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

Henrique Moreira de Oliveira

ANÁLISE DE PREVISIBILIDADE DO RESULTADO DAS ELEIÇÕES DE 2018 PARA OS CARGOS DE DEPUTADOS ESTADUAIS E FEDERAIS

Belo Horizonte
2021

Henrique Moreira de Oliveira

ANÁLISE DE PREVISIBILIDADE DO RESULTADO DAS ELEIÇÕES DE 2018 PARA OS CARGOS DE DEPUTADOS ESTADUAIS E FEDERAIS

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

Belo Horizonte

2021

SUMÁRIO

1. Introdução

1.1. Contextualização

A história do voto no Brasil e, consequentemente, a capacidade de votar e ser votado remontam ao ano de 1532, início do período colonial, quando da eleição para definição dos membros do Conselho Municipal de São Vicente. Nessa eleição os que poderiam votar eram os nobres de linhagem, os senhores de engenho, e os membros da alta burocracia militar, a esses se acrescentando os homens novos, burgueses enriquecidos pelo comércio. Esses eleitores escolhiam, indiretamente, quem exerceria determinados cargos nas vilas nos 3 anos seguintes.

Com a vinda da família real portuguesa para o Brasil em 1808 e a consequente elevação do Brasil a Reino do Império Português, oficialmente denominado de Reino Unido de Portugal, Brasil e Algarves, os brasileiros foram convocados a escolher os deputados das cortes de Lisboa em 1821, ainda com a corte no Brasil e de forma indireta, já que os eleitores nomeavam os compromissários, que escolhiam os eleitores de paróquia, que designavam os eleitores de comarca que, por sua vez, elegiam os deputados.

Após a declaração de independência em 1822, o Brasil passou a ter eleições para as, eventuais, Assembleias Constituintes e para as Assembleias Legislativas. Essas eleições continuavam sendo, durante a maior parte do período, indiretas e restritas, tanto para se poder votar como para ser votado. Faziam parte dessas restrições o sexo do eleitor (apenas homens poderiam votar), idade, renda anual, origem, religião etc. Destaca-se que, até 1881, pessoas analfabetas podiam votar e, nesse ano, com a publicação da Lei Saraiva, esse direito deixou de existir, reduzindo drasticamente o número de eleitores no país, tanto que, em 1886, votaram nas eleições parlamentares pouco mais de 100 mil eleitores, ou 0,8% da população total.

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 instituição do primeiro Código Eleitoral, em 1932, trouxe importantes avanços nos direitos políticos com a introdução do voto facultativo feminino, a fixação do voto secreto, a instituição do sistema representativo proporcional e a regulação em todo o país das eleições federais, estaduais e municipais. Contudo, os analfabetos continuavam impedidos de votar.

Entre os anos de 1945 e 1964, novos avanços aconteceram com a exclusividade dos partidos políticos na apresentação das candidaturas, a redução da idade mínima para votar, de 21 para 18 anos, e a obrigatoriedade do voto em sufrágio universal, ampliando, assim, o número de cidadãos que participavam do processo eleitoral que ainda era baixo, já que apenas 15% da população compareceu às urnas em 1945.

Com a instituição do regime militar em 1964, o voto passou a ter funções básicas: legitimar as decisões do governo e servir como uma espécie de laboratório eleitoral, no qual a população podia exercer, controladamente, o direito de votar. Entre 1966 e 1982 houve um aumento de 163% no eleitorado brasileiro, devido, entre outras coisas, ao aumento vegetativo da população e das sanções impostas a quem não se alistasse. Esse aumento, porém, não teve efeitos práticos. pois houve um alto número de abstenções, votos nulos e brancos, muito pelo descrédito do processo eleitoral no período.

A partir de meados da década de 1970, foi iniciada uma abertura lenta e gradual do regime militar que, aliada à vitória da oposição nas eleições para governador em 1982 e ao esgotamento do regime, culminaram na eleição do primeiro presidente civil desde 1964, mesmo de forma indireta.

Após o movimento “Diretas já!” e a elaboração da Constituição de 1988, conhecida como a Constituição cidadã, houve significativos avanços no processo eleitoral brasileiro e em seu eleitorado, com a diminuição da idade para a faculdade do voto e a permissão de voto ao analfabeto, modelo adotado até hoje.

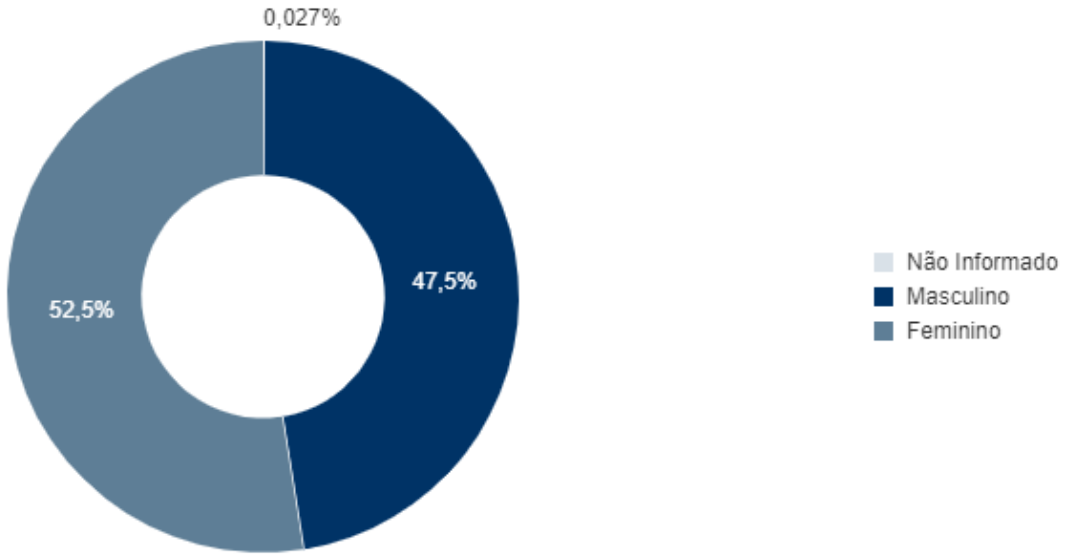
Atualmente, após todas as evoluções, tem-se mais de 147 milhões de eleitores aptos a votar de acordo com o site do Tribunal Superior Eleitoral (https://www.tse.jus.br/eleicoes/estatisticas/estatisticas-eleitorais).

