir, resumimos as características de cada uma delas:

#	Coluna	Descrição	tipo	classificação
indice	SQ_CANDIDATO	Número único do candidato gerado pelo TSE	int64	chave identificadora
0	NM_UE	Nome do município da eleição	object	categórica
1	SG_PARTIDO	Sigla do partido do candidato	object	categorica
2	DS_OCUPACAO	Descrição da ocupação do candidato	object	categorica
3	IDADE_DATA_POSSE	Idade do candidato em 01/01/2009	int64	numérica discreta
4	DS_GENERO	Sexo do candidato	object	categorica
5	DS_GRAU_INSTRUCAO	Grau de instrução do candidato	object	categorica ordinal
6	DS_ESTADO_CIVIL	Estado civil do candidato	object	categorica
7	DS_NACIONALIDADE	Nacionalidade do candidato	object	categorica
8	SG_UF_NASCIMENTO	Estado de nascimento do candidato	object	categorica
9	CD_MUNICIPIO_NASCIMENTO	Município de nascimento do candidato	int64	categorica
10	TP_AGREMIACAO	Tipo, se: coligação ou partido isolado	object	categorica
11	COMP_COLIGACAO	Composição da coligação	object	categórica
12	QT_PART_COL	Quantidade de partidos na coligação	int64	numérica discreta
13	QT_BENS	Quantidade de bens informados	int64	numérica discreta
14	VL_TOT_BENS	Valor total dos bens informados	float64	numérica contínua
15	REC_PROP	Receitas de campanha próprias	float64	numérica contínua
16	REC_PF	Receitas de Pessoas Físicas	float64	numérica contínua
17	REC_PJ	Receitas de Pessoas Jurídicas	float64	numérica contínua
18	REC_PARTCAND	Receitas de partidos, comitês ou candidatos	float64	numérica contínua
19	REC_OUTRAS	Outras receitas	float64	numérica contínua
20	VR_DESPESA	Vlr total das despesas de campanha	float64	numérica contínua
21	VOTOS_PART_ELEC_ANT	Votos do partido para vereador no mesmo munic	int64	numérica discreta
22	QT_ELEITOS_ELEC_ANT	Quantidadde de vereadores eleitos pelo partido n	int64	numérica discreta
23	RESULTADO	Target, eleito (1) ou não eleito (0)	int64	target

Definidas as colunas finais do *dataset*, iniciamos a preparação da base para a aplicação do modelo de Árvore de Decisão: separar o *dataset* em 2 outros, um de “treino” e outro de “teste”. A base de treino será usada para “treinar o modelo”, para que ele aprenda como classificar um candidato (eleito ou não). Já a base de testes é a base sobre a qual vamos, de fato, verificar a efetividade do nosso modelo. Com o que o modelo “aprendeu” da base de treinamento, ele tentará inferir o resultado (*target*) na base de testes. A seguir, a codificação para criar o *dataset*, trocar o *index* e verificar o balanceamento da base:

```
# base de treino/testes dos modelos
baseml = base.copy()

# colocando SQ_CANDIDATO como index do dataframe
baseml.set_index('SQ_CANDIDATO', inplace=True)

# verificando o balanceamento de nosso dataset:
def resumo_base_treino(train,alvo):
    n_predi = train.shape[1] -1
    observ = train.shape[0]
    a0,a1 = alvo.value_counts()
    print(f'*** Resumo Amostra de treinamento ***')
    print(f'{observ} observações na amostra de treinamento')
    print(f'{n_predi} colunas preditoras')
    print(f'{int(observ / n_predi)} observações por coluna preditora')
    print(f'{round(a0/(a0+a1),2)} : {round(a1/(a0+a1),2)} - balanceamento da coluna target')

resumo_base_treino(baseml, baseml.RESULTADO)

*** Resumo Amostra de treinamento ***
56500 observações na amostra de treinamento
23 colunas preditoras
2456 observações por coluna preditora
0.89 : 0.11 - balanceamento da coluna target
```

Como podemos perceber, nosso *dataset* é desbalanceado. Ou seja, a nossa coluna alvo (eleito ou não eleito) apresenta muitos valores de uma classe (não eleito) e poucos de outra (eleito). No caso específico, apenas 11% dos resultados são “eleito” e 89% “não eleito”. Tal fato, como veremos mais adiante, influencia no resultado do modelo¹⁹ bem como na escolha das melhores métricas para avaliação de desempenho do mesmo.

5.1. Arvore de Decisões para a base de todo o estado de SP

A seguir, usamos o “*LabelEncoder*” da biblioteca “*scikit-learn*²⁰” para “transformar” as colunas categóricas em numéricas. Aqui, poderíamos ter usado as próprias colunas de código do TSE (anteriormente excluídas, por questão de redundância) mas, preferimos criar novas colunas de codificação, até para mostrar o funcionamento dessa ferramenta “*LabelEncoder*”.

```
# usando pre-processing do scikit learn para criar colunas numéricas com códigos, para substituir as categóricas
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df_temp['ue_le'] = le.fit_transform(df_temp.NM_UE)
df_temp['partido_le'] = le.fit_transform(df_temp.SG_PARTIDO)
df_temp['ocup_le'] = le.fit_transform(df_temp.DS_OCUPACAO)
df_temp['genero_le'] = le.fit_transform(df_temp.DS_GENERO)
df_temp['instrucao_le'] = le.fit_transform(df_temp.DS_GRAU_INSTRUCAO)
df_temp['estcivil_le'] = le.fit_transform(df_temp.DS_ESTADO_CIVIL)
df_temp['nac_le'] = le.fit_transform(df_temp.DS_NACIONALIDADE)
df_temp['uf_le'] = le.fit_transform(df_temp.SG_UF_NASCIMENTO)
df_temp['mn_le'] = le.fit_transform(df_temp.CD_MUNICIPIO_NASCIMENTO)
df_temp['agre_le'] = le.fit_transform(df_temp.TP_AGREMIAÇÃO)
df_temp['comp_le'] = le.fit_transform(df_temp.COMP_COLIGACAO)
```

Uma vez substituídas as colunas categóricas pelas numéricas geradas pelo *LabelEncoder*, passamos para:

- separação da colunas preditoras da coluna *target* e
- separação da base em “treino” e “teste”.

Usamos a ferramenta “*train_test_split*” do “*scikit-learn*” para fazer divisão da base em teste e treino, definindo o tamanho da base teste como 30% do tamanho total da base (parâmetro *test_size* = 0.3). Abaixo, código para fazer as 2 separações:

¹⁹ Dados desbalanceados tendem a gerar modelos com viés, ou seja, o modelo tende a apresentar bons resultados no acerto da classe majoritária, porém resultados ruins no acerto da classe minoritária.

²⁰ A biblioteca *scikit-learn* é um conjunto de ferramentas para uso de aprendizagem de máquina com python, largamente utilizado e extremamente poderosa. Vide link: <https://scikit-learn.org/stable/>

```
# Separar preditoras do target
df_y = df[['RESULTADO']].copy()
df_X = df.drop(columns='RESULTADO')

# separar a amostra de treinamento
i_size = 0.3
i_Seed = 2021

from sklearn.model_selection import train_test_split
X_treinamento, X_teste, y_treinamento, y_teste = train_test_split(df_X, df_y, test_size = i_size, random_state = i_Seed)

print(f'X treino: {X_treinamento.shape}')
print(f'y treino: {y_treinamento.shape}')
print(f'X teste : {X_teste.shape}')
print(f'y teste : {y_teste.shape}')

X treino: (39550, 23)
y treino: (39550, 1)
X teste : (16950, 23)
y teste : (16950, 1)
```

Com isso, temos 4 *dataframes* para treinar e testar o modelo:

- X_treino: colunas preditoras da base de treino
- y_treino: coluna *target* (resultado) da base de treino
- X_teste: colunas preditoras da base de testes
- y_teste: coluna *target* (resultado) da base de testes

Agora, vamos efetivamente criar e treinar nosso modelo de Classificação com Árvore de Decisões, usando ferramentas disponíveis no pacote *scikit-learn*.

```
from sklearn.tree import DecisionTreeClassifier # Library para Decision Tree (Classificação)
from sklearn import tree

# Instancia (configuração do Decision Trees) objeto com os parâmetros padrão:
ml_DT = DecisionTreeClassifier()

# Treina o modelo
ml_DT.fit(X_treinamento, y_treinamento)

DecisionTreeClassifier()

# Acuracias modelo
print(f'Acurácia base treino: {ml_DT.score(X_treinamento, y_treinamento)}')
print(f'Acurácia base teste : {ml_DT.score(X_teste, y_teste)}')

Acurácia base treino: 1.0
Acurácia base teste : 0.8363421828908555
```

Nosso modelo apresenta acurácia média de 100% na base de treinamento e de 83,63% na base de testes. Parece excelente! Entretanto, a acurácia mede o quanto frequente o modelo está correto, ou seja, a quantidade de acertos do modelo dividido pelo total de registros. Ocorre que, para um *dataset* desbalanceado como o nosso, se o modelo considerar TODOS os registros como “não eleitos”, ele já teria uma acurácia de 89%. Assim, para modelos desbalanceados, a acurácia não é uma

boa métrica de avaliação de desempenho. Aqui, cabe um breve resumo sobre o conceito de matriz de confusão e as principais métricas de avaliação.

A matriz de confusão é uma tabela que mostra as frequências de classificação para cada classe do modelo. Em outras palavras, ela compara a classificação final inferida pelo modelo com a classificação efetiva / real. Para ilustração, segue uma matriz de confusão hipotética de um modelo que tenta prever se uma mulher está grávida (1) ou não está grávida (0):

		ATUAL / REAL		
		0	1	
PREDITO	0	TN 50	FN 50	
	1	FP 47	TP 45	

Da matriz de confusão temos as seguintes informações:

True positive — TP: quantidade de observações que o modelo classificou como positivas e eram, de fato, positivas. Por exemplo, quando a mulher está grávida e o modelo previu corretamente que ela está grávida.

False positive — FP: quantidade de observações que o modelo classificou como positivas mas, na verdade, eram negativas. Exemplo: a mulher não está grávida, mas o modelo disse que ela está.

True negative — TN: quantidade de observações que o modelo classificou como negativas e eram, de fato, negativas. Exemplo: a mulher não estava grávida, e o modelo previu corretamente que ela não está.

False negative — FN: quantidade de observações que o modelo classificou como negativas mas, na verdade, eram positivas. Por exemplo, quando a mulher está grávida e o modelo previu incorretamente que ela não está grávida.

A partir dos valores da matriz de confusão, podemos calcular métricas para avaliação do desempenho do modelo, como por exemplo:

- **Accuracy** (acurácia): mede o quanto frequente o modelo está correto, ou seja, a quantidade de acertos do modelo dividido pelo total de registros.

$$\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN} = \frac{\text{previsões corretas}}{\text{todas as previsões}}$$

- **Recall**: mede a proporção de positivos que foi identificado corretamente. Em outras palavras, quanto bom o modelo é para prever positivos (assim entendido como a classe que se quer prever, por exemplo, se a mulher está grávida).

$$\text{recall} = \frac{TP}{TP + FN}$$

- **Precision**: mede a proporção entre as identificações corretamente classificadas como positivas e todas que o modelo classificou como positivo.

$$\text{precision} = \frac{TP}{TP + FP}$$

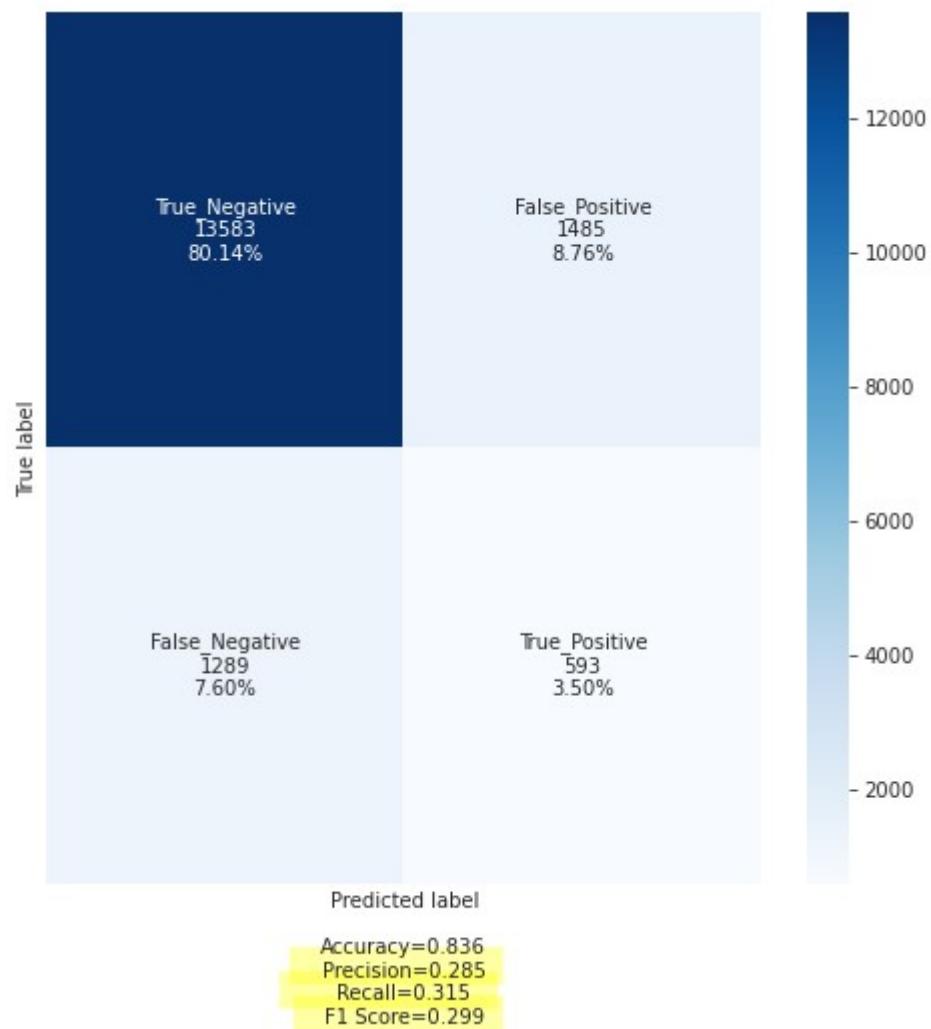
- **F1-score**: combina *precision* e *recall* de modo a trazer um número único que leva em consideração as duas métricas. Trabalha bem com *datasets* desbalanceados.

$$\text{F1-score} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Portanto, no nosso caso, em que queremos prever a eleição de candidatos (classe positiva) e que temos um *dataset* desbalanceado, a situação ideal seria bons valores de *Recall* e de *Precision*. Logo, a métrica *F1-score*, que leva em consideração as duas, parece ser uma boa opção para avaliação de nosso modelo.

A seguir, a matriz de confusão²¹ do nosso primeiro modelo de árvore de decisões para previsão de eleição de todos os candidatos a vereadores no estado de São Paulo em 2008, bem como o resultado das métricas “accuracy”, “precision”, “recall” e “f1-score”:

```
from sklearn.metrics import confusion_matrix
y_pred = ml_DT.predict(X_teste)
cf_matrix = confusion_matrix(y_teste, y_pred)
cf_labels = ['True_Negative', 'False_Positive', 'False_Negative', 'True_Positive']
cf_categories = ['Zero', 'One']
mostra_confusion_matrix(cf_matrix, group_names= cf_labels, categories= cf_categories)
```



²¹ **Matriz de confusão** com informações adicionais como métricas de classificação. O código para geração da mesma foi extraído de: <https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea>

Como pode-se perceber, apesar da alta “acurácia”, as métricas *precision*, *recall* e *f1-score* foram baixas (próximas a 30%). Obviamente, esta foi apenas primeira tentativa com um modelo simples e ainda sem nenhum ajuste. Além disso, estamos diante de um *dataset* desbalanceado que, por si só, já “direciona” o modelo para um vies.

