PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA

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

Felipe Gaudêncio da Rocha

ANÁLISE DO PERFIL DOS RESULTADOS DAS ELEIÇÕES DE 2020 PARA OS CARGOS DE PREFEITOS E VEREADORES

Belo Horizonte

2022

Felipe Gaudêncio da Rocha

ANÁLISE DO PERFIL DOS RESULTADOS DAS ELEIÇÕES DE 2020 PARA OS CARGOS DE PREFEITOS E VEREADORES

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

2022

SUMÁRIO

1. Introdução

1.1. Contextualização

A história eleitoral do Brasil é tão rica que hoje representa o terceiro maior eleitoral do planeta, perdendo apenas para Índia e Estados Unidos. Em 1822, após a declaração de independência, o Brasil passou a ter eleições para as, eventuais, Assembleias Constituintes e para Assembleias Legislativa.

Em 1875, tivemos modificações que foram introduzidas com a Lei Saraiva: as eleições passaram a ser diretas, as juntas paroquiais de qualificação foram extintas, o alistamento foi entregue à magistratura, o título de eleitor foi instituído, substituindo o título de qualificação, e o analfabeto foi proibido de votar.

Apesar da Proclamação da República em 1889 e do surgimento de uma nova constituição em 1891, não houve grandes evoluções dos direitos políticos dos cidadãos na Primeira República, pois os que podiam votar eram os maiores de 21 anos que tivessem se alistado conforme determinação legal, vedada a participação de analfabetos e mulheres. Com esse cenário, apenas 2,2% da população votou na eleição de 1894. Nesse período, até 1930, teve-se a popularização do termo “voto de cabresto” que retrata a situação de um eleitor sendo coagido a ir votar sob a ordem das elites da época.

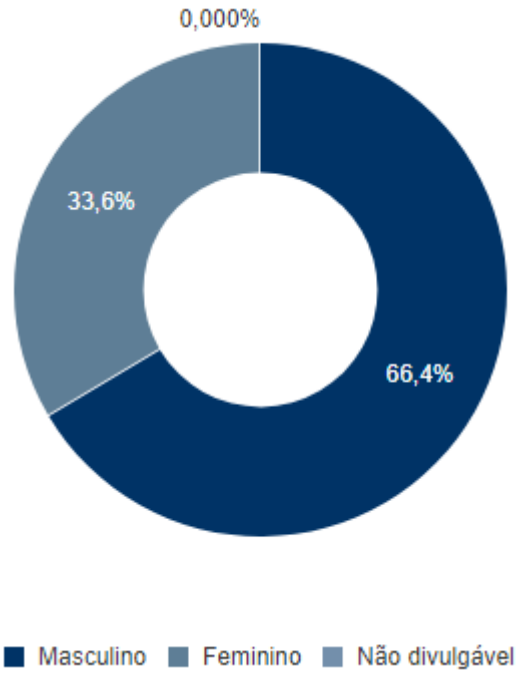
A criação da Justiça Eleitoral em 1932 faz parte do projeto de modernização imposto ao país pelo movimento de 1930. Sem dúvida, a inserção do Brasil no rol dos países civilizados passava pela confiabilidade do sistema eleitoral. Para atingir esse objetivo, além de um processo eleitoral transparente, era fundamental transformar os próprios eleitores.

1.2. Definição do problema

O objetivo com a presente análise é a identificação do perfil do eleitorado brasileiro, seria possível prever, com base no gênero, estado civil, idade, raça, escolaridade, e poder aquisitivo dos candidatos, os resultados das eleições para os cargos de prefeito e vereador na eleição de 2020?

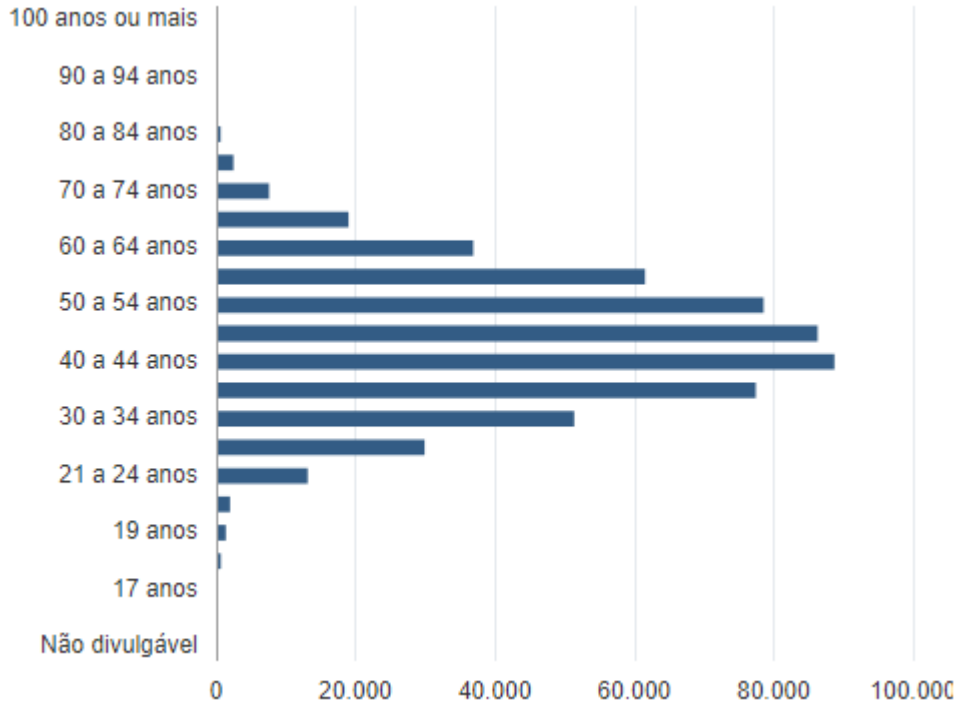
As informações e os gráficos foram extraídos do portal do TSE (Tribunal Superior Eleitoral), traz alguns dados das características desses eleitorados, mais especificamente sobre seu gênero, faixa etária, estado civil e grau de instrução.

- Gênero



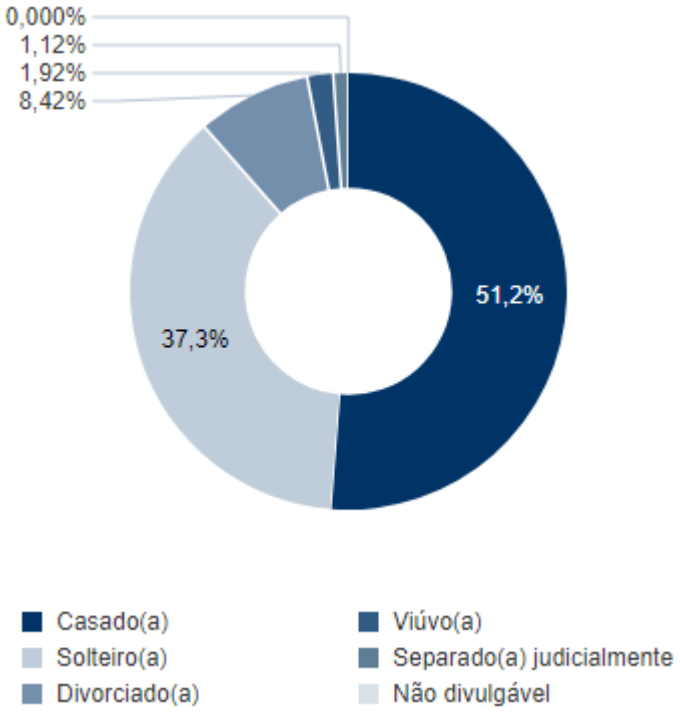
Gênero	Quantitativo de eleitorado	Porcentagem (%)
Masculino	370.381	66,4 %
Feminino	187.025	33,6 %
Não Informado	1	0,000 %

- Faixa etária



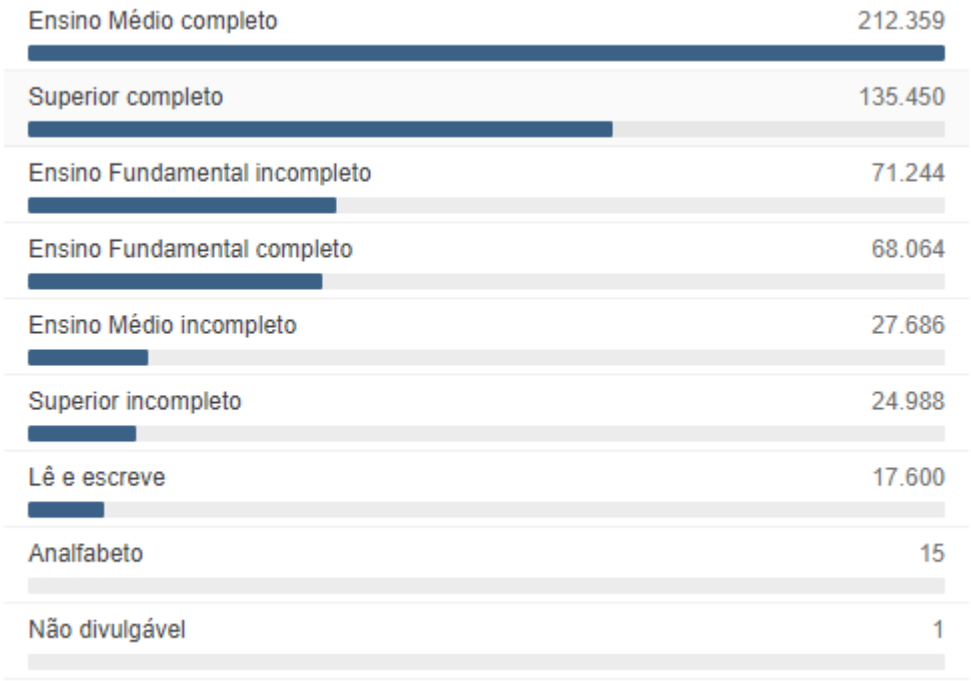
Faixa etária	Quantitativo de eleitorado	Porcentagem (%)
100 anos ou mais	6	0
95 a 99 anos	8	0
90 a 94 anos	20	0
85 a 89 anos	123	0,02
80 a 84 anos	628	0,11
75 a 79 anos	2470	0,44
70 a 74 anos	7590	1,36
65 a 69 anos	18986	3,41
60 a 64 anos	36864	6,61
55 a 59 anos	61514	11,04
50 a 54 anos	78497	14,08
45 a 49 anos	86264	15,48
40 a 44 anos	88660	15,91
35 a 39 anos	77386	13,88
30 a 34 anos	51349	9,21
25 a 29 anos	29894	5,36
21 a 24 anos	13131	2,36
20 anos	1974	0,35
19 anos	1390	0,25
18 anos	649	0,12
17 anos	2	0
16 anos	1	0
Não divulgável	1	0

- Estado civil



Estado Civil	Quantitativo de eleitorado	Porcentagem (%)
Casado(a)	285.546	51,23
Divorciado(a)	46.915	8,42
Não divulgável	1	0
Separado(a) judicialmente	6.228	1,12
Solteiro(a)	208.001	37,32
Viúvo(a)	10.716	1,92

- Grau de instrução



Grau instrução	Quantitativo de eleitorado	Porcentagem (%)
Analfabeto	15	0
Ensino Fundamental completo	68.064	12,21
Ensino Fundamental incompleto	71.244	12,78
Ensino Médio completo	212.359	38,1
Ensino Médio incompleto	27.686	4,97
Lê e escreve	17.600	3,16
Não divulgável	1	0
Superior completo	135.450	24,3
Superior incompleto	24.988	4,48

O eleitorado brasileiro é constituído por maioria do gênero masculino (66,4%), tem menos de 50 anos (62,92%), é solteiro (37,3%) e não possui Ensino Superior Completo (75,7%). Ao analisar o perfil dos candidatos eleitos para o cardo de prefeito e vereador, podemos

perceber que o perfil é bem diferente do que o eleitorado, sendo que dos 537.864 prefeitos e vereadores eleitos, são homens 84,3%, casados 62,5% e possuem superior completos 33,45%. O perfil dos representantes do povo no Congresso Nacional ainda é bem diferente do eleitorado, a representatividade feminina, por exemplo, que é de 15,7%, é bem abaixo que o perfil da população.

A diferença no perfil do eleitorado e dos candidatos eleitos mostram, que o perfil dos representantes do povo ainda segue o padrão, ou seja, homens brancos com elavado grau de escolaridade.

Portanto, o problema que se propõe é se seria possível expandir essa análise também para os cargos de Prefeito e Vereador e prever, com base no gênero, estado civil, idade, raça, escolaridade e poder aquisitivo dos candidatos, o resultado das eleições para os cargos de Prefeito e Vereador na eleição de 2020.

https://www.tse.jus.br/eleicoes/estatisticas/estatisticas-eleitorais

2. Coleta de Dados

Para realizar o tratamento do problema proposto, foram utilizados quatro datasets (conjunto de dados) relativos às eleições de 2020, extraídos em 17/09/2020 do repositório de dados eleitorais do Tribunal Superior Eleitoral.

O primeiro dataset, "consulta_cand_2020_BRASIL", disponível em https://cdn.tse.jus.br/estatistica/sead/odsele/consulta_cand/consulta_cand_2020.zip, traz toas as informações do candidato, a situação de sua candidatura e seu resultado na eleição. Esse dataset possui os seguintes campos:

Nome da coluna/campo	Descrição
DT_GERACAO	Data da extração dos dados para geração do arquivo.
HH_GERACAO	Hora da extração dos dados para geração do arquivo com base no horário de Brasília.

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. Por exemplo: Em 2016 houve eleições ordinárias. Após a data desta eleição ordinária e antes da próxima, houve eleições suplementares em 2017, 2018 e 2019. As informações destas eleições suplementares estarão divulgadas no arquivo gerado para as Eleições 2016.
CD_TIPO_ELEICAO	Código do tipo de eleição. Pode assumir os valores: 1 - Eleição Suplementar, 2 - Eleição Ordinária e 3 - Consulta Popular.
NM_TIPO_ELEICAO	Nome do tipo de eleição. Observação: As eleições ordinárias são previstas em Lei, possuem data certa para serem realizadas, ocorrem em anos pares e possuem a periodicidade de 04 em 04 anos.
NR_TURNO	Número do turno da eleição. Observação: No Brasil, as eleições realizam-se por meio de dois sistemas: o sistema majoritário (aplicado aos cargos de Presidente, Vice Presidente, Governador, Vice-Governador, Prefeito, Vice Prefeito e Senador) e o sistema proporcional (aplicado aos cargos de Deputado Federal, Deputado Estadual, Deputado Distrital e Vereador). O sistema majoritário consiste em declarar eleito o candidato que tenha recebido a maioria dos votos válidos (excluídos os votos em brancos e os votos nulos). Caso o candidato ao cargo indicado no sistema majoritário, com exceção do cargo de Senador, não alcance maioria absoluta destes votos válidos no primeiro turno (mínimo de 50% + 1), haverá segundo turno em que concorrerão

	apenas os dois candidatos mais votados. O segundo turno das eleições no Brasil ocorre para os cargos de Presidente, Vice-Presidente da República, Governadores e Vice-Governadores dos Estados e do Distrito Federal e para Prefeitos e Vice-Prefeitos de Municípios com mais de 200 mil eleitores. Nos municípios cujo eleitorado é igual ou menor que 200 mil e para o cargo de Senador elege-se o
CD_ELEICAO	Código único da eleição no âmbito da Justiça Eleitoral. Observação: Este código é único por eleição e por turno, ou seja, cada turno possui seu código de eleição.
DS_ELEICAO	Descrição da eleição.
DT_ELEICAO	Data em que ocorreu a eleição.
TP_ABRANGENCIA	Abrangência da eleição. Pode assumir os valores: Municipal, Estadual e Federal. Observação: A abrangência territorial da eleição está diretamente relacionada aos cargos eletivos e suas circunscrições eleitorais. As eleições realizadas na circunscrição Municipal são as eleições para os cargos de Prefeito, Vice-Prefeito e Vereador; as realizadas na circunscrição Estadual são para os cargos de Governador, Vice-Governador, Senador, Deputado Estadual, Deputado Federal e Deputado Distrital e; as realizadas na circunscrição Federal são para os cargos de Presidente e Vice-Presidente da República.
SG_UF	Sigla da Unidade da Federação em que ocorreu a eleição.
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 Federal (cargo de Presidente e Vice-Presidente) a sigla é BR. Em caso de abrangência Estadual (cargos de Governador, Vice Governador, Senador, Deputado Federal, Deputado Estadual e Deputado Distrital) a sigla é a UF da candidatura. Em caso de abrangência Municipal (cargos de Prefeito, Vice-Prefeito e Vereador) é o código de identificação do município da candidatura.
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.
CD_CARGO	Código do cargo ao qual o candidato concorre na eleição.
DS_CARGO	Cargo ao qual o candidato concorre na eleição.
SQ_CANDIDATO	Número sequencial do candidato, gerado internamente pelos sistemas eleitorais para cada eleição. Observação: não é o número de campanha do candidato.
NR_CANDIDATO	Número do candidato na urna.
NM_CANDIDATO	Nome completo do candidato.
NM_URNA_CANDIDATO	Nome do candidato que aparece na urna.
NM_SOCIAL_CANDIDATO	Nome social do candidato. Observação: Nome social é o nome pelo qual pessoas travestir ou transexuais preferem ser chamadas cotidianamente, em contraste com o nome oficialmente registrado, que não reflete sua identidade de gênero. A identidade

	do nome social é vinculada com a identidade civil original. Em âmbito federal, o Decreto nº 8.727 de 2016, garante o direito ao uso do nome social e reconhecimento da identidade de gênero de pessoas travestis e transexuais no âmbito da administração pública federal direta, autárquica e fundacional.
NR_CPF_CANDIDATO	Número do CPF do candidato.
NM_EMAIL	Endereço de e-mail do candidato.
CD_SITUACAO_CANDIDATURA	Código da situação do registro de candidatura do candidato.
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.
CD_DETALHE_SITUACAO_CAND	Código do detalhe da situação do registro de candidatura do candidato.
DS_DETALHE_SITUACAO_CAND	Detalhe da situação do registro de candidatura do candidato que especifica o motivo pelo qual a candidatura foi julgada como 'Apta' ou 'Inapta'
TP_AGREMIACAO	Tipo de agremiação da candidatura do candidato, ou seja, forma como o candidato concorrerá nas eleições. Pode assumir os valores: Coligação (quando o candidato concorre por coligação) e Partido Isolado (quando o candidato concorre somente pelo partido).
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.
SG_PARTIDO	Sigla do partido de origem do candidato.
NM_PARTIDO	Nome do partido de origem do candidato.
SQ_COLIGACAO	Sequencial da coligação da qual o candidato pertence, gerado pela Justiça Eleitoral.
NM_COLIGACAO	Nome da coligação da qual o candidato pertence.
DS_COMPOSICAO_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 intercarladas com o símbolo /.
CD_NACIONALIDADE	Código da nacionalidade do candidato.
DS_NACIONALIDADE	Nacionalidade do candidato.
SG_UF_NASCIMENTO	Sigla da Unidade da Federação de nascimento do candidato.
CD_MUNICIPIO_NASCIMENTO	Código de identificação do município de nascimento do candidato.
NM_MUNICIPIO_NASCIMENTO	Nome do município de nascimento do candidato.
DT_NASCIMENTO	Data de nascimento do candidato.
NR_IDADE_DATA_POSSE	Idade do candidato na data da posse. A idade é calculada com base na data da posse do referido candidato para o cargo e unidade eleitoral constantes no arquivo de vagas.
NR_TITULO_ELEITORAL_CANDIDATO	Número do título eleitoral do candidato.
CD_GENERO	Código do gênero do candidato.
DS_GENERO	Gênero do candidato.
CD_GRAU_INSTRUCAO	Código do grau de instrução do candidato.
DS_GRAU_INSTRUCAO	Grau de instrução do candidato.
CD_ESTADO_CIVIL	Código do estado civil do candidato.
DS_ESTADO_CIVIL	Estado civil do candidato.

CD_COR_RACA	Código da cor/raça do candidato. (autodeclaração)
DS_COR_RACA	Cor/raça do candidato. (autodeclaração)
CD_OCUPACAO	Código da ocupação do candidato.
DS_OCUPACAO	Ocupação do candidato.
VR_DESPESA_MAX_CAMPANHA	Valor máximo, em reais, de despesas de campanha declarada pelo partido para aquele candidato.
CD_SIT_TOT_TURNO	Código da situação de totalização do candidato, naquele turno da eleição, após a totalização dos votos.
DS_SIT_TOT_TURNO	Situação de totalização do candidato, naquele turno da eleição, após a totalização dos votos.
ST_REELEICAO	Indica se o candidato está concorrendo ou não à reeleição. Pode assumir os valores: S - Sim e N - Não. Informação autodeclarada pelo candidato. Observação: Reeleição é a renovação do mandato para o mesmo cargo eletivo, por mais um período, na mesma circunscrição eleitoral na qual o representante, no pleito imediatamente anterior, se elegeu. Pelo sistema eleitoral brasileiro, o presidente da República, os governadores de estado e os prefeitos podem ser reeleitos para um único período subsequente, o que se aplica também ao vice-presidente da República, aos vice-governadores e aos vice-prefeitos. Já os parlamentares (senadores, deputados federais e estaduais/distritais e vereadores) podem se reeleger ilimitadas vezes. A possibilidade da reeleição compreende algumas regras mais específicas detalhadas no sistema eleitoral brasileiro.
ST_DECLARAR_BENS	Indica se o candidato tem ou não bens a declarar. Pode assumir os valores: S - Sim e N - Não. Esta

	informação é fornecida pelo próprio candidato no momento do pedido da candidatura.
NR_PROTOCOLO_CANDIDATURA	Número do protocolo de registro de candidatura do candidato.
NR_PROCESSO	Número do processo de registro de candidatura do candidato.
CD_SITUACAO_CANDIDATO_PLEITO	Código da situação da candidatura no dia do Pleito.
DS_SITUACAO_CANDIDATO_PLEITO	Situação da candidatura no dia do Pleito.
CD_SITUACAO_CANDIDATO_URNA	Código da situação da candidatura na urna.
DS_SITUACAO_CANDIDATO_URNA	Situação da candidatura na urna.
ST_CANDIDATO_INSERIDO_URNA	Informa se o candidato foi inserido na urna eletrônica. (S/N)

O segundo *dataset*, "bem_candidato_2020_BRASIL", disponível em https://cdn.tse.jus.br/estatistica/sead/odsele/bem_candidato/bem_candidato_2020.zip, traz a declaração de bens de cada candidato para a eleição em questão e possui os seguintes campos:

Nome da coluna/campo	Descrição
DT_GERACAO	Data de geração do arquivo (data da extração dos dados).
HH_GERACAO	Hora de geração do arquivo (hora da extração) - Horário de Brasília.
ANO_ELEICAO	Ano da eleição referente ao ano eleitoral de pesquisa.
CD_TIPO_ELEICAO	Código do tipo de eleição.
NM_TIPO_ELEICAO	Nome do tipo de eleição.
CD_ELEICAO	Código da eleição.
DS_ELEICAO	Descrição da eleição.
DT_ELEICAO	Data em que ocorreu a eleição.
SG_UF	Sigla da Unidade da Federação em que ocorreu a eleição.
SG_UE	Sigla da Unidade Eleitoral do candidato (Em caso de eleição majoritária é a sigla da UF que o candidato concorre e em caso de eleição municipal é o código TSE do município).