1.2. O problema proposto

Para se definir o problema proposto, inicialmente será analisado o perfil do eleitorado brasileiro. O sítio eletrônico do Tribunal Superior Eleitoral, mencionado anteriormente e de onde os gráficos e informações a seguir foram extraídas, traz alguns dados das características desse eleitorado, mais especificamente sobre seu gênero, faixa etária, estado civil e grau de instrução.

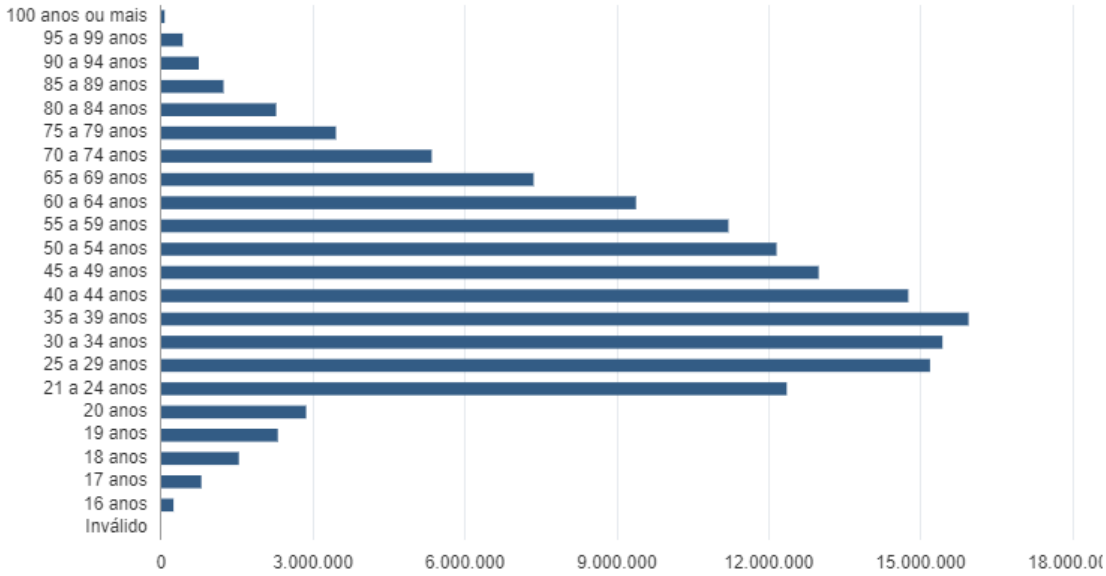
Os dados serão, primeiramente, apresentados e posteriormente analisados:

- Gênero



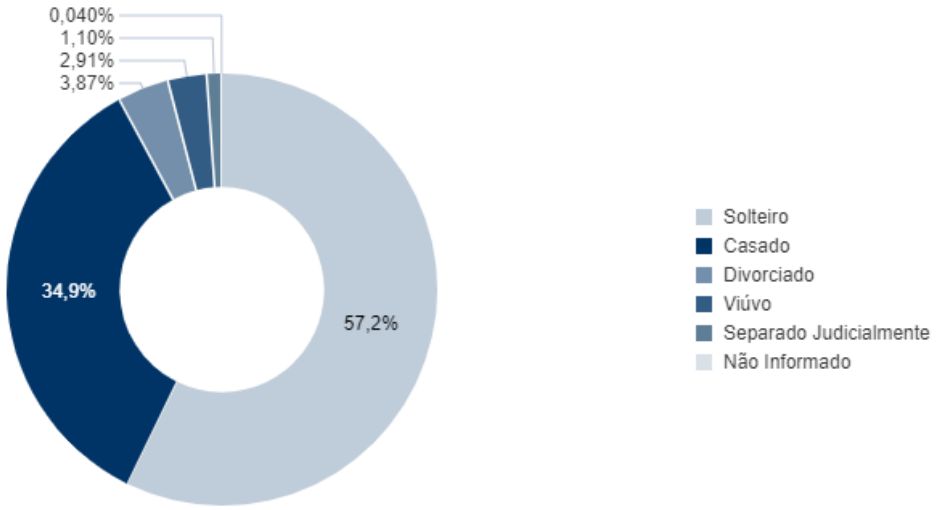
Gênero	Quantitativo de eleitorado	Porcentagem (%)
Não Informado	40.457	0,03%
Masculino	70.228.457	47,48%
Feminino	77.649.569	52,49%