Agora, vamos tentar melhorar este modelo pelo ajuste de seus parâmetros (Hyperparameter optimization²²). Para isto, definimos uma lista com diversas opções de “hiperparâmetros” possíveis para o “Classificador por Árvores de Decisões”:

```
# Dicionário com hiperparâmetros para o parameter tuning. Ao todo serão ajustados 2X15X5X7 = 5.250 modelos.
# Contando com 10 folds no Cross-Validation, então são 52.500 modelos.

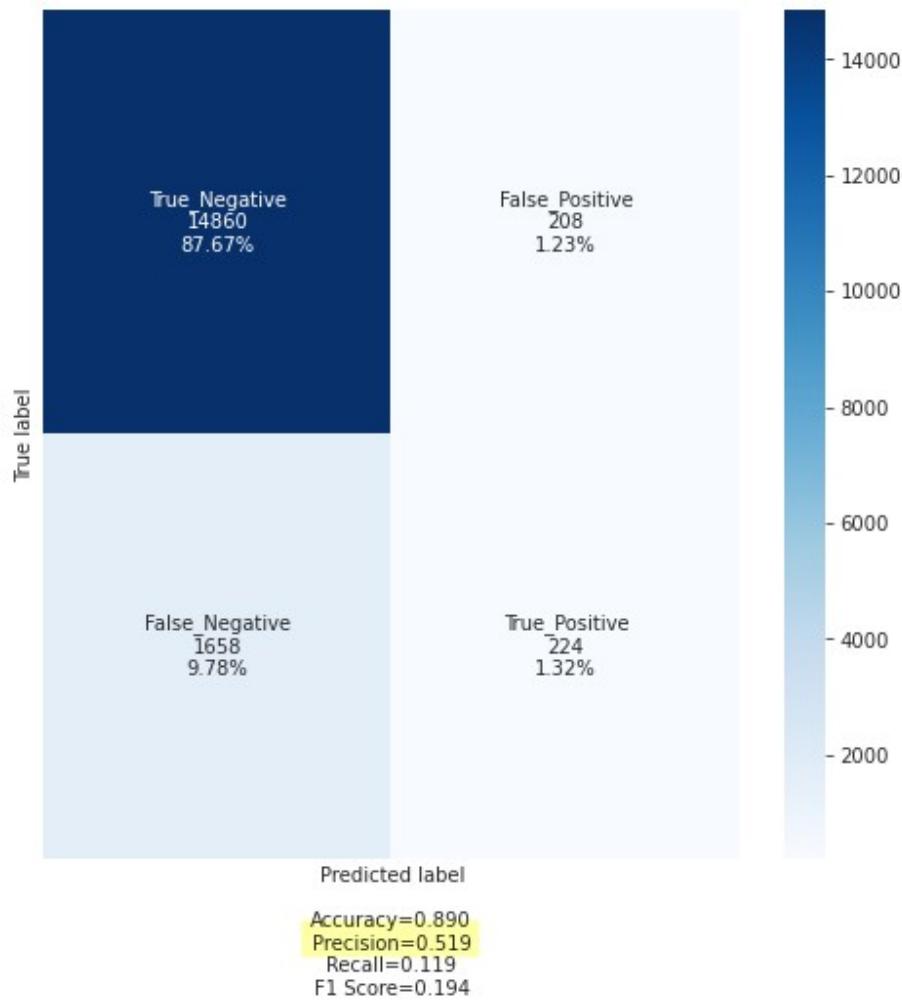
d_hiperparametros_DT = {"criterion": ["gini", "entropy"],
                        "min_samples_split": [2, 5, 10, 30, 50, 70, 90, 120, 150, 180, 210, 240, 270, 350, 400],
                        "max_depth": [None, 2, 5, 9, 15],
                        "min_samples_leaf": [20, 40, 60, 80, 100],
                        "max_leaf_nodes": [None, 2, 3, 4, 5, 10, 15]}
```

Desta relação de opções de hiperparâmetros, vamos buscar qual combinação traz o melhor resultado (são 5.200 possíveis). Para cada combinação ainda vamos fazer um *cross-validation* com 10 folds. Portanto, ao todo, serão testadas 52 mil árvores de decisões diferentes em busca dos melhores parâmetros. As ferramentas para fazer o *grid search* com *cross-validation* foram todas do scikit-learn e as funções que usamos estão todas detalhadas no notebook jupyter constante no github do projeto. Após a realização do “ajuste de hiperparâmetros” descrito acima, chegamos ao seguinte modelo:

- Matriz de confusão e métricas de avaliação:

```
from sklearn.metrics import confusion_matrix
y_pred = ml_DT2.predict(X_teste)
cf_matrix = confusion_matrix(y_teste, y_pred)
cf_labels = ['True_Negative', 'False_Positive', 'False_Negative', 'True_Positive']
cf_categories = ['Zero', 'One']
mostra_confusion_matrix(cf_matrix, group_names= cf_labels, categories= cf_categories)
```

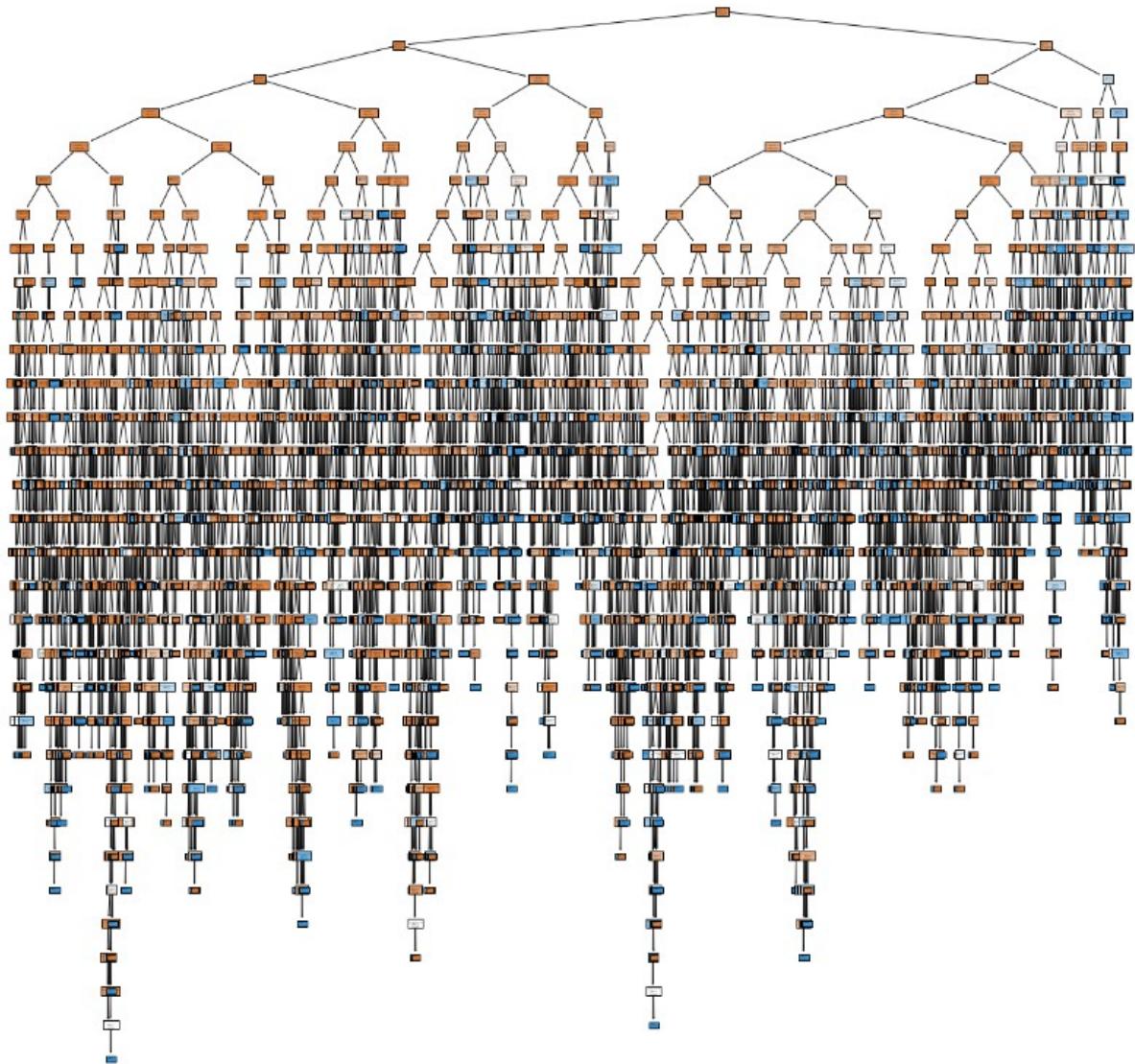
²² https://en.wikipedia.org/wiki/Hyperparameter_optimization



Apesar da precisão ter melhorado (de 28,5% para 51,9%), o *recall* e *F1-score* pioraram e ficaram abaixo de 20%.

Antes de fazermos outras tentativas de melhoria no modelo ou mesmo fazermos testes com outros modelos de aprendizado de máquina ou tentarmos minimizar a questão do desbalanceamento do *dataset*, entendemos que o fato de estarmos tratando 645 municípios com características distintas, de forma conjunta, também seja um fator que prejudica o desempenho de nosso modelo. A seguir, o desenho da árvore de decisões do modelo inicial, antes do grid-search, e que ilustra bem a questão da complexidade:

```
plt.figure(figsize=(20,20))
features = df.columns
classes = ['Eleito','Não eleito']
tree.plot_tree(ml_DT,feature_names=features,class_names=classes,filled=True)
plt.show()
```



Em razão do acima exposto, optamos por segmentar o nosso dataset de forma a testarmos o modelo de Árvore de Decisão para um único município. No ano de 2008, apenas o município de São Paulo atende à recomendação da PUC-Minas de tamanho de *dataset* com, no mínimo, 1 mil registros. Portanto, construiremos um novo modelo de árvore de decisão para os candidatos a vereador da capital paulista (São Paulo).

```
# agrupar por município e ordenar pela quantidade de candidatos
# o único que atende, individualmente, ao requisito mínimo da Puc-Minas (de 1000 observação) é São Paulo
base.groupby('NM_UE').agg({'SG_PARTIDO':'count'}).sort_values('SG_PARTIDO', ascending=False).head()
```

SG_PARTIDO	
NM_UE	
SÃO PAULO	1060
GUARULHOS	841
CAMPINAS	686
SÃO BERNARDO DO CAMPO	456
OSASCO	448

5.2. Arvore de Decisões para dataset do município de São Paulo

Seguimos a mesma metodologia usada no item 5.1. O primeiro passo foi criar o *dataset* com dados exclusivos do município de São Paulo:

```
# cria o dataset apenas com os registros de candidaturas do município de São Paulo
df_temp_c = df_temp_b.query('ue_le == 572').copy()

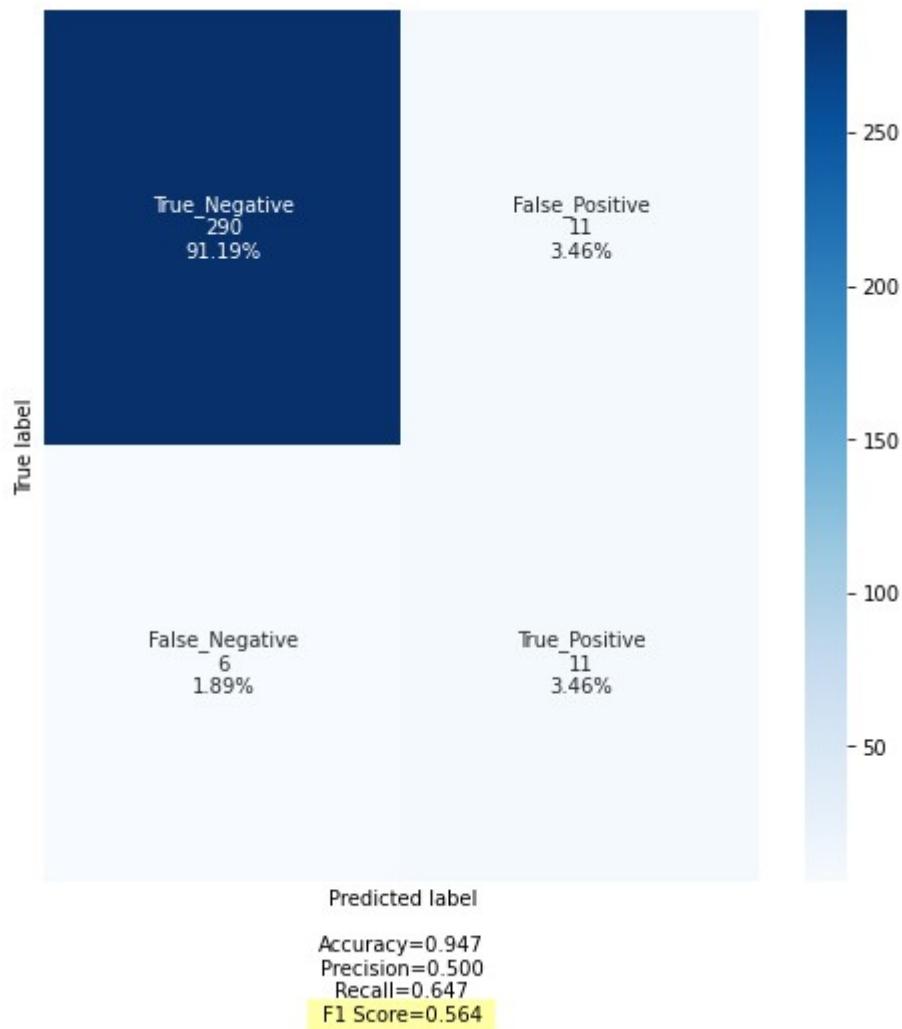
# avaliação do balanceamento do dataset
resumo_base_treino(df_temp_c, df_temp_c.RESULTADO)

*** Resumo Amostra de treinamento ***
1060 observações na amostra de treinamento
23 colunas preditoras
46 observações por coluna preditora
0.95 : 0.05 - balanceamento da coluna target
```

Percebe-se que este *dataset* é ainda mais desbalanceado que o anterior, com dados de todos os municípios. Para o município de São Paulo, temos 95% das observações como “eleito” e apenas 5% como “não eleito”.

Criamos, então, um modelo de “árvore de decisões” com os parâmetros padrões do *scikit-learn*. Mostramos a seguir, a matriz de confusão e as métricas de desempenho deste modelo:

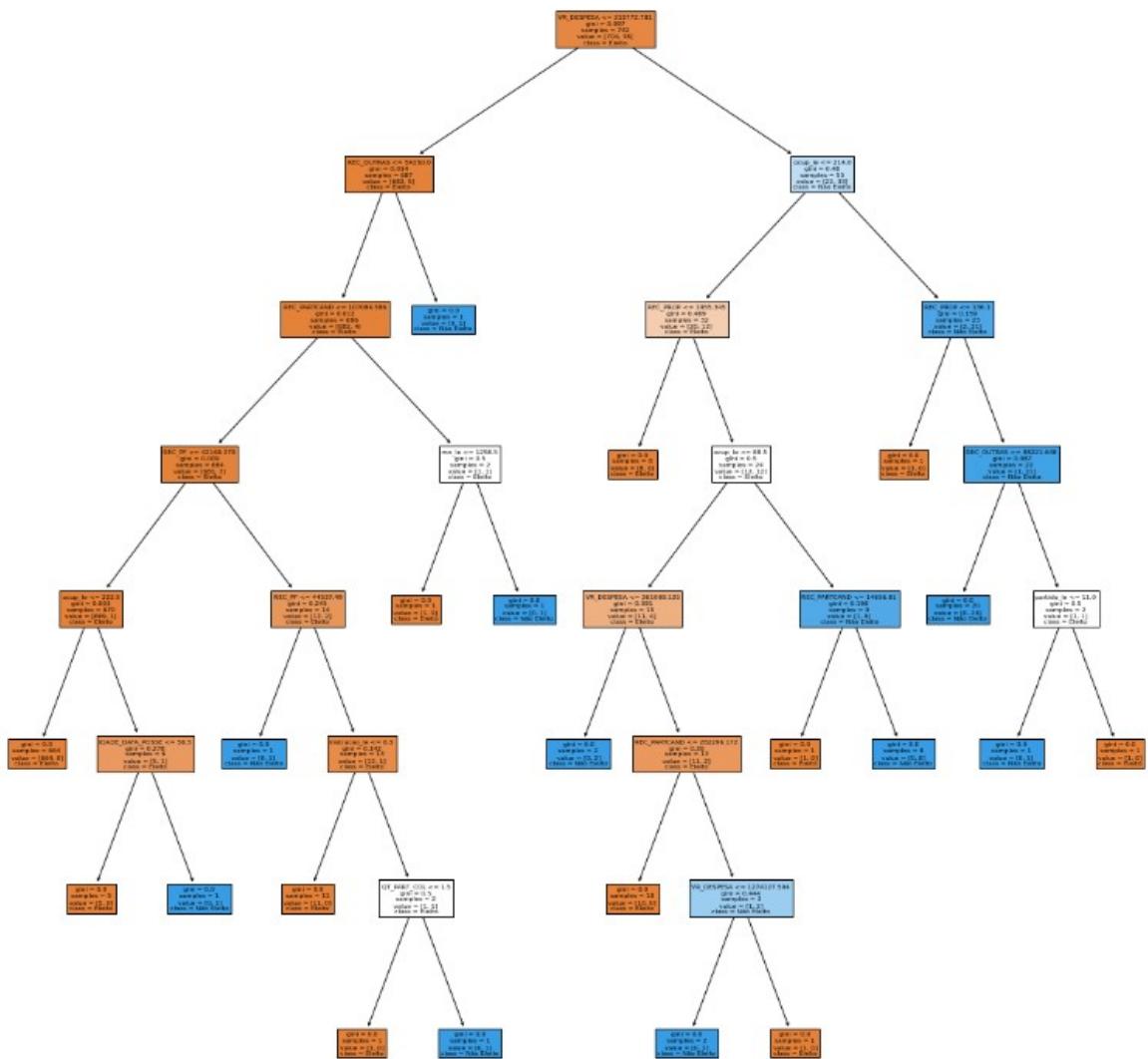
```
from sklearn.metrics import confusion_matrix
y_pred = ml_DT.predict(X_teste)
cf_matrix = confusion_matrix(y_teste, y_pred)
cf_labels = ['True_Negative', 'False_Positive', 'False_Negative', 'True_Positive']
cf_categories = ['Zero', 'One']
mostra_confusion_matrix(cf_matrix, group_names= cf_labels, categories= cf_categories)
```



Como podemos verificar, este modelo teve um desempenho bem interessante, com F1-score de 56,4% - muito superior aos 29% do modelo inicial para todo o estado, apesar da base atual ser ainda mais desbalanceada que a anterior. Tal fato, em princípio, confirma nossa tese de que tratar todos os municípios em conjunto pode não ser uma boa opção.