NM_UE	Nome de Unidade Eleitoral do candidato (Em caso de eleição majoritária é o nome da UF que o candidato concorre e em caso de eleição municipal é o nome do município).
SQ_CANDIDATO	Número sequencial do candidato gerado internamente pelos sistemas eleitorais. Não é o número de campanha do candidato.
NR_ORDEM_CANDIDATO	Número da ordem do bem declarado do candidato.
CD_TIPO_BEM_CANDIDATO	Código do tipo do bem do candidato.
DS_TIPO_BEM_CANDIDATO	Descrição do tipo do bem do candidato.
DS_BEM_CANDIDATO	Descrição detalhada do bem do candidato.
VR_BEM_CANDIDATO	Valor declarado em reais do bem do candidato.
DT_ULTIMA_ATUALIZACAO	Data da última atualização do registro do bem.
HH_ULTIMA_ATUALIZACAO	Hora da última atualização do registro do bem.

O terceiro dataset, “despesas_contratadas_candidatos_2020_BRASIL.csv” e o quarto, “despesas_pagas_candidatos_2020_BRASIL.csv”, foram obtidos no link https://cdn.tse.jus.br/estatistica/sead/odsele/prestacao_contas/prestacao_de_contas_eleitorais_candidatos_2020.zip.

O dataset “despesas_contratadas_candidatos_2020_BRASIL.csv” traz as despesas contratadas pelos candidatos na eleição e possui a seguinte estrutura:

Nome da coluna/campo	Descrição
DT_GERACAO	Data de geração das informações (data da extração dos dados).
HH_GERACAO	Hora de geração das informações (hora da extração dos dados) - Horário de Brasília.
ANO_ELEICAO	Ano da eleição referente ao ano eleitoral de pesquisa.
CD_TIPO_ELEICAO	Código do tipo de eleição.
NM_TIPO_ELEICAO	Nome do tipo de eleição.
CD_ELEICAO	Código da eleição.
DS_ELEICAO	Descrição da eleição.

DT_ELEICAO	Data em que ocorreu a eleição.
ST_TURNO	O indicativo se prestador de contas teve prestação para 2º turno.
TP_PRESTACAO_CONTAS	Tipo de entrega da prestação de contas. Pode assumir os valores: Parcial (referente à entrega parcial de prestação de contas); Final (referente à entrega final da prestação); Relatório financeiro (referente à entrega do relatório financeiro).
DT_PRESTACAO_CONTAS	Data de entrega da prestação de contas junto ao TSE.
SQ_PRESTADOR_CONTAS	Sequencial de identificação do prestador de contas junto ao TSE.
SG_UF	Sigla da unidade da federação de abrangência do prestador de contas.
SG_UE	Sigla da Unidade Eleitoral do candidato (Em caso de eleição majoritária é a sigla da UF que o candidato concorre e em caso de eleição municipal é o código TSE do município).
NM_UE	Nome de Unidade Eleitoral do candidato (Em caso de eleição majoritária é o nome da UF que o candidato concorre e em caso de eleição municipal é o nome do município).
NR_CNPJ_PRESTADOR_CONTA	Numero do CNPJ do prestador de contas.
CD_CARGO	Código do cargo do candidato prestador de contas.
DS_CARGO	Descrição do cargo do candidato prestador de contas.
SQ_CANDIDATO	Sequencial do candidato prestador de contas.
NR_CANDIDATO	Número do candidato prestador de contas.
NM_CANDIDATO	Nome completo do candidato.
NR_CPF_CANDIDATO	CPF do candidato registrado na Justiça Eleitoral
NR_CPF_VICE_CANDIDATO	CPF do candidato à vice/suplente do titular, se houver.
NR_PARTIDO	Número do partido do candidato.
SG_PARTIDO	Sigla do partido do candidato.

NM_PARTIDO	Nome do partido do candidato.
CD_TIPO_FORNECEDOR	Código de identificação do tipo de fornecedor informada pelo prestador de contas em relação à despesa.
DS_TIPO_FORNECEDOR	Descrição do tipo de fornecedor informada pelo prestador de contas em relação à despesa. Pode assumir os valores: 'Pessoa Física' ou 'Pessoa Jurídica'.
CD_CNAE_FORNECEDOR	Código CNAE do fornecedor de bens e/ou serviços, se pessoa jurídica.
DS_CNAE_FORNECEDOR	Descrição do CNAE (Código do Setor Econômico) do fornecedor de bens e/ou serviços, se pessoa jurídica.
NR_CPF_CNPJ_FORNECEDOR	Número do CPF/CNPJ do fornecedor de bens e/ou serviços informada pelo prestador de contas em relação à despesa.
NM_FORNECEDOR	Nome do fornecedor de bens e/ou serviços declarado a Justiça Eleitoral, informada pelo prestador de contas em relação à despesa.
NM_FORNECEDOR_RFB	Nome do fornecedor cadastrado na Receita Federal do Brasil, informada pelo prestador de contas em relação à despesa.
CD_ESFERA_PART_FORNECEDOR	Código do tipo de esfera partidária do fornecedor, quando fornecedor 'Órgão partidário'.
DS_ESFERA_PART_FORNECEDOR	Descrição do tipo de esfera partidária do fornecedor. Pode assumir os valores: 'Nacional', 'Estadual', 'Distrital' e 'Municipal'. Válida para quando fornecedor 'Órgão partidário'.
SG_UF_FORNECEDOR	Sigla da unidade da federação do fornecedor, quando fornecedor candidato ou órgão partidário.
CD_MUNICIPIO_FORNECEDOR	Código do município do fornecedor, quando a esfera partidária do fornecedor for municipal.
NM_MUNICIPIO_FORNECEDOR	Descrição do município do fornecedor, quando a esfera partidária do fornecedor for municipal.

SQ_CANDIDATO_FORNECEDOR	Sequencial do candidato fornecedor, quando fornecedor candidato.
NR_CANDIDATO_FORNECEDOR	Número do candidato declarado pelo prestador de contas, quando fornecedor candidato.
CD_CARGO_FORNECEDOR	Código do cargo do candidato declarado pelo prestador de contas, quando fornecedor.
DS_CARGO_FORNECEDOR	Descrição do cargo do candidato declarado pelo prestador de contas, quando fornecedor.
NR_PARTIDO_FORNECEDOR	Número do partido declarado pelo prestador de contas, quando fornecedor candidato ou órgão partidário.
SG_PARTIDO_FORNECEDOR	Sigla do partido declarado pelo prestador de contas, quando fornecedor candidato ou órgão partidário.
NM_PARTIDO_FORNECEDOR	Nome do partido declarado pelo prestador de contas, quando fornecedor candidato ou órgão partidário.
DS_TIPO_DOCUMENTO	Tipo de documento. Podendo assumir os valores 'Recibo', 'Cupom Fiscal', 'Fatura', 'NotaFiscal', 'Duplicata', 'Outros'.
NR_DOCUMENTO	Número de documento que comprove a despesa.
SQ_DESPESA	Sequencial de identificação do registro da despesa declarada pelo prestador de contas.
DT_DESPESA	Data da despesa declarada à Justiça Eleitoral.
DS_DESPESA	Descrição do gasto no elenco de aplicações informada pelo prestador de contas em relação à despesa.
VR_DESPESA_CONTRATADA	Valor da despesa contratada em Reais (R$), informada pelo prestador de contas em relação à despesa.

O *dataset* "despesas_pagas_candidatos_2020_BRASIL.csv" traz as despesas efetivamente pagas pelos candidatos na eleição e possui a seguinte estrutura:

Nome da coluna/campo	Descrição
CD_ELEICAO	Código da eleição.
DS_ELEICAO	Descrição da eleição.
DT_ELEICAO	Data em que ocorreu a eleição.

ST_TURNO	Indicativo se prestador de contas participou do 2º turno.
TP_PRESTACAO_CONTAS	Tipo de entrega da prestação de contas. Pode assumir os valores: Relatório financeiro (referente à entrega do relatório financeiro); Parcial (referente à entrega parcial de prestação de contas); Final (referente à entrega final da prestação)
DT_PRESTACAO_CONTAS	Data de entrega da prestação de contas junto ao TSE.
SQ_PRESTADOR_CONTAS	Sequencial de identificação do prestador de contas junto ao TSE.
SG_UF	Sigla da unidade da federação de abrangência do prestador de contas.
DS_TIPO_DOCUMENTO	Tipo de documento. Podendo assumir os valores 'Recibo', 'Cupom Fiscal', 'Fatura', 'NotaFiscal', 'Duplicata', 'Outros'.
NR_DOCUMENTO	Número de documento que comprove a despesa.
CD_FONTE_DESPESA	Código de identificação do tipo de fonte de recursos da despesa, informado pelo prestador de contas.
DS_FONTE_DESPESA	Descrição do tipo de fonte do recurso da despesa, informado pelo prestador de contas.
CD_ORIGEM_DESPESA	Código de identificação do tipo de origem da despesa (DRD), informado pelo prestador de contas em relação à despesa
DS_ORIGEM_DESPESA	Descrição do tipo de origem da despesa (DRD), informado pelo prestador de contas em relação à despesa.
CD_NATUREZA_DESPESA	Código da natureza de recursos da despesa.
DS_NATUREZA_DESPESA	Descrição da natureza de recursos da despesa. Pode assumir os valores 'Financeiro' ou 'Estimável'.
CD_ESPECIE_RECURSO	Código de identificação da espécie de recursos para pagamento da despesa
DS_ESPECIE_RECURSO	Descrição da espécie do recurso para pagamento da despesa.
SQ_DESPESA	Sequencial de identificação do registro da despesa

	declarada pelo prestador de contas.
SQ_PARCELAMENTO_DESPESA	Sequencial de identificação do registro da de
DT_PAGTO_DESPESA	Data de pagamento da despesa declarada à Justiça Eleitoral.
DS_DESPESA	Descrição da despesa.
VR_PAGTO_DESPESA	Valor pago da despesa em Reais (R$), informada pelo prestador de conta

3. Processamento/Tratamento de Dados

Para analisar os dados utilizei a biblioteca “pandas”, criada para a linguagem Python para manipulação e análise de dados. Foram utilizadas seguintes ferramentas, Python versão 3.9.7 e Jupyter Notebook versão 6.4.5.

Serão realizados a leitura e tratamento de 4 datasets, são eles:

“consulta_cand_2020_BRASIL.csv”;
"bem_candidato_2020_BRASIL.csv";
"despesas_contratadas_candidatos_2020_BRASIL.csv";
"despesas_pagas_candidatos_2020_BRASIL.csv".

Para realizar a leitura e o tratamento dos datasets, precisamos realizar a importação da biblioteca “pandas”. Em seguida, foi realizado a leitura do dataset “consulta_cand_2020_BRASIL.csv” com 557.990 entradas e 63 colunas.

```
import pandas as pd
```

```
df_consulta = pd.read_csv("consulta_cand_2020_BRASIL.csv", encoding = "Latin 1", sep = ";", decimal = ",")
```

```
df_consulta.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 557990 entries, 0 to 557989
Data columns (total 63 columns):
 #   Column                           Non-Null Count   Dtype
---  ------                           --------------   -----
 0   DT_GERACAO                       557990 non-null  object
 1   HH_GERACAO                       557990 non-null  object
 2   ANO_ELEICAO                      557990 non-null  int64
 3   CD_TIPO_ELEICAO                  557990 non-null  int64
 4   NM_TIPO_ELEICAO                  557990 non-null  object
 5   NR_TURNO                         557990 non-null  int64
 6   CD_ELEICAO                       557990 non-null  int64
 7   DS_ELEICAO                       557990 non-null  object
 8   DT_ELEICAO                       557990 non-null  object
 9   TP_ABRANGENCIA                   557990 non-null  object
 10  SG_UF                            557990 non-null  object
 11  SG_UE                            557990 non-null  int64
 12  NM_UE                            557990 non-null  object
 13  CD_CARGO                         557990 non-null  int64
 14  DS_CARGO                         557990 non-null  object
 15  SQ_CANDIDATO                     557990 non-null  int64
 16  NR_CANDIDATO                     557990 non-null  int64
 17  NM_CANDIDATO                     557990 non-null  object
 18  NM_URNA_CANDIDATO                557987 non-null  object
 19  NM_SOCIAL_CANDIDATO              557990 non-null  object
 20  NR_CPF_CANDIDATO                 557990 non-null  object
 21  NM_EMAIL                         557990 non-null  object
 22  CD_SITUACAO_CANDIDATURA          557990 non-null  int64
 23  DS_SITUACAO_CANDIDATURA          557990 non-null  object
 24  CD_DETALHE_SITUACAO_CAND         557990 non-null  int64
 25  DS_DETALHE_SITUACAO_CAND         557990 non-null  object
 26  TP_AGREMIACAO                    557990 non-null  object
 27  NR_PARTIDO                       557990 non-null  int64
 28  SG_PARTIDO                       557990 non-null  object
 29  NM_PARTIDO                       557990 non-null  object
 30  SQ_COLIGACAO                     557990 non-null  int64
 31  NM_COLIGACAO                     557990 non-null  object
 32  DS_COMPOSICAO_COLIGACAO          557990 non-null  object
 33  CD_NACIONALIDADE                 557990 non-null  int64
 34  DS_NACIONALIDADE                 557990 non-null  object
 35  SG_UF_NASCIMENTO                 557990 non-null  object
 36  CD_MUNICIPIO_NASCIMENTO          557990 non-null  int64
 37  NM_MUNICIPIO_NASCIMENTO          557990 non-null  object
 38  DT_NASCIMENTO                    557989 non-null  object
 39  NR_IDADE_DATA_POSSE              557989 non-null  float64
 40  NR_TITULO_ELEITORAL_CANDIDATO    557990 non-null  int64
 41  CD_GENERO                        557990 non-null  int64
 42  DS_GENERO                        557990 non-null  object
 43  CD_GRAU_INSTRUCAO                557990 non-null  int64
 44  DS_GRAU_INSTRUCAO                557990 non-null  object
 45  CD_ESTADO_CIVIL                  557990 non-null  int64
 46  DS_ESTADO_CIVIL                  557990 non-null  object
 47  CD_COR_RACA                      557990 non-null  int64
 48  DS_COR_RACA                      557990 non-null  object
 49  CD_OCUPACAO                      557990 non-null  int64
 50  DS_OCUPACAO                      557990 non-null  object
 51  VR_DESPESA_MAX_CAMPANHA          557990 non-null  object
 52  CD_SIT_TOT_TURNO                 557990 non-null  int64
 53  DS_SIT_TOT_TURNO                 557990 non-null  object
 54  ST_REELEICAO                     557990 non-null  object
 55  ST_DECLARAR_BENS                 557990 non-null  object
 56  NR_PROTOCOLO_CANDIDATURA         557990 non-null  int64
 57  NR_PROCESSO                      557990 non-null  int64
 58  CD_SITUACAO_CANDIDATO_PLEITO     557990 non-null  int64
 59  DS_SITUACAO_CANDIDATO_PLEITO     557990 non-null  object
 60  CD_SITUACAO_CANDIDATO_URNA       557990 non-null  int64
 61  DS_SITUACAO_CANDIDATO_URNA       557990 non-null  object
 62  ST_CANDIDATO_INSERIDO_URNA       557990 non-null  object
dtypes: float64(1), int64(25), object(37)
memory usage: 268.2+ MB
```

O *dataframe* traz informações dos candidatos a Prefeitos, Vice-Prefeitos e Vereador, sendo que será considerada apenas candidaturas aptas, para isso, vamos selecionar a coluna 'DS_SITUACAO_CANDIDATURA'.

```
df_consulta['DS_SITUACAO_CANDIDATURA'].unique()
```

```
array(['APTO', 'INAPTO', 'CADASTRADO'], dtype=object)
```

```
df_consulta = df_consulta[df_consulta.DS_SITUACAO_CANDIDATURA.isin(['APTO'])]
```

```
df_consulta.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 532281 entries, 0 to 557989
Data columns (total 63 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   DT_GERACAO                      532281 non-null  object
 1   HH_GERACAO                      532281 non-null  object
 2   ANO_ELEICAO                     532281 non-null  int64
 3   CD_TIPO_ELEICAO                 532281 non-null  int64
 4   NM_TIPO_ELEICAO                 532281 non-null  object
 5   NR_TURNO                        532281 non-null  int64
 6   CD_ELEICAO                      532281 non-null  int64
 7   DS_ELEICAO                      532281 non-null  object
 8   DT_ELEICAO                      532281 non-null  object
 9   TP_ABRANGENCIA                  532281 non-null  object
 10  SG_UF                           532281 non-null  object
```

Como podemos perceber, o dataframe possui agora 532.281 entradas, com 63 colunas. Dessa forma, vamos utilizar apenas as colunas com as seguintes relevâncias:

Variável	Descrição
SQ_CANDIDATO	Número sequencial do candidato, gerado internamente pelos sistemas eleitorais para cada eleição. Observação: não é o número de campanha do candidato.
NR_IDADE_DATA_POSSE	Idade do candidato na data da posse. A idade é calculada com base na data da posse do referido candidato para o cargo e unidade eleitoral constantes no arquivo de vagas.
DS_GENERO	Gênero do candidato.
DS_GRAU_INSTRUCAO	Grau de instrução do candidato.

DS_ESTADO_CIVIL	Estado civil do candidato.
DS_COR_RACA	Cor/raça do candidato. (autodeclaração)
DS_SIT_TOT_TURNO	Situação de totalização do candidato, naquele turno da eleição, após a totalização dos votos.
VR_DESPESA_MAX_CAMPANHA	Valor máximo, em reais, de despesas de campanha declarada pelo partido para aquele candidato.

Para selecionar as colunas acima, foi utilizado o seguinte código:

```
df_consulta = df_consulta[['SQ_CANDIDATO','NR_IDADE_DATA_POSSE','DS_GENERO','DS_GRAU_INSTRUCAO',
                           'DS_ESTADO_CIVIL','DS_COR_RACA','DS_SIT_TOT_TURNO','VR_DESPESA_MAX_CAMPANHA']]
```

```
df_consulta.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 532281 entries, 0 to 557989
Data columns (total 8 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   SQ_CANDIDATO            532281 non-null  int64
 1   NR_IDADE_DATA_POSSE     532280 non-null  float64
 2   DS_GENERO               532281 non-null  object
 3   DS_GRAU_INSTRUCAO       532281 non-null  object
 4   DS_ESTADO_CIVIL         532281 non-null  object
 5   DS_COR_RACA             532281 non-null  object
 6   DS_SIT_TOT_TURNO        532281 non-null  object
 7   VR_DESPESA_MAX_CAMPANHA  532281 non-null  object
dtypes: float64(1), int64(1), object(6)
memory usage: 36.5+ MB
```

Percebemos que temos 532.281 entradas filtradas e com apenas 8 colunas escolhidas. Para validar se há algum dado nulo no dataframe, utilizamos a função “isnull()”, juntamente com a função “sum()”, que soma a os valores encontrados.

```
df_consulta.isnull().sum()
```

```
SQ_CANDIDATO            0
NR_IDADE_DATA_POSSE     1
DS_GENERO               0
DS_GRAU_INSTRUCAO       0
DS_ESTADO_CIVIL         0
DS_COR_RACA             0
DS_SIT_TOT_TURNO        0
VR_DESPESA_MAX_CAMPANHA    0
dtype: int64
```

Após o comando, identificamos que na coluna “NR_IDADE_DATA_POSSE” há um valor nulo, portanto, sabemos que um candidato na data da posse não informou a sua idade.

Também identificamos que a um que na data da posse um candidato informou sua data errada como 999 anos.

```
df_consulta.NR_IDADE_DATA_POSSE.max()
```

```
999.0
```

Portanto, iremos alterar o “NR.IDADE_DATA_POSSE” para 0, senão a visualização dos histogramas será prejudicada, já que o valor da idade da posse do candidato está errado, sendo impossível uma pessoa ter 999 anos.

A leitura do segundo *dataset*, “bem_candidato_2020_BRASIL”, é feita da mesma forma do primeiro, também seguindo os parâmetros do documento que o especifica.

```
df_bem = pd.read_csv("bem_candidato_2020_BRASIL.csv", encoding = "Latin 1", sep = ";", decimal = ",")
```

```
df_bem.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1013809 entries, 0 to 1013808
Data columns (total 19 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   DT_GERACAO             1013809 non-null  object
 1   HH_GERACAO             1013809 non-null  object
 2   ANO_ELEICAO            1013809 non-null  int64
 3   CD_TIPO_ELEICAO        1013809 non-null  int64
 4   NM_TIPO_ELEICAO        1013809 non-null  object
 5   CD_ELEICAO             1013809 non-null  int64
 6   DS_ELEICAO             1013809 non-null  object
 7   DT_ELEICAO             1013809 non-null  object
 8   SG_UF                  1013809 non-null  object
 9   SG_UE                  1013809 non-null  int64
 10  NM_UE                  1013809 non-null  object
 11  SQ_CANDIDATO           1013809 non-null  int64
 12  NR_ORDEM_CANDIDATO     1013809 non-null  int64
 13  CD_TIPO_BEM_CANDIDATO  1013809 non-null  int64
 14  DS_TIPO_BEM_CANDIDATO  1013809 non-null  object
 15  DS_BEM_CANDIDATO       1013809 non-null  object
 16  VR_BEM_CANDIDATO       1013809 non-null  float64
 17  DT_ULTIMA_ATUALIZACAO  1013809 non-null  object
 18  HH_ULTIMA_ATUALIZACAO  1013809 non-null  object
dtypes: float64(1), int64(7), object(11)
memory usage: 147.0+ MB
```

Esse d*ataframe* possui 1.013.808 entradas divididas em 19 colunas, conforme demonstrado após a utilização do comando “info()”

Neste caso, iremos selecionar as seguintes colunas abaixo:

Variável	Descrição
SQ_CANDIDATO	Número sequencial do candidato, gerado internamente pelos sistemas eleitorais para cada eleição. Observação: não é o número de campanha do candidato.
DS_TIPO_BEM_CANDIDATO	Descrição do tipo do bem do candidato.
DS_BEM_CANDIDATO	Descrição detalhada do bem do candidato.
VR_BEM_CANDIDATO	Valor declarado em reais do bem do candidato.