- Faixa etária



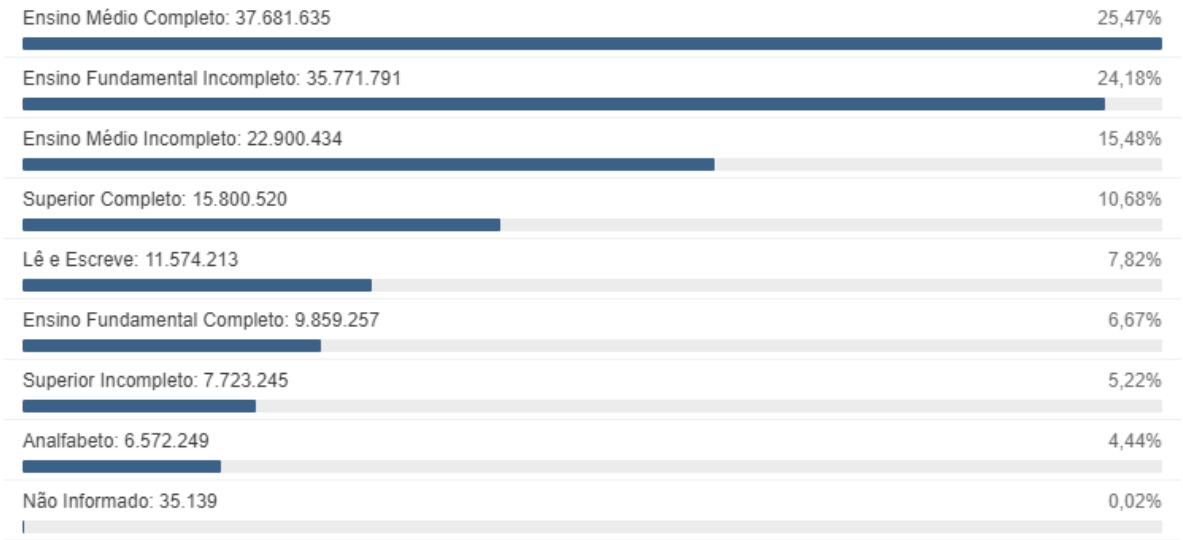
Faixa etária	Quantitativo de eleitorado	Porcentagem (%)
100 anos ou mais	65.589	0,04%
95 a 99 anos	423.377	0,29%
90 a 94 anos	741.147	0,50%
85 a 89 anos	1.227.487	0,83%
80 a 84 anos	2.266.484	1,53%
75 a 79 anos	3.446.550	2,33%
70 a 74 anos	5.337.454	3,61%
65 a 69 anos	7.350.599	4,97%
60 a 64 anos	9.372.283	6,34%
55 a 59 anos	11.198.821	7,57%
50 a 54 anos	12.147.351	8,21%
45 a 49 anos	12.979.183	8,77%
40 a 44 anos	14.748.141	9,97%
35 a 39 anos	15.938.174	10,77%
30 a 34 anos	15.423.585	10,43%
25 a 29 anos	15.178.770	10,26%
21 a 24 anos	12.349.829	8,35%
20 anos	2.859.883	1,93%
19 anos	2.299.649	1,55%
18 anos	1.531.395	1,04%
17 anos	790.602	0,53%
16 anos	239.961	0,16%
Inválido	2.169	0,00%

- Estado civil



Estado civil	Quantitativo de eleitorado	Porcentagem (%)
Solteiro	84.630.900	57,21%
Casado	51.575.693	34,87%
Divorciado	5.719.757	3,87%
Viúvo	4.309.584	2,91%
Separado Judicialmente	1.623.642	1,10%
Não Informado	58.907	0,04%

- Grau de instrução



Grau de instrução	Quantitativo de eleitorado	Porcentagem (%)
Ensino Médio Completo	37.681.635	25,47%
Ensino Fundamental Incompleto	35.771.791	24,18%
Ensino Médio Incompleto	22.900.434	15,48%
Superior Completo	15.800.520	10,68%
Lê e Escreve	11.574.213	7,82%
Ensino Fundamental Completo	9.859.257	6,67%
Superior Incompleto	7.723.245	5,22%
Analfabeto	6.572.249	4,44%
Não Informado	35.139	0,02%

Constata-se que a maioria do eleitorado brasileiro é do gênero feminino (52,49%), tem menos de 50 anos (63,76%), é solteiro (57,21%) e não possui Ensino Superior Completo (89,32%). Contudo, ao se analisar, por exemplo, o perfil dos candidatos eleitos para o cargo de deputado federal que, segundo à Constituição Federal em seu artigo 45, são os representantes do povo, percebe-se que o seu perfil é bem diferente do que o do eleitorado, pois dos 513 deputados eleitos, 436 são homens (85%), 355 são casados (69,2%) e 415 possuem ensino superior completo (80,8%). Apenas a idade dos deputados se aproxima da do eleitorado, já que 53% deles possuem menos de 50 anos de idade. Outro ponto que merece destaque é que as 3 profissões mais comuns entre os eleitos, 42% dos deputados, são empresário, advogado e médico, atividades que representam altos ganhos financeiros, indicando que eles possuem alto poder aquisitivo, mesmo antes da eleição. Essas informações foram extraídas do sítio eletrônico da Câmara: https://www.camara.leg.br/internet/agencia/infograficos-html5/composicaocamara2019/index.html

Essa situação mostra que o perfil dos representantes do povo no Congresso Nacional é bem diferente do eleitorado que representam. Apesar de a representatividade feminina, por exemplo, vir aumentando desde 1998, de 5,7% para 15%, ela ainda é muito distante do perfil da população.

Essas diferenças entre o perfil do eleitorado e dos candidatos eleitos para deputado federal mostram que, apesar de evoluções no eleitorado, o perfil dos representantes do povo ainda segue o padrão histórico, ou seja, homens brancos com elevado grau de escolaridade. Assim, o problema que se propõe é se seria possível expandir essa análise também para os cargos de deputados estaduais e

prever, com base no gênero, estado civil, idade, raça, escolaridade e poder aquisitivo dos candidatos, o resultado das eleições para os cargos de deputado federal e estadual na eleição de 2018, que é a última realizada até o momento.

2. Coleta de Dados

Para o tratamento do problema proposto, foram utilizados quatro *datasets* (conjunto de dados) relativos às eleições de 2018, extraídos em 17/09/2020 do repositório de dados eleitorais do Tribunal Superior Eleitoral.

O primeiro *dataset*, "consulta_cand_2018_BRASIL", disponível em https://cdn.tse.jus.br/estatistica/sead/odsele/consulta_cand/consulta_cand_2018.zip, traz as informações cadastrais de cada candidato, bem como a situação de sua candidatura e seu resultado na eleição. Esse *dataset* possui os seguintes campos:

Variável	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.