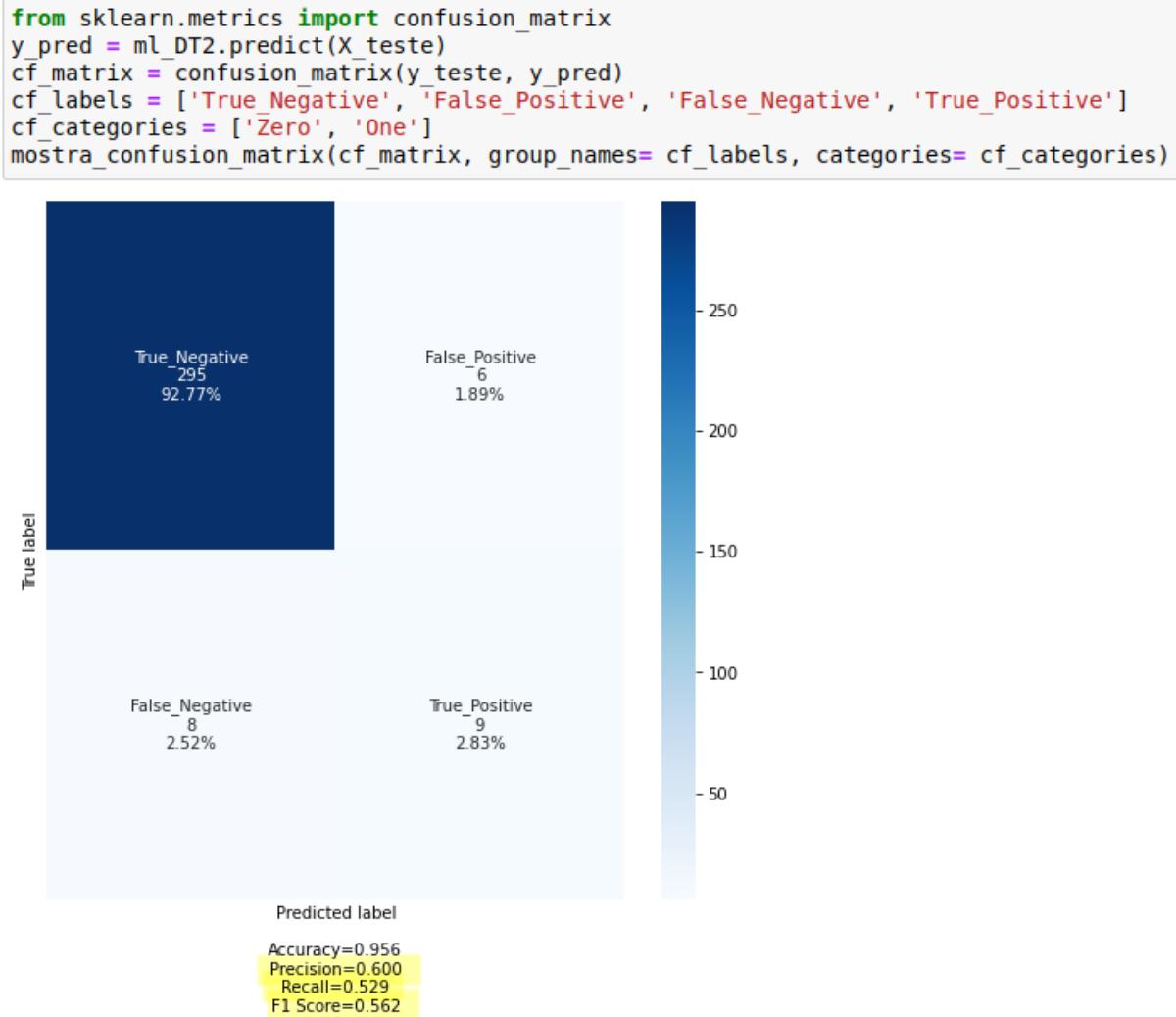
O desenho da árvore de decisões deste modelo também nos mostra como diminuiu a complexidade quando comparado com o modelo para todos os municípios do estado de São Paulo:

```
plt.figure(figsize=(20,20))
features = df.columns
classes = ['Eleito', 'Não Eleito']
tree.plot_tree(ml_DT, feature_names=features, class_names=classes, filled=True)
plt.show()
```



Passamos, então, para o “modelo ajustado” com o processo de ajuste de hiperparâmetros (*hyperparameter tuning*), com *grid search* e *cross-validation* nos mesmos moldes do modelo anterior.

Como pode ser visto nos resultados abaixo, as métricas de avaliação não tiveram variações muito consideráveis após o *grid search*. Entretanto, percebe-se as métricas com valores mais próximos entre si.



Analisando as colunas preditoras mais relevantes no nosso modelo, identificamos que apenas 3 delas (destacadas em amarelo, a seguir) tem relevância acima de 5%:

```
: # importância das colunas preditoras neste modelo
pd.DataFrame(zip(X_treinamento.columns, ml_DT2.feature_importances_),
              columns=['preditora','importância']).sort_values('importância', ascending=False).head(10)
```

	preditora	importância
15	VR_DESPESA	0.62
2	ocup_le	0.26
16	REC_PROP	0.07
14	VL_TOT_BENS	0.03
17	REC_PF	0.02
0	ue_le	0.00
12	QT_PART_COL	0.00
21	VOTOS_PART_ELEC_ANT	0.00
20	REC_OUTRAS	0.00
19	REC_PARTCAND	0.00

Uma técnica para tentar melhorar o desempenho de modelos de *Machine Learning* é a chamada *feature selection*²³, ou seleção das melhores *features* (colunas preditoras). Usando esta técnica, vamos então treinar um novo modelo, idêntico a este no que diz respeito aos parâmetros da árvore de decisões, porém apenas com as colunas preditoras de maior relevância. Escolhemos apenas as preditoras com relevância igual ou superior a 5%. Teremos, portanto, um modelo com apenas 3 colunas preditoras: “*ocupação do candidato*”, “*valor total das despesas de campanha*” e “*receitas próprias usadas na campanha*”. A seguir, a codificação para criação e treinamento do modelo:

```
# resumo do modelo 2, no qual vamos baser este modelo 3
ml_DT2
DecisionTreeClassifier(min_samples_leaf=2, min_samples_split=20,
                      random_state=2021)

# melhores parâmetros encontrados pelo grid search:
best_params
{'criterion': 'gini',
 'max_depth': None,
 'max_leaf_nodes': None,
 'min_samples_leaf': 20,
 'min_samples_split': 2}

# seleciona colunas relevantes
X_treinamento_DT, X_teste_DT = seleciona_colunas_relevantes(ml_DT2, X_treinamento, X_teste)

***** COLUNAS Relevantes *****
[ 2 15 16]
```

²³ Mais sobre feature selection em: https://en.wikipedia.org/wiki/Feature_selection

```
# Treina usando as COLUNAS relevantes...
ml_DT2.fit(X_treinamento_DT, y_treinamento)

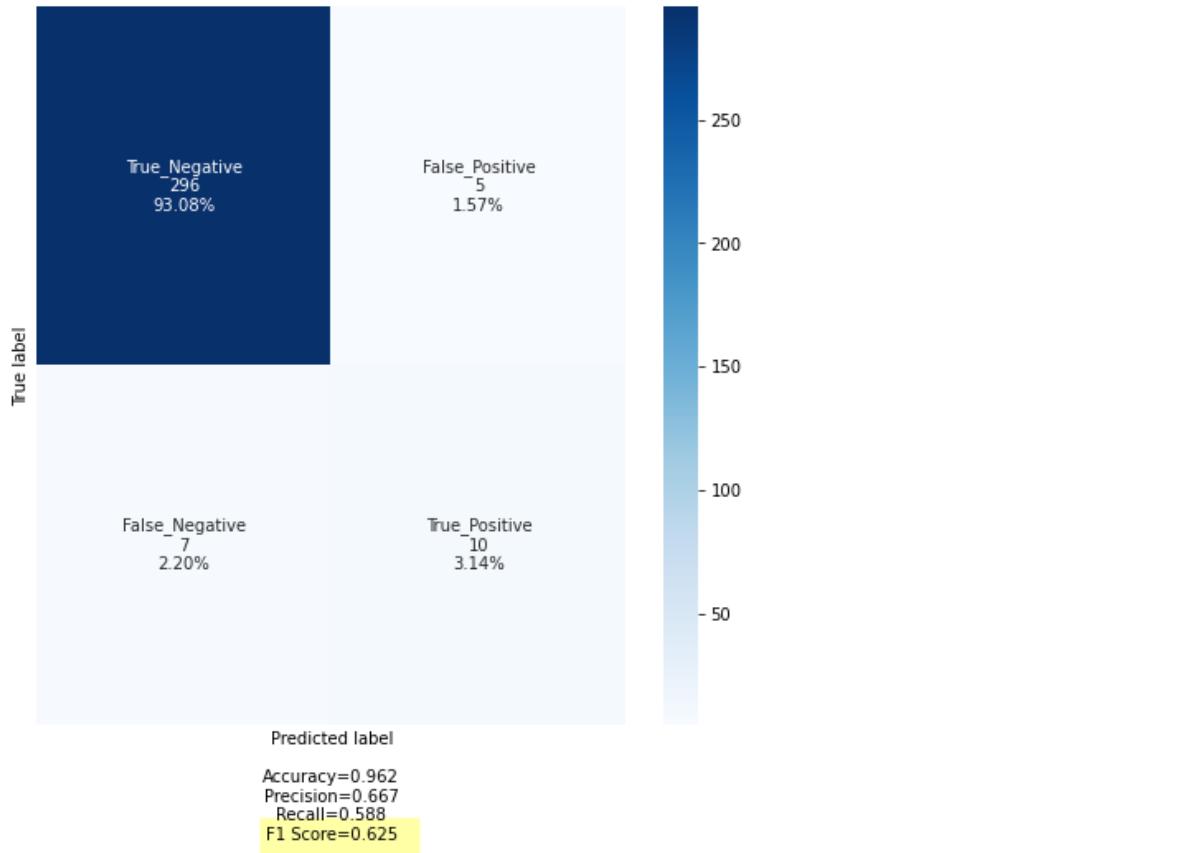
DecisionTreeClassifier(min_samples_leaf=2, min_samples_split=20,
                      random_state=2021)

# Cross-Validation com 10 folds
a_scores_CV = funcao_cross_val_score(ml_DT2, X_treinamento_DT, y_treinamento, i_CV)

Média das Acurárias calculadas pelo CV....: 97.57000000000001
std médio das Acurárias calculadas pelo CV: 1.18
```

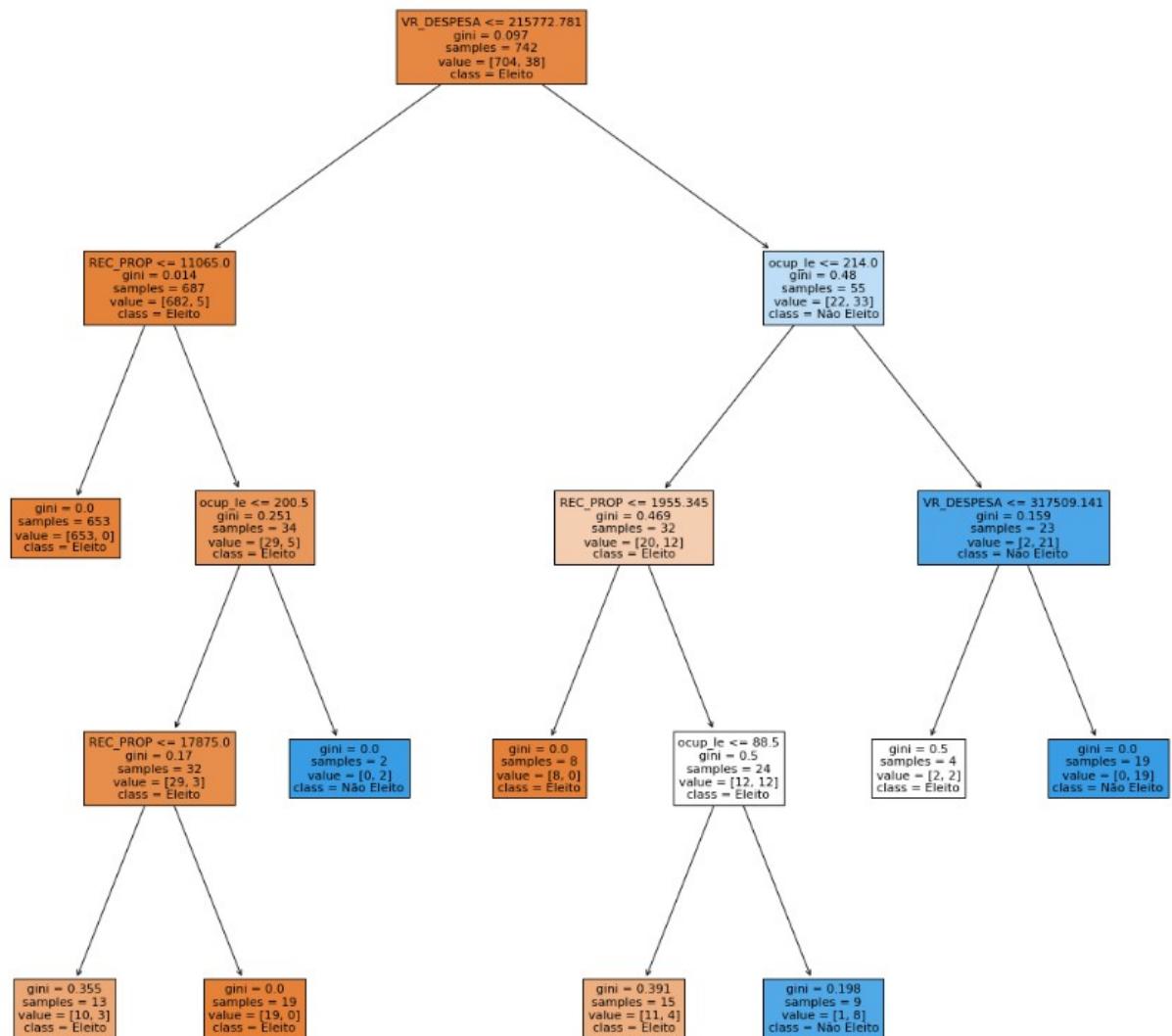
E a seguir, os resultados do modelo pela matriz de confusão e métricas de avaliação:

```
from sklearn.metrics import confusion_matrix
y_pred = ml_DT2.predict(X_teste_DT)
cf_matrix = confusion_matrix(y_teste, y_pred)
cf_labels = ['True_Negative', 'False_Positive', 'False_Negative', 'True_Positive']
cf_categories = ['Zero', 'One']
mostra_confusion_matrix(cf_matrix, group_names= cf_labels, categories= cf_categories)
```



Após os processos de “ajustes de hiperparâmetros” com *grid search* e *cross-validation* e também após o processo de seleção das colunas preditoras mais relevantes, descartando as demais para fins de treinamento, chegamos a um resultado que melhorou bastante o desempenho do modelo inicial. Saindo de um *F1-score* de 56,4% para 62,5% e um *recall* de 52,9% para 58,8%. Da leitura da matriz de confusão, identificamos que nosso modelo classificou como eleito 10 candidatos do total de 17 que efetivamente foram eleitos. Além disso, classificou como eleitos outros 5 que não foram eleitos de fato.