Para selecionar a coluna citada utilizamos o seguinte comando abaixo, com isso podemos verificar que o dataframe continua com 1.013.809 entradas e 4 colunas:

```
df_bem = df_bem[['SQ_CANDIDATO','DS_TIPO_BEM_CANDIDATO','DS_BEM_CANDIDATO','VR_BEM_CANDIDATO']]
```

```
df_bem.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1013809 entries, 0 to 1013808
Data columns (total 4 columns):
 #   Column                Non-Null Count    Dtype
---  ------                --------------    -----
 0   SQ_CANDIDATO          1013809 non-null  int64
 1   DS_TIPO_BEM_CANDIDATO 1013809 non-null  object
 2   DS_BEM_CANDIDATO      1013809 non-null  object
 3   VR_BEM_CANDIDATO      1013809 non-null  float64
dtypes: float64(1), int64(1), object(2)
memory usage: 30.9+ MB
```

Esse dataframe não possui dados nulos.

```
df_bem.isnull().sum()
```

```
SQ_CANDIDATO          0
DS_TIPO_BEM_CANDIDATO 0
DS_BEM_CANDIDATO      0
VR_BEM_CANDIDATO      0
dtype: int64
```

O terceiro *dataset*, "despesas_contratadas_candidatos_2020_BRASIL.csv", será lido como das outras vezes por meio do seguinte comando:

```
df_despesas_contratadas = pd.read_csv("despesas_contratadas_candidatos_2020_BRASIL.csv",
                                      encoding = "Latin 1", sep = ";", decimal = ",")
```

```
df_despesas_contratadas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3868709 entries, 0 to 3868708
Data columns (total 53 columns):
 #   Column                       Dtype
---  ------                       -----
 0   DT_GERACAO                   object
 1   HH_GERACAO                   object
 2   ANO_ELEICAO                  int64
 3   CD_TIPO_ELEICAO              int64
 4   NM_TIPO_ELEICAO              object
 5   CD_ELEICAO                   int64
 6   DS_ELEICAO                   object
 7   DT_ELEICAO                   object
 8   ST_TURNO                     int64
 9   TP_PRESTACAO_CONTAS          object
 10  DT_PRESTACAO_CONTAS          object
 11  SQ_PRESTADOR_CONTAS          int64
 12  SG_UF                        object
 13  SG_UE                        int64
 14  NM_UE                        object
 15  NR_CNPJ_PRESTADOR_CONTA      int64
 16  CD_CARGO                     int64
 17  DS_CARGO                     object
 18  SQ_CANDIDATO                 int64
 19  NR_CANDIDATO                 int64
 20  NM_CANDIDATO                 object
 21  NR_CPF_CANDIDATO             int64
 22  NR_CPF_VICE_CANDIDATO        int64
 23  NR_PARTIDO                   int64
 24  SG_PARTIDO                   object
 25  NM_PARTIDO                   object
 26  CD_TIPO_FORNECEDOR           int64
 27  DS_TIPO_FORNECEDOR           object
 28  CD_CNAE_FORNECEDOR           int64
 29  DS_CNAE_FORNECEDOR           object
 30  NR_CPF_CNPJ_FORNECEDOR       int64
 31  NM_FORNECEDOR                object
 32  NM_FORNECEDOR_RFB            object
 33  CD_ESFERA_PART_FORNECEDOR    int64
 34  DS_ESFERA_PART_FORNECEDOR    object
 35  SG_UF_FORNECEDOR             object
 36  CD_MUNICIPIO_FORNECEDOR      int64
 37  NM_MUNICIPIO_FORNECEDOR      object
 38  SQ_CANDIDATO_FORNECEDOR      int64
 39  NR_CANDIDATO_FORNECEDOR      int64
 40  CD_CARGO_FORNECEDOR          int64
 41  DS_CARGO_FORNECEDOR          object
 42  NR_PARTIDO_FORNECEDOR        int64
 43  SG_PARTIDO_FORNECEDOR        object
 44  NM_PARTIDO_FORNECEDOR        object
 45  DS_TIPO_DOCUMENTO            object
 46  NR_DOCUMENTO                 object
 47  CD_ORIGEM_DESPESA            int64
 48  DS_ORIGEM_DESPESA            object
 49  SQ_DESPESA                   int64
 50  DT_DESPESA                   object
 51  DS_DESPESA                   object
 52  VR_DESPESA_CONTRATADA        float64
dtypes: float64(1), int64(24), object(28)
memory usage: 1.5+ GB
```

O dataframe criadas despesas eleitorais contratadas possuem 3.868.709 entradas e 53 colunas.

Para permitir análises e avaliações posteriores, serão selecionadas as seguintes colunas para esse dataframe:

Variável	Descrição
SQ_CANDIDATO	Descrição do tipo do bem do candidato.
SQ_DESPESA	Sequencial de identificação do registro da despesa declarada pelo prestador de contas.
DS_DESPESA	Descrição do gasto no elenco de aplicações informada pelo prestador de contas em relação à despesa.
VR_DESPESA_CONTRATADA	Descrição detalhada do bem do candidato.

```
df_despesas_contratadas = df_despesas_contratadas[['SQ_DESPESA','SQ_CANDIDATO','DS_DESPESA','VR_DESPESA_CONTRATADA']]
```

```
df_despesas_contratadas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3868709 entries, 0 to 3868708
Data columns (total 4 columns):
 #   Column                Dtype
---  ------                -----
 0   SQ_DESPESA            int64
 1   SQ_CANDIDATO          int64
 2   DS_DESPESA            object
 3   VR_DESPESA_CONTRATADA  float64
dtypes: float64(1), int64(2), object(1)
memory usage: 118.1+ MB
```

Esse *dataframe* também não apresenta valores nulos, como pode ser verificado a seguir:

```
df_despesas_contratadas.isnull().sum()
```

```
SQ_DESPESA            0
SQ_CANDIDATO          0
DS_DESPESA            0
VR_DESPESA_CONTRATADA    0
dtype: int64
```

A leitura e obtenção dos detalhes do quarto e último *dataset*, "despesas_pagas_candidatos_2020_BRASIL.csv", seguiu os mesmos passos anteriores apresentando 3.054.879 entradas em 28 colunas. Como esperado, o número de entradas é menor do que o *dataframe* anterior, pois nem todas as despesas contratadas precisam, necessariamente, ser pagas.

```
df_despesas_paga = pd.read_csv("despesas_pagas_candidatos_2020_BRASIL.csv",
                               encoding = "Latin 1", sep = ";", decimal = ",")
```

```
df_despesas_paga.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3054880 entries, 0 to 3054879
Data columns (total 28 columns):
 #   Column                    Dtype
---  ------                    -----
 0   DT_GERACAO                object
 1   HH_GERACAO                object
 2   ANO_ELEICAO               int64
 3   CD_TIPO_ELEICAO           int64
 4   NM_TIPO_ELEICAO           object
 5   CD_ELEICAO                int64
 6   DS_ELEICAO                object
 7   DT_ELEICAO                object
 8   ST_TURNO                  int64
 9   TP_PRESTACAO_CONTAS       object
 10  DT_PRESTACAO_CONTAS       object
 11  SQ_PRESTADOR_CONTAS       int64
 12  SG_UF                     object
 13  DS_TIPO_DOCUMENTO         object
 14  NR_DOCUMENTO              object
 15  CD_FONTE_DESPESA          int64
 16  DS_FONTE_DESPESA          object
 17  CD_ORIGEM_DESPESA         int64
 18  DS_ORIGEM_DESPESA         object
 19  CD_NATUREZA_DESPESA       int64
 20  DS_NATUREZA_DESPESA       object
 21  CD_ESPECIE_RECURSO        int64
 22  DS_ESPECIE_RECURSO        object
 23  SQ_DESPESA                int64
 24  SQ_PARCELAMENTO_DESPESA   int64
 25  DT_PAGTO_DESPESA          object
 26  DS_DESPESA                object
 27  VR_PAGTO_DESPESA          float64
dtypes: float64(1), int64(11), object(16)
memory usage: 652.6+ MB
```

Das informações apresentadas nesse dataset, apenas 2 se mostraram relevantes para o estudo e são as seguintes:

Variável Descrição	Variável Descrição
SQ_DESPESA	Sequencial de identificação do registro da despesa declarada pelo prestador de contas.
VR_PAGTO_DESPESA	VR_PAGTO_DESPESA

Dessa forma, nos demais *dataframes*, não foram identificados valores nulos:

```
df_despesas_paga = df_despesas_paga[['SQ_DESPESA','VR_PAGTO_DESPESA']]
```

```
df_despesas_paga.isnull().sum()
```

```
SQ_DESPESA          0
VR_PAGTO_DESPESA    0
dtype: int64
```

Na seção seguinte será feita a análise e exploração desses dados que justificarão a escolha dos dados feitos até o momento e mostrará a relação entre eles.

4. Análise e Exploração dos Dados

Para a realizar a análise e exploração de dados será utilizada a função “Counter” do módulo “Collections”, que conta quantas vezes uma determinada opção aparece em uma série de dados a armazena esses valores em um dicionário, a biblioteca “Matplotlib” para a criação de gráficos e, novamente, a biblioteca “pandas”.

O primeiro passo para a análise é a importação das bibliotecas adicionais que serão utilizadas.

```
from collections import Counter
import matplotlib.pyplot as plt
```

```
df_consulta['DS_SIT_TOT_TURNO'].unique()
```

```
array(['SUPLENTE', 'NÃO ELEITO', 'ELEITO POR QP', 'ELEITO',
       'ELEITO POR MÉDIA', '#NULO#', '2º TURNO'], dtype=object)
```

```
df_consulta_eleitos = df_consulta[df_consulta.DS_SIT_TOT_TURNO.isin(['ELEITO POR MÉDIA','ELEITO POR QP'])]
```

```
df_consulta_eleitos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 57999 entries, 6 to 557982
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   SQ_CANDIDATO            57999 non-null  int64
 1   NR_IDADE_DATA_POSSE     57999 non-null  float64
 2   DS_GENERO               57999 non-null  object
 3   DS_GRAU_INSTRUCAO       57999 non-null  object
 4   DS_ESTADO_CIVIL         57999 non-null  object
 5   DS_COR_RACA             57999 non-null  object
 6   DS_SIT_TOT_TURNO        57999 non-null  object
 7   VR_DESPESA_MAX_CAMPANHA  57999 non-null  object
dtypes: float64(1), int64(1), object(6)
memory usage: 4.0+ MB
```

O *dataframe* original, tinha 532.281 entradas, esse possui apenas 57.999, que são os candidatos que efetivamente foram eleitos.

A avaliação desses *dataframes* será iniciada pela análise do gênero dos candidatos. Primeiramente, serão contados quantos candidatos de cada gênero concorreram e quantos foram eleitos na eleição analisada.

```
genero_candidatos = Counter(df_consulta['DS_GENERO'])
```

```
genero_candidatos
```

```
Counter({'MASCULINO': 354282, 'FEMININO': 177998, 'NÃO DIVULGÁVEL': 1})
```

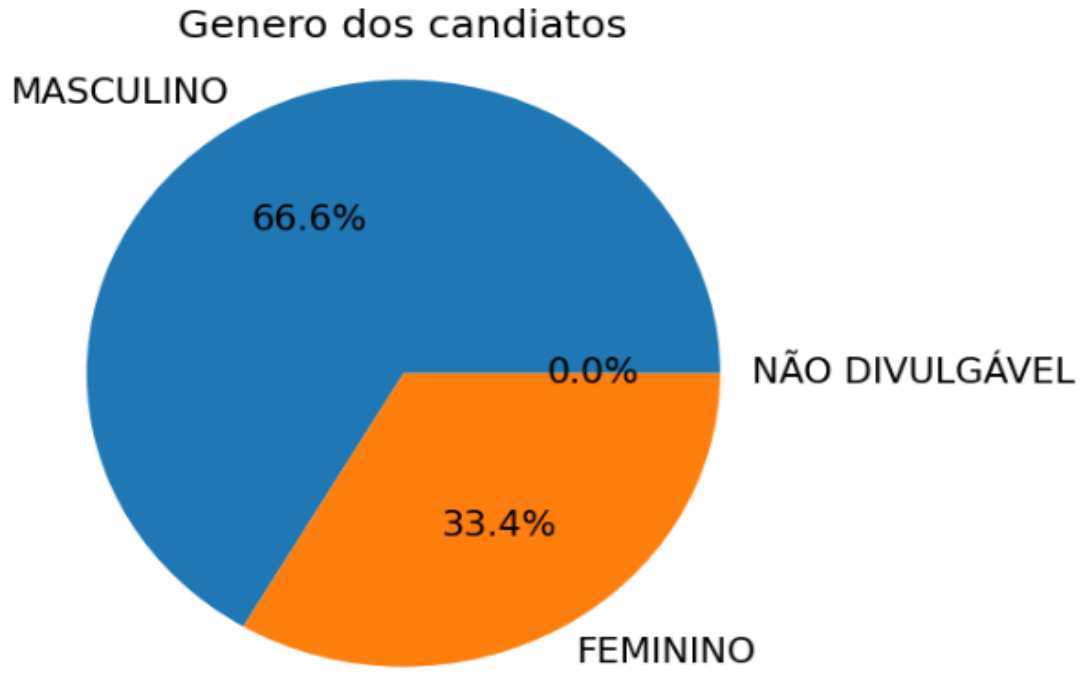
```
genero_candidato_eleitos = Counter(df_consulta_eleitos['DS_GENERO'])
```

```
genero_candidato_eleitos
```

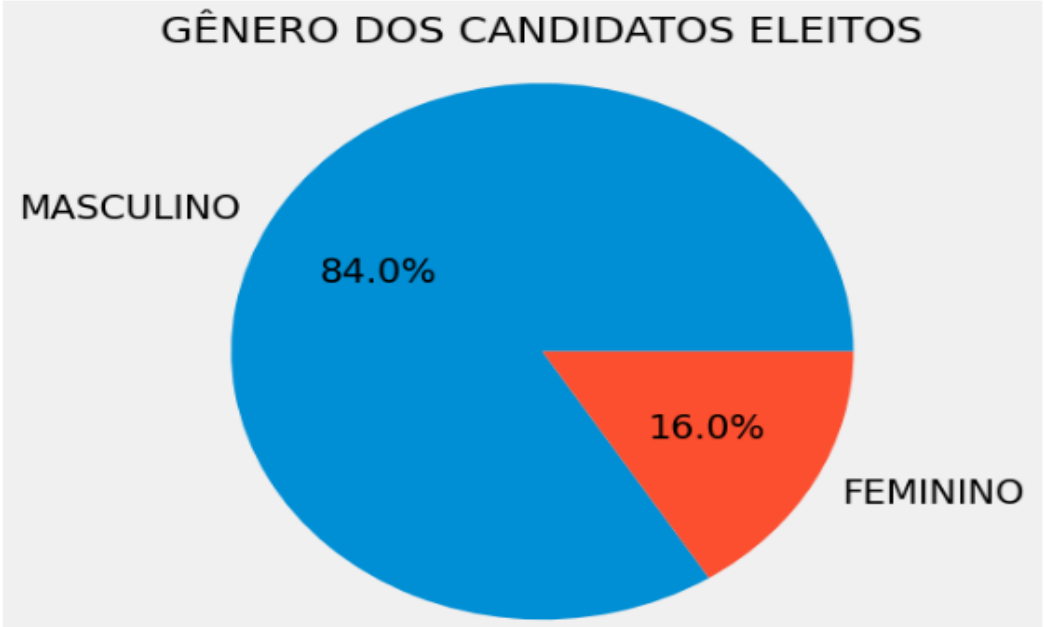
```
Counter({'MASCULINO': 48716, 'FEMININO': 9283})
```

Apesar de já ser possível analisar as informações, com o objetivo de facilitar a visualização, serão criados gráficos de pizza para cada dicionário criado. Para diferenciar os gráficos serão utilizados estilos diferentes: o “seaborn-pastel” para todos os candidatos e o “ggplot” para os candidatos eleitos. Para gerar os gráficos com seus títulos e porcentagens foram utilizados os comandos a seguir:

```
plt.style.use('default')
plt.pie(genero_candidatos.values(), labels = genero_candidatos.keys(),
        autopct = '%1.1f%%', textprops={'fontsize':'16'})
plt.axis("image")
plt.title("Genero dos candiatos",fontsize = 18)
plt.tight_layout()

plt.show()
```




```
plt.style.use('fivethirtyeight')
plt.pie(genero_candidato_eleitos.values(), labels = genero_candidato_eleitos.keys(),
        autopct = '%1.1f%%', textprops={'fontsize':'16'})
plt.axis("image")
plt.title("GÊNERO DOS CANDIDATOS ELEITOS",fontsize = 18)
plt.tight_layout()

plt.show()
```



Ao analisar os gráficos podemos perceber que o número de candidatos de cada gênero já não é proporcional ao eleitorado. O eleitorado masculino sendo de 66,6%, esse aumento proporcional se torna mais expressivo quando se analisa os candidatos eleitos, pois esse percentual salta para 84,0%, enquanto os candidatos eleitos do gênero feminino representam apenas 16%, sendo que elas representam 33,4% do eleitorado.

O próximo item a ser analisado é a idade que os candidatos teriam no momento da posse. Para essa análise será utilizada a função “describe” que apresenta os principais indicadores estatísticos de uma série de dados. Na visualização da distribuição dessas idades serão utilizados histogramas, seguindo os mesmos estilos descritos anteriormente.

```
df_consulta['NR_IDADE_DATA_POSSE'].describe()
```

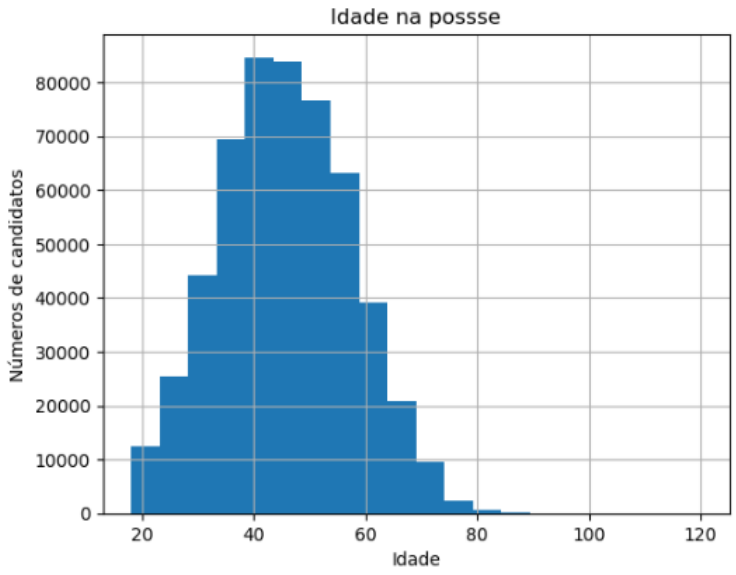
```
count    532279.000000
mean         45.549255
std          11.518701
min          18.000000
25%          37.000000
50%          45.000000
75%          54.000000
max         120.000000
Name: NR_IDADE_DATA_POSSE, dtype: float64
```

```
df_consulta_eleitos['NR_IDADE_DATA_POSSE'].describe()
```

```
count    57999.000000
mean        44.602114
std         10.486738
min         18.000000
25%         37.000000
50%         44.000000
75%         52.000000
max        116.000000
Name: NR_IDADE_DATA_POSSE, dtype: float64
```

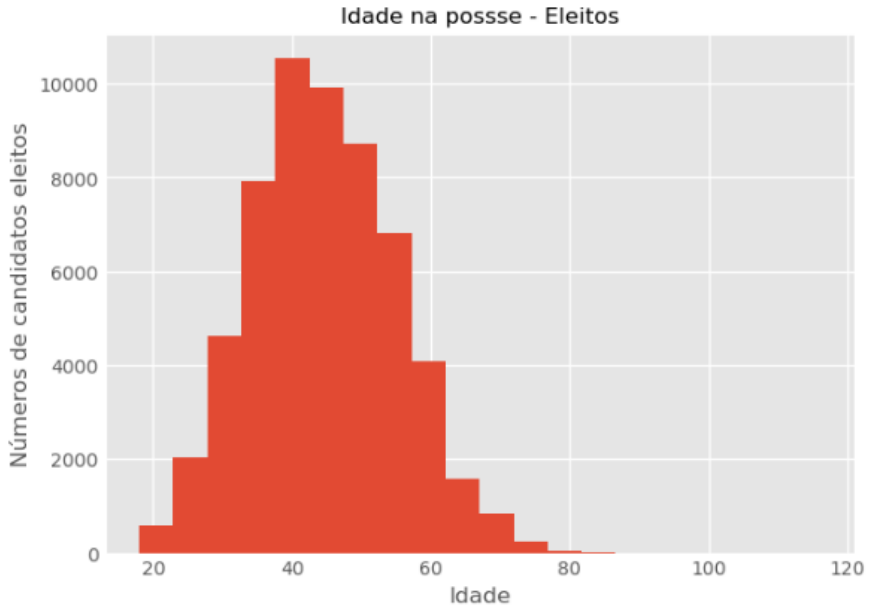
```
df_consulta.NR_IDADE_DATA_POSSE.hist(bins=20)
plt.style.use('seaborn-pastel')
plt.xlabel('Idade')
plt.ylabel('Números de candidatos')
plt.title("Idade na possse")
```

```
Text(0.5, 1.0, 'Idade na possse')
```



```
df_consulta_eleitos.NR_IDADE_DATA_POSSE.hist(bins=20)
plt.style.use('default')
plt.xlabel('Idade')
plt.ylabel('Números de candidatos eleitos')
plt.title("Idade na possse - Eleitos")
```

```
Text(0.5, 1.0, 'Idade na possse - Eleitos')
```



Após analisar os histogramas, os candidatos eleitos não terem um pico tão acentuado, não se verifica nenhuma mudança significativa. Essa constatação pode ser confirmada por meio dos indicadores estatísticos das duas séries de dados que são muito similares, com variações mínimas na média de idade, por exemplo: 39,48 anos para todos os candidatos e 38,43 anos para os eleitos. Percebe-se também que a distribuição de idade é muito similar à do eleitorado.