Variável	Descrição
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. Nas eleições ordinárias nacionais são eleitos os cargos de Presidente, Governadores, Deputados (Federais e Estaduais) e Senadores. Nas eleições ordinárias municipais são eleitos os cargos de Prefeito e Vereadores. As eleições suplementares são aquelas que não têm periodicidade pré-determinada ou definida e ocorrem quando, eventualmente, se fizerem necessárias. As consultas populares ocorrem sempre que a população é convocada a opinar diretamente sobre um assunto específico e importante. Ela pode ser realizada de duas formas: plebiscito (quando o cidadão opina previamente sobre a possível criação de uma lei) e referendo (quando uma lei aprovada por um órgão legislativo é submetida à aceitação ou não dos eleitores).
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

Variável	Descrição
	candidato que tenha alcançado a maioria simples dos votos.
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.

Variável	Descrição
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'

Variável	Descrição
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)

Variável	Descrição
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_2018_BRASIL”, disponível em https://cdn.tse.jus.br/estatistica/sead/odsele/bem_candidato/bem_candidato_2018.zip, traz a declaração de bens de cada candidato para a eleição em questão e possui os seguintes campos:

Variável	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_2018_BRASIL.csv” e o quarto, “despesas_pagas_candidatos_2018_BRASIL.csv”, foram obtidos no link https://cdn.tse.jus.br/estatistica/sead/odsele/prestacao_contas/prestacao_de_contas_eleitorais_candidatos_2018.zip.

O *dataset* “despesas_contratadas_candidatos_2018_BRASIL.csv” traz as despesas contratadas pelos candidatos na eleição e possui a seguinte estrutura:

Variável	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).

Variável	Descrição
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.

Variável	Descrição
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_2018_BRASIL.csv” traz as despesas efetivamente pagas pelos candidatos na eleição e possui a seguinte estrutura:

Variável	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.

Variável	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 despesa parcelada paga, declarada pelo prestador de contas.

Variável	Descrição
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

O processamento e o tratamento dos dados foram feitos utilizando a linguagem Python, versão 3.7.0, no ambiente Jupyter Notebook, versão 5.6.0. Dentro da linguagem, utilizou-se a biblioteca “pandas” que é uma poderosa ferramenta para tratamento e análise de dados.

O primeiro passo é a importação da biblioteca “pandas”.

```
import pandas as pd
```

Em seguida, deve-se fazer a leitura e tratamento dos *datasets*, que, nesse caso, será feita individualmente para cada um deles. O primeiro *dataset* que será processado será o *dataset* “consulta_cand_2018_BRASIL.csv” que será importado por meio do comando “pd.read.csv”, seguindo os parâmetros estabelecidos no arquivo que traz as características do *dataset*.

```
df_consulta = pd.read_csv("consulta_cand_2018_BRASIL.csv",
                          encoding = "Latin 1", sep = ";", decimal = ',')
```

O uso desse comando cria um *dataframe*, que é uma estrutura bidimensional de dados similar a uma planilha. Para se obter informações do *dataframe*, pode-se utilizar a função “info()” que, no caso específico, mostrou que o *dataframe* possui 29.145 entradas, divididas nas 58 colunas apresentadas anteriormente.

```
df_consulta.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29145 entries, 0 to 29144
Data columns (total 58 columns):
```

O *dataframe* traz informações de candidatos a todos os cargos da eleição em questão, mas, como o objetivo do estudo é avaliar o resultado para os cargos de deputado federal e estadual, foi necessário, antes de selecionar apenas as entradas de interesse, identificar quais eram as descrições dos cargos que constavam no arquivo, o que foi feito por meio da função “unique()”, que mostra quais são as opções existentes na coluna “DS_CARGO”, que traz o cargo ao qual o candidato concorre na eleição.

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

```
array(['DEPUTADO ESTADUAL', 'DEPUTADO FEDERAL', 'VICE-GOVERNADOR',
       '2º SUPLENTE', 'SENADOR', '1º SUPLENTE', 'DEPUTADO DISTRITAL',
       'GOVERNADOR', 'PRESIDENTE', 'VICE-PRESIDENTE'], dtype=object)
```

Sabendo as opções existentes, o próximo passo é filtrar apenas as informações que são de interesse do estudo, nesse caso os cargos ‘DEPUTADO ESTADUAL’ e ‘DEPUTADO FEDERAL’. Para esse procedimento utilizou-se a função “isin()” na coluna ‘DS_CARGO’, que retorna apenas os valores desejados.

```
df_consulta = df_consulta[df_consulta.DS_CARGO.isin(['DEPUTADO ESTADUAL',
                                                     'DEPUTADO FEDERAL'])]
```

```
df_consulta.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26529 entries, 0 to 29144
Data columns (total 58 columns):
```

Após esse comando é possível perceber que o número de entradas agora é de 26.529, menor do que o original.

Outro filtro necessário é que sejam consideradas apenas candidaturas aptas e, para isso, os passos anteriores serão repetidos, mas agora para a coluna ‘DS_SITUACAO_CANDIDATURA’.

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

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

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

```
df_consulta.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23831 entries, 0 to 29144
Data columns (total 58 columns):
```

Como é possível perceber, o *dataframe* possui agora 23.831 entradas, ainda divididas em 58 colunas. Muitas dessas colunas, porém, não trazem informações relevantes para o estudo em questão. Dessa forma, após analisar a descrição e relevância de cada uma das colunas disponíveis, optou-se por manter as informações da tabela 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.
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.

Interessante perceber que uma informação potencialmente útil para o estudo não foi considerada, a informação da coluna ‘VR_DESPESA_MAX_CAMPANHA’. Isso se deu pelo fato de que essa coluna, apesar de descrita na documentação do *dataset*, não existe, mas sim a coluna ‘NR_DESPESA_MAX_CAMPANHA’ que traz apenas valores ‘0’ e ‘-1’ e por isso também foi desconsiderada.