O uso apenas das preditoras mais significativas também deixou a árvore de decisões de nosso modelo bem mais simplificada, como pode ser visto abaixo:



Poderíamos continuar com outras técnicas para tentar melhorar nosso modelo de árvore de decisão, ou técnicas para tratar a questão do desbalanceamento²⁴ do nosso *dataset*. Entretanto, neste ponto, entendemos ser mais interessante testarmos novos modelos de *Machine Learning*.

5.3. Outros algoritmos de Aprendizagem de Máquina para classificação

Existem diversos modelos de *Machine Learning* para classificação. Para fins didáticos, nos estendemos um pouco mais na árvore de decisões. Usaremos a seguir, a ferramenta de AutoML²⁵ “Pycaret” para fazer a comparação de como se comportam alguns outros algorítimos de ML com o nosso *dataset* de candidaturas do município de São Paulo.

De forma bem simplista, podemos dizer que a ferramenta *Pycaret* automatiza o trabalho “braçal” de codificação como, por exemplo, do tratamento do *dataset*, a separação do mesmo em base de treino e base de testes, transformações diversas nas colunas preditoras, instanciamento e criação de modelos, “ajustes” nos hiperparâmetros dos modelos, com *grid search* e *cross-validation*, dentre vários outros. Além disso, faz a comparação de desempenho entre diversos modelos.

Criamos um notebook jupyter específico (disponível no github deste trabalho) para registrar os testes que fizemos com o *dataset* de candidaturas ao cargo de vereador para o município de São Paulo com a ferramenta *Pycaret*.

Feita a leitura do *dataset*, o passo mais importante é a definição dos parâmetros da ferramenta: aqui é onde definimos como o *Pycaret* deve trabalhar, com a definição dos tipos de colunas preditoras, eventuais transformações ou normalizações nas colunas, tratamento de “*missing values*”, técnicas para balanceamento do *dataset*, se vamos ou não priorizar preditoras mais relevantes,

²⁴ De forma simplista, o desbalanceamento é tratado pela exclusão de registros da classe majoritária (Undersampling) ou pelo acréscimo de registros na classe minoritária (Oversampling). O acréscimo/exclusão de registros pode ser feito de forma randômica ou usando técnicas específicas, como por exemplo SMOTE, Nearmiss, Cluster, etc. Vide: https://en.wikipedia.org/wiki/Oversampling_and_undersampling_in_data_analysis

²⁵ AutoML, ou Machine Learning automatizado é o processo de automatizar algumas do desenvolvimento de modelo de machine learning.

dentre diversos outros parâmetros. Podemos também não definir nenhum parâmetro e deixar que o PyCaret use suas definições padrão. Para o nosso caso, fizemos as seguintes definições:

- indicação das colunas numéricas e categóricas;
- normalização das colunas numéricas: o objetivo é deixá-las em uma escala comum sem, contudo, distorcer as diferenças nos intervalos de valores e nem perder informações;
- remoção de multi-colinearidade: caso existam colunas que estejam muito correlacionadas entre si, deve considerar apenas uma delas;
- ignorar *low variance*: em colunas categóricas com múltiplas categorias, quando uma categoria for claramente dominante e a variância for baixa, desconsidera a coluna;
- escolha das melhores *features*: descarta colunas preditoras menos importantes e considera apenas as *features* mais significativas.

O código para leitura do *dataset*²⁶ e definição dos parâmetros foi este:

```
# lê base de treino 2008
base = pd.read_csv('base_para_ml_cod_2008.csv', index_col=0)
# filtra o dataset apenas com os registros de candidaturas do município de São Paulo
train_df = base.query('SG_UE == 71072').copy()
train_df.drop(columns='SG_UE', inplace=True)

# importando ferramentas/ modelos de classificação
from pycaret.classification import *

clf=setup(data = basett,
           target = 'RESULTADO',
           numeric_features = ['IDADE_DATA_POSSE', 'QT_PART_COL', 'QT_BENS', 'VL_TOT_BENS', 'VR_DESPESA', 'REC_PROP',
                                'REC_PF', 'REC_PJ', 'REC_PARTCAND', 'REC_OUTRAS', 'VOTOS_PART_ELEC_ANT', 'QT_ELEITOS_ELEC_ANT'],
           categorical_features = ['NR_PARTIDO', 'CD_OUPACAO', 'CD_GENERO', 'CD_GRAU_INSTRUCAO', 'CD_ESTADO_CIVIL',
                                    'CD_NACIONALIDADE', 'SG_UF_NASCIMENTO', 'CD_MUNICIPIO_NASCIMENTO', 'TP_AGREMIACAO', 'COMP_COLIGACAO'],
           normalize = True, normalize_method = 'zscore',
           ignore_low_variance = True,
           remove_multicollinearity = True, multicollinearity_threshold = 0.95,
           feature_selection = True, feature_selection_threshold= 0.7,
           session_id = 1234, silent = False,
           log_experiment = True, experiment_name = 'vereador2')
```

²⁶ Estamos partindo do dataset já tratado, de candidaturas do município de São Paulo e que foi usado para criação do modelo de Árvore de Decisões do item 5.2 deste trabalho.

Em seguida, fizemos a comparação de desempenho de 15 modelos de classificação. Por padrão, o *Pycaret* separa o *dataset* em 70% para treino e 30% para teste e usa *cross-validation* com 10 desdobramentos (*folds*). Estas duas configurações foram as mesmas que usamos na árvore de decisões do item 5.2. Abaixo, o código e a saída com os modelos já comparados e destacados quais tiveram melhor desempenho em cada uma das métricas de avaliação:

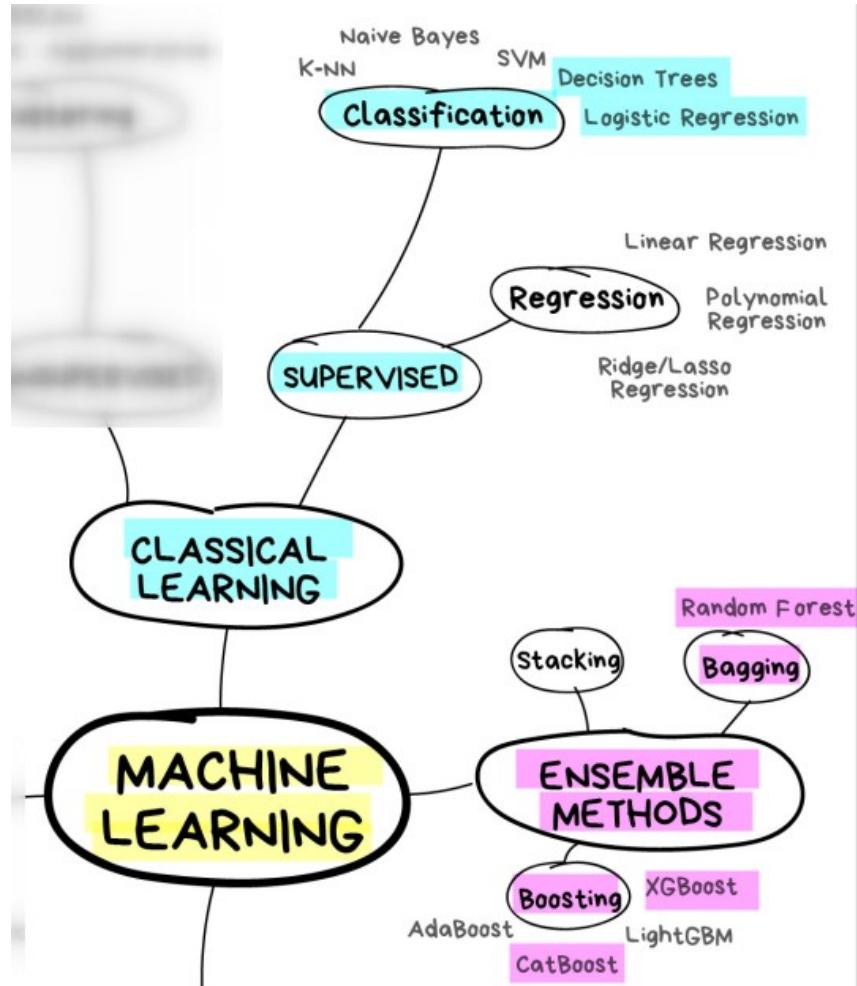
```
# compara modelos, tendo como parâmetro a métrica F1
best_model = compare_models(sort='F1')
```

Model		Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.9717	0.9797	0.6500	0.8333	0.7159	0.7015	0.7150	0.1860
catboost	CatBoost Classifier	0.9703	0.9850	0.6750	0.7950	0.7095	0.6943	0.7077	0.8070
gbc	Gradient Boosting Classifier	0.9703	0.9843	0.6750	0.7983	0.7087	0.6939	0.7084	0.0410
xgboost	Extreme Gradient Boosting	0.9703	0.9854	0.6500	0.7883	0.6813	0.6667	0.6851	0.0560
ridge	Ridge Classifier	0.9703	0.0000	0.6167	0.8317	0.6729	0.6587	0.6839	0.0060
et	Extra Trees Classifier	0.9703	0.9773	0.5917	0.8333	0.6686	0.6542	0.6756	0.1790
lightgbm	Light Gradient Boosting Machine	0.9690	0.9839	0.6250	0.8150	0.6670	0.6520	0.6782	0.0180
ada	Ada Boost Classifier	0.9663	0.9813	0.5917	0.7850	0.6431	0.6268	0.6489	0.0250
lda	Linear Discriminant Analysis	0.9649	0.9479	0.6167	0.7588	0.6401	0.6229	0.6465	0.0070
dt	Decision Tree Classifier	0.9595	0.7976	0.6167	0.7461	0.6219	0.6029	0.6323	0.0060
lr	Logistic Regression	0.9636	0.9491	0.5167	0.7850	0.5860	0.5688	0.5997	0.0870
knn	K Neighbors Classifier	0.9649	0.8984	0.5250	0.6917	0.5793	0.5647	0.5784	0.0510
svm	SVM - Linear Kernel	0.9528	0.0000	0.3917	0.5317	0.4261	0.4066	0.4231	0.0060
nb	Naive Bayes	0.5775	0.6356	0.7000	0.0815	0.1454	0.0571	0.1207	0.0060
qda	Quadratic Discriminant Analysis	0.1107	0.4912	0.9167	0.0523	0.0989	-0.0007	-0.0462	0.0080

Nesta comparação, o *Pycaret* “testa” os modelos com os parâmetros padrão, ou seja, sem nenhum ajuste. Neste ponto, escolhemos 5 modelos para fazermos ajustes e posterior comparação entre eles: *Random Forest*, *Catboost*, *Extreme Gradient Boosting*, *Decision Tree* e *Logistic Regression*²⁷.

Abrimos aqui um “parêntesis” para fazermos um breve comentário sobre os 5 modelos que escolhemos. Inicialmente, mostramos onde cada um destes modelos se encontram na classificação de algoritmos de ML:

²⁷ Escolhemos Random Forest e Catboost pois foram os 2 com melhor F1-score. Xgboost porque teve o quarto melhor F1-score e, além disso é muito usado em competições de ML, como kaggle. Decision Tree e Logistic Regeression, por serem modelos “puros”.



Árvore de Decisão (*Decision Tree*) e Regressão Logística (*Logistic Regression*) são algoritmos “clássicos” de Machine Learning e, por este motivo, mesmo não estando entre os melhores na comparação inicial do Pycaret, foram escolhidos para posterior ajustes. Já havíamos feito uma breve introdução sobre Árvore de Decisão e “*A regressão logística é uma técnica estatística que tem como objetivo produzir, a partir de um conjunto de observações, um modelo que permita a predição de valores tomados por uma variável categórica, frequentemente binária, a partir de uma série de variáveis explicativas contínuas e/ou binárias*²⁸.”

Os outros 3 modelos que escolhemos para ajustes são do tipo “ensemble”, assim definido:

²⁸ https://pt.wikipedia.org/wiki/Regress%C3%A3o_log%C3%ADstica

“Estes métodos constroem vários modelos de machine learning, utilizando o resultado de cada modelo na definição de um único resultado, obtendo-se assim um valor final único.

Isso significa que a resposta agregada de todos esses modelos é que será dada como o resultado final para cada dado que se está testando. Aqui estamos falando de algoritmos mais robustos e complexos, que envolvem mais operações, com um custo computacional um pouco maior, mas que, geralmente, têm uma performance melhor!²⁹”

Logo, modelos ensamble são na verdade um conjunto de diversos outros modelos que trabalham em conjunto para chegarem ao resultado final. Cada um deles tem suas peculiaridades e detalhes próprios que não vamos detalhar neste trabalho.

Feita esta breve introdução, o passo seguinte de nosso trabalho foi a criação dos 5 modelos escolhidos. O *Pycaret*, ao criar um modelo, calcula o resultado de cada uma dos desdobramentos (*k-folds*) da validação cruzada (*cross-validation*). Como não definimos o número *k*, de folds, é usado o valor padrão do *pycaret* (*k=10*). O “resultado” do modelo é a média dos 10 folds da validação cruzada. O modelo inicialmente criado é o mesmo da comparação inicial, ou seja, apenas com os parâmetros padrão do algoritmo de classificação.

Entretanto, o *Pycaret* nos dá, também, a opção ajustar (*tunning*) os hiperparâmetros do modelo. Este ajuste é feito usando *grid search* de forma randômica em uma tabela pré determinada de opções de parâmetros³⁰. Como temos uma base desbalanceada, nossa principal métrica de avaliação de desempenho dos modelos foi o F1-score, sem contudo perder de vista o Recall. Assim, fizemos o ajuste dos hiperparâmetros dos 5 modelos, a procura de melhorar o desempenho em F1-Score.

A seguir, os comandos e respectivas saídas para criação (lado esquerdo) e para tunning (lado direito) do modelo *Random Forest*. As saídas para os outros 4 modelos podem ser vistas no notebook *jupyter*, no *githib* deste trabalho.

²⁹ <https://didatica.tech/metodos-ensemble/>

³⁰ A ferramenta oferece, também, opção de usar um grid serach customizado.

```
# cria modelo Random Forest com parâmetros padrão
rf = create_model('rf')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9733	0.9683	0.5000	1.0000	0.6667	0.6544	0.6974
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.9865	0.9929	0.7500	1.0000	0.8571	0.8502	0.8599
3	0.9595	0.9857	0.5000	0.6667	0.5714	0.5506	0.5569
4	0.9595	0.9661	0.7500	0.6000	0.6667	0.6454	0.6499
5	0.9324	0.9518	0.5000	0.4000	0.4444	0.4089	0.4118
6	0.9865	0.9964	0.7500	1.0000	0.8571	0.8502	0.8599
7	0.9730	0.9571	0.5000	1.0000	0.6667	0.6542	0.6972
8	0.9865	1.0000	0.7500	1.0000	0.8571	0.8502	0.8599
9	0.9595	0.9786	0.5000	0.6667	0.5714	0.5506	0.5569
Mean	0.9717	0.9797	0.6500	0.8333	0.7159	0.7015	0.7150
SD	0.0186	0.0171	0.1658	0.2155	0.1623	0.1715	0.1706

```
# Tuning Random Forest
tuned_rf = tune_model(rf, optimize='Recall')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9333	0.9683	0.5000	0.4000	0.4444	0.4094	0.4123
1	0.9595	1.0000	1.0000	0.5000	0.6667	0.6476	0.6920
2	0.9324	0.9893	1.0000	0.4444	0.6154	0.5843	0.6424
3	0.9054	1.0000	1.0000	0.3636	0.5333	0.4932	0.5721
4	0.9459	0.9893	1.0000	0.5000	0.6667	0.6408	0.6866
5	0.9054	0.9500	1.0000	0.3636	0.5333	0.4932	0.5721
6	0.9054	0.9857	1.0000	0.3636	0.5333	0.4932	0.5721
7	0.9459	0.9929	1.0000	0.5000	0.6667	0.6408	0.6866
8	0.9189	1.0000	1.0000	0.4000	0.5714	0.5356	0.6047
9	0.9189	1.0000	1.0000	0.4000	0.5714	0.5356	0.6047
Mean	0.9271	0.9875	0.9500	0.4235	0.5803	0.5473	0.6046
SD	0.0184	0.0156	0.1500	0.0551	0.0699	0.0755	0.0792

Uma vez feito os ajustes de hiperparâmetros, vamos ver como cada um dos 5 modelos se comportou na amostra de testes do dataset de 2008 (30% da amostra separado pelo *Pycaret* e que ainda eram desconhecidos dos algoritmos).

```
# Realiza predições na amostra de testes (30% da amostra total)
predict_model(tuned_rf);
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Random Forest Classifier	0.9592	0.9821	1.0000	0.5517	0.7111	0.6911	0.7267