A escolaridade dos candidatos será o próximo critério ser analisado, optamos pelo gráfico de barras horizontais para a visualização desses dados. Para a construção desse tipo de gráfico foi necessário construir listas com os rótulos e os valores dos dados, criando a necessidade de alguns passos adicionais.

O primeiro passo é a contagem dos valores de cada ocorrência como feito anteriormente.

```
escolaridade_candidatos = Counter(df_consulta['DS_GRAU_INSTRUCAO'])
escolaridade_candidatos
```

```
Counter({'ENSINO MÉDIO COMPLETO': 203236,
         'SUPERIOR COMPLETO': 130975,
         'SUPERIOR INCOMPLETO': 23907,
         'ENSINO FUNDAMENTAL INCOMPLETO': 67298,
         'ENSINO FUNDAMENTAL COMPLETO': 64561,
         'LÊ E ESCREVE': 16064,
         'ENSINO MÉDIO INCOMPLETO': 26237,
         'NÃO DIVULGÁVEL': 1,
         'ANALFABETO': 2})
```

Buscando trazer mais dados para a análise, foi criado um laço de repetição “for” para apresentação dos percentuais de cada um dos níveis de instrução:

```
n_cadidatos=sum(escolaridade_candidatos.values())
for x, y in escolaridade_candidatos.items():
    a = y/n_cadidatos*100
    print(str(x) + ': ' +'\n'+ str(round(a,2)) + '%' + '\n')
```

```
ENSINO MÉDIO COMPLETO:
38.18%

SUPERIOR COMPLETO:
24.61%

SUPERIOR INCOMPLETO:
4.49%

ENSINO FUNDAMENTAL INCOMPLETO:
12.64%

ENSINO FUNDAMENTAL COMPLETO:
12.13%

LÊ E ESCREVE:
3.02%

ENSINO MÉDIO INCOMPLETO:
4.93%

NÃO DIVULGÁVEL:
0.0%

ANALFABETO:
0.0%
```

A construção do gráfico de barras horizontais, foram criadas a lista com os rótulos e os valores por meio do laço de repetição a seguir:

```
lista_escolaridade_candidatos_lables = []
lista_escolaridade_candidatos_valores = []
for x, y in escolaridade_candidatos.items():
    lista_escolaridade_candidatos_lables.append(x)
    lista_escolaridade_candidatos_valores.append(y)
```

```
lista_escolaridade_candidatos_lables
```

```
['ENSINO MÉDIO COMPLETO',
 'SUPERIOR COMPLETO',
 'SUPERIOR INCOMPLETO',
 'ENSINO FUNDAMENTAL INCOMPLETO',
 'ENSINO FUNDAMENTAL COMPLETO',
 'LÊ E ESCREVE',
 'ENSINO MÉDIO INCOMPLETO',
 'NÃO DIVULGÁVEL',
 'ANALFABETO']
```

```
lista_escolaridade_candidatos_valores
```

```
[203236, 130975, 23907, 67298, 64561, 16064, 26237, 1, 2]
```

A seguir, para a construção do gráfico, utilizou-se os seguintes comandos:

```
plt.style.use('seaborn-pastel')
plt.barh(lista_escolaridade_candidatos_lables, lista_escolaridade_candidatos_valores)
plt.ylabel('escolaridade')
plt.xlabel('número de candidatos')
plt.title('Candidatos por escolaridade')
plt.show()
```

Para os candidatos eleitos, a mesma lógica foi utilizada, alterando apenas a fonte de dados que nesse caso foi o *dataframe* "df_consulta_eleitos":

```
escolaridade_candidatos_eleitos = Counter(df_consulta_eleitos['DS_GRAU_INSTRUCAO'])
escolaridade_candidatos_eleitos
```

```
Counter({'ENSINO MÉDIO COMPLETO': 21680,
         'SUPERIOR INCOMPLETO': 2225,
         'SUPERIOR COMPLETO': 17725,
         'ENSINO FUNDAMENTAL INCOMPLETO': 6627,
         'ENSINO FUNDAMENTAL COMPLETO': 6446,
         'LÊ E ESCREVE': 1079,
         'ENSINO MÉDIO INCOMPLETO': 2217})
```

```
n_eleitos=sum(escolaridade_candidatos_eleitos.values())
for x, y in escolaridade_candidatos_eleitos.items():
    a = y/n_eleitos*100
    print(str(x) + ': ' +'\n'+ str(round(a,2)) + '%' + '\n')
```

```
ENSINO MÉDIO COMPLETO:
37.38%

SUPERIOR INCOMPLETO:
3.84%

SUPERIOR COMPLETO:
30.56%

ENSINO FUNDAMENTAL INCOMPLETO:
11.43%

ENSINO FUNDAMENTAL COMPLETO:
11.11%

LÊ E ESCREVE:
1.86%

ENSINO MÉDIO INCOMPLETO:
3.82%
```

```
lista_escolaridade_candidatos_eleitos_lables = []
lista_escolaridade_candidatos_eleitos_valores = []
for x, y in escolaridade_candidatos_eleitos.items():
    lista_escolaridade_candidatos_eleitos_lables.append(x)
    lista_escolaridade_candidatos_eleitos_valores.append(y)
```

```
lista_escolaridade_candidatos_eleitos_lables
```

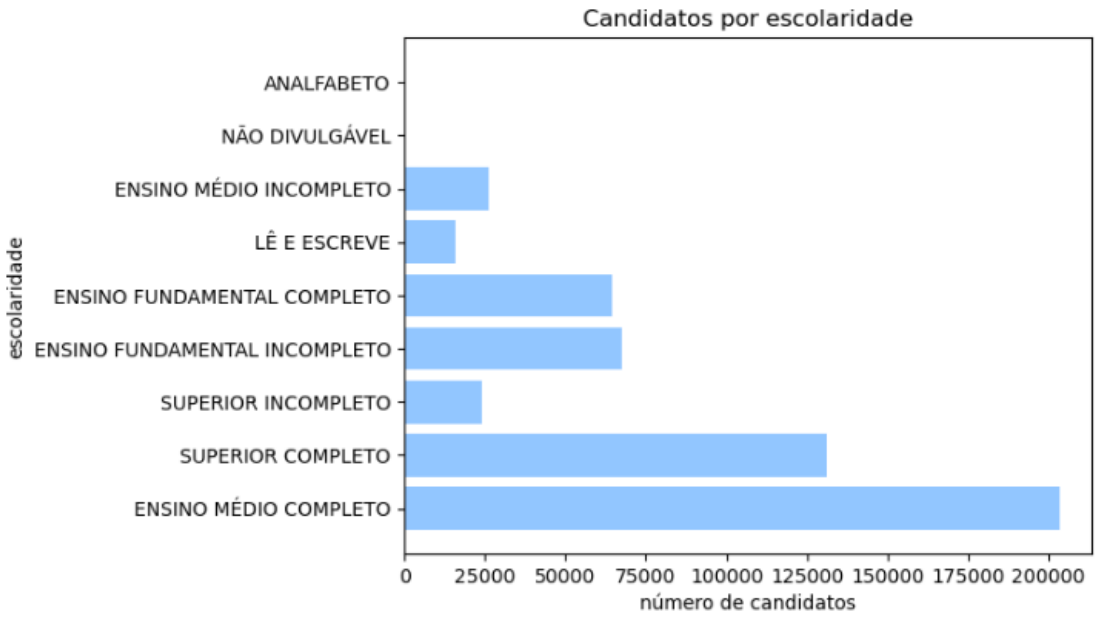
```
['ENSINO MÉDIO COMPLETO',
 'SUPERIOR INCOMPLETO',
 'SUPERIOR COMPLETO',
 'ENSINO FUNDAMENTAL INCOMPLETO',
 'ENSINO FUNDAMENTAL COMPLETO',
 'LÊ E ESCREVE',
 'ENSINO MÉDIO INCOMPLETO']
```

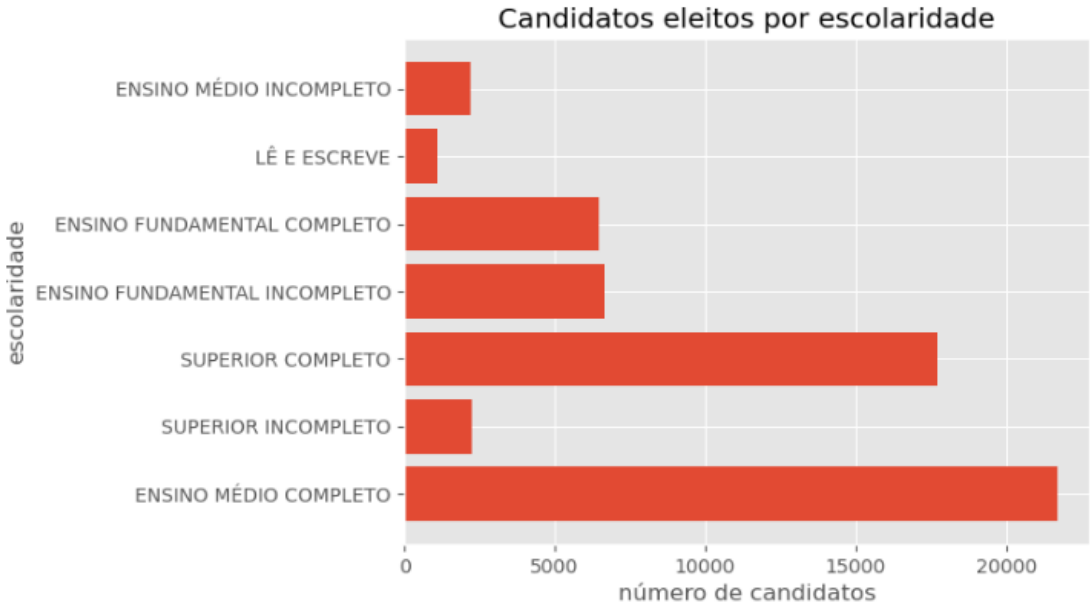
```
lista_escolaridade_candidatos_eleitos_valores
```

```
[21680, 2225, 17725, 6627, 6446, 1079, 2217]
```

```
plt.style.use('ggplot')
plt.barh(lista_escolaridade_candidatos_eleitos_lables, lista_escolaridade_candidatos_eleitos_valores)
plt.ylabel('escolaridade')
plt.xlabel('número de candidatos')
plt.title('Candidatos eleitos por escolaridade')
plt.show()
```

Esses comandos geraram os seguintes gráficos:





No gráfico candidatos eleitos por escolaridade, podemos perceber que a porcentagem do ensino médio completo é maior que o ensino superior. O importante destacar que esse percentual elevado de eleitos com ensino médio não é um problema, já que a educação é algo fundamental para todos os cidadãos. O preocupante é o baixo percentual do eleitorado que possui ensino superior completo.

Nesse próximo ponto, será analisado a raça dos candidatos. Para essa análise serão utilizadas as mesmas técnicas descritas anteriormente, apenas com a diferença na apresentação dos dados que será por meio do gráfico de barras. Assim o primeiro passo é realizar a contagem dos valores da série desejada.

```
raca_candidatos = Counter(df_consulta['DS_COR_RACA'])
raca_candidatos
```

```
Counter({'PRETA': 55648,
         'BRANCA': 256502,
         'PARDA': 210113,
         'NÃO INFORMADO': 6070,
         'INDÍGENA': 2082,
         'AMARELA': 1865,
         'NÃO DIVULGÁVEL': 1})
```

Em seguida, obtém-se os percentuais de cada ocorrência:

```
n_raca = sum(raca_candidatos.values())
for x, y in raca_candidatos.items():
    a = y/n_raca*100
    print(str(x) + ': ' +'\n'+ str(round(a,2)) + '%' + '\n')
```

```
PRETA:
10.45%

BRANCA:
48.19%

PARDA:
39.47%

NÃO INFORMADO:
1.14%

INDÍGENA:
0.39%

AMARELA:
0.35%

NÃO DIVULGÁVEL:
0.0%
```

Com o dicionário “raca_candidatos” é possível construir o gráfico de barras sem a necessidade de se criar listas por meio do seguinte código:

```
plt.style.use('seaborn-pastel')
plt.bar(raca_candidatos.keys(), raca_candidatos.values())
plt.ylabel('Número de candidatos')
plt.xlabel('Raça')
plt.title('Candidatos por raça')
plt.show()
```

Todo o processo é repetido para os candidatos eleitos:

```
raca_candidatos_eleitos = Counter(df_consulta_eleitos['DS_COR_RACA'])
raca_candidatos_eleitos
```

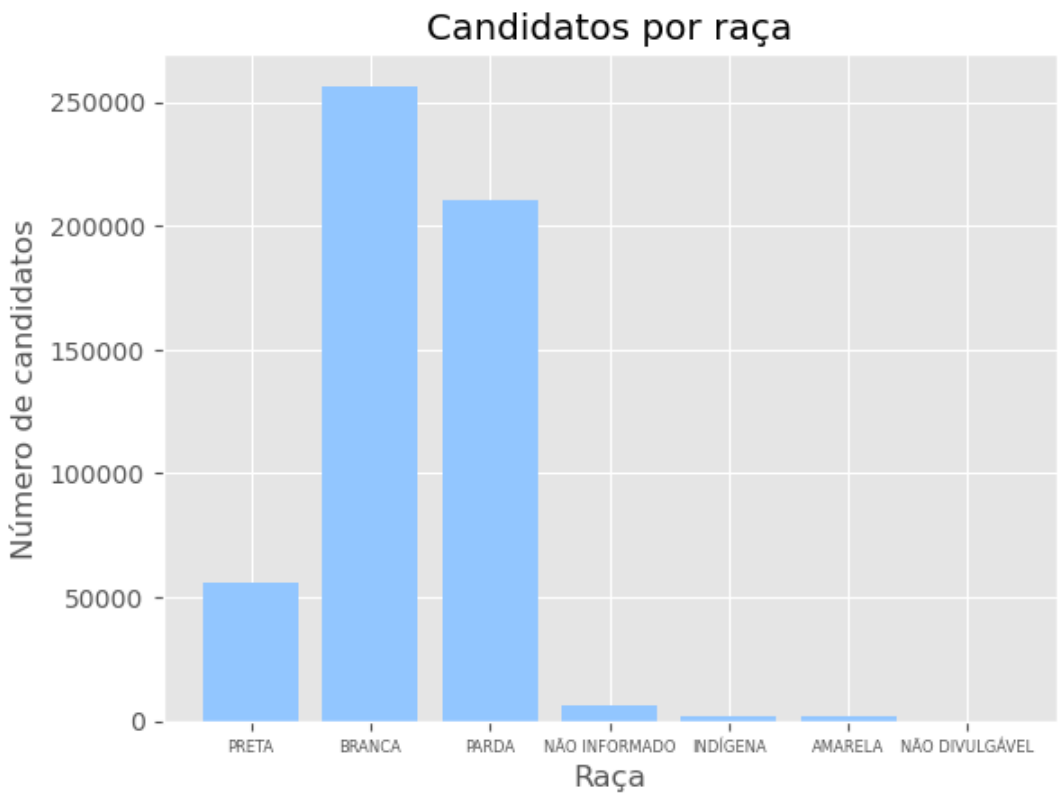
```
Counter({'BRANCA': 31088,
         'PRETA': 3588,
         'PARDA': 22336,
         'NÃO INFORMADO': 578,
         'AMARELA': 232,
         'INDÍGENA': 177})
```

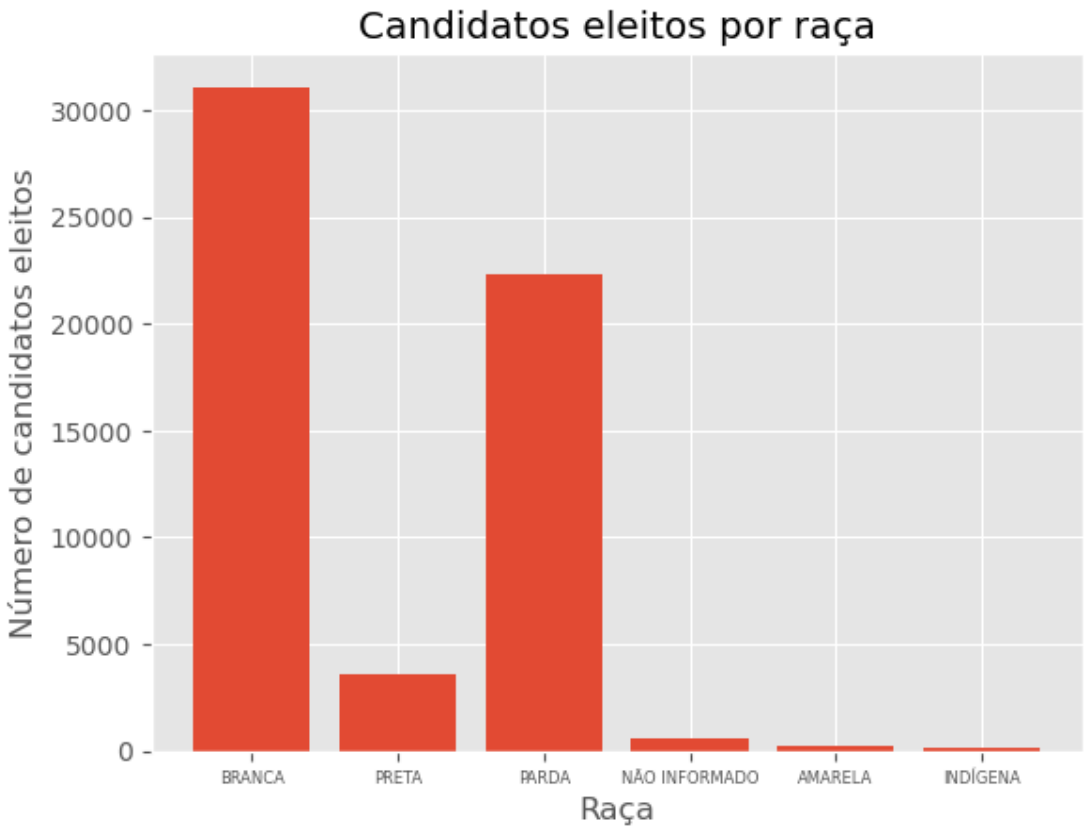
```
n_raca_eleitos = sum(raca_candidatos_eleitos.values())
for x, y in raca_candidatos_eleitos.items():
    a = y/n_raca_eleitos*100
    print(str(x) + ': ' +'\n'+ str(round(a,2)) + '%' + '\n')
```

```
BRANCA:
53.6%

PRETA:
6.19%

PARDA:
38.51%

NÃO INFORMADO:
1.0%

AMARELA:
0.4%

INDÍGENA:
0.31%
```

```
plt.style.use('ggplot')
plt.bar(raca_candidatos_eleitos.keys(), raca_candidatos_eleitos.values())
plt.ylabel('Número de candidatos eleitos')
plt.xlabel('Raça')
plt.title('Candidatos eleitos por raça')
plt.show()
```

Após esses passos, são gerados os seguintes gráficos





O próximo ponto, podemos perceber uma semelhança com a análise de gênero, em que há diferenças significativas entre a população, ou o eleitorado, e o resultado das eleições. De acordo com dados da Pesquisa Nacional por Amostra de Domicílios (PNAD) 2019, 42,7% da população se declara branca e esse percentual sobe para 52,81% para os candidatos e 72,61% quando se fala de candidatos eleitos.