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

```
array([ 0, -1], dtype=int64)
```

Para selecionar apenas as colunas desejadas, 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']]
```

```
df_consulta.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23831 entries, 0 to 29144
Data columns (total 7 columns):
SQ_CANDIDATO           23831 non-null int64
NR_IDADE_DATA_POSSE    23831 non-null int64
DS_GENERO              23831 non-null object
DS_GRAU_INSTRUCAO      23831 non-null object
DS_ESTADO_CIVIL        23831 non-null object
DS_COR_RACA            23831 non-null object
DS_SIT_TOT_TURNO       23831 non-null object
dtypes: int64(2), object(5)
memory usage: 1.5+ MB
```

Percebe-se que agora há apenas as 7 colunas escolhidas, mas ainda com as 23.831 entradas filtradas anteriormente. Para verificar se há dados nulos no *dataframe*, utilizou-se a a função “isnull()”, que faz essa análise, em conjunto com a função “sum()”, que, nesse caso, soma os valores encontrados na função anterior. Após esse comando, verificou-se que nenhum deles é nulo, razão pela qual não há necessidade de maiores tratamentos no momento.

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

```
SQ_CANDIDATO           0
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
dtype: int64
```

A leitura do segundo *dataset*, “bem_candidato_2018_BRASIL”, é feita da mesma forma do primeiro, também seguindo os parâmetros do documento que o especifica.

```
df_bem = pd.read_csv("bem_candidato_2018_BRASIL.csv", encoding = "Latin 1", sep = ";", decimal = ',')
```

Esse novo *dataframe* possui 93.271 entradas divididas em 19 colunas, conforme demonstrado após a utilização do comando “info()”

```
df_bem.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 93271 entries, 0 to 93270
Data columns (total 19 columns):
```

Diferentemente do arquivo anterior, não há o que se falar em filtros nesse *dataframe*, já que não se tem informações relacionadas à situação de candidatura, nem ao cargo ao qual o candidato está concorrendo. Por essa razão, nenhuma entrada será retirada e o tratamento, igual descrito anteriormente, será apenas nas colunas selecionadas, que serão as seguintes:

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.

```
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: 93271 entries, 0 to 93270
Data columns (total 4 columns):
SQ_CANDIDATO             93271 non-null int64
DS_TIPO_BEM_CANDIDATO    93271 non-null object
DS_BEM_CANDIDATO         93271 non-null object
VR_BEM_CANDIDATO         93271 non-null float64
dtypes: float64(1), int64(1), object(2)
memory usage: 2.8+ MB
```

Esse *dataframe* também não apresenta dados nulos que necessitem de tratamento.

```
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_2018_BRASIL.csv", será lido como das outras vezes por meio do seguinte comando:

```
df_despesas_contratadas = pd.read_csv("despesas_contratadas_candidatos_2018_BRASIL.csv",
                                      encoding = "Latin 1", sep = ";", decimal = ',')
```

Por se tratar de um arquivo com todas as despesas eleitorais contratadas, de todos os candidatos da eleição de 2018, o *dataframe* criado possui 1.711.062 entradas, divididas em 53 colunas.

```
df_despesas_contratadas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1711062 entries, 0 to 1711061
Data columns (total 53 columns):
```

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: 1711062 entries, 0 to 1711061
Data columns (total 4 columns):
SQ_DESPESA              int64
SQ_CANDIDATO            int64
DS_DESPESA              object
VR_DESPESA_CONTRATADA   float64
dtypes: float64(1), int64(2), object(1)
memory usage: 52.2+ 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_2018_BRASIL.csv", seguiu os mesmos passos anteriores apresentando 1.647.553 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_pagas = pd.read_csv("despesas_pagas_candidatos_2018_BRASIL.csv",
                                encoding = "Latin 1", sep = ";", decimal = ',')
```

```
df_despesas_pagas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1647553 entries, 0 to 1647552
Data columns (total 28 columns):
```

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

Variável	Descrição
SQ_DESPESA	Sequencial de identificação do registro da despesa declarada pelo prestador de contas.
VR_PAGTO_DESPESA	Valor pago da despesa em Reais (R$), informada pelo prestador de conta

Assim como nos demais *dataframes*, não foram identificados valores nulos:

```
df_despesas_pagas.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 realização da 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
```

Inicialmente será feita a análise dos dados dos *dataframes* de maneira individual e, em seguida, as junções necessárias para continuação dos estudos. Portanto, as análises serão iniciadas pelo *dataframe* que traz as informações cadastrais de cada candidato, chamado de “df_consulta”.

Como um dos objetivos é avaliar a discrepância de característica entre os candidatos que concorreram nas eleições e os que efetivamente foram eleitos,

sempre que possível as informações serão separadas nesses critérios e, para isso, a partir do *dataframe* “df_consulta” será criado um *dataframe* apenas com os candidatos eleitos, que será chamado de “df_consulta_eleitos”.

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

```
array(['SUPLENTE', 'NÃO ELEITO', 'ELEITO POR MÉDIA', 'ELEITO POR QP'],
      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: 1548 entries, 64 to 29139
Data columns (total 7 columns):
```

Percebe-se que diferentemente do *dataframe* original, que tinha 23.831 entradas, esse possui apenas 1.548, 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': 16378, 'FEMININO': 7453})
```

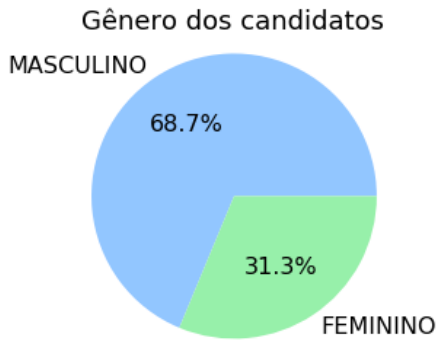
```
genero_candidatos_eleitos = Counter(df_consulta_eleitos['DS_GENERO'])
genero_candidatos_eleitos
```