```
# Realiza predições na amostra de testes (30% da amostra total)
predict_model(tuned_catb);
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	CatBoost Classifier	0.9592	0.9862	0.3750	0.6667	0.4800	0.4605	0.4813

```
# Realiza predições na amostra de testes (30% da amostra total)
predict_model(tuned_xg);
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Extreme Gradient Boosting	0.9530	0.9831	0.7500	0.5217	0.6154	0.5912	0.6023

```
# Realiza previsões na amostra de testes (30% da amostra total)
predict_model(tuned_dt);
```

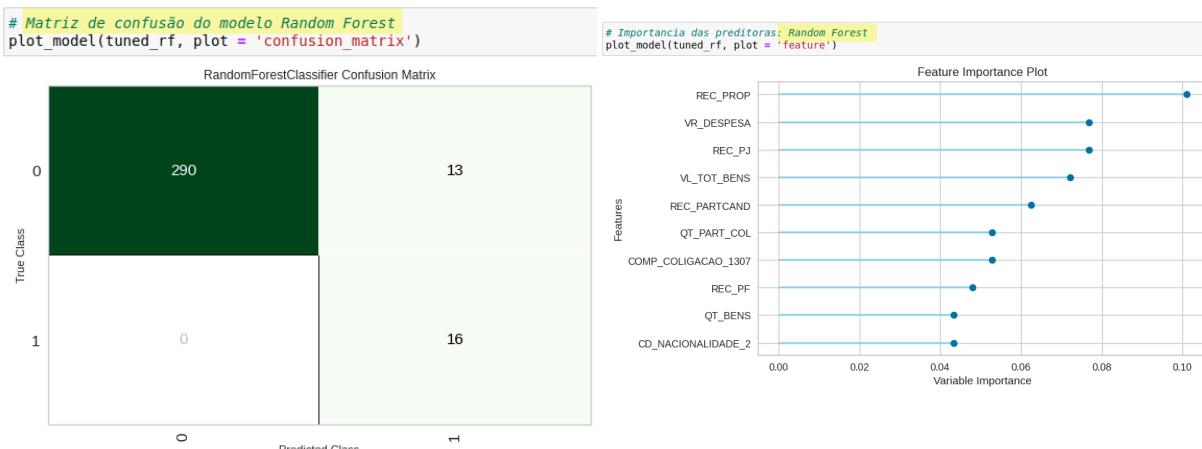
Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0 Decision Tree Classifier	0.9624	0.9210	0.8750	0.5833	0.7000	0.6808	0.6967

```
# Realiza previsões na amostra de testes (30% da amostra total)
predict_model(tuned_rl);
```

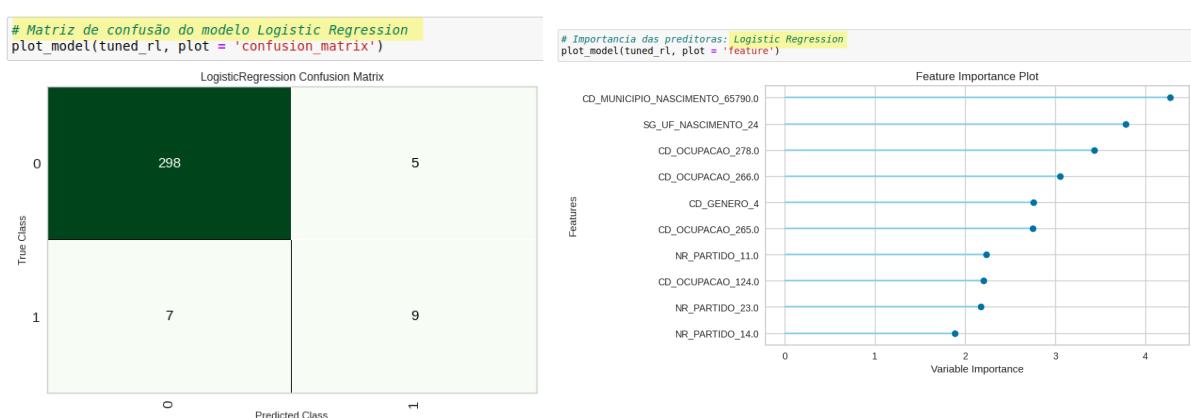
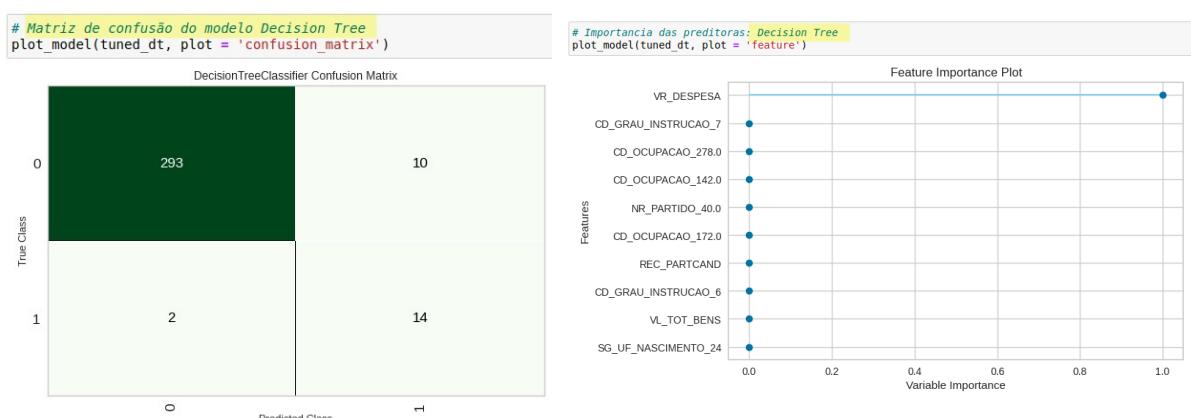
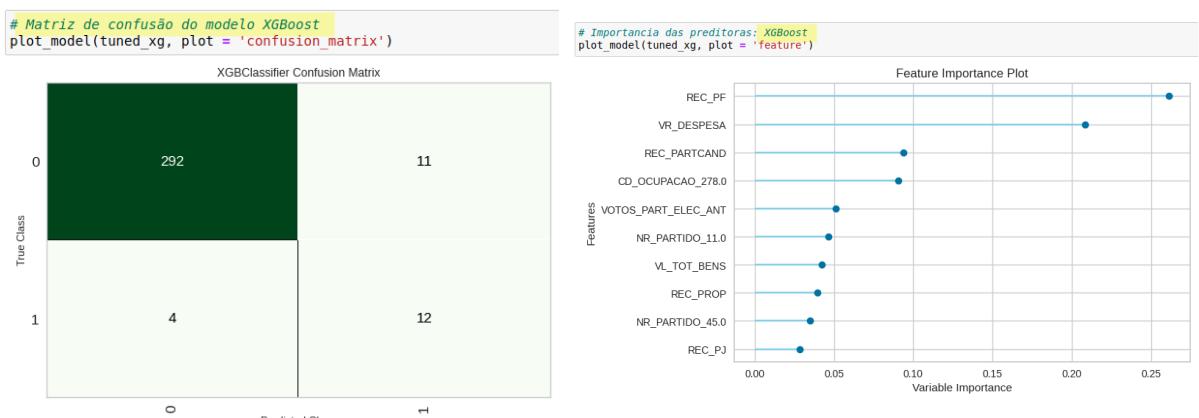
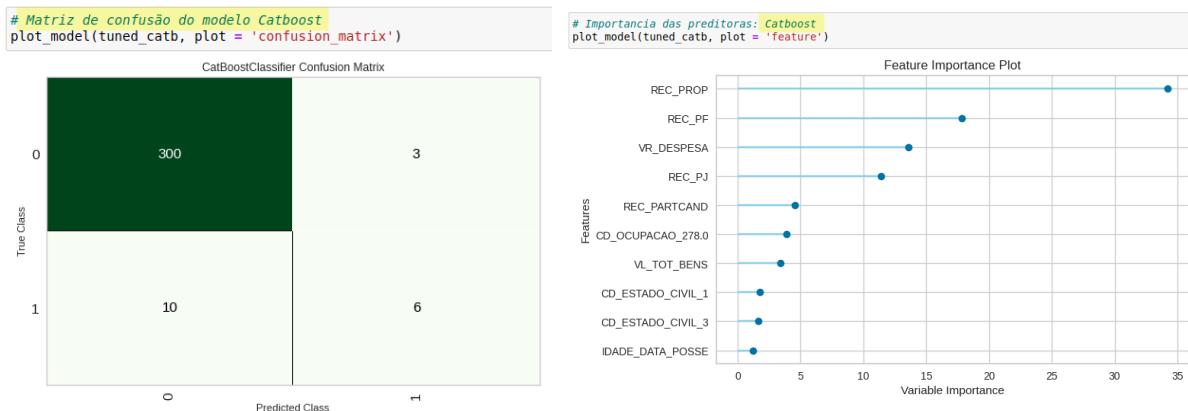
Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0 Logistic Regression	0.9624	0.9249	0.5625	0.6429	0.6000	0.5804	0.5818

À exceção do *catboost*, todos os outros modelos tiveram performance maior ou igual a 0,6 no *F1-score*. Como já havíamos comentado, no nosso caso, a situação ideal seriam bons valores de *F1-score* e *Recall* e, sob este aspecto, o modelo *Random Forest* teve um desempenho muito bom com 100% de *Recall* (acertou todos os positivos) além de um bom *F1-Score* 71%. Já o modelo *Catboost*, que na comparação inicial era o segundo melhor, teve desempenho muito ruim, com *Recall* de 37% e *F1-score* de 48%, o que pode ser um indicativo de que houve *overfitting*³¹ durante o processo de ajustes de hiperparâmetros.

O *Pycaret* tem implementado diversas funções para “plotagem” de gráficos que nos ajudam a compreender o funcionamento e o desempenho dos modelos. A seguir, mostramos a matriz de confusão da base de testes e as melhores *features* (colunas) para cada um dos 5 modelos que criamos:



³¹ Overfitting é quando o modelo se especializa demasiadamente na amostra de testes, de tal forma que acaba perdendo a capacidade de generalização para novos valores.



Interessante notar como variam as colunas mais importantes, a depender do modelo escolhido. Inclusive, para o modelo Regressão Logística, a principal *feature* é um desdobramento da coluna código do município de nascimento. Coluna esta que conseguimos recuperar graças ao esforço na imputação destes valores, que originalmente vieram como *NaN* na base do TSE.

Outro ponto que merece atenção diz respeito a avaliação dos modelos. Mesmo usando a métrica *F1-score*, somente ela não nos dá uma real ideia de como o modelo performou. Uma análise mais apurada das matrizes de confusão ajuda a ter um melhor cenário:

	TP	FP	TN	FN	F1-score	Recall	Precision
Random Forest	16	13	290	0	71,11%	100,00%	55,17%
Catboost	6	3	300	10	48,00%	37,50%	66,67%
XGBoost	12	11	292	4	61,54%	75,00%	52,17%
Decision Tree	14	10	293	2	70,00%	87,50%	58,33%
Logistic Regression	9	5	298	7	60,00%	56,25%	64,29%

Mesmo os 2 modelos com pior desempenho na métrica *F1-score* tem seus pontos positivos. Basta ver que são eles que tem as melhores performances na métrica *Precision*. Como última tentativa de melhorar o desempenho, vamos criar um novo modelo que será uma mistura (*ensamble* por *voting*) dos 4 melhores modelos em desempenho na métrica *F1-score*, que foram: *Random Forest*, *XGBoost*, *Decision Tree* e *Logistic Regression*. De forma simplista, este novo modelo faz uma “votação” entre os resultados dos 4 modelos e escolhe como *target* o valor mais “votado”. Nos modelos ensamble por votação, a “votação” pode ocorrer de 2 formas: *hard*, onde cada modelo que compõe o ensamble tem peso igual na votação ou *soft*, no qual, ao invés de “contar” os votos de cada modelo, faz-se uma média das probabilidades de cada modelo.

O *Pycaret* já tem implementada função para criação de modelos ensamble por votação. A seguir, a codificação para criar o nosso novo modelo de ensamble por votação, do tipo “soft” e respectiva saída gerada pelo *Pycaret*:

```
# Ensemble (blend) dos 4 melhores modelos "individuais" que haviamos criado
estimadores = [tuned_dt, tuned_xg, tuned_rf, tuned_rl]
blend_4s = blend_models(estimator_list = estimadores, method = 'soft')
```

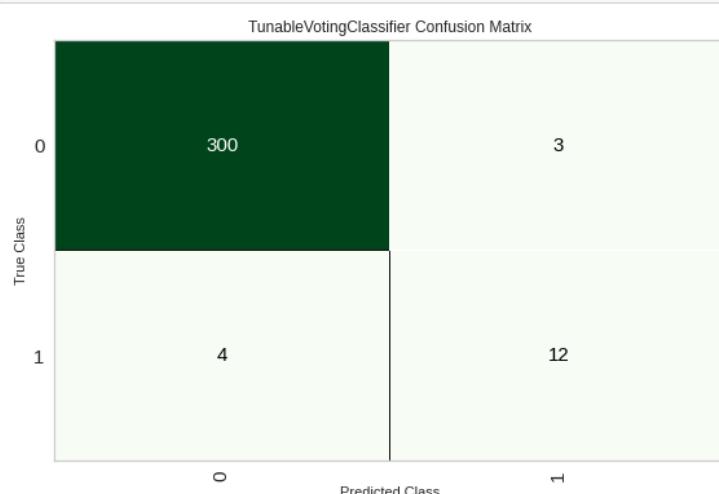
	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9600	0.9754	0.5000	0.6667	0.5714	0.5509	0.5572
1	0.9865	1.0000	1.0000	0.7500	0.8571	0.8502	0.8599
2	0.9730	1.0000	1.0000	0.6667	0.8000	0.7861	0.8047
3	0.9730	0.9929	1.0000	0.6667	0.8000	0.7861	0.8047
4	0.9324	0.9857	0.7500	0.4286	0.5455	0.5119	0.5353
5	0.9189	0.9679	0.5000	0.3333	0.4000	0.3584	0.3669
6	0.9730	0.9929	1.0000	0.6667	0.8000	0.7861	0.8047
7	0.9595	0.9786	0.5000	0.6667	0.5714	0.5506	0.5569
8	0.9595	0.9893	0.7500	0.6000	0.6667	0.6454	0.6499
9	0.9595	0.9929	1.0000	0.5714	0.7273	0.7071	0.7396
Mean	0.9595	0.9875	0.8000	0.6017	0.6739	0.6533	0.6680
SD	0.0191	0.0101	0.2179	0.1208	0.1403	0.1492	0.1521

E, abaixo, como se comporta esse modelo ensamble por votação na predição da base de testes: melhorou consideravelmente o F1-score, que era de 71,11% no melhor dos 4 modelos, e passou para 77,4%.