O próximo dataframe a ser analisado é referente ao estado civil dos candidatos. Nesse sentido, os passos serão os mesmos utilizados para a análise da escolaridade dos candidatos, ou seja, serão utilizados comandos para obtenção do percentual dos valores de cada categoria, transformação dos dados do dicionário em listas e, por fim, apresentação dos resultados em gráficos de barras horizontais.

```
estado_civil_candidatos = Counter(df_consulta['DS_ESTADO_CIVIL'])
estado_civil_candidatos
```

```
Counter({'CASADO(A)': 274387,
         'DIVORCIADO(A)': 44636,
         'SEPARADO(A) JUDICIALMENTE': 5910,
         'SOLTEIRO(A)': 197248,
         'VIÚVO(A)': 10099,
         'NÃO DIVULGÁVEL': 1})
```

```
n_estado_civil=sum(estado_civil_candidatos.values())
for x, y in estado_civil_candidatos.items():
    a = y/n_estado_civil*100
    print(str(x) + ':' + '\n' + str (round(a,2)) + '%'+'\n')
```

```
CASADO(A):
51.55%

DIVORCIADO(A):
8.39%

SEPARADO(A) JUDICIALMENTE:
1.11%

SOLTEIRO(A):
37.06%

VIÚVO(A):
1.9%

NÃO DIVULGÁVEL:
0.0%
```

```
lista_estado_civil_candidatos_labels = []
lista_estado_civil_candidatos_valores = []
for x, y in estado_civil_candidatos.items():
    lista_estado_civil_candidatos_labels.append(x)
    lista_estado_civil_candidatos_valores.append(y)
lista_estado_civil_candidatos_labels
```

```
['CASADO(A)',
 'DIVORCIADO(A)',
 'SEPARADO(A) JUDICIALMENTE',
 'SOLTEIRO(A)',
 'VIÚVO(A)',
 'NÃO DIVULGÁVEL']
```

```
lista_estado_civil_candidatos_valores
```

```
[274387, 44636, 5910, 197248, 10099, 1]
```

```
plt.style.use('seaborn-pastel')
plt.barh(lista_estado_civil_candidatos_labels, lista_estado_civil_candidatos_valores),
plt.ylabel('Estado Civil')
plt.xlabel('Número de canditados')
plt.title('CANDIDATOS POR ESTADO CIVIL')
plt.show()
```

```
estado_civil_candidatos_eleitos = Counter(df_consulta_eleitos['DS_ESTADO_CIVIL'])
estado_civil_candidatos_eleitos
```

```
Counter({'CASADO(A)': 35230,
         'SOLTEIRO(A)': 18081,
         'DIVORCIADO(A)': 3630,
         'SEPARADO(A) JUDICIALMENTE': 482,
         'VIÚVO(A)': 576})
```

```
n_estado_civil_eleitos = sum(estado_civil_candidatos_eleitos.values())
for x, y in estado_civil_candidatos_eleitos.items():
    a = y/n_estado_civil_eleitos*100
    print(str(x) + ':' + '\n' + str (round(a,2)) + '%'+'\n')
```

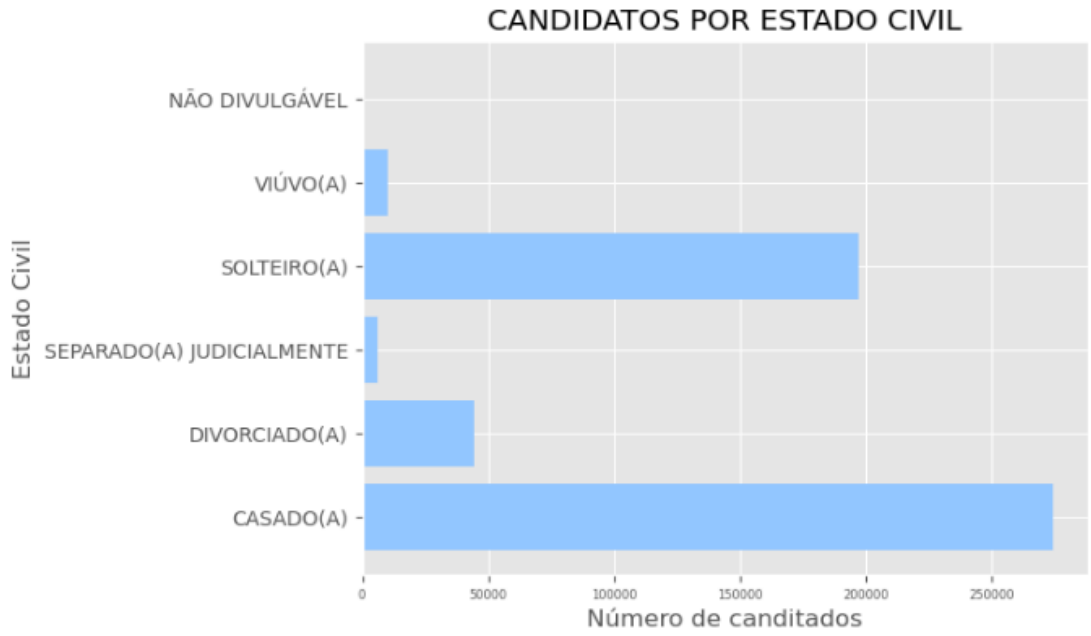
```
CASADO(A):
60.74%

SOLTEIRO(A):
31.17%

DIVORCIADO(A):
6.26%

SEPARADO(A) JUDICIALMENTE:
0.83%

VIÚVO(A):
0.99%
```

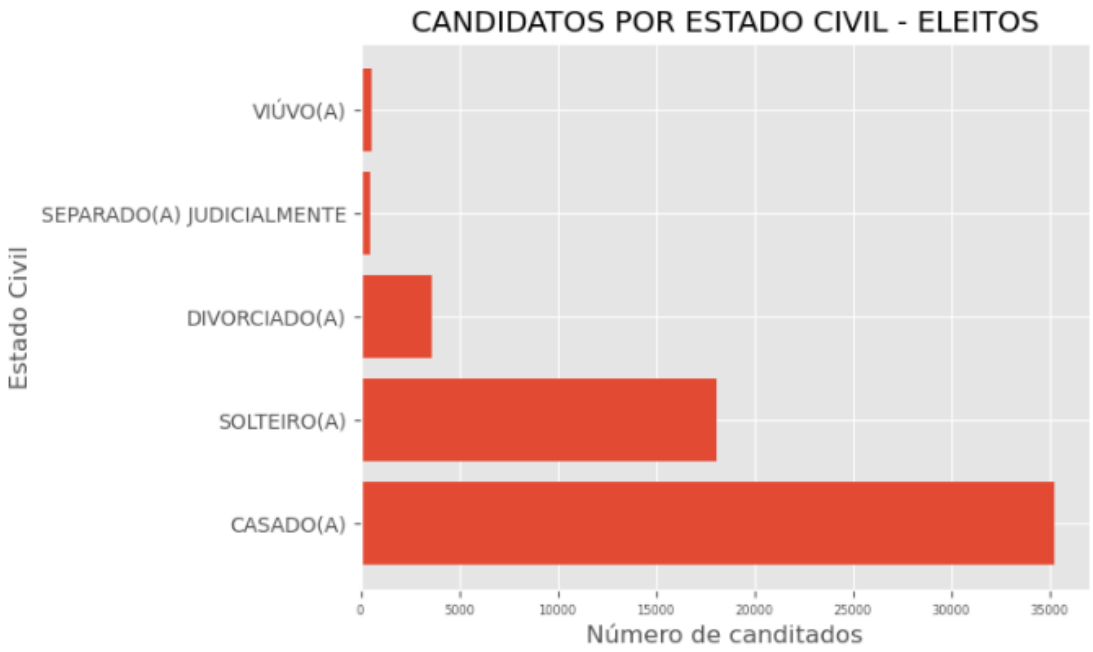
```
lista_estado_civil_candidatos_eleitos_labels = []
lista_estado_civil_candidatos_eleitos_valores = []
for x, y in estado_civil_candidatos_eleitos.items():
    lista_estado_civil_candidatos_eleitos_labels.append(x)
    lista_estado_civil_candidatos_eleitos_valores.append(y)
lista_estado_civil_candidatos_eleitos_labels
```

```
['CASADO(A)',
 'SOLTEIRO(A)',
 'DIVORCIADO(A)',
 'SEPARADO(A) JUDICIALMENTE',
 'VIÚVO(A)']
```

```
lista_estado_civil_candidatos_eleitos_valores
```

```
[35230, 18081, 3630, 482, 576]
```





Percebesse que por meio dos gráficos a maioria dos candidatos é casado, sendo que esse número eleva quando avaliamos os eleitos. Ao analisar essa informação comparada com a média de idade é esperado que o número de candidatos casados seja a que se destaca, mas é interessante perceber que, mesmo a média de idade se mantendo entre os candidatos e os eleitos.

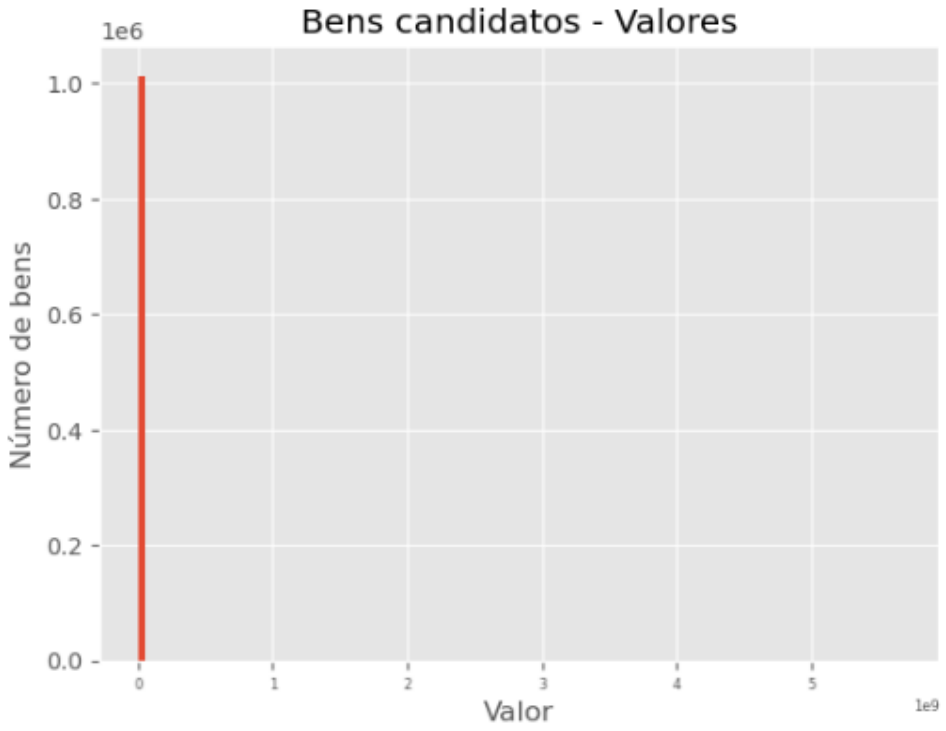
O *dataframe* analisado é o que diz respeito aos bens declarados pelos candidatos. A primeira verificação será na análise estatística dos valores dos bens, o que será feito por meio da função “describe” e, para melhorar a visualização, os valores serão transformados para números inteiros usando a função “astype”.

```
df_bem['VR_BEM_CANDIDATO'].describe().astype('int')
```

```
count       1013809
mean         115434
std         8183459
min            -113
25%            7000
50%           25000
75%           80000
max     -2147483648
Name: VR_BEM_CANDIDATO, dtype: int32
```

Com base nos valores apresentados, percebe-se que os dados estão muito concentrados em valores mais baixos dos bens. Em contrapartida o valor máximo identificado é de R$5.000.000.000,00 o que parece claramente um equívoco no lançamento da informação. Visualmente, essa situação é verificada ao plotarmos as informações em um histograma, onde se visualiza apenas uma barra no gráfico.

```
df_bem.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel('Valor')
plt.ylabel('Número de bens')
plt.title('Bens candidatos - Valores')
plt.show()
```



Com o objetivo de verificar possíveis erros de cadastro que pudessem impactar significantemente a análise, optamos por verificar individualmente os bens que tivessem o seu valor superior a R$ 10.000.000,00, já que, apesar de possível, bens com valores superiores não são tão comuns. Inicialmente, serão identificadas quantas entradas apresentam valor superior ao estabelecido.

```
df_bem.loc[(df_bem['VR_BEM_CANDIDATO'] > 10000000)].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 278 entries, 13546 to 1009959
Data columns (total 4 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   SQ_CANDIDATO           278 non-null    int64
 1   DS_TIPO_BEM_CANDIDATO  278 non-null    object
 2   DS_BEM_CANDIDATO       278 non-null    object
 3   VR_BEM_CANDIDATO       278 non-null    float64
dtypes: float64(1), int64(1), object(2)
memory usage: 10.9+ KB
```

Para que seja possível avaliar cada um dos 278 registros identificados, será feito um laço de repetição para que seja apresentado o índice do registro, o número sequencial do candidato, o tipo de bem, sua descrição e seu valor.

```
for index, row in df_bem.loc[(df_bem['VR_BEM_CANDIDATO'] > 100000000)].iterrows():
    print(index)
    print('Candidato: ' + str(row['SQ_CANDIDATO']))
    print('Tipo de bem: ' + str(row['DS_TIPO_BEM_CANDIDATO']))
    print('Descrição: ' + str(row['DS_BEM_CANDIDATO']))
    print('Valor: ' + str(row['VR_BEM_CANDIDATO']) + '\n')
```

O resultado desse comando imprimirá na tela os 278 registros da seguinte forma:

```
85540
Candidato: 260000725370
Tipo de bem: Veículo automotor terrestre: caminhão, automóvel, moto, etc.
Descrição: Palio winqueno
Valor: 160000000.0
```

Como pode-se perceber nesse exemplo, há um claro equívoco no lançamento do valor, já que na descrição menciona que o valor do veículo é R$ 16.000,00 e não R$ 160.000.000,00. Infelizmente nem todos os registros apresentam uma descrição clara, sendo inclusive identificada como “#NULO#”, que é um valor vazio diretamente do *dataset*, como pode-se verificar no exemplo abaixo.

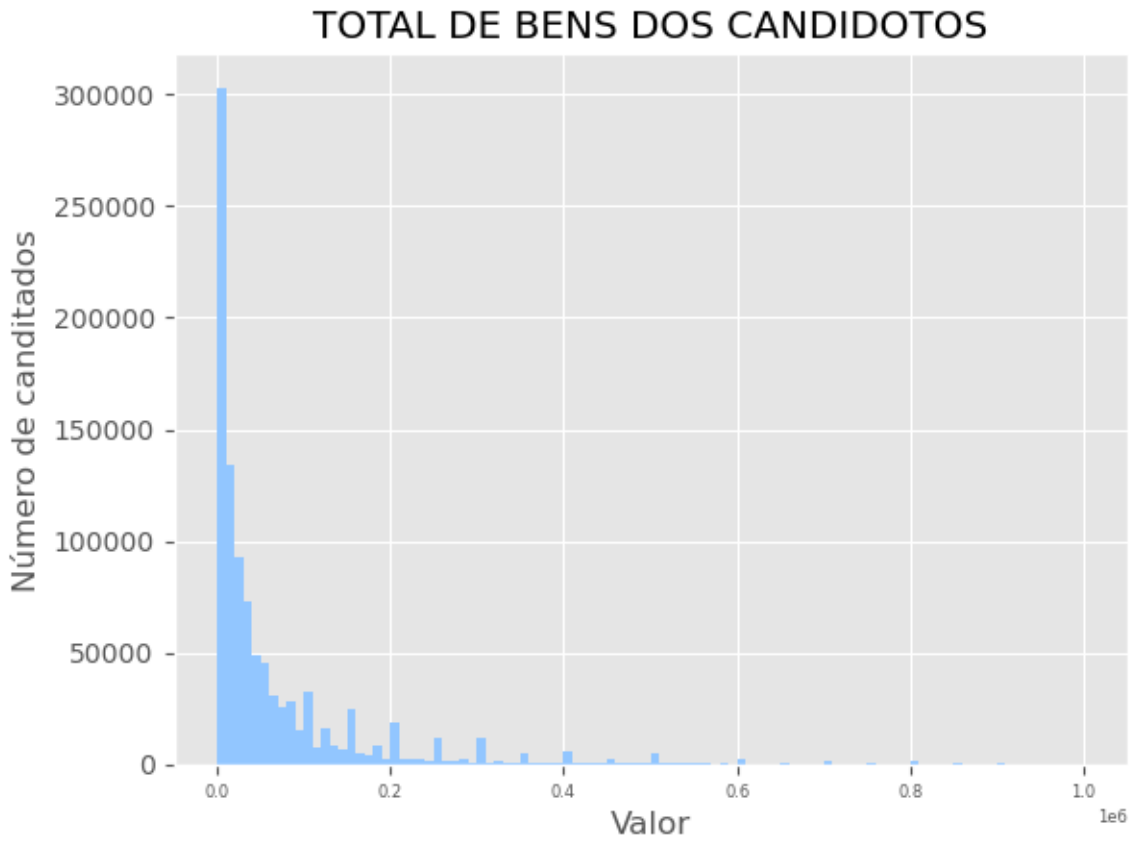
```
df_bem.loc[8425,'DS_BEM_CANDIDATO']
```

```
'#NULO#'
```

Com o objetivo de melhorar a visualização dessa informação, será criado um *dataframe* apenas com os valores inferiores a R$ 1.000.000,00, exclusivamente para plotagem em um histograma para se mostrar como a distribuição dos bens se comporta.

```
df_bem1 = df_bem.loc[(df_bem['VR_BEM_CANDIDATO'] < 1000000)]
```

```
df_bem1.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel('Valor')
plt.ylabel('Número de canditados')
plt.title('TOTAL DE BENS DOS CANDIDOTOS')
plt.show()
```



In [ ]:

Esse histograma confirma que a grande maioria dos candidatos não possui um grande valor em bens declarados. Contudo, esse *dataframe* ainda apresenta bens de todos os candidatos que concorreram na eleição, independentemente do cargo. Para que se possa avaliar apenas os candidatos aos cargos de prefeito e vereador, deve-se fazer a junção entre o *dataframe* "df_consulta", que contém os dados cadastrais dos candidatos a deputado, e o *dataframe* "df_bem". O primeiro passo para essa junção é definir no *dataframe* "df_consulta" o número sequencial do candidato como índice, já que esse é o valor comum entre os *dataframes*. Para realizar essa operação, será utilizada a função "set_index".

```
df_consulta = df_consulta.set_index('SQ_CANDIDATO')
```

```
df_consulta.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 532281 entries, 50001247384.0 to 50001133253.0
Data columns (total 7 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   NR_IDADE_DATA_POSSE      532279 non-null  float64
 1   DS_GENERO                532281 non-null  object
 2   DS_GRAU_INSTRUCAO        532281 non-null  object
 3   DS_ESTADO_CIVIL          532281 non-null  object
 4   DS_COR_RACA              532281 non-null  object
 5   DS_SIT_TOT_TURNO         532281 non-null  object
 6   VR_DESPESA_MAX_CAMPANHA  532281 non-null  object
dtypes: float64(1), object(6)
memory usage: 32.5+ MB
```

Percebe-se agora que a coluna “SQ_CANDIDATO” não aparece mais como uma das colunas, já que ela foi transformada em índice. Para se fazer a junção entre as duas tabelas será utilizada a função “join” sem nenhum parâmetro adicional, já que o interesse é que os valores dos bens dos candidatos a deputado sejam trazidos para o *dataframe* “df_consulta” por meio do seu índice.

```
df_consolidado = df_consulta.join(df_bem)
```

```
df_consolidado.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 532281 entries, 10000641345.0 to 270001000000.0
Data columns (total 11 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   NR_IDADE_DATA_POSSE      532279 non-null  float64
 1   DS_GENERO                532281 non-null  object
 2   DS_GRAU_INSTRUCAO        532281 non-null  object
 3   DS_ESTADO_CIVIL          532281 non-null  object
 4   DS_COR_RACA              532281 non-null  object
 5   DS_SIT_TOT_TURNO         532281 non-null  object
 6   VR_DESPESA_MAX_CAMPANHA  532281 non-null  object
 7   SQ_CANDIDATO             0 non-null       float64
 8   DS_TIPO_BEM_CANDIDATO    0 non-null       object
 9   DS_BEM_CANDIDATO         0 non-null       object
 10  VR_BEM_CANDIDATO         0 non-null       float64
dtypes: float64(3), object(8)
memory usage: 48.7+ MB
```

Verifica-se que o novo *dataframe* “df_consolidado” possui 11 colunas, que são as 6 originais do *dataframe* “df_consulta” mais a coluna “VR_BEM_CANDIDATO” do *dataframe* “df_bem”. Contudo, agora temos valores nulos no novo *dataframe*, como pode ser comprovado a seguir:

```
df_consolidado.isnull().sum()
```

```
NR_IDADE_DATA_POSSE          2
DS_GENERO                    0
DS_GRAU_INSTRUCAO            0
DS_ESTADO_CIVIL              0
DS_COR_RACA                  0
DS_SIT_TOT_TURNO             0
VR_DESPESA_MAX_CAMPANHA      0
SQ_CANDIDATO            532281
DS_TIPO_BEM_CANDIDATO   532281
DS_BEM_CANDIDATO        532281
VR_BEM_CANDIDATO        532281
dtype: int64
```

A análise dos próximos dois *dataframes* tem por principal objetivo identificar o quanto cada candidato efetivamente gastou na eleição. O processo para fazer essa relação será o seguinte: identificação do pagamento total de cada despesa no *dataframe* “df_despesas_pagas”, vinculação dessa informação com as despesas contratadas no *dataframe* “df_despesas_contratadas”. que contém o número sequencial do candidato, e, por fim, junção com o *dataframe* consolidado dos candidatos. Como as informações estão cadastradas no site do Tribunal Superior Eleitoral entendeu-se que elas estão aprovadas e, por esse motivo, a análise dos valores será feita apenas após a junção com os dados consolidados.

Utilizando a função “head” no *dataframe* “df_despesas_pagas” tem-se uma amostra dos primeiros itens do *dataframe* e percebe-se, conforme procedimento realizado anteriormente, que há apenas duas colunas, uma identificando a despesa e outra com o seu valor.

SQ_DESPESA	SQ_PARCELAMENTO_DESPESA	DT_PAGTO_DESPESA	DS_DESPESA	VR_PAGTO_DESPESA
38797095	25882911	12/11/2020	Despesa com Impulsionamento de Conteúdos	1000.0
38826744	25906986	13/11/2020	Atividades de militância e mobilização de rua	500.0
38734630	25832961	14/11/2020	Despesas com pessoal	250.0
38734631	25832968	14/11/2020	Despesas com pessoal	250.0
38734575	25832991	14/11/2020	Despesas com pessoal	250.0

Como cada despesa poderia ser paga por meio de parcelas, será utilizado o mesmo processo para agrupamento realizado para o patrimônio dos candidatos, ou seja, somando o valor de cada despesa.

```
df_despesas_paga = df_despesas_paga.groupby(['SQ_DESPESA']).sum()
```

```
df_despesas_paga.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2776968 entries, 25641558 to 41360384
Data columns (total 11 columns):
 #   Column                  Dtype
---  ------                  -----
 0   ANO_ELEICAO             int64
 1   CD_TIPO_ELEICAO         int64
 2   CD_ELEICAO              int64
 3   ST_TURNO                int64
 4   SQ_PRESTADOR_CONTAS     int64
 5   CD_FONTE_DESPESA        int64
 6   CD_ORIGEM_DESPESA       int64
 7   CD_NATUREZA_DESPESA     int64
 8   CD_ESPECIE_RECURSO      int64
 9   SQ_PARCELAMENTO_DESPESA int64
 10  VR_PAGTO_DESPESA        float64
dtypes: float64(1), int64(10)
memory usage: 254.2 MB
```

Percebe-se a redução de entradas no *dataframe* após esse procedimento, de 3.054.880 para 2.776.968, indicando a soma das despesas com o mesmo índice.