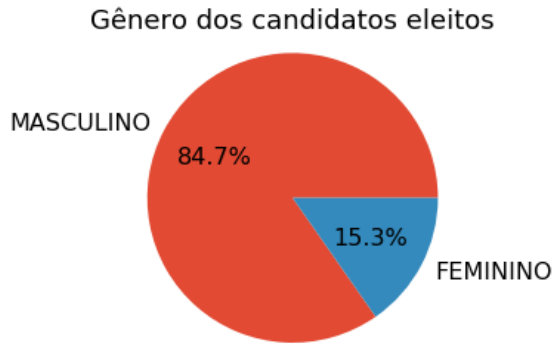
```
Counter({'MASCULINO': 1311, 'FEMININO': 237})
```

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('seaborn-pastel')
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.show()
```

```
plt.style.use('ggplot')
plt.pie(genero_candidatos_eleitos.values(), labels = genero_candidatos_eleitos.keys(),
        autopct = '%1.1f%%', textprops={'fontsize': 16})
plt.title("Gênero dos candidatos eleitos", fontsize=18)
plt.axis("image")
plt.show()
```





Analisando os gráficos pode-se perceber que o número de candidatos de cada gênero já não é proporcional ao eleitorado. Apesar do eleitorado masculino ser de 47,48%, o percentual de candidatos desse mesmo gênero é de 68,7%. Esse aumento proporcional se torna mais expressivo quando se analisa os candidatos eleitos, pois esse percentual salta para 84,7%, enquanto os candidatos eleitos do gênero feminino representam apenas 15,3%, sendo que ele representa 52,49% 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()
```

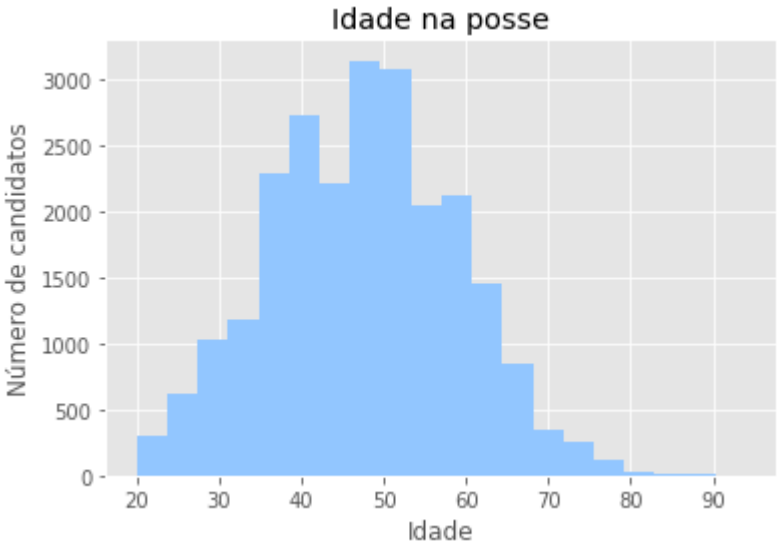
```
count    23831.000000
mean        47.460409
std         11.336135
min         20.000000
25%         39.000000
50%         47.000000
75%         55.000000
max         94.000000
Name: NR_IDADE_DATA_POSSE, dtype: float64
```

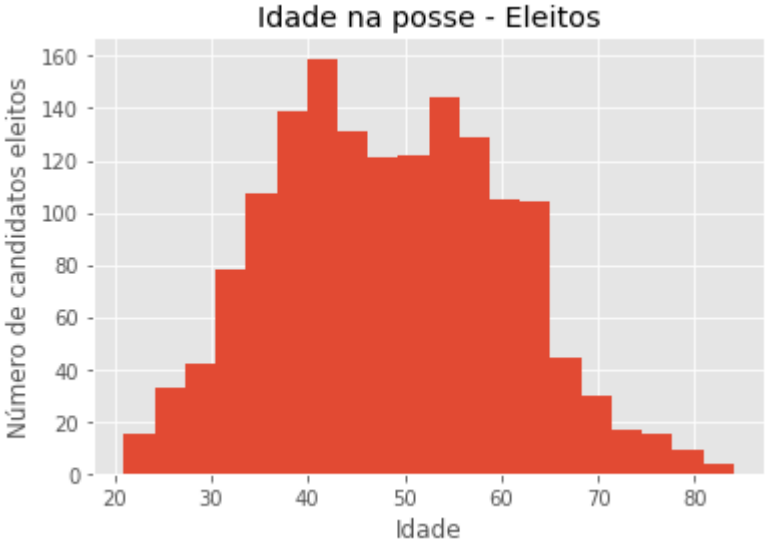
```
df_consulta_eleitos['NR_IDADE_DATA_POSSE'].describe()
```

```
count    1548.000000
mean       48.446382
std        12.000984
min        21.000000
25%        39.000000
50%        48.000000
75%        57.000000
max        84.000000
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("Número de candidatos")
plt.title("Idade na posse")
plt.show()
```

```
df_consulta_eleitos.NR_IDADE_DATA_POSSE.hist(bins=20)
plt.style.use('ggplot')
plt.xlabel("Idade")
plt.ylabel("Número de candidatos eleitos")
plt.title("Idade na posse - Eleitos")
plt.show()
```





Analisando os histogramas, apesar de o dos 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: 47,46 anos para todos os candidatos e 48,44 anos para os eleitos. Percebe-se também que a distribuição de idade é muito similar à do eleitorado.