```
# Realiza predições na amostra de testes (30% da amostra total)
predict_model(blend_4s);
```

Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0 Voting Classifier	0.9781	0.9851	0.7500	0.8000	0.7742	0.7627	0.7631

```
# matriz de decisão do modelo "ensamble" que criamos
plot_model(blend_4s, plot = 'confusion_matrix')
```



5.4. Testando os modelos em situação “de produção”

A etapa final de nosso trabalho foi fazer um teste efetivo com o nosso modelo. Ou seja, ver como ele desempenha com dados novos e totalmente desconhecidos, simulando assim uma situação de “produção”. Para isso, usamos os dados de candidaturas do ano de 2012. Os passos para o teste final e resultados estão a seguir descritos.

Normalmente, em situações de produção, depois de todos os testes com diversos modelos, escolhemos o melhor deles para entrar em produção. Entretanto, aqui, como tratam-se de “ensaios sobre viabilidade de uso de ML”, e por termos 5 modelos próximos em desempenho (*Random Forest*, *XGBoost*, *Decision Tree*, *Logistic Regression* e *Ensemble* por Votação desses 4 modelos), decidimos fazer a simulação de produção com todos eles.

Antes de inciarmos os testes dos modelos com os dados de 2012, ainda precisamos fazer um passo de suma importância: finalizar os modelos com os dados de treinamento. Neste momento, cada um deles é “retreinado” com todo o conjunto de dados do ano de 2008³². A seguir, os comandos para finalização dos 5 modelos no pycaret:

```
# finaliza os 4 melhores modelos e o ensamble
rf_final = finalize_model(tuned_rf)
xg_final = finalize_model(tuned_xg)
dt_final = finalize_model(tuned_dt)
rl_final = finalize_model(tuned_rl)
blend4s_final = finalize_model(blend_4s)
```

O próximo passo para fazer o teste de predições com os dados de candidaturas do ano de 2012 é fazer a leitura do *dataset* de 2012. Importante frisar que a preparação da base de dados de 2012 seguiu todos os mesmos passos que havíamos feito para tratar o *dataset* de 2008. Criamos um *notebook jupyter* específico para fazer a leitura e tratamento dos dados de candidaturas de 2012 (*tcc_eleicoes_002_leitura_tratamento_base_testes_2012.ipynb*) que encontra-se no *github* deste trabalho.

³² Durante o treinamento, o dataset de 2008 foi dividido em 2 partes: treino (70%) e testes (30%). Na finalização do modelo, o mesmo é retreinado com 100% do dataset.

A seguir, a codificação para a leitura do *dataset* 2012 no *Pycaret*:

```
: # lê base teste 2012
test_df = pd.read_csv('base_para_ml_cod_2012.csv', index_col=0)
test_df.drop(columns='SG_UE', inplace=True)
```

Temos nossos modelos finalizados e os dados de teste carregados (*dataset* 2012), logo passamos para a criação dos *dataframes* com as previsões para os dados de candidaturas de 2012. Para isso, usamos a função *predict_model* do *Pycaret*:

```
: # realiza as previsões, a partir dos modelos finais, para os dados de 2012
pred12_rf = predict_model(rf_final, data=test_df)
pred12_xg = predict_model(xg_final, data=test_df)
pred12_dt = predict_model(dt_final, data=test_df)
pred12_rl = predict_model(rl_final, data=test_df)
pred12_b4s = predict_model(blend4s_final, data=test_df)
```

Por fim, o desempenho das previsões, com base na métrica F1-score, para eleição de candidatos a vereador no município de São Paulo para o ano de 2012, baseado no treinamento feito com os dados de 2008 para cada um destes 5 modelos:

```
from pycaret.utils import check_metric

# calcula métrica "F1-score" do modelo Random Forest
check_metric(pred12_rf['RESULTADO'], pred12_rf['Label'], metric = 'F1')
0.5934

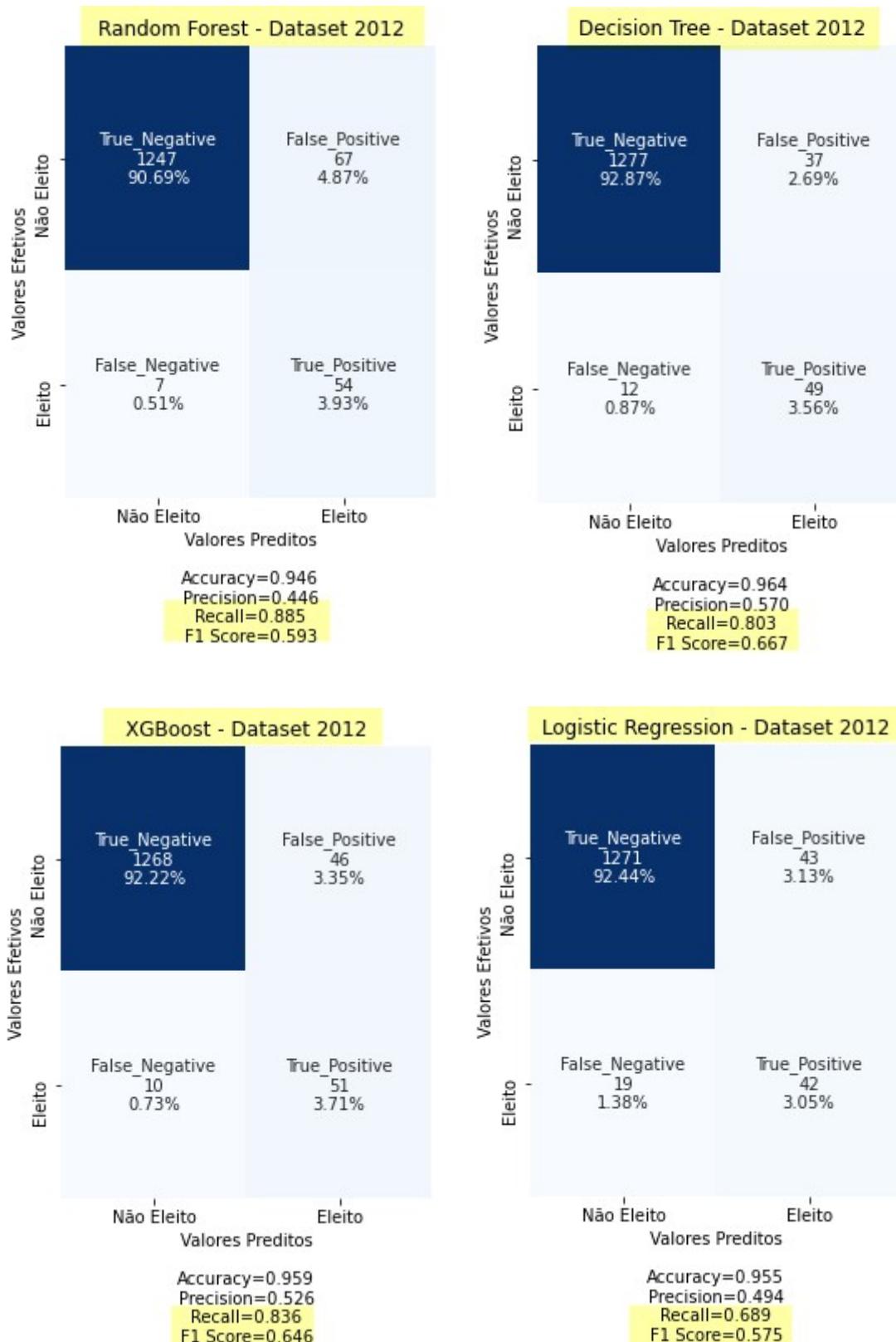
# calcula métrica "F1-score" do modelo XGBoost para 2012
check_metric(pred12_xg['RESULTADO'], pred12_xg['Label'], metric = 'F1')
0.6456

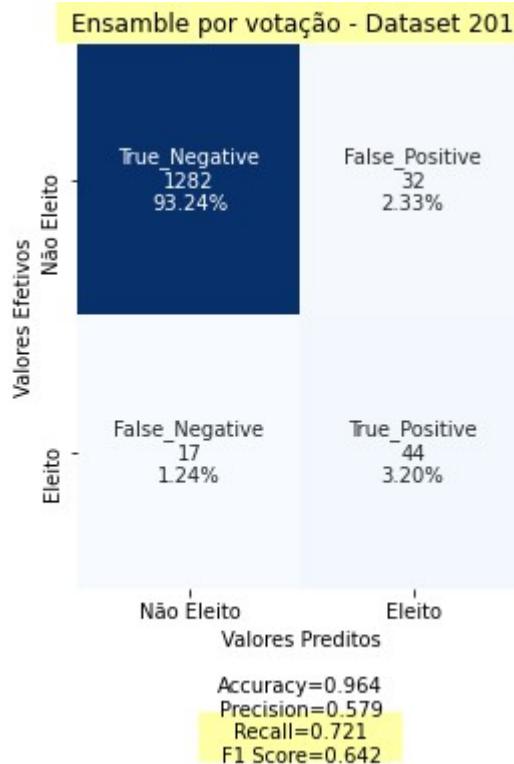
# calcula métrica "F1-score" do modelo Decision Tree para 2012
check_metric(pred12_dt['RESULTADO'], pred12_dt['Label'], metric = 'F1')
0.6667

# calcula métrica "acurácia" do modelo Logistic Regression
check_metric(pred12_rl['RESULTADO'], pred12_rl['Label'], metric = 'F1')
0.5753