No *dataframe* "df_despesas_contratadas", assim como no *dataframe* "df_despesas_pagas", a coluna "SQ_DESPESA" será transformada em índice, mas agora pelo comando "set_index", também utilizado anteriormente, e pode-se perceber que se terá apenas 3 colunas no *dataframe:*

```
df_despesas_contratadas = df_despesas_contratadas.set_index('SQ_DESPESA')
```

```
df_despesas_contratadas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3868709 entries, 36238272 to 39555359
Data columns (total 3 columns):
 #   Column                Dtype
---  ------                -----
 0   SQ_CANDIDATO          int64
 1   DS_DESPESA            object
 2   VR_DESPESA_CONTRATADA float64
dtypes: float64(1), int64(1), object(1)
memory usage: 118.1+ MB
```

O próximo passo é fazer a junção do *dataframe* “df_despesas_pagas” ao *dataframe* “df_despesas_contratadas” por meio do comando “join” em um novo *dataframe* chamado “df_despesas”.

```
df_despesas = df_despesas_contratadas.join(df_despesas_paga)
```

```
df_despesas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3868709 entries, 25641558 to 41360384
Data columns (total 14 columns):
 #   Column                   Dtype
---  ------                   -----
 0   SQ_CANDIDATO             int64
 1   DS_DESPESA               object
 2   VR_DESPESA_CONTRATADA    float64
 3   ANO_ELEICAO              float64
 4   CD_TIPO_ELEICAO          float64
 5   CD_ELEICAO               float64
 6   ST_TURNO                 float64
 7   SQ_PRESTADOR_CONTAS      float64
 8   CD_FONTE_DESPESA         float64
 9   CD_ORIGEM_DESPESA        float64
 10  CD_NATUREZA_DESPESA      float64
 11  CD_ESPECIE_RECURSO       float64
 12  SQ_PARCELAMENTO_DESPESA  float64
 13  VR_PAGTO_DESPESA         float64
dtypes: float64(12), int64(1), object(1)
memory usage: 442.7+ MB
```

Identifica-se que esse novo *dataframe* possui as colunas dos dois *dataframe* anteriores. Contudo, não se verifica a identificação de valores nulos por meio desse comando, portanto, será utilizado a função “isnull()” em conjunto com a função “sum()” para identificar esse número.

```
df_despesas.isnull().sum()
```

```
SQ_CANDIDATO                0
DS_DESPESA                  0
VR_DESPESA_CONTRATADA       0
VR_PAGTO_DESPESA        48765
dtype: int64
```

Verifica-se que há 48.765 registros nulos que podem ser explicados pelo fato de uma despesa ter sido contratada, mas não paga. Por esse motivo, esses registros serão preenchidos com o valor 0 (zero), por meio da função “fillna”.

```
df_despesas.fillna(0, inplace=True)
```

```
df_despesas.isnull().sum()
```

```
SQ_CANDIDATO               0
DS_DESPESA                 0
VR_DESPESA_CONTRATADA      0
ANO_ELEICAO                0
CD_TIPO_ELEICAO            0
CD_ELEICAO                 0
ST_TURNO                   0
SQ_PRESTADOR_CONTAS        0
CD_FONTE_DESPESA           0
CD_ORIGEM_DESPESA          0
CD_NATUREZA_DESPESA        0
CD_ESPECIE_RECURSO         0
SQ_PARCELAMENTO_DESPESA    0
VR_PAGTO_DESPESA           0
dtype: int64
```

Em seguida, os dados serão agrupados pelo número sequencial do candidato, somando-se todos os valores de despesas do *dataframe*.

```
df_despesas = df_despesas.groupby(['SQ_CANDIDATO']).sum()
```

```
df_despesas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 312710 entries, 10000641345 to 270001274657
Data columns (total 12 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   VR_DESPESA_CONTRATADA    312710 non-null  float64
 1   ANO_ELEICAO              312710 non-null  float64
 2   CD_TIPO_ELEICAO          312710 non-null  float64
 3   CD_ELEICAO               312710 non-null  float64
 4   ST_TURNO                 312710 non-null  float64
 5   SQ_PRESTADOR_CONTAS      312710 non-null  float64
 6   CD_FONTE_DESPESA         312710 non-null  float64
 7   CD_ORIGEM_DESPESA        312710 non-null  float64
 8   CD_NATUREZA_DESPESA      312710 non-null  float64
 9   CD_ESPECIE_RECURSO       312710 non-null  float64
 10  SQ_PARCELAMENTO_DESPESA  312710 non-null  float64
 11  VR_PAGTO_DESPESA         312710 non-null  float64
dtypes: float64(12)
memory usage: 31.0 MB
```

Tem-se agora um *dataframe* com 312.710 entradas, que representam os candidatos com despesas pagas na eleição. Contudo, ainda se verifica duas colunas no *dataframe*: “VR_DESPESA_CONTRATADA” e VR_PAGTO_DESPESA”. O que se deseja é apenas o pagamento efetivo da despesa, por isso, apenas essa coluna será mantida.

```
df_despesas = df_despesas[['VR_PAGTO_DESPESA']]
```

```
df_despesas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 312710 entries, 10000641345 to 270001274657
Data columns (total 1 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   VR_PAGTO_DESPESA  312710 non-null  float64
dtypes: float64(1)
memory usage: 4.8 MB
```

Com os valores das despesas pagas consolidados, deve-se, agora, fazer a junção entre esse *dataframe* e o que contém os demais dados dos candidatos. Para esse procedimento, será utilizado, novamente, o comando "join" para criar o *dataframe* "df_candidatos_final".

```
df_deputados_final = df_consolidado.join(df_despesas)
```

```
df_deputados_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 532281 entries, 10000641345.0 to 270001000000.0
Data columns (total 12 columns):
 #   Column                   Non-Null Count   Dtype  
---  ------                   --------------   -----  
 0   NR_IDADE_DATA_POSSE      532281 non-null  float64
 1   DS_GENERO                532281 non-null  object 
 2   DS_GRAU_INSTRUCAO        532281 non-null  object 
 3   DS_ESTADO_CIVIL          532281 non-null  object 
 4   DS_COR_RACA              532281 non-null  object 
 5   DS_SIT_TOT_TURNO         532281 non-null  object 
 6   VR_DESPESA_MAX_CAMPANHA  532281 non-null  object 
 7   SQ_CANDIDATO             532281 non-null  float64
 8   DS_TIPO_BEM_CANDIDATO    532281 non-null  int64  
 9   DS_BEM_CANDIDATO         532281 non-null  int64  
 10  VR_BEM_CANDIDATO         532281 non-null  float64
 11  VR_PAGTO_DESPESA         66482 non-null   float64
dtypes: float64(4), int64(2), object(6)
memory usage: 52.8+ MB
```

Vê-se agora que esse *dataframe* contém, também, a coluna do valor de pagamento das despesas, porém, novamente, identifica-se valores nulos, como pode ser confirmado a seguir.

```
df_deputados_final.isnull().sum()
```

```
NR_IDADE_DATA_POSSE             0
DS_GENERO                       0
DS_GRAU_INSTRUCAO               0
DS_ESTADO_CIVIL                 0
DS_COR_RACA                     0
DS_SIT_TOT_TURNO                0
VR_DESPESA_MAX_CAMPANHA         0
SQ_CANDIDATO                    0
DS_TIPO_BEM_CANDIDATO           0
DS_BEM_CANDIDATO                0
VR_BEM_CANDIDATO                0
VR_PAGTO_DESPESA           465799
dtype: int64
```

Novamente esses valores serão preenchidos com 0 (zero), pois entende-se que esses valores se devem ao fato de o candidato não ter tido despesas pagas na eleição.

```
df_deputados_final.fillna(0, inplace=True)
```

```
df_deputados_final.isnull().sum()
```

```
NR_IDADE_DATA_POSSE        0
DS_GENERO                  0
DS_GRAU_INSTRUCAO          0
DS_ESTADO_CIVIL            0
DS_COR_RACA                0
DS_SIT_TOT_TURNO           0
VR_DESPESA_MAX_CAMPANHA    0
SQ_CANDIDATO               0
DS_TIPO_BEM_CANDIDATO      0
DS_BEM_CANDIDATO           0
VR_BEM_CANDIDATO           0
VR_PAGTO_DESPESA           0
dtype: int64
```

Com os dados concentrados em um único *dataframe*, é possível analisar as informações de valores pagos de todos os candidatos e dos candidatos eleitos. O procedimento para essa verificação será idêntico aos descritos anteriormente.

```
df_deputados_final.eleitos = df_deputados_final[df_deputados_final.DS_SIT_TOT_TURNO.isin
                                                (['ELEITO POR MÉDIA', 'ELEITO POR QP'])]
```

```
df_deputados_final.eleitos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 57999 entries, 10000641352.0 to 270001000000.0
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   NR_IDADE_DATA_POSSE      57999 non-null  float64
 1   DS_GENERO                57999 non-null  object 
 2   DS_GRAU_INSTRUCAO        57999 non-null  object 
 3   DS_ESTADO_CIVIL          57999 non-null  object 
 4   DS_COR_RACA              57999 non-null  object 
 5   DS_SIT_TOT_TURNO         57999 non-null  object 
 6   VR_DESPESA_MAX_CAMPANHA  57999 non-null  object 
 7   SQ_CANDIDATO             57999 non-null  float64
 8   DS_TIPO_BEM_CANDIDATO    57999 non-null  int64  
 9   DS_BEM_CANDIDATO         57999 non-null  int64  
 10  VR_BEM_CANDIDATO         57999 non-null  float64
 11  VR_PAGTO_DESPESA         57999 non-null  float64
dtypes: float64(4), int64(2), object(6)
memory usage: 5.8+ MB
```

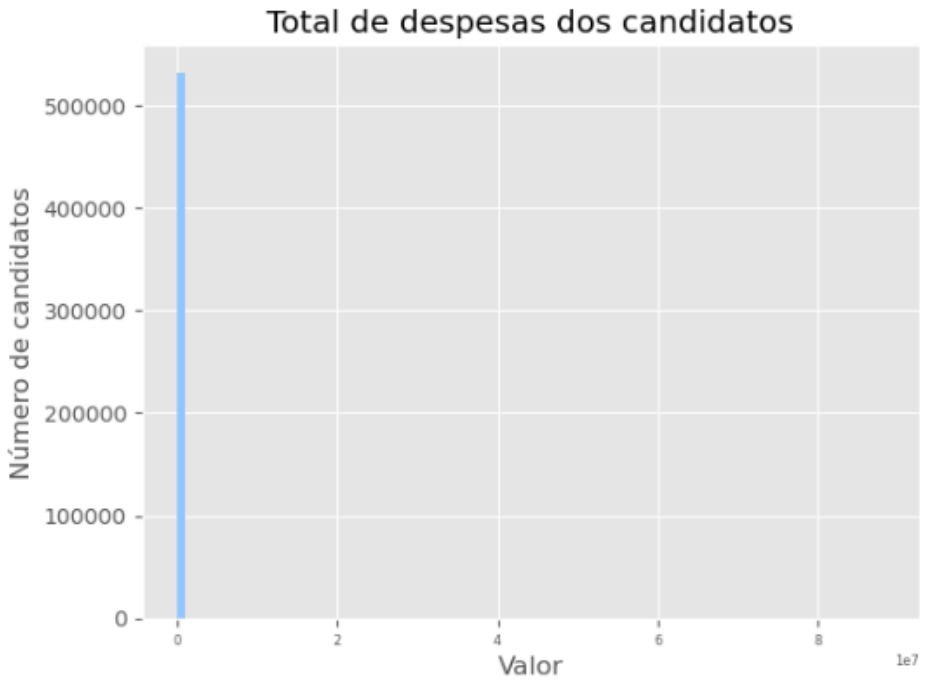
```
df_deputados_final['VR_PAGTO_DESPESA'].describe().astype('int')
```

```
count      532281
mean         4238
std        219435
min             0
25%             0
50%             0
75%             0
max      88207497
Name: VR_PAGTO_DESPESA, dtype: int32
```

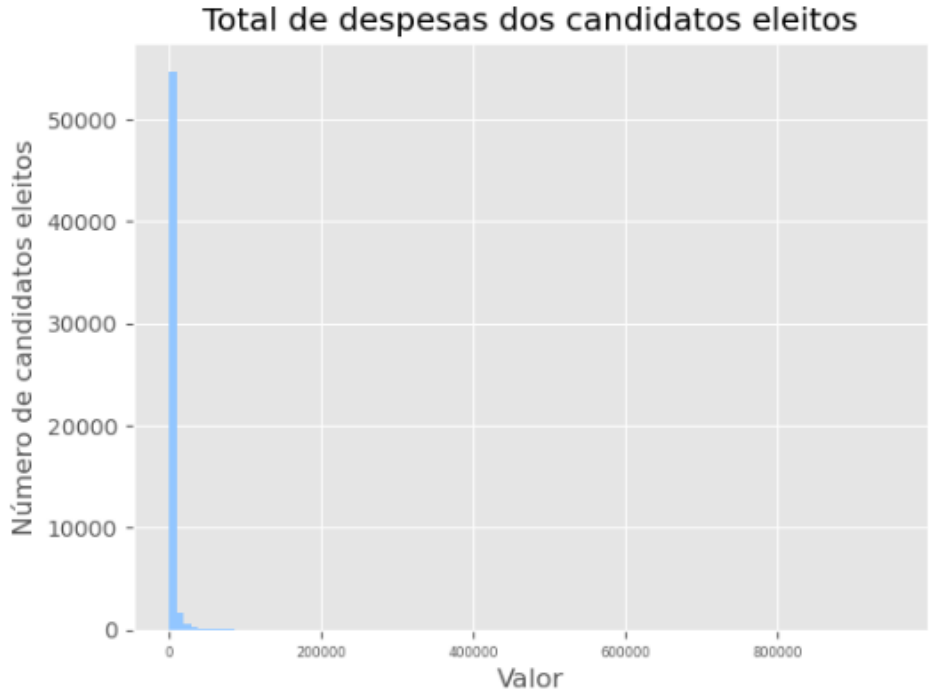
```
df_deputados_final.eleitos['VR_PAGTO_DESPESA'].describe().astype('int')
```

```
count     57999
mean       2558
std       18391
min           0
25%           0
50%           0
75%           0
max      949987
Name: VR_PAGTO_DESPESA, dtype: int32
```

```
df_deputados_final.VR_PAGTO_DESPESA.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel('Valor')
plt.ylabel('Número de candidatos')
plt.title('Total de despesas dos candidatos')
plt.show()
```




```
df_deputados_final.eleitos.VR_PAGTO_DESPESA.hist(bins=100)
plt.style.use('ggplot')
plt.xlabel('Valor')
plt.ylabel('Número de candidatos eleitos')
plt.title('Total de despesas dos candidatos eleitos')
plt.show()
```



Como era esperado, percebe-se que os candidatos eleitos gastaram significativamente mais do que a média de todos os candidatos. O valor médio variou de R$ 500.000,00 para R$ 50.000,00.

5. Criação de Modelos de Machine Learning

Como o machine learning é um computador que analisa dados, identifica padrões e então usa esses insights para concluir melhor sua tarefa atribuída, vamos utiliza-lo para identificar quais candidatos seriam eleitos com um conjunto de características. Sendo assim, vamos utilizar algoritmos de classificação, que são cálculos preditivos usados para atribuir dados a categorias predefinidas, analisando um conjunto de dados de treinamento.

Serão utilizados 4 tipos de algoritmos de classificação, são eles: Árvore de Decisão, Regressão Logística, Gradiente Descendente e Randon Forest.

Para avaliar os algoritmos, optamos por realizar alguns ajustes no *dataframe* "df_deputados_final". Por recomendação, os dados categóricos serão transformados em valores inteiros, mais especificamente nas colunas DS_GENERO, DS_GRAU_INSTRUCAO, DS_ESTADO_CIVIL, DS_COR_RACA e DS_SIT_TOT_TURNO que possuem os seguintes valores:

```
df_deputados_final['DS_GENERO'].unique()
```

```
array(['MASCULINO', 'FEMININO', 'NÃO DIVULGÁVEL'], dtype=object)
```

```
df_deputados_final['DS_GRAU_INSTRUCAO'].unique()
```

```
array(['ENSINO MÉDIO COMPLETO', 'ENSINO FUNDAMENTAL COMPLETO',
       'ENSINO MÉDIO INCOMPLETO', 'ENSINO FUNDAMENTAL INCOMPLETO',
       'SUPERIOR COMPLETO', 'LÊ E ESCREVE', 'SUPERIOR INCOMPLETO',
       'ANALFABETO', 'NÃO DIVULGÁVEL'], dtype=object)
```

```
df_deputados_final['DS_ESTADO_CIVIL'].unique()
```

```
array(['CASADO(A)', 'SOLTEIRO(A)', 'DIVORCIADO(A)', 'VIÚVO(A)',
       'SEPARADO(A) JUDICIALMENTE', 'NÃO DIVULGÁVEL'], dtype=object)
```

```
df_deputados_final['DS_COR_RACA'].unique()
```

```
array(['PARDA', 'INDÍGENA', 'BRANCA', 'PRETA', 'AMARELA', 'NÃO INFORMADO',
       'NÃO DIVULGÁVEL'], dtype=object)
```

```
df_deputados_final['DS_SIT_TOT_TURNO'].unique()
```

```
array(['SUPLENTE', 'ELEITO POR QP', 'ELEITO POR MÉDIA', 'NÃO ELEITO',
       'ELEITO', '2º TURNO', '#NULO#'], dtype=object)
```

A escolaridade dos candidatos será a primeira informação a ser transformada, a coluna DS_GRAU_INSTRUCAO. É possível ordenar as informações, do grau de escolaridade mais baixo que é “LÊ E ESCREVE” até o mais alto “SUPERIOR COMPLETO”. Para fazer esse ajuste, será criado um dicionário ordenando esses níveis de 1 a 7 e em seguida os valores serão substituídos no *dataframe* utilizando a função “map”.

```
ajuste_ensino = {'LÊ E ESCREVE': 1, 'ENSINO FUNDAMENTAL INCOMPLETO': 2,
'ENSINO FUNDAMENTAL COMPLETO': 3, 'ENSINO MÉDIO INCOMPLETO': 4,
'ENSINO MÉDIO COMPLETO': 5, 'SUPERIOR INCOMPLETO': 6, 'SUPERIOR COMPLETO': 7}
df_deputados_final['DS_GRAU_INSTRUCAO'] = df_deputados_final['DS_GRAU_INSTRUCAO'].map(ajuste_ensino)
```

```
df_deputados_final['DS_GRAU_INSTRUCAO'].unique()
```

```
array([ 5.,  3.,  4.,  2.,  7.,  1.,  6., nan])
```

A coluna “DS_SIT_TOT_TURNO” que indica se o candidato foi ou não eleito. Neste caso, não será possível ordenar as opções, mas elas serão substituídas por “0” e “1”, sendo “0” as situações em que o candidato não foi eleito e “1” as que ele foi eleito.

```
ajuste_eleito = {'ELEITO POR QP': 1, 'SUPLENTE': 0, 'NÃO ELEITO': 0, 'ELEITO POR MÉDIA': 1}
df_deputados_final['DS_SIT_TOT_TURNO'] = df_deputados_final['DS_SIT_TOT_TURNO'].map(ajuste_eleito)
```

```
df_deputados_final['DS_SIT_TOT_TURNO'].unique()
```

```
array([ 0.,  1., nan])
```

As demais colunas não permitem uma ordenação ou uma classificação binária, pois tratam do gênero, raça e estado civil dos candidatos. Nesse caso, a alternativa escolhida foi utilizar a função “get_dummies” que cria novas colunas no *dataframe* que terão como nome os valores de cada coluna com informações categóricas e preenche cada linha com “0” e “1” dependendo do valor que a entrada possuía. A função é aplicada a todas a colunas, mas ela atuará apenas nas categóricas.

```
df_deputados_final = pd.get_dummies(df_deputados_final[['NR_IDADE_DATA_POSSE', 'DS_GENERO',
'DS_GRAU_INSTRUCAO', 'DS_ESTADO_CIVIL', 'DS_COR_RACA', 'DS_SIT_TOT_TURNO','VR_BEM_CANDIDATO', 'VR_PAGTO_DESPESA' ]])
```

```
df_deputados_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 532281 entries, 10000641345.0 to 270001000000.0
Data columns (total 21 columns):
 #   Column                                  Non-Null Count   Dtype
---  ------                                  --------------   -----
 0   NR_IDADE_DATA_POSSE                     532281 non-null  float64
 1   DS_GRAU_INSTRUCAO                       532278 non-null  float64
 2   DS_SIT_TOT_TURNO                        518867 non-null  float64
 3   VR_BEM_CANDIDATO                        532281 non-null  float64
 4   VR_PAGTO_DESPESA                        532281 non-null  float64
 5   DS_GENERO_FEMININO                      532281 non-null  uint8
 6   DS_GENERO_MASCULINO                     532281 non-null  uint8
 7   DS_GENERO_NÃO DIVULGÁVEL                532281 non-null  uint8
 8   DS_ESTADO_CIVIL_CASADO(A)               532281 non-null  uint8
 9   DS_ESTADO_CIVIL_DIVORCIADO(A)           532281 non-null  uint8
 10  DS_ESTADO_CIVIL_NÃO DIVULGÁVEL          532281 non-null  uint8
 11  DS_ESTADO_CIVIL_SEPARADO(A) JUDICIALMENTE  532281 non-null  uint8
 12  DS_ESTADO_CIVIL_SOLTEIRO(A)             532281 non-null  uint8
 13  DS_ESTADO_CIVIL_VIÚVO(A)                532281 non-null  uint8
 14  DS_COR_RACA_AMARELA                     532281 non-null  uint8
 15  DS_COR_RACA_BRANCA                      532281 non-null  uint8
 16  DS_COR_RACA_INDÍGENA                    532281 non-null  uint8
 17  DS_COR_RACA_NÃO DIVULGÁVEL              532281 non-null  uint8
 18  DS_COR_RACA_NÃO INFORMADO               532281 non-null  uint8
 19  DS_COR_RACA_PARDA                       532281 non-null  uint8
 20  DS_COR_RACA_PRETA                       532281 non-null  uint8
dtypes: float64(5), uint8(16)
memory usage: 32.5 MB
```

As colunas DS_GENERO, DS_ESTADO_CIVIL e DS_COR_RACA agora estão divididas em suas categorias e apresentam números inteiros.

```
df_deputados_final['DS_GENERO_FEMININO'].unique()
```

```
array([0, 1], dtype=uint8)
```

A análise busca avaliar a previsibilidade de um candidato ser eleito ou não com base em alguns parâmetros. Portanto, a informação que utilizaremos como resultado é a coluna DS_SIT_TOT_TURNO, que traz se o candidato foi eleito ou não. Contudo, como esperado, tem-se muito mais candidatos não eleitos do que eleitos, respectivamente 460.868 e 57.999.

```
df_deputados_final['DS_SIT_TOT_TURNO'].value_counts()
```

```
0.0    460868
1.0     57999
Name: DS_SIT_TOT_TURNO, dtype: int64
```

Essa situação é prejudicial para a construção do algoritmo porque uma simples afirmação de que o candidato não será eleito acertaria em 93% das vezes, nesse caso concreto. Para contornar esse problema, será utilizada a biblioteca Scikit-learn, mais

especificamente sua função resample, que, após alguns passos, igualará a quantidade de amostras de candidatos eleitos e não eleitos.