O próximo critério a ser analisado é a escolaridade dos candidatos e, para a visualização desses dados, optou-se pelo gráfico de barras horizontais. 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': 7087,
         'SUPERIOR COMPLETO': 11497,
         'SUPERIOR INCOMPLETO': 2177,
         'ENSINO FUNDAMENTAL COMPLETO': 1386,
         'ENSINO FUNDAMENTAL INCOMPLETO': 764,
         'ENSINO MÉDIO INCOMPLETO': 697,
         'LÊ E ESCREVE': 223})
```

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

```
ENSINO MÉDIO COMPLETO:
29.74%

SUPERIOR COMPLETO:
48.24%

SUPERIOR INCOMPLETO:
9.14%

ENSINO FUNDAMENTAL COMPLETO:
5.82%

ENSINO FUNDAMENTAL INCOMPLETO:
3.21%

ENSINO MÉDIO INCOMPLETO:
2.92%

LÊ E ESCREVE:
0.94%
```

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

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

```
lista_escolaridade_candidatos_labels
```

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

```
lista_escolaridade_candidatos_valores
```

```
[7087, 11497, 2177, 1386, 764, 697, 223]
```

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

```
plt.style.use('seaborn-pastel')
plt.barh(lista_escolaridade_candidatos_labels, 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({'SUPERIOR COMPLETO': 1153,
         'ENSINO FUNDAMENTAL COMPLETO': 29,
         'ENSINO MÉDIO INCOMPLETO': 8,
         'ENSINO MÉDIO COMPLETO': 207,
         'SUPERIOR INCOMPLETO': 141,
         'ENSINO FUNDAMENTAL INCOMPLETO': 9,
         'LÊ E ESCREVE': 1})
```

```
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')
```

```
SUPERIOR COMPLETO:
74.48%

ENSINO FUNDAMENTAL COMPLETO:
1.87%

ENSINO MÉDIO INCOMPLETO:
0.52%

ENSINO MÉDIO COMPLETO:
13.37%

SUPERIOR INCOMPLETO:
9.11%

ENSINO FUNDAMENTAL INCOMPLETO:
0.58%

LÊ E ESCREVE:
0.06%
```

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

```
lista_escolaridade_candidatos_eleitos_labels
```

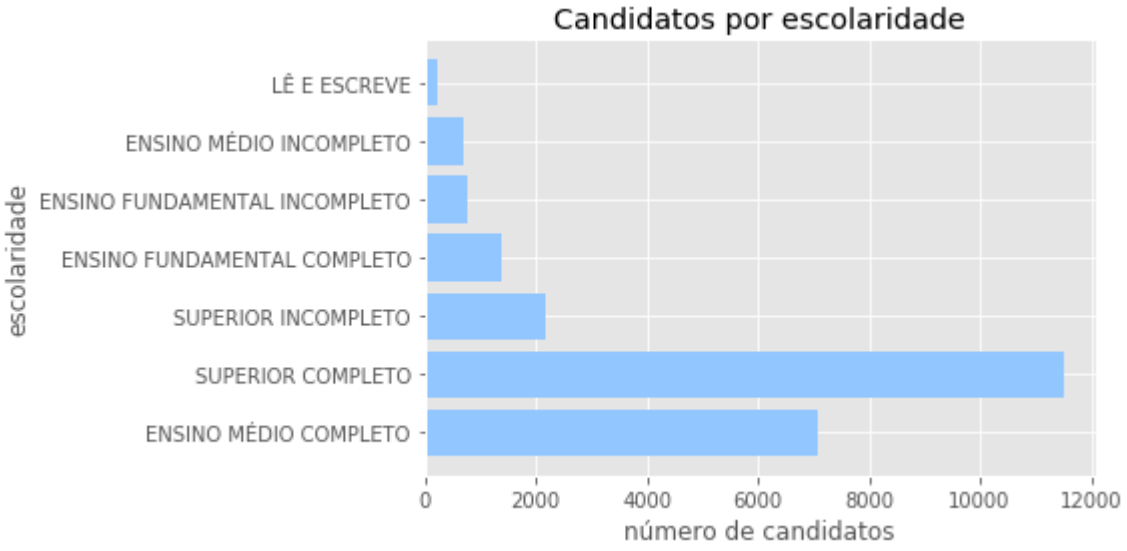
```
['SUPERIOR COMPLETO',
 'ENSINO FUNDAMENTAL COMPLETO',
 'ENSINO MÉDIO INCOMPLETO',
 'ENSINO MÉDIO COMPLETO',
 'SUPERIOR INCOMPLETO',
 'ENSINO FUNDAMENTAL INCOMPLETO',
 'LÊ E ESCREVE']
```

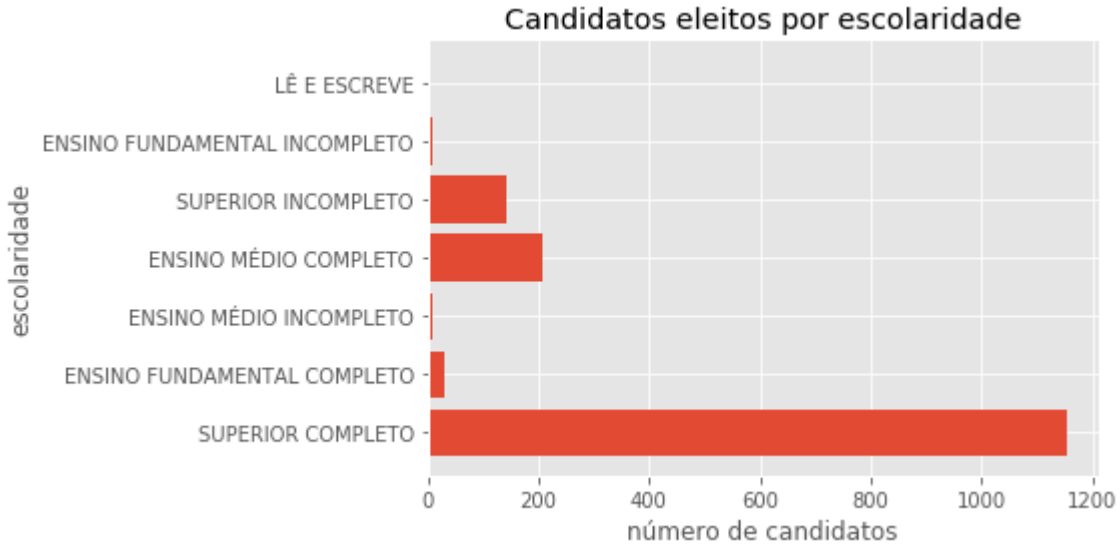
```
lista_escolaridade_candidatos_eleitos_valores
```