# calcula métrica "acurácia" do modelo Blending por Votação
check_metric(pred12_b4s['RESULTADO'], pred12_b4s['Label'], metric = 'F1')
0.6423
```

Apenas a métrica F1-score não nos mostra efetivamente como performou o modelo. A seguir, apresentamos a matriz de confusão e principais métrica para cada um dos 5 modelos testados:





5.4.1. Interpretação dos Resultados

Como podemos ver todos os modelos tiveram um excelente desempenho pela métrica “Accuracy”. O melhor desempenho na métrica *F1-score* foi do modelo *Decision Tree*. Por outro lado, o melhor desempenho no *Recall* foi do *Random Forest*. Já na métrica *Precision*, o melhor desempenho foi do modelo *Ensamble*. Também ficou claro uma piora geral no desempenho do F1-Score quando comparado aos resultados obtidos na base de treinamento de 2008.

O primeiro ponto a observar na interpretação dos resultados diz respeito ao cuidado que devemos ter na escolha das métricas de avaliação de um modelo de Machine Learning. Já havíamos comentado o assunto no decorrer do trabalho (vide item 5.1), mas é importante frisar a ineficácia da métrica “Accuracy” para avaliar modelos com classes desbalanceadas, que é o nosso caso. Para avaliar nossos modelos, precisamos de bons índices de “Recall” – que nos mostra o percentual dos candidatos efetivamente eleitos que nosso modelo acertou. Por outro lado, precisamos também de bons índices de “Precision” – que nos mostra qual a probabilidade de nosso modelo acertar quando ele diz que o candidato foi eleito.

Portanto, a métrica F1-score, que faz um balanceamento de *Precision* e *Recall*, se mostra adequada para avaliarmos nossos modelos, mas, ainda assim, é muito importante analisar a própria matriz de confusão para ter uma visão completa da eficiência do modelo.

A seguir, um resumo dos resultados de nossos modelos quando aplicados aos dados de candidaturas de 2012:

	TP	FP	TN	FN	F1-score	Recall	Precision
Random Forest	54	67	1247	7	59,34%	88,52%	44,63%
Decision Tree	49	37	1277	12	66,67%	80,33%	56,98%
XGBoost	51	46	1268	10	64,56%	83,61%	52,58%
Logistic Regression	42	43	1271	19	57,53%	68,85%	49,41%
Ensamble 4modelos	44	32	1282	17	64,23%	72,13%	57,89%

Apesar de não termos um resultado muito expressivo, entendemos que pode ser considerado bastante promissor, afinal conseguimos *F1-score* próximo a 65% em 3 dos modelos. Se levarmos em conta que ainda existe muito caminho para ajuste nos modelos e melhora na qualidade das informações do *dataset*, parece bastante promissor o uso de *Machine Learning* em previsões eleitorais.

6. Apresentação dos Resultados

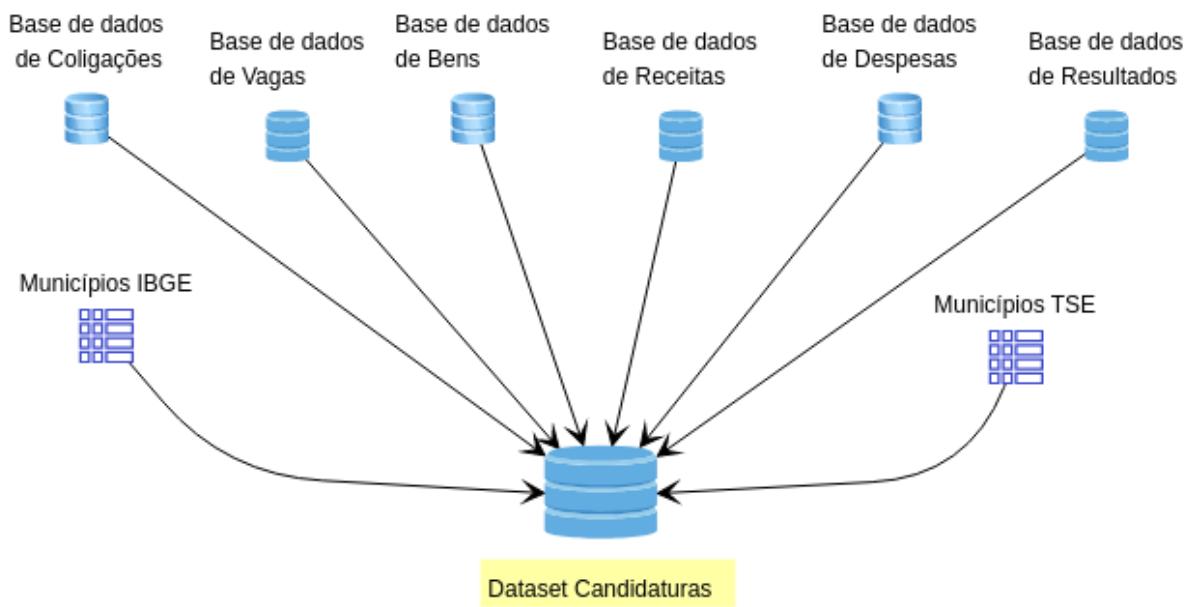
A proposta do presente trabalho foi testar a viabilidade de uso de algoritmos de *Machine Learning* para previsão de “eleição” ou “não eleição” de candidatos ao cargo de vereador. Para ajudar no direcionamento do mesmo, criamos um “*Canvas Workflow*”³³³⁴ que mostra a ideia principal e alguns tópicos chave relacionados ao projeto.

MACHINE LEARNING CANVAS		
Designed for: TCC PUC Minas	Designed by: Silvio Finotti	Date: 27/07/2021 Iteration: 5
TÍTULO: Ensaios sobre uso de ML para previsões de resultado de eleições municipais para vereador		
DEFINIÇÃO DO PROBLEMA É viável o uso de Machine Learning para previsões eleitorais de candidatos a vereador ? Quais colunas preditoras que melhor explicam a possível eleição de um candidato ?	RESULTADOS E PREVISÕES Identificar, dentre candidaturas registradas para o cargo de vereador, quais candidatos têm maiores chances de serem eleitos, ou seja: fazer previsões dos eleitos. Colunas preditoras: características pessoais do candidato, características do partido do candidato, desempenho do partido em pleitos anteriores, valores gastos na campanha, etc.	AQUISIÇÃO DE DADOS Do repositório de dados do TSE usamos informações de sete bases em arquivo csv: candidaturas (dataset principal), coligações partidárias, bens de candidatos, quantitativo de vagas, receitas e despesas de campanha e resultados de eleições anteriores. Informações de nomes dos municípios, distritos e subdistritos do IBGE (arquivo xls).
MODELAGEM Temos um problema de classificação binária (eleito ou não eleito). Logo, precisamos de algoritmos de Machine Learning supervisionados. Iniciaremos com Decision Tree e posteriormente testamos Random Forest, XGBoost, Catboost, Logistic Regression e um Ensamble por votação de 4 desses modelos.	AVALIAÇÃO DO MODELO Temos classes muito desbalanceadas logo, a Acurácia não é uma métrica muito indicada. Maior interesse está na previsão correta de positivos, portanto a métrica Recall é importante, sem descuidar da Precision. F1-score, que leva em consideração ambas, é uma boa métrica para avaliação em conjunto com dados da matriz de confusão	PREPARAÇÃO DOS DADOS Tratamos dados ausentes tanto no dataset principal, como nas colunas agregadas posteriormente. Imputação variada, de acordo com a necessidade de cada coluna. Ajustamos, quando possível, colunas com informações inconsistentes.

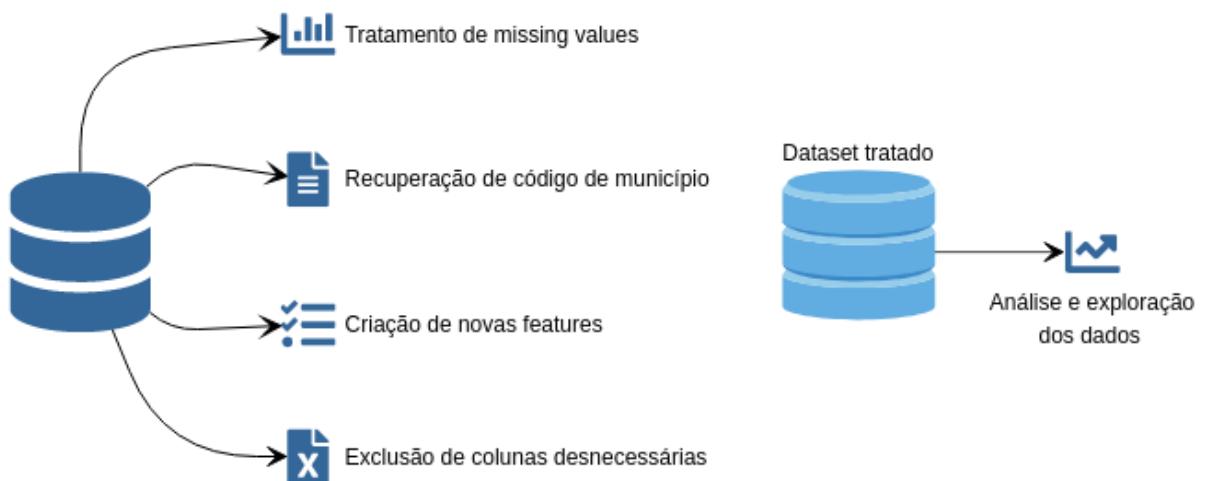
Para testar a nossa hipótese, buscamos informações em 9 bases de dados distintas para criar um *dataset* com informações de candidatos ao cargo de vereador.

³³ Nosso CANVAS foi baseado no modelo criado por Jasmine Vassandani, em 2019: <https://towardsdatascience.com/a-data-science-workflow-canvas-to-kickstart-your-projects-db62556be4d0>

³⁴ Versão em PDF em tamanho original disponível no github deste trabalho.

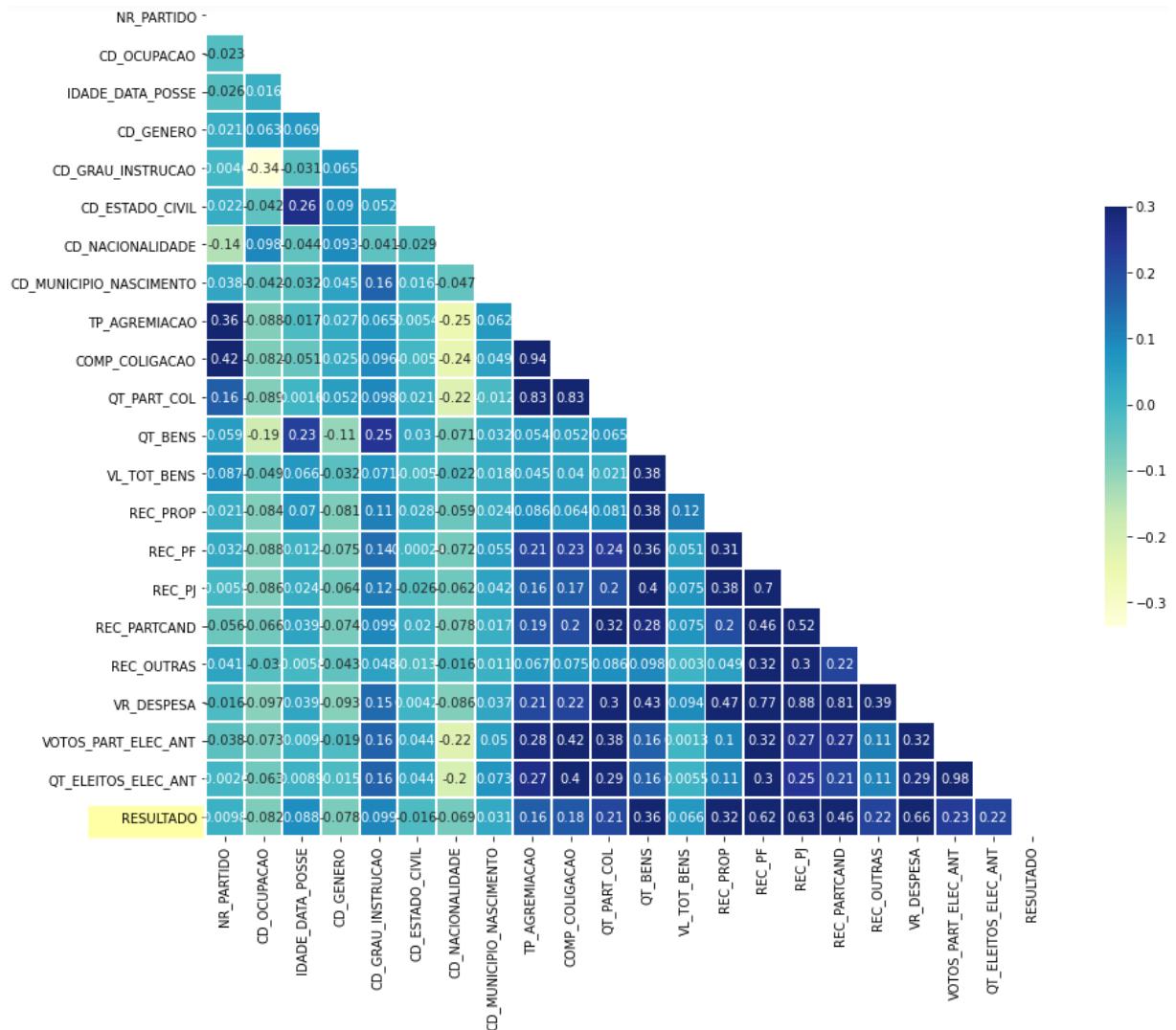


Em seguida, fizemos a preparação dos dados com: remoção de duplicados, imputação de valores faltantes, remoção de colunas sem utilidade além de outros ajustes e correções nos dados para deixá-los adequados ao nosso propósito.



Uma vez pronto o *dataset*, fizemos a análise exploratória do mesmo em busca de informações e *insights* que pudessem auxiliar no entendimento dos dados. O gráfico de correlação entre as diversas colunas preditoras (*features*) do *dataset* mostra como cada uma delas influencia nas demais. Percebe-se que as colunas que mais “explicam” o resultado (“eleição” ou “não eleição”) do candidato estão

relacionadas às receitas e despesas de campanha, bens e desempenho do partido nas eleições anteriores.



Na escolha dos algoritmos de *Machine Learning*, testamos diversos modelos e selecionamos 5 deles para ajustes e comparações mais aprofundadas. Com os dados de teste do ano de 2008, obtivemos resultados relativamente satisfatórios no que se refere ao desempenho dos modelos. Destes 5, selecionamos os 4 “melhores” para criarmos um novo modelo Ensamble por votação. De uma amostra com registro de 319 candidaturas ao cargo de vereador ao município de São Paulo em 2008, os modelos performaram da seguinte forma:

	TP	FP	TN	FN	F1-score	Recall	Precision
Random Forest	16	13	290	0	71,11%	100,00%	55,17%
Catboost	6	3	300	10	48,00%	37,50%	66,67%
XGBoost	12	11	292	4	61,54%	75,00%	52,17%
Decision Tree	14	10	293	2	70,00%	87,50%	58,33%
Logistic Regression	9	5	298	7	60,00%	56,25%	64,29%
Ensamble 4modelos	12	3	300	4	77,42%	75,00%	80,00%

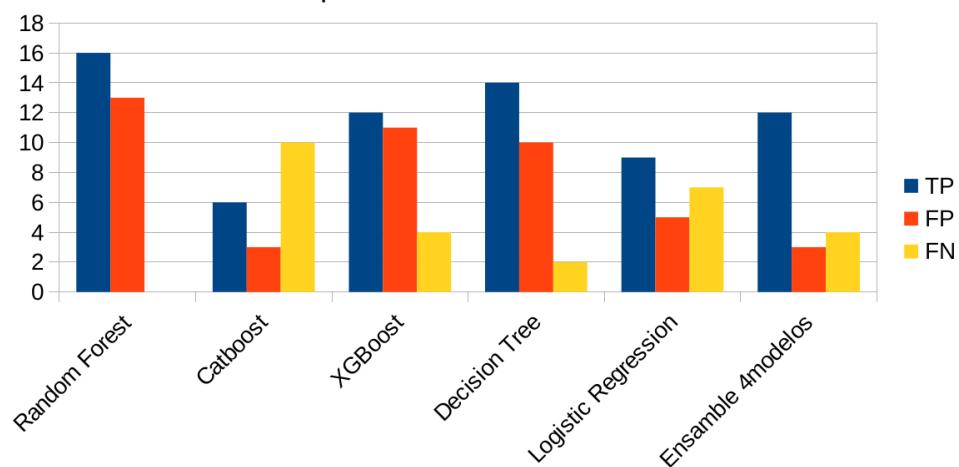
TP = eleitos acertados (modelo classificou corretamente como eleito, pois foi, de fato, eleito)

FP = não eleito incorreto (modelo classificou erroneamente como eleito, pois não foi eleito)

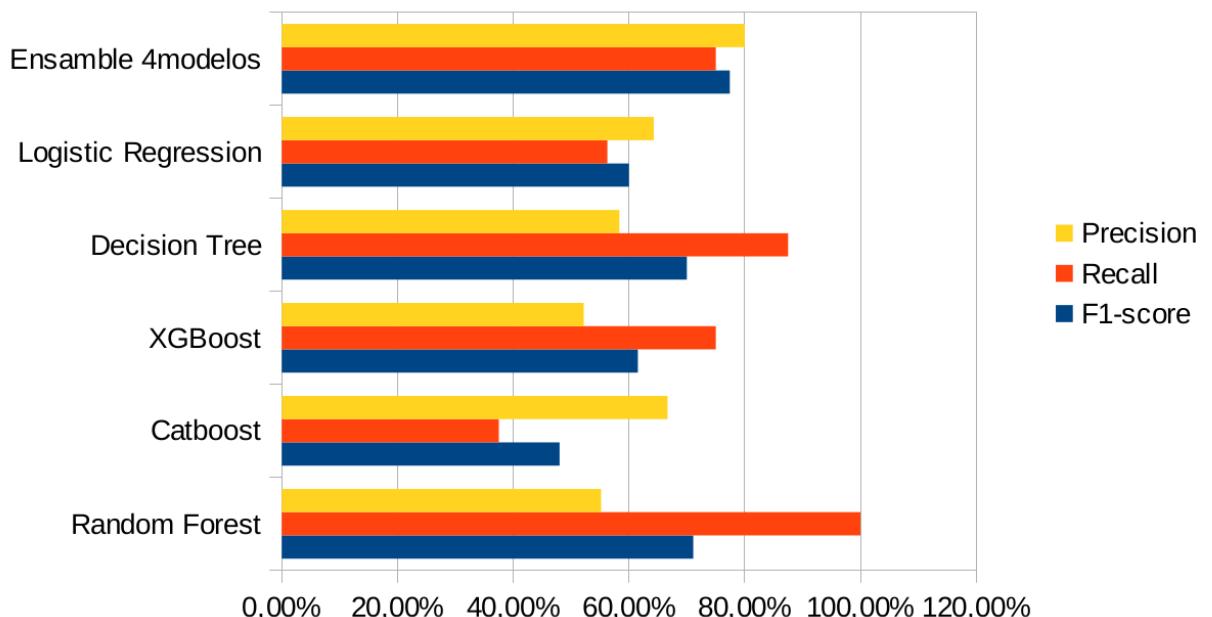
TN = não eleito acertado (modelo classificou corretamente como não eleito, pois foi de fato não eleito)

FN = eleito incorreto (modelo classificou erroneamente como não eleito, pois foi, de fato, eleito)

Frequencias na base testes 2008



Performance modelos base testes 2008



Precision: do total que o modelo classificou como eleito, qual percentual ele acertou ?

Recall: qual percentual dos efetivamente eleitos o modelo acertou ?

F1-score: métrica que mede um “balanço” entre precision e recall

Por fim, com o intuito de simular uma situação de produção, testamos 5 modelos com os dados de candidaturas a vereador do ano de 2012, no mesmo município de São Paulo. Os resultados estão resumidos na tabela e gráficos a seguir:

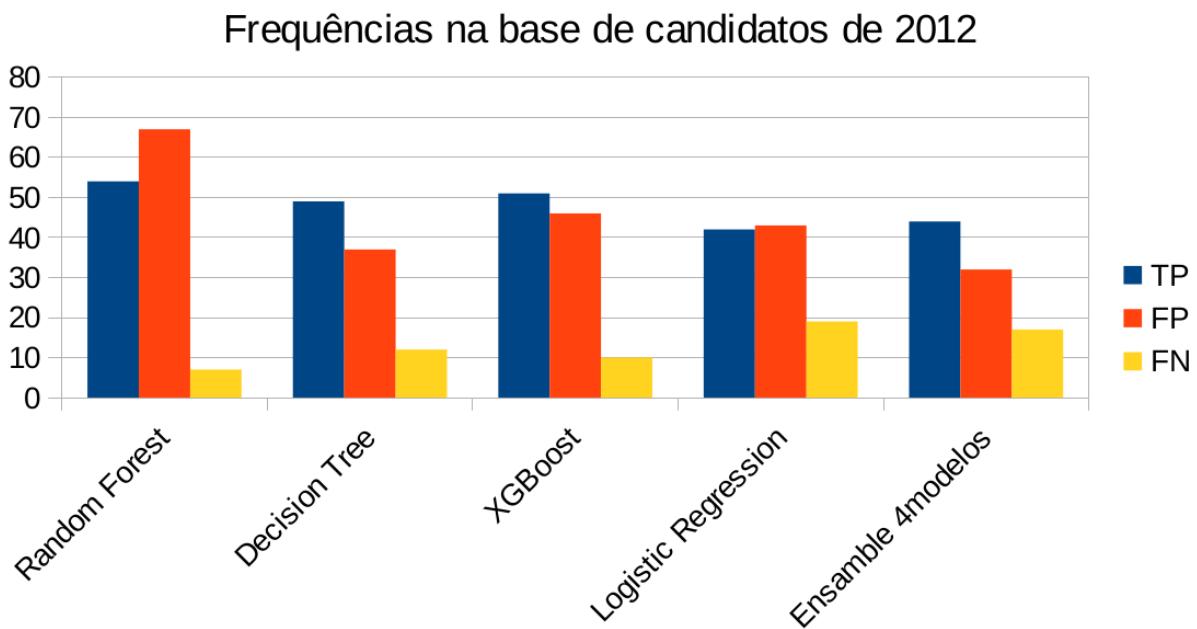
	TP	FP	TN	FN	F1-score	Recall	Precision