Em seguida iremos importar a função:

```
from sklearn.utils import resample
```

O dataframe será dividido entre as entradas que possuem valor 0 e valor 1 na coluna DS_SIT_TOT_TURNO:

```
df_deputados_final_majority = df_deputados_final[df_deputados_final.DS_SIT_TOT_TURNO==0]
df_deputados_final_majority.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 460868 entries, 10000641345.0 to 270001000000.0
Data columns (total 21 columns):
 #   Column                                  Non-Null Count   Dtype
---  ------                                  --------------   -----
 0   NR_IDADE_DATA_POSSE                     460868 non-null  float64
 1   DS_GRAU_INSTRUCAO                       460865 non-null  float64
 2   DS_SIT_TOT_TURNO                        460868 non-null  float64
 3   VR_BEM_CANDIDATO                        460868 non-null  float64
 4   VR_PAGTO_DESPESA                        460868 non-null  float64
 5   DS_GENERO_FEMININO                      460868 non-null  uint8
 6   DS_GENERO_MASCULINO                     460868 non-null  uint8
 7   DS_GENERO_NÃO DIVULGÁVEL                460868 non-null  uint8
 8   DS_ESTADO_CIVIL_CASADO(A)               460868 non-null  uint8
 9   DS_ESTADO_CIVIL_DIVORCIADO(A)           460868 non-null  uint8
 10  DS_ESTADO_CIVIL_NÃO DIVULGÁVEL          460868 non-null  uint8
 11  DS_ESTADO_CIVIL_SEPARADO(A) JUDICIALMENTE  460868 non-null  uint8
 12  DS_ESTADO_CIVIL_SOLTEIRO(A)             460868 non-null  uint8
 13  DS_ESTADO_CIVIL_VIÚVO(A)                460868 non-null  uint8
 14  DS_COR_RACA_AMARELA                     460868 non-null  uint8
 15  DS_COR_RACA_BRANCA                      460868 non-null  uint8
 16  DS_COR_RACA_INDÍGENA                    460868 non-null  uint8
 17  DS_COR_RACA_NÃO DIVULGÁVEL              460868 non-null  uint8
 18  DS_COR_RACA_NÃO INFORMADO               460868 non-null  uint8
 19  DS_COR_RACA_PARDA                       460868 non-null  uint8
 20  DS_COR_RACA_PRETA                       460868 non-null  uint8
dtypes: float64(5), uint8(16)
memory usage: 28.1 MB
```

```
df_deputados_final_minority = df_deputados_final[df_deputados_final.DS_SIT_TOT_TURNO==1]
df_deputados_final_minority.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 57999 entries, 10000641352.0 to 270001000000.0
Data columns (total 21 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   NR_IDADE_DATA_POSSE                       57999 non-null  float64
 1   DS_GRAU_INSTRUCAO                         57999 non-null  float64
 2   DS_SIT_TOT_TURNO                          57999 non-null  float64
 3   VR_BEM_CANDIDATO                          57999 non-null  float64
 4   VR_PAGTO_DESPESA                          57999 non-null  float64
 5   DS_GENERO_FEMININO                        57999 non-null  uint8
 6   DS_GENERO_MASCULINO                       57999 non-null  uint8
 7   DS_GENERO_NÃO DIVULGÁVEL                  57999 non-null  uint8
 8   DS_ESTADO_CIVIL_CASADO(A)                 57999 non-null  uint8
 9   DS_ESTADO_CIVIL_DIVORCIADO(A)             57999 non-null  uint8
 10  DS_ESTADO_CIVIL_NÃO DIVULGÁVEL            57999 non-null  uint8
 11  DS_ESTADO_CIVIL_SEPARADO(A) JUDICIALMENTE 57999 non-null  uint8
 12  DS_ESTADO_CIVIL_SOLTEIRO(A)               57999 non-null  uint8
 13  DS_ESTADO_CIVIL_VIÚVO(A)                  57999 non-null  uint8
 14  DS_COR_RACA_AMARELA                       57999 non-null  uint8
 15  DS_COR_RACA_BRANCA                        57999 non-null  uint8
 16  DS_COR_RACA_INDÍGENA                      57999 non-null  uint8
 17  DS_COR_RACA_NÃO DIVULGÁVEL                57999 non-null  uint8
 18  DS_COR_RACA_NÃO INFORMADO                 57999 non-null  uint8
 19  DS_COR_RACA_PARDA                         57999 non-null  uint8
 20  DS_COR_RACA_PRETA                         57999 non-null  uint8
dtypes: float64(5), uint8(16)
memory usage: 3.5 MB
```

O objetivo agora é igualar a quantidade de entradas desses novos *dataframes*, aumentando a quantidade de registros do que possui menor número. Esse processo é feito pelo comando a seguir:

```
df_deputados_final_minority_upsampled = resample(df_deputados_final_minority, replace=True, n_samples=22283,
random_state=123)
```

```
df_deputados_final_minority_upsampled.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 22283 entries, 250001000000.0 to 200001000000.0
Data columns (total 21 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   NR_IDADE_DATA_POSSE                       22283 non-null  float64
 1   DS_GRAU_INSTRUCAO                         22283 non-null  float64
 2   DS_SIT_TOT_TURNO                          22283 non-null  float64
 3   VR_BEM_CANDIDATO                          22283 non-null  float64
 4   VR_PAGTO_DESPESA                          22283 non-null  float64
 5   DS_GENERO_FEMININO                        22283 non-null  uint8
 6   DS_GENERO_MASCULINO                       22283 non-null  uint8
 7   DS_GENERO_NÃO DIVULGÁVEL                  22283 non-null  uint8
 8   DS_ESTADO_CIVIL_CASADO(A)                 22283 non-null  uint8
 9   DS_ESTADO_CIVIL_DIVORCIADO(A)             22283 non-null  uint8
 10  DS_ESTADO_CIVIL_NÃO DIVULGÁVEL            22283 non-null  uint8
 11  DS_ESTADO_CIVIL_SEPARADO(A) JUDICIALMENTE 22283 non-null  uint8
 12  DS_ESTADO_CIVIL_SOLTEIRO(A)               22283 non-null  uint8
 13  DS_ESTADO_CIVIL_VIÚVO(A)                  22283 non-null  uint8
 14  DS_COR_RACA_AMARELA                       22283 non-null  uint8
 15  DS_COR_RACA_BRANCA                        22283 non-null  uint8
 16  DS_COR_RACA_INDÍGENA                      22283 non-null  uint8
 17  DS_COR_RACA_NÃO DIVULGÁVEL                22283 non-null  uint8
 18  DS_COR_RACA_NÃO INFORMADO                 22283 non-null  uint8
 19  DS_COR_RACA_PARDA                         22283 non-null  uint8
 20  DS_COR_RACA_PRETA                         22283 non-null  uint8
dtypes: float64(5), uint8(16)
memory usage: 1.4 MB
```

Agora, com os dois *dataframes* com o mesmo número de entradas, eles devem ser concatenados para se ter uma única fonte de dados novamente. Esse processo será feito utilizando a função “pd.concat”.

```
df_deputados_final_upsampled = pd.concat([df_deputados_final_majority,
                                          df_deputados_final_minority_upsampled])

df_deputados_final_upsampled.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 483151 entries, 10000641345.0 to 200001000000.0
Data columns (total 21 columns):
 #   Column                                  Non-Null Count   Dtype
---  ------                                  --------------   -----
 0   NR_IDADE_DATA_POSSE                     483151 non-null  float64
 1   DS_GRAU_INSTRUCAO                       483148 non-null  float64
 2   DS_SIT_TOT_TURNO                        483151 non-null  float64
 3   VR_BEM_CANDIDATO                        483151 non-null  float64
 4   VR_PAGTO_DESPESA                        483151 non-null  float64
 5   DS_GENERO_FEMININO                      483151 non-null  uint8
 6   DS_GENERO_MASCULINO                     483151 non-null  uint8
 7   DS_GENERO_NÃO DIVULGÁVEL                483151 non-null  uint8
 8   DS_ESTADO_CIVIL_CASADO(A)               483151 non-null  uint8
 9   DS_ESTADO_CIVIL_DIVORCIADO(A)           483151 non-null  uint8
 10  DS_ESTADO_CIVIL_NÃO DIVULGÁVEL          483151 non-null  uint8
 11  DS_ESTADO_CIVIL_SEPARADO(A) JUDICIALMENTE  483151 non-null  uint8
 12  DS_ESTADO_CIVIL_SOLTEIRO(A)             483151 non-null  uint8
 13  DS_ESTADO_CIVIL_VIÚVO(A)                483151 non-null  uint8
 14  DS_COR_RACA_AMARELA                     483151 non-null  uint8
 15  DS_COR_RACA_BRANCA                      483151 non-null  uint8
 16  DS_COR_RACA_INDÍGENA                    483151 non-null  uint8
 17  DS_COR_RACA_NÃO DIVULGÁVEL              483151 non-null  uint8
 18  DS_COR_RACA_NÃO INFORMADO               483151 non-null  uint8
 19  DS_COR_RACA_PARDA                       483151 non-null  uint8
 20  DS_COR_RACA_PRETA                       483151 non-null  uint8
dtypes: float64(5), uint8(16)
memory usage: 29.5 MB
```

```
df_deputados_final_upsampled['DS_SIT_TOT_TURNO'].value_counts()
```

```
0.0    460868
1.0     22283
Name: DS_SIT_TOT_TURNO, dtype: int64
```

Percebe-se que o novo *dataframe* possui 483.151 entradas, divididas igualmente entre candidatos eleitos e não eleitos.

O próximo passo para a aplicação dos algoritmos de classificação é dividir o *dataframe* em bases de treinamento e de teste. Para esse procedimento será utilizado a função “train_test_split”, também da biblioteca Scikit-learn.

```
from sklearn.model_selection import train_test_split
```

Antes de utilizar a função mencionada, o *dataframe* será dividido em dois: um onde consta apenas os atributos que serão analisados e outra que mostra se o candidato foi ou não

eleito. O *dataframe* que contém todos os atributos será chamado do de X_train e o que contém o resultado será chamado de y_train. Importante perceber que y_train agora é uma série de valores.

```
X_train = df_deputados_final_upsampled.drop(['DS_SIT_TOT_TURNO'], axis = 1)
y_train = df_deputados_final_upsampled.DS_SIT_TOT_TURNO
X_train.info()
type(y_train)
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 483151 entries, 10000641345.0 to 200001000000.0
Data columns (total 20 columns):
 #   Column                                  Non-Null Count   Dtype
---  ------                                  --------------   -----
 0   NR_IDADE_DATA_POSSE                     483151 non-null  float64
 1   DS_GRAU_INSTRUCAO                       483148 non-null  float64
 2   VR_BEM_CANDIDATO                        483151 non-null  float64
 3   VR_PAGTO_DESPESA                        483151 non-null  float64
 4   DS_GENERO_FEMININO                      483151 non-null  uint8
 5   DS_GENERO_MASCULINO                     483151 non-null  uint8
 6   DS_GENERO_NÃO DIVULGÁVEL                483151 non-null  uint8
 7   DS_ESTADO_CIVIL_CASADO(A)               483151 non-null  uint8
 8   DS_ESTADO_CIVIL_DIVORCIADO(A)           483151 non-null  uint8
 9   DS_ESTADO_CIVIL_NÃO DIVULGÁVEL          483151 non-null  uint8
 10  DS_ESTADO_CIVIL_SEPARADO(A) JUDICIALMENTE  483151 non-null  uint8
 11  DS_ESTADO_CIVIL_SOLTEIRO(A)             483151 non-null  uint8
 12  DS_ESTADO_CIVIL_VIÚVO(A)                483151 non-null  uint8
 13  DS_COR_RACA_AMARELA                     483151 non-null  uint8
 14  DS_COR_RACA_BRANCA                      483151 non-null  uint8
 15  DS_COR_RACA_INDÍGENA                    483151 non-null  uint8
 16  DS_COR_RACA_NÃO DIVULGÁVEL              483151 non-null  uint8
 17  DS_COR_RACA_NÃO INFORMADO               483151 non-null  uint8
 18  DS_COR_RACA_PARDA                       483151 non-null  uint8
 19  DS_COR_RACA_PRETA                       483151 non-null  uint8
dtypes: float64(4), uint8(16)
memory usage: 25.8 MB

pandas.core.series.Series
```

Agora será feita a divisão entre bases de treinamento e de teste utilizando a função “train_test_split”. Para a divisão de percentual de dados para treinamento e para teste, será utilizado o padrão da função, que é de 75% para treinamento e 25% para teste.

```
xtreinamento, xteste, ytreinamento, yteste = train_test_split(X_train, y_train, random_state = 0)
```

```
xtreinamento.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 362363 entries, 240001000000.0 to 150001000000.0
Data columns (total 20 columns):
 #   Column                                  Non-Null Count   Dtype  
---  ------                                  --------------   -----  
 0   NR_IDADE_DATA_POSSE                     362363 non-null  float64
 1   DS_GRAU_INSTRUCAO                       362360 non-null  float64
 2   VR_BEM_CANDIDATO                        362363 non-null  float64
 3   VR_PAGTO_DESPESA                        362363 non-null  float64
 4   DS_GENERO_FEMININO                      362363 non-null  uint8  
 5   DS_GENERO_MASCULINO                     362363 non-null  uint8  
 6   DS_GENERO_NÃO DIVULGÁVEL                362363 non-null  uint8  
 7   DS_ESTADO_CIVIL_CASADO(A)               362363 non-null  uint8  
 8   DS_ESTADO_CIVIL_DIVORCIADO(A)           362363 non-null  uint8  
 9   DS_ESTADO_CIVIL_NÃO DIVULGÁVEL          362363 non-null  uint8  
 10  DS_ESTADO_CIVIL_SEPARADO(A) JUDICIALMENTE  362363 non-null  uint8  
 11  DS_ESTADO_CIVIL_SOLTEIRO(A)             362363 non-null  uint8  
 12  DS_ESTADO_CIVIL_VIÚVO(A)                362363 non-null  uint8  
 13  DS_COR_RACA_AMARELA                     362363 non-null  uint8  
 14  DS_COR_RACA_BRANCA                      362363 non-null  uint8  
 15  DS_COR_RACA_INDÍGENA                    362363 non-null  uint8  
 16  DS_COR_RACA_NÃO DIVULGÁVEL              362363 non-null  uint8  
 17  DS_COR_RACA_NÃO INFORMADO               362363 non-null  uint8  
 18  DS_COR_RACA_PARDA                       362363 non-null  uint8  
 19  DS_COR_RACA_PRETA                       362363 non-null  uint8  
dtypes: float64(4), uint8(16)
memory usage: 19.4 MB
```

```
xteste.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 120788 entries, 240001000000.0 to 90000722946.0
Data columns (total 20 columns):
 #   Column                                  Non-Null Count   Dtype  
---  ------                                  --------------   -----  
 0   NR_IDADE_DATA_POSSE                     120788 non-null  float64
 1   DS_GRAU_INSTRUCAO                       120788 non-null  float64
 2   VR_BEM_CANDIDATO                        120788 non-null  float64
 3   VR_PAGTO_DESPESA                        120788 non-null  float64
 4   DS_GENERO_FEMININO                      120788 non-null  uint8  
 5   DS_GENERO_MASCULINO                     120788 non-null  uint8  
 6   DS_GENERO_NÃO DIVULGÁVEL                120788 non-null  uint8  
 7   DS_ESTADO_CIVIL_CASADO(A)               120788 non-null  uint8  
 8   DS_ESTADO_CIVIL_DIVORCIADO(A)           120788 non-null  uint8  
 9   DS_ESTADO_CIVIL_NÃO DIVULGÁVEL          120788 non-null  uint8  
 10  DS_ESTADO_CIVIL_SEPARADO(A) JUDICIALMENTE  120788 non-null  uint8  
 11  DS_ESTADO_CIVIL_SOLTEIRO(A)             120788 non-null  uint8  
 12  DS_ESTADO_CIVIL_VIÚVO(A)                120788 non-null  uint8  
 13  DS_COR_RACA_AMARELA                     120788 non-null  uint8  
 14  DS_COR_RACA_BRANCA                      120788 non-null  uint8  
 15  DS_COR_RACA_INDÍGENA                    120788 non-null  uint8  
 16  DS_COR_RACA_NÃO DIVULGÁVEL              120788 non-null  uint8  
 17  DS_COR_RACA_NÃO INFORMADO               120788 non-null  uint8  
 18  DS_COR_RACA_PARDA                       120788 non-null  uint8  
 19  DS_COR_RACA_PRETA                       120788 non-null  uint8  
dtypes: float64(4), uint8(16)
memory usage: 6.5 MB
```

```
ytreinamento.count()
```

```
362363
```

```
yteste.count()
```

```
120788
```

Ao aplicar os algoritmos de classificação, será necessário definir qual medida de avaliação será analisada para verificar a eficiência do modelo. Com esse objetivo, serão utilizadas duas funções da biblioteca Scikit-learn: “accuracy_score” e “classification_report”. Essas funções mostram as medidas de acurácia, precisão, revocação (recall) e f1-score.

A acurácia é a quantidade de acertos do modelo dividido pelo total da amostra e indica uma performance geral do modelo:

$$Acurácia = \frac{Verdadeiros\ Positivos + Verdadeiros\ Negativos}{Total\ de\ Amostras}$$

A precisão define os chamados positivos verdadeiros, ou seja, dentre os exemplos classificados como verdadeiros, quantos eram realmente verdadeiros.

$$Precisão = \frac{Verdadeiros\ Positivos}{Verdadeiros\ Positivos + Falsos\ Positivos}$$

A revocação (recall) indica qual a porcentagem de dados classificados como verdadeiros comparado com a quantidade real de resultados verdadeiros que existem na amostra.

$$Revocação = \frac{Verdadeiros\ Positivos}{Verdadeiros\ Positivos + Falsos\ Negativos}$$

A F1-score traz a média ponderada de precisão e revocação e traz um número único que determina a qualidade geral do modelo.

$$F1 - score = \frac{2 * Precisão\ * Revocação}{Precisão + Revocação}$$

A principal medida de avaliação que será utilizada nesse trabalho é a acurácia, porém todas as outras também serão consideradas na avaliação. Para fazer a importação do pacote que apresentará essas medidas, foi utilizado o comando a seguir:

```
from sklearn.metrics import accuracy_score, classification_report
```

Definidos os *dataframes* de treinamento e de teste e escolhidas as medidas de avaliação, o próximo passo é a efetiva utilização dos algoritmos listados anteriormente. O processo para todos será o mesmo, começando pela importação do respectivo algoritmo, realizando o treinamento por meio da função "fit" nas duas bases de treinamento e registrando sua acurácia por meio da função score. Em seguida, será utilizada a função "predict" na base teste xteste e sua saída será comparada com a série yteste, tendo suas medidas de avaliação sendo geradas por meio das funções "accuracy_score" e "classification_report".

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
deputados_tree = DecisionTreeClassifier()
deputados_tree = decision_tree.fit(xtreinamento, ytreinamento)

print("Acurácia: ", deputados_tree.score(xtreinamento, ytreinamento))
Train_predict = deputados_tree.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, Train_predict))
print(classification_report(yteste, Train_predict))
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr = lr.fit(xtreinamento, ytreinamento)
print("Acurácia: ", lr.score(xtreinamento, ytreinamento))
tp_lr = lr.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_lr))
print(classification_report(yteste, tp_lr))
```

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()
sgd = sgd.fit(xtreinamento, ytreinamento)
print("Acurácia: ", sgd.score(xtreinamento, ytreinamento))
tp_sgd = sgd.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_sgd))
print(classification_report(yteste, tp_sgd))
```

```
from sklearn.ensemble import RandomForestClassifier
rfm = RandomForestClassifier()
rfm = rfm.fit(xtreinamento, ytreinamento)
print("Acurácia: ", rfm.score(xtreinamento, ytreinamento))
tp_rfm = rfm.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_rfm))
print(Classifiction_report(yteste, tp_rfm))
```

6. Interpretação dos Resultados

Os resultados da aplicação dos algoritmos de Machine Learning, podem ser verificados na tabela e gráficos a seguir:

Algoritmo	Acurácia de Previsão	Precisão	Revocação	F1-Score
Árvore de Decisão	0,98	0,99	0,98	0,98
Regressão Logística	0,60	0,78	0,61	0,52
Gradiente Descendente	0,60	0,79	0,61	0,53
Randon Forest	0,99	0,99	0,99	0,99

Ao analisar as informações da tabela e o gráfico, podemos perceber que Árvore de Decisão e Randon Forest tiveram os melhores resultados. Mas o algoritmo Randon Forest apresentou o melhor resultado, tendo atingindo o valor de 0,98 para todas as medidas de avaliação(acurácia, precisão, revocação e F1-score).

Sendo assim, os resultados obtidos demostram que a o resultado da eleição é muito previsível, afirmar com 98% de certeza se um candidato será ou não eleito.

O perfil dos candidatos a eleitos são homem, branco, casado, com ensino médio completo e com 44 anos.

Conclui-se que durante anos de eleições não houve mudança significativa em relação aos anos anteriores.

7. Links

Link para o vídeo:

https://youtu.be/21mavMwUWbk

Link para o repositório Github:

https://github.com/felipegaudencio/TCC

Link para o repositório Google Drive:

https://drive.google.com/drive/folders/1Y01d61yEIrOm1psTv5fuD5973PmO-wLi?usp=sharing

APÊNDICE

#Importação da biblioteca pandas

import pandas as pd

#leitura do arquivo consulta_cand_2020_BRASIL.csv

df_consulta = pd.read_csv("consulta_cand_2020_BRASIL.csv", encoding = "Latin 1", sep = ";", decimal = ',')

#informações do dataframe df_consulta

df_consulta.info()

#identificação das opções da coluna 'DS_SITUACAO_CANDIDATURA' do dataframe df_consulta

df_consulta['DS_SITUACAO_CANDIDATURA'].unique()

#Seleção das opções de interesse na coluna DS_SITUACAO_CANDIDATURA

df_consulta = df_consulta[df_consulta.DS_SITUACAO_CANDIDATURA.isin(['APTO'])]

#Informações do dataframe após o filtro aplicado

df_consulta.info()

#Seleção das colunas de interesse

df_consulta = df_consulta[['SQ_CANDIDATO', NR_IDADE_DATA_POSSE','DS_GENERO', 'DS_GRAU_INSTRUCAO', 'DS_ESTADO_CIVIL', 'DS_COR_RACA', 'DS_SIT_TOT_TURNO', 'VR_DESPESA_MAX_CAMPANHA']]

#Informações do dataframe após o filtro aplicado

df_consulta.info()

#Verificação dos valores nulos no dataframe df_consulta

df_consulta.isnull().sum()

#Verificação de idade no momento da posse

```
df_consulta.NR_IDADE_DATA_POSSE.max()

#leitura do arquivo bem_candidato_2020_BRASIL.csv
df_bem = pd.read_csv("bem_candidato_2020_BRASIL.csv", encoding = "Latin 1", sep = ";", decimal = ",")

#informações do dataframe df_bem
df_bem.info()

#Seleção das colunas de interesse
df_bem = df_bem[['SQ_CANDIDATO', 'DS_TIPO_BEM_CANDIDATO', 'DS_BEM_CANDIDATO', 'VR_BEM_CANDIDATO']]

#informações do dataframe df_bem após seleção das colunas
df_bem.info()

#Verificação dos valores nulos no dataframe df_bem
df_bem.isnull().sum()

#leitura do arquivo despesas_contratadas_candidatos_2020_BRASIL.csv
df_despesas_contratadas = pd.read_csv
("despesas_contratadas_candidatos_2020_BRASIL.csv", encoding = "Latin 1", sep = ";", decimal = ",")

#informações do dataframe df_despesas_contratadas
df_despesas_contratadas.info()

#Seleção das colunas de interesse
df_despesas_contratadas = df_despesas_contratadas [['SQ_DESPESA','SQ_CANDIDATO', 'DS_DESPESA','VR_DESPESA_CONTRATADA']]

#informações do dataframe df_despesas_contratadas após seleção das colunas
```

```
df_despesas_contratadas.info()

#Verificação dos valores nulos
df_despesas_contratadas.isnull().sum()

#leitura do arquivo despesas_pagas_candidatos_2020_BRASIL.csv
df_despesas_paga = pd.read_csv ("despesas_pagas_candidatos_2020_BRASIL.csv", encoding = "Latin 1", sep = ";", decimal = ",")

#informações do dataframe df_despesas_pagas
df_despesas_paga.info()

#Seleção das colunas de interesse
df_despesas_paga = df_despesas_paga[['SQ_DESPESA','VR_PAGTO_DESPESA']]

#informações do dataframe df_despesas_contratadas após seleção das colunas
df_despesas_contratadas.info()

#Verificação dos valores nulos
df_despesas_paga.isnull().sum()

#Importação da função Counter e da biblioteca matplotlib.pyplot
from collections import Counter
import matplotlib.pyplot as plt

#identificação das opções da coluna DS_SIT_TOT_TURNO do dataframe df_consulta
df_consulta['DS_SIT_TOT_TURNO'].unique()

#Criação de um dataframe df_consulta apenas com candidatos eleitos
df_consulta_eleitos = df_consulta[df_consulta.DS_SIT_TOT_TURNO.isin (['ELEITO POR MÉDIA','ELEITO POR QP'])]
#informações do dataframe df_consulta_eleitos
```

```
df_consulta_eleitos.info()

#Contagem das opções da coluna DS_GENERO
genero_candidatos = Counter(df_consulta['DS_GENERO'])
genero_candidatos

#Plotagem das informações de gênero dos candidatos
plt.style.use('default')
plt.pie(genero_candidatos.values(), labels = genero_candidatos.keys(),
      autopct = '%1.1f%%', textprops={'fontsize':'16'})
plt.axis("image")
plt.title("GÊNERO DOS CANDIDATOS",fontsize = 18)
plt.tight_layout()
plt.legend()
plt.show()

#Plotagem das informações de gênero dos candidatos eleitos
plt.style.use('fivethirtyeight')
plt.pie(genero_candidato_eleitos.values(), labels = genero_candidato_eleitos.keys(),
      autopct = '%1.1f%%', textprops={'fontsize':'16'})
plt.axis("image")
plt.title("GÊNERO DOS CANDIDATOS ELEITOS",fontsize = 18)
plt.tight_layout()
plt.legend()
plt.show()

#Descrição estatística da idade dos candidatos
df_consulta['NR_IDADE_DATA_POSSE'].describe()

#Descrição estatística da idade dos candidatos eleitos
df_consulta_eleitos['NR_IDADE_DATA_POSSE'].describe()
#Plotagem de histograma com as idades dos candidatos
```

```
df_consulta.NR_IDADE_DATA_POSSE.hist(bins=20)
plt.style.use('seaborn-pastel')
plt.xlabel('Idade')
plt.ylabel('Números de candidatos')
plt.title("Idade na possse")

#Plotagem de histograma com as idades dos candidatos eleitos
df_consulta_eleitos.NR_IDADE_DATA_POSSE.hist(bins=20)
plt.style.use('default')
plt.xlabel('Idade')
plt.ylabel('Números de candidatos eleitos')
plt.title("Idade na possse - Eleitos")

#Contagem das opções da coluna DS_GRAU_INSTRUCAO dos candidatos
escolaridade_candidatos = Counter(df_consulta['DS_GRAU_INSTRUCAO'])
escolaridade_candidatos

#Identificação dos percentuais de escolaridade dos candidatos
n_cadidatos=sum(escolaridade_candidatos.values())
for x, y in escolaridade_candidatos.items():
a = y/n_cadidatos*100
        print(str(x) + ': ' +'\n'+ str(round(a,2)) + '%' + '\n')

#Criação de listas para construção do gráfico de escolaridade
lista_escolaridade_candidatos_lables = []
lista_escolaridade_candidatos_valores = []
for x, y in escolaridade_candidatos.items():
   lista_escolaridade_candidatos_lables.append(x)
   lista_escolaridade_candidatos_valores.append(y)

#Verificação da lista criada
```

```
lista_escolaridade_candidatos_lables

#Verificação da lista criada com valores
lista_escolaridade_candidatos_valores

#Plotagem do gráfico de escolaridade dos candidatos
plt.style.use('seaborn-pastel')
plt.barh(lista_escolaridade_candidatos_lables, lista_escolaridade_candidatos_valores)
plt.ylabel('escolaridade')
plt.xlabel('número de candidatos')
plt.title('Candidatos por escolaridade')
plt.show()

#Contagem das opções da coluna DS_GRAU_INSTRUCAO dos candidatos eleitos
escolaridade_candidatos_eleitos = Counter(df_consulta_eleitos['DS_GRAU_INSTRUCAO'])
escolaridade_candidatos_eleitos

#Identificação dos percentuais de escolaridade dos candidatos eleitos
n_eleitos=sum(escolaridade_candidatos_eleitos.values())
for x, y in escolaridade_candidatos_eleitos.items():
    a = y/n_eleitos*100
    print(str(x) + ': ' +'\n'+ str(round(a,2)) + '%' + '\n')

#Criação de listas para construção do gráfico de escolaridade dos candidatos eleitos
lista_escolaridade_candidatos_eleitos_lables = []
lista_escolaridade_candidatos_eleitos_valores = []
for x, y in escolaridade_candidatos_eleitos.items():
    lista_escolaridade_candidatos_eleitos_lables.append(x)
    lista_escolaridade_candidatos_eleitos_valores.append(y)

#Verificação da lista criada
```

```
lista_escolaridade_candidatos_eleitos_lables

#Verificação da lista criada com valores
lista_escolaridade_candidatos_eleitos_valores

#Plotagem do gráfico de escolaridade dos candidatos eleitos
plt.style.use('ggplot')
plt.barh(lista_escolaridade_candidatos_eleitos_lables,
lista_escolaridade_candidatos_eleitos_valores)
plt.ylabel('escolaridade')
plt.xlabel('número de candidatos')
plt.title('Candidatos eleitos por escolaridade')
plt.show()

#Contagem das opções da coluna DS_COR_RACA dos candidatos
raca_candidatos = Counter(df_consulta['DS_COR_RACA'])
raca_candidatos

#Identificação dos percentuais de raça dos candidatos
n_raca = sum(raca_candidatos.values())
for x, y in raca_candidatos.items():
    a = y/n_raca*100
    print(str(x) + ': ' +'\n'+ str(round(a,2)) + '%' + '\n')

#Plotagem do gráfico de raça dos candidatos
plt.style.use('seaborn-pastel')
label_size = 6
plt.rcParams['xtick.labelsize'] = label_size
plt.bar(raca_candidatos.keys(), raca_candidatos.values())
plt.ylabel('Número de candidatos')
plt.xlabel('Raça')
plt.title('Candidatos por raça')
```

```
plt.show()

#Contagem das opções da coluna DS_COR_RACA dos candidatos eleitos
raca_candidatos_eleitos = Counter(df_consulta_eleitos['DS_COR_RACA'])
raca_candidatos_eleitos

#Identificação dos percentuais de raça dos candidatos eleitos
n_raca_eleitos = sum(raca_candidatos_eleitos.values())
for x, y in raca_candidatos_eleitos.items():
    a = y/n_raca_eleitos*100
    print(str(x) + ': ' +'\n'+ str(round(a,2)) + '%' + '\n')

#Plotagem do gráfico de raça dos candidatos eleitos
plt.style.use('ggplot')
label_size = 6
plt.rcParams['xtick.labelsize'] = label_size
plt.bar(raca_candidatos_eleitos.keys(), raca_candidatos_eleitos.values())
plt.ylabel('Número de candidatos eleitos')
plt.xlabel('Raça')
plt.title('Candidatos eleitos por raça')
plt.show()

#Contagem das opções da coluna DS_ESTADO_CIVIL dos candidatos
estado_civil_candidatos = Counter(df_consulta['DS_ESTADO_CIVIL'])
estado_civil_candidatos

#Identificação dos percentuais de estado civil dos candidatos
n_estado_civil=sum(estado_civil_candidatos.values())
for x, y in estado_civil_candidatos.items():
    a = y/n_estado_civil*100
    print(str(x) + ':' + '\n' + str (round(a,2)) + '%'+'\n')
```

#Criação de listas para construção do gráfico de estado civil dos candidatos

```
lista_estado_civil_candidatos_labels = []
lista_estado_civil_candidatos_valores = []
for x, y in estado_civil_candidatos.items():
    lista_estado_civil_candidatos_labels.append(x)
    lista_estado_civil_candidatos_valores.append(y)
lista_estado_civil_candidatos_labels
```

#Verificação da lista criada

```
lista_estado_civil_candidatos_labels
```

#Verificação da lista criada com valores

```
lista_estado_civil_candidatos_valores
```

#Plotagem do gráfico de estado civil dos candidatos

```
plt.style.use('seaborn-pastel')
lt.barh(lista_estado_civil_candidatos_labels, lista_estado_civil_candidatos_valores),
plt.ylabel('Estado Civil')
plt.xlabel('Número de canditados')
plt.title('CANDIDATOS POR ESTADO CIVIL')
plt.show()
```

#Contagem das opções da coluna DS_ESTADO_CIVIL dos candidatos eleitos

```
estado_civil_candidatos_eleitos = Counter(df_consulta_eleitos['DS_ESTADO_CIVIL'])
estado_civil_candidatos_eleitos
```

#Identificação dos percentuais de estado civil dos candidatos eleitos

```
n_estado_civil_eleitos = sum(estado_civil_candidatos_eleitos.values())
for x, y in estado_civil_candidatos_eleitos.items():
    a = y/n_estado_civil_eleitos*100
    print(str(x) + ':' + '\n' + str (round(a,2)) + '%'+'\n')
```

```
#Criação de listas para construção do gráfico de estado civil dos candidatos eleitos
lista_estado_civil_candidatos_eleitos_labels = []
lista_estado_civil_candidatos_eleitos_valores = []
for x, y in estado_civil_candidatos_eleitos.items():
    lista_estado_civil_candidatos_eleitos_labels.append(x)
    lista_estado_civil_candidatos_eleitos_valores.append(y)
lista_estado_civil_candidatos_eleitos_labels

#Verificação da lista criada com valores
lista_estado_civil_candidatos_eleitos_valores

#Plotagem do gráfico de estado civil dos candidatos eleitos
plt.style.use('ggplot')
plt.barh(lista_estado_civil_candidatos_eleitos_labels,
lista_estado_civil_candidatos_eleitos_valores),
plt.ylabel('Estado Civil')
plt.xlabel('Número de canditados')
plt.title('CANDIDATOS POR ESTADO CIVIL - ELEITOS')
plt.show()

#descrição estatística dos valores dos bens dos candidatos
df_bem['VR_BEM_CANDIDATO'].describe().astype('int')

#Plotagem do histograma dos valores dos bens dos candidatos
df_bem.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel('Valor')
plt.ylabel('Número de bens')
plt.title('Bens candidatos - Valores')
plt.show()

#Quantidade de bens com valores acima de R$ 10.000.000,00
```

```
df_bem.loc[(df_bem['VR_BEM_CANDIDATO'] > 10000000)].info()

#Impressão dos detalhes dos bens com valores acima de R$ 10.000.000,00
for index, row in df_bem.loc[(df_bem['VR_BEM_CANDIDATO'] > 10000000)].iterrows():
    print(index)
    print('Candidato: ' + str(row['SQ_CANDIDATO']))
    print('Tipo de bem: ' + str(row['DS_TIPO_BEM_CANDIDATO']))
    print('Descrição: ' + str(row['DS_BEM_CANDIDATO']))
    print('Valor: ' + str(row['VR_BEM_CANDIDATO']) + '\n')

#Verificar valor nulo
df_bem.loc[8425,'DS_BEM_CANDIDATO']

#Criação de um dataframe com bens abaixo de R$ 1.000.000,00
df_bem = df_bem.loc[(df_bem['VR_BEM_CANDIDATO'] < 1000000)]

#Plotagem do histograma dos valores dos bens dos candidatos até R$ 1.000.000,00
df_bem1.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel('Valor')
plt.ylabel('Número de canditados')
plt.title('TOTAL DE BENS DOS CANDIDOTOS')
plt.show()

#Definição do índice do dataframe df_consulta
df_consulta = df_consulta.set_index('SQ_CANDIDATO')

#Informações do dataframe df_consulta após definição do índice
df_consulta.info()

#Criação de um dataframe consolidado a partir da junção de df_consulta e df_bem
df_consolidado = df_consulta.join(df_bem)
#Informações do dataframe df_consolidado
```

```
df_consolidado.info()

#Verificação dos valores nulos no dataframe df_consolidado
df_consolidado.isnull().sum()

#Preenchimento dos campos nulos no dataframe df_consolidado
df_consolidado.fillna(0, inplace = True)

#Informações do dataframe df_consolidado após preenchimento dos valores nulos
df_consolidado.info()

#Criação de um dataframe consolidado de candidatos eleitos
df_consolidado_eleitos = df_consolidado[df_consulta.DS_SIT_TOT_TURNO.isin(['ELEITO POR MÉDIA', 'ELEITO POR QP'])]

#Informações do dataframe consolidado de candidatos eleitos
df_consolidado_eleitos.info()

#Descrição estatística dos bens dos candidatos do dataframe consolidado
df_consolidado['VR_BEM_CANDIDATO'].describe().astype('int')

#Descrição estatística dos bens dos candidatos eleitos do dataframe consolidado
df_consolidado_eleitos['VR_BEM_CANDIDATO'].describe().astype('int')

#Plotagem de histograma dos valores das despesas dos candidatos
df_deputados_final.VR_PAGTO_DESPESA.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel('Valor')
plt.ylabel('Número de candidatos')
plt.title('Total de despesas dos candidatos')
plt.show()
#Plotagem de histograma dos valores das despesas dos candidatos eleitos
```

```
df_deputados_final.eleitos.VR_PAGTO_DESPESA.hist(bins=100)
plt.style.use('ggplot')
plt.xlabel('Valor')
plt.ylabel('Número de candidatos eleitos')
plt.title('Total de despesas dos candidatos eleitos')
plt.show()

#Identificação dos valores da coluna DS_GENERO
df_deputados_final['DS_GENERO'].unique()

#Identificação dos valores da coluna DS_GRAU_INSTRUCAO
df_deputados_final['DS_GRAU_INSTRUCAO'].unique()

#Identificação dos valores da coluna DS_ESTADO_CIVIL
df_deputados_final['DS_ESTADO_CIVIL'].unique()

#Identificação dos valores da coluna DS_COR_RACA
df_deputados_final['DS_COR_RACA'].unique()

#Identificação dos valores da coluna DS_SIT_TOT_TURNO
df_deputados_final['DS_SIT_TOT_TURNO'].unique()

#Transformação de valores categóricos em valores inteiros
ajuste_ensino = {'LÊ E ESCREVE': 1, 'ENSINO FUNDAMENTAL INCOMPLETO': 2,
'ENSINO FUNDAMENTAL COMPLETO': 3, 'ENSINO MÉDIO INCOMPLETO': 4,
'ENSINO MÉDIO COMPLETO': 5, 'SUPERIOR INCOMPLETO': 6, 'SUPERIOR COMPLETO': 7}
df_deputados_final['DS_GRAU_INSTRUCAO']                                    =
df_deputados_final['DS_GRAU_INSTRUCAO'].map(ajuste_ensino)

#Verificação da alteração na coluna DS_GRAU_INSTRUCAO
df_deputados_final['DS_GRAU_INSTRUCAO'].unique()
#Transformação de valores categóricos em valores inteiros
```

```
ajuste_eleito = {'ELEITO POR QP': 1, 'SUPLENTE': 0, 'NÃO ELEITO': 0, 'ELEITO POR MÉDIA': 1}
df_deputados_final['DS_SIT_TOT_TURNO'] =
df_deputados_final['DS_SIT_TOT_TURNO'].map(ajuste_eleito)

#Verificação da alteração na coluna DS_SIT_TOT_TURNO
df_deputados_final['DS_SIT_TOT_TURNO'].unique()

#Transformação de valores categóricos em valores inteiros
df_deputados_final = pd.get_dummies(df_deputados_final[['NR_IDADE_DATA_POSSE',
'DS_GENERO', 'DS_GRAU_INSTRUCAO', 'DS_ESTADO_CIVIL', 'DS_COR_RACA',
'DS_SIT_TOT_TURNO', 'VR_BEM_CANDIDATO', 'VR_PAGTO_DESPESA' ]])

#Informações do dataframe df_deputados_final após os ajustes feitos
df_deputados_final.info()

#Verificação dos valores da coluna DS_GENERO_FEMININO
df_deputados_final['DS_GENERO_FEMININO'].unique()

#Contagem dos candidatos eleitos e não eleitos
df_deputados_final['DS_SIT_TOT_TURNO'].value_counts()

#Importação da função resample da biblioteca sklearn
from sklearn.utils import resample

#Criação de um dataframe apenas com candidatos não eleitos
df_deputados_final_majority = df_deputados_final[df_deputados_final.DS_SIT_TOT_TURNO
== 0 ]
df_deputados_final_majority.info()

#Criação de um dataframe apenas com candidatos eleitos
```

```
df_deputados_final_minority = df_deputados_final[df_deputados_final.DS_SIT_TOT_TURNO == 1]
df_deputados_final_minority.info()

#Ajuste no número de entradas do dataframe de candidatos eleitos
df_deputados_final_minority_upsampled = resample(df_deputados_final_minority, replace=True, n_samples=57999,
random_state=123)

#Informações do dataframe ajustado
df_deputados_final_minority_upsampled.info()

#Concatenação dos dataframes de candidatos eleitos e não eleitos
df_deputados_final_upsampled = pd.concat([df_deputados_final_majority,
                                df_deputados_final_minority_upsampled])

#Informações do dataframe ajustado
df_deputados_final_upsampled.info()

#Contagem dos valores de candidatos eleitos e não eleitos
df_deputados_final_upsampled['DS_SIT_TOT_TURNO'].value_counts()

#Importação da função train_test_split
from sklearn.model_selection import train_test_split

#Divisão para as bases de treinamento
X_train = df_deputados_final_upsampled.drop(['DS_SIT_TOT_TURNO'], axis = 1)
y_train = df_deputados_final_upsampled.DS_SIT_TOT_TURNO
X_train.info()
type(y_train)

#Informações do dataframe com os dados para treinamento
```

```
X_train.info()

#Tipo da serie y_train
type(y_train)

#Criação das bases de teste e treinamento
xtreinamento, xteste, ytreinamento, yteste = train_test_split(X_train, y_train, random_state = 0)

#Informação do dataframe de treinamento
xtreinamento.info()

#Informação do dataframe de teste
xteste.info()

#Contagem dos resultados para treinamento
ytreinamento.count()

#Contagem dos resultados para teste
yteste.count()

#Importação das funções para as medidas de avaliação dos algoritmos
from sklearn.metrics import accuracy_score, classification_report

#Criação do modelo utilizando a Árvore de decisão
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
deputados_tree = DecisionTreeClassifier()
deputados_tree = decision_tree.fit(xtreinamento, ytreinamento)
print("Acurácia: ", deputados_tree.score(xtreinamento, ytreinamento))
Train_predict = deputados_tree.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, Train_predict))
```

```
print(classification_report(yteste, Train_predict))

#Criação do modelo utilizando a Regressão Logística
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr = lr.fit(xtreinamento, ytreinamento)
print("Acurácia: ", lr.score(xtreinamento, ytreinamento))
tp_lr = lr.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_lr))
print(classification_report(yteste, tp_lr))

#Criação do modelo utilizando Gradiente Descendente
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()
sgd = sgd.fit(xtreinamento, ytreinamento)
print("Acurácia: ", sgd.score(xtreinamento, ytreinamento))
tp_sgd = sgd.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_sgd))
print(classification_report(yteste, tp_sgd))

#Criação do modelo utilizando Randon Forest
from sklearn.ensemble import RandomForestClassifier
rfm = RandomForestClassifier()
rfm = rfm.fit(xtreinamento, ytreinamento)
print("Acurácia: ", rfm.score(xtreinamento, ytreinamento))
tp_rfm = rfm.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_rfm))
print(Classifiction_report(yteste, tp_rfm))
```