Random Forest	54	67	1247	7	59,34%	88,52%	44,63%
Decision Tree	49	37	1277	12	66,67%	80,33%	56,98%
XGBoost	51	46	1268	10	64,56%	83,61%	52,58%
Logistic Regression	42	43	1271	19	57,53%	68,85%	49,41%
Ensamble 4modelos	44	32	1282	17	64,23%	72,13%	57,89%

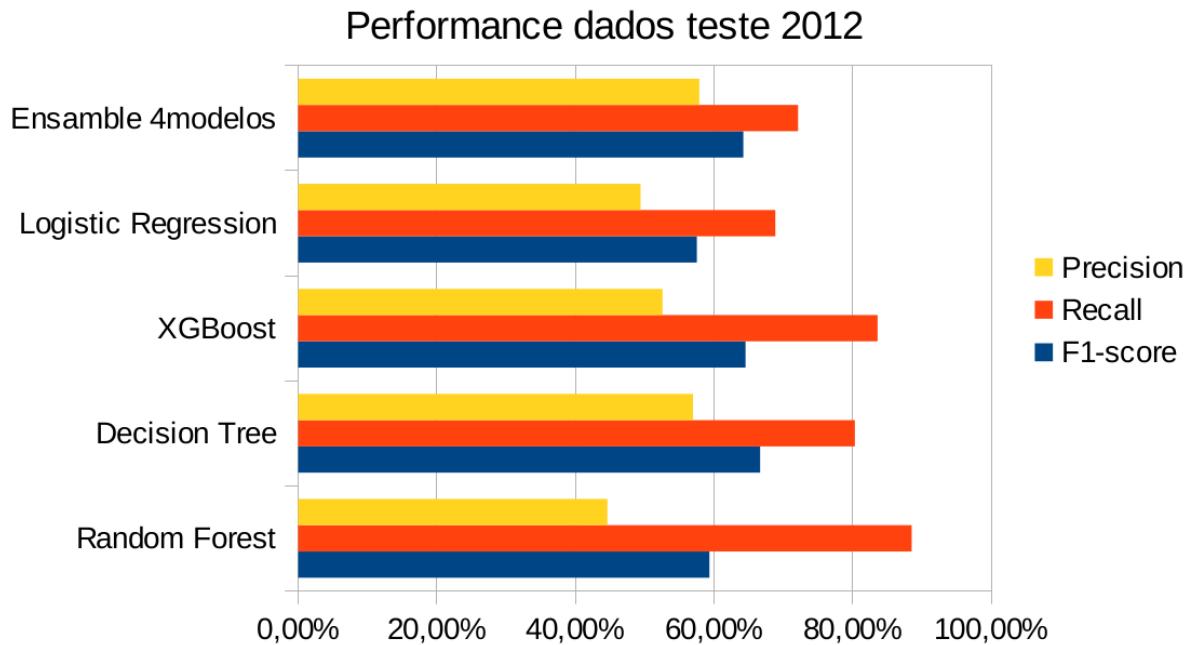
TP = eleitos acertados (modelo classificou corretamente como eleito, pois foi, de fato, eleito)

FP = não eleito incorreto (modelo classificou erroneamente como eleito, pois não foi eleito)

TN = não eleito acertado (modelo classificou corretamente como não eleito, pois foi de fato não eleito)

FN = eleito incorreto (modelo classificou erroneamente como não eleito, pois foi, de fato, eleito)





Precision: do total que o modelo classificou como eleito, qual percentual ele acertou ?

Recall: qual percentual dos efetivamente eleitos o modelo acertou ?

F1-score: métrica que mede um “balanço” entre precision e recall

Acreditamos que, de todos estes modelos, o “Ensamble” e o “Decision Tree” sejam os mais equilibrados. Vamos tentar ilustrar melhor:

fatos:

- Total de candidatos em 2012 no município de São Paulo: **1.375**
- Total de candidatos eleitos: **61**

Modelo Ensamble:

- classificou **76** candidatos como eleitos;
- destes 76, **acertou 44** candidatos e **errou 32**;
- outros **17** candidatos eleitos, nosso modelo classificou como não eleito;
- **classificou corretamente 1.282** candidatos como não eleitos.

Modelo Decision Tree:

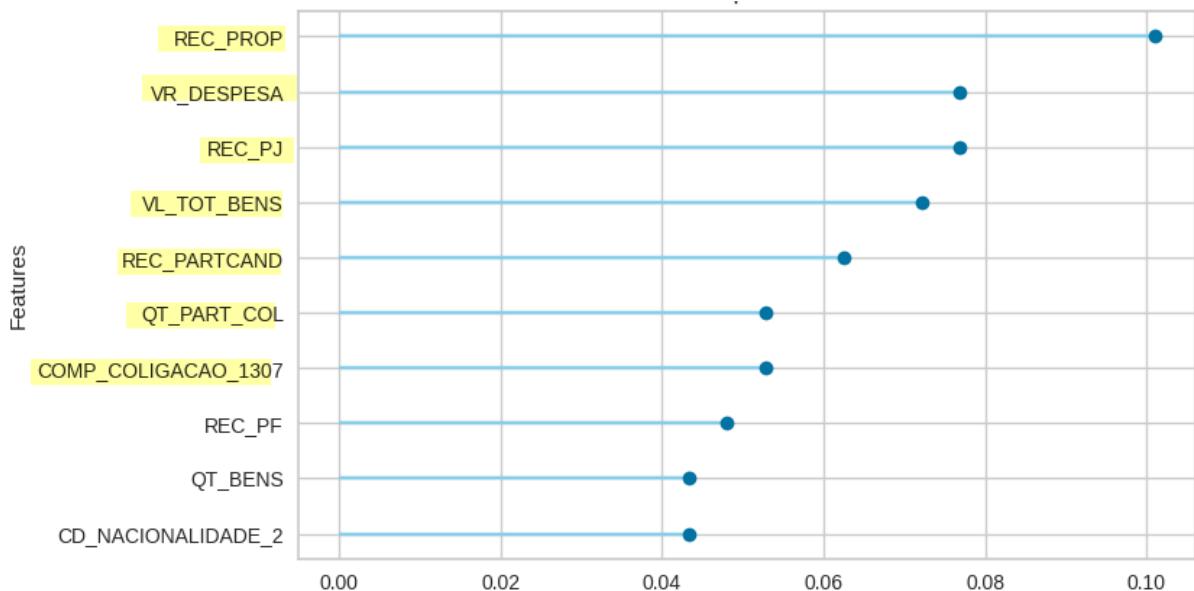
- classificou **86** candidatos como eleitos;
- destes 86, **acertou 49** candidatos e **errou 37**;
- outros **12** candidatos eleitos, nosso modelo classificou como não eleito;

- ***classificou corretamente 1.277*** candidatos como não eleitos.

O modelo Random Forest teve o maior número de acertos de candidatos eleitos (métrica Recall = 88,52%): **54**. Entretanto para acertar esses 54, classificou um total de **121** candidatos como eleitos, ou seja, 67 “falsos positivos” (classificou como eleito incorretamente), número muito superior que dos modelos Ensamble e Decision Tree. O fato do modelo *Random Forest* ter classificado um número grande de “falsos positivos” está refletido na métrica “*Precision*”. Por sinal, foi o modelo que teve o pior desempenho nesta métrica (44,6%). Reparem aqui, como a métrica *F1-Score*, representa um “equilíbrio” entre *Recall* e *Precision* e acaba sendo uma boa métrica para avaliação dos modelos.

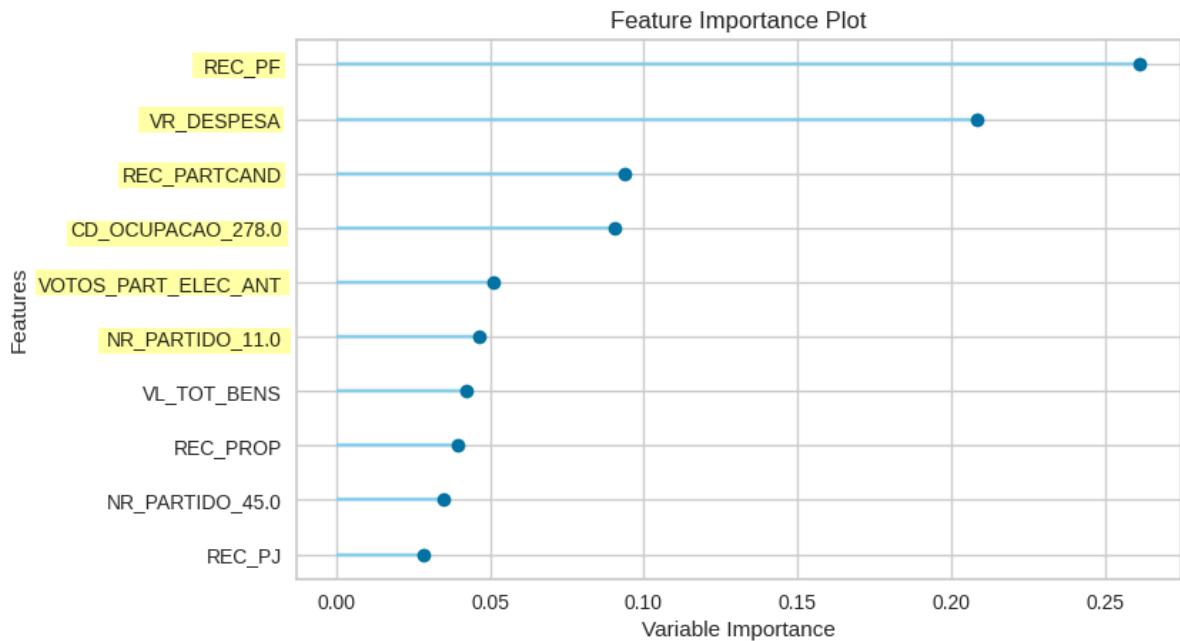
A seguir, mostramos quais as *features* (colunas) mais significativas, ou seja, aquelas que mais contribuem para explicar a “eleição” ou “não eleição” de um candidato para cada um dos modelos.³⁵ Interessante observar como cada modelo usa características variadas para a tomada de decisão.

- ***Random Forest***

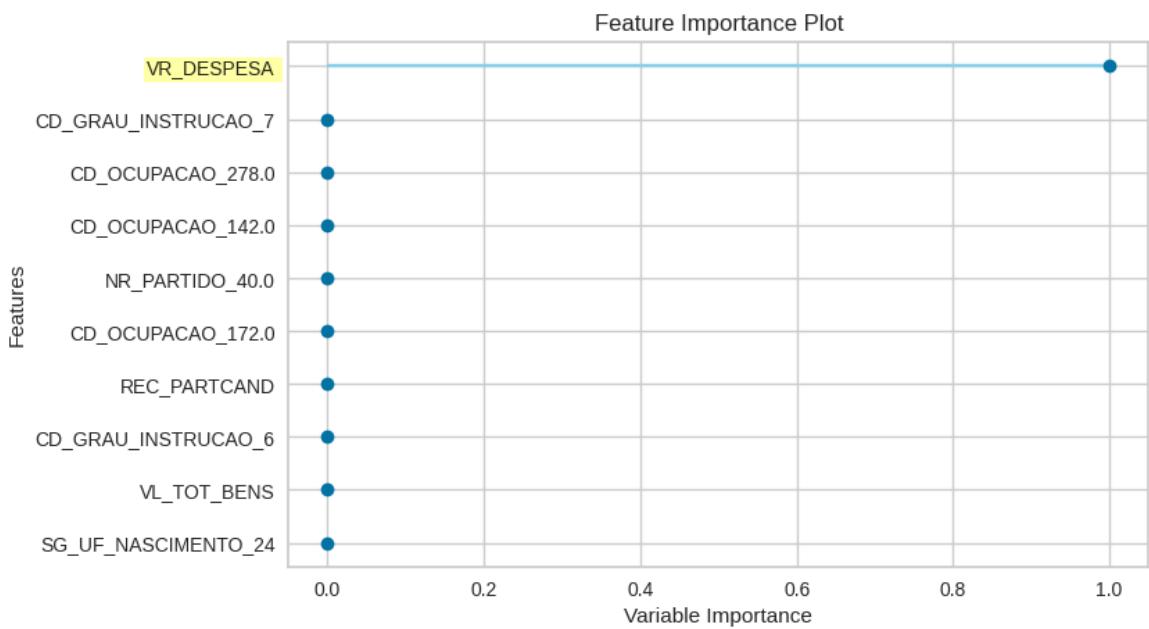


³⁵ Modelo ensemble, é uma votação dos 4 deamais, logo, todas as features dos 4 modelos influenciam nele.

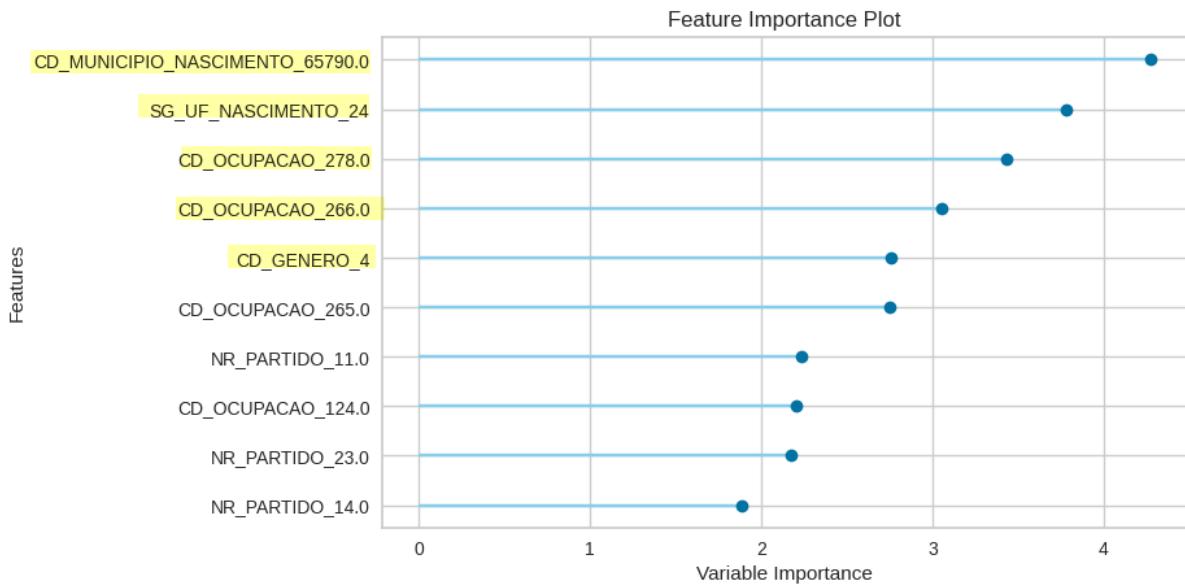
- ***Extra Gradient Boosting (XGBoost)***



- ***Decision Tree***



- ***Logistic Regression***



6.1. Observações finais e próximos passos

Nossa proposta inicial de tratar todos os municípios do estado de São Paulo ficou comprometida, em nossa opinião, por estarmos tentando tratar realidades muito diferentes de cada um dos municípios em um único conjunto de dados. Quando passamos a tratar apenas um único município (São Paulo – Capital) já pudemos notar uma melhora nos resultados do algorítimo de ML: saiu de F1-score³⁶ de 29% - todo o estado de SP, para 56% - somente São Paulo – capital.

Ao testarmos outras possibilidades de algorítimos de ML, com o *Pycaret*, conseguimos melhorar ainda mais o desempenho, chegando a até 74% de F1-score para a base de testes de 2008.

³⁶ F1-score aplicado à base de testes do ano 2008

Como prova final de nossos modelos, fizemos as previsões para as candidaturas do município de São Paulo para o ano de 2012 – dados totalmente desconhecidos dos nossos algorítimos de ML. Nesta prova, final conseguimos *F1-score* superior a 65% com 3 dos 5 modelos testados.

Um ponto que precisa ser implementado para que os resultados sejam mais realistas é colocar uma limitação na quantidade de “positivos” que cada modelo pode classificar. Como temos de antemão a quantidade de vagas para vereador, devemos limitar também a quantidade de *targets* classificados como “eleito” pelos modelos. Trata-se de uma implementação relativamente simples, bastando usarmos, ao invés do *target* “eleito” / “não eleito”, as probabilidades de cada *target* ser eleito. Assim, com estas probabilidades, bastaria escolher as 61 maiores (no caso do ano de 2012 em que haviam 61 vagas para vereador no município de São Paulo) como *target* “eleitos” pelos modelos. Desta forma, a comparação de desempenho dos modelos seria muito mais direta.

Assim, entendemos como bastante válido os ensaios aqui realizados e vemos bom potencial de melhora dos mesmos para dados mais recentes. Nossa escolha pelo ano de 2008 se deu principalmente pelo fato de precisarmos de 3 eleições sequenciais³⁷ sem muitas alterações na legislação entre elas. Neste aspecto, as alterações legislativas nas questões de coligações e proporcionalidade acabaram inviabilizando o nosso estudo em dados de eleições mais recentes.

Por outro lado, se partirmos das últimas eleições, teremos possibilidades muito maiores de agregar informações úteis à base de candidaturas, como por exemplo dados de redes sociais: quantidade de seguidores, análise de sentimentos dos *feedbacks* recebidos, tipos de postagens etc. Além disso, existem pontos como o desbalanceamento do *dataset* que também podem ser atacados.

Um outro ponto que também vemos possibilidades de avanço diz respeito ao tratamento isolado de municípios *versus* tratamento em conjunto (dentro do estado, por exemplo). Podemos fazer, com base em dados do IBGE, um agrupamento

³⁷ Foi necessário usar dados de 2004 para enriquecer o dataset e dados de 2012 para fazer o teste final, de simulação de produção.

(clusterização) de municípios que tenham realidades econômicas, sociais e culturais mais parecidas e, então, tentar tratar as candidaturas destes municípios em conjunto.

Por fim, o uso de Aprendizado de Máquina para previsões eleitorais nos parece um campo muito promissor e que vale o aprofundamento nos estudos.

Acreditamos que os elementos aqui tratados foram de suma importância para a fixação e aprofundamento de vários temas tratados ao longo da Especialização em Ciência de Dados e Big Data da PUC Minas. Além disso, nos abre um vasto horizonte de possibilidades para aplicação prática em nosso ambiente profissional.

7. Links

Link para o vídeo: <https://youtu.be/wU1AMn5kFoQ>

Link para o repositório: https://github.com/sfinotti/TCC_Puc_Minas.git

REFERÊNCIAS

- [1] PYTHON SOFTWARE FOUNDATION. **Python Language Site: Documentation.** Disponível em: <<https://www.python.org/doc/>>. Acesso em: 27 de jul. de 2021.
- [2] JUPYTER Notebook Disponível em: <<https://jupyter.org/>>. Acesso em: 27 de jul. de 2021.
- [3] PANDAS, Disponível em: <<https://pandas.pydata.org/>>. Acesso em: 27 de jul. de 2021.
- [4] NUMPY: The fundamental package for scientific computing with PythonDisponível em: <<https://numpy.org/>>. Acesso em: 27 de jul. de 2021.
- [5] MATPLOTLIB: Visualization with Python Disponível em: <<https://matplotlib.org/>>. Acesso em: 27 de jul. de 2021.
- [6] SEABORN: statistical data visualization Disponível em: <<https://seaborn.pydata.org/>>. Acesso em: 27 de jul. de 2021.
- [7] DIFFLIB — Helpers for computing deltas Disponível em: <<https://docs.python.org/3/library/difflib.html>>. Acesso em: 27 de jul. de 2021.
- [8] SCIKIT – LEARN Machine Learning in python. Disponível em: <<https://scikit-learn.org/stable/>>. Acesso em: 27 de jul. de 2021.
- [9] WAGACHA, Peter Waiganjo. **Induction of Decision Trees.** Nairobi: Institute of Computer Science, University of Nairobi, 2003.
- [10] VASSANDANI, Jasmine **A Data Science Workflow Canvas to Kickstart Your Projects.** Disponível em: <<https://towardsdatascience.com/a-data-science-workflow-canvas-to-kickstart-your-projects-db62556be4d0>>