```
[1153, 29, 8, 207, 141, 9, 1]
```

```
plt.style.use('ggplot')
plt.barh(lista_escolaridade_candidatos_eleitos_labels,
         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:





Nesse ponto, percebe-se mais uma vez uma grande discrepância entre o perfil do eleitorado, dos candidatos e dos eleitos. Enquanto apenas 10,68% do eleitorado possui ensino superior completo, esse percentual é de 48,24% para os candidatos e de 74,48% para os eleitos. Aqui é importante destacar que esse

percentual elevado de eleitos com ensino superior completo 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.

O próximo critério a ser analisado é 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({'BRANCA': 12585,
         'PRETA': 2580,
         'PARDA': 8420,
         'AMARELA': 139,
         'INDÍGENA': 107})
```

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')
```

```
BRANCA:
52.81%

PRETA:
10.83%

PARDA:
35.33%

AMARELA:
0.58%

INDÍGENA:
0.45%
```

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
```

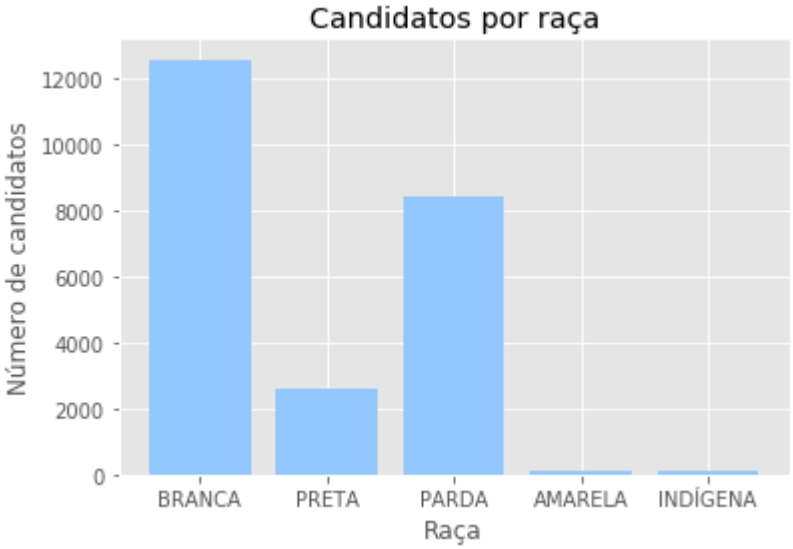
```
Counter({'BRANCA': 1124,
         'PARDA': 361,
         'PRETA': 59,
         'INDÍGENA': 1,
         'AMARELA': 3})
```

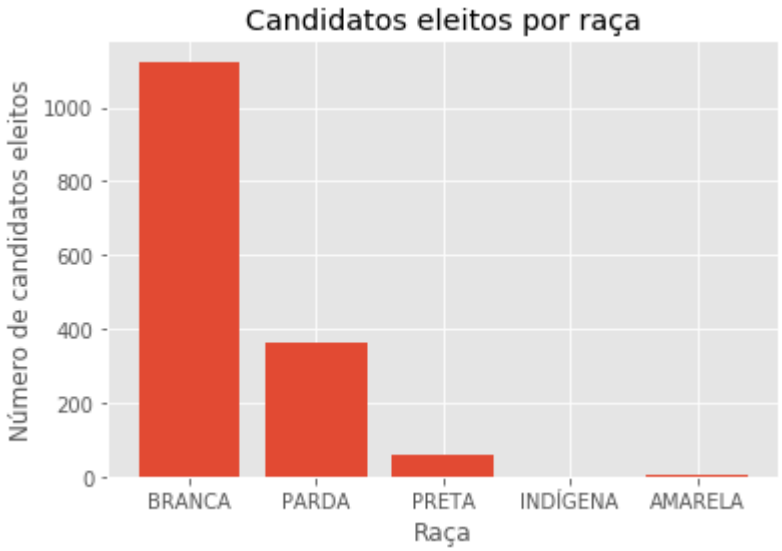
```
n_raca_eleitos=sum(raca_candidatos_eleitos.values())
for x, y in raca_candidatos_eleitos.items():
    a = y/n_raca_eleitos*100
    print(str(x) + ': ' +'\n'+ str(round(a,2)) + '%'+'\n')
```

```
BRANCA:
72.61%

PARDA:
23.32%

PRETA:
3.81%

INDÍGENA:
0.06%

AMARELA:
0.19%
```

```
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





Na análise desse ponto, percebe-se 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.

A última análise do *dataframe* em questão 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)': 12995,
         'SOLTEIRO(A)': 7543,
         'SEPARADO(A) JUDICIALMENTE': 261,
         'DIVORCIADO(A)': 2654,
         'VIÚVO(A)': 378})
```

```
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):
54.53%

SOLTEIRO(A):
31.65%

SEPARADO(A) JUDICIALMENTE:
1.1%

DIVORCIADO(A):
11.14%

VIÚVO(A):
1.59%
```

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

```
lista_estado_civil_candidatos_valores
```

```
[12995, 7543, 261, 2654, 378]
```

```
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 candidatos')
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)': 1112,
         'DIVORCIADO(A)': 125,
         'SOLTEIRO(A)': 289,
         'SEPARADO(A) JUDICIALMENTE': 9,
         'VIÚVO(A)': 13})
```

```
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):
71.83%

DIVORCIADO(A):
8.07%

SOLTEIRO(A):
18.67%

SEPARADO(A) JUDICIALMENTE:
0.58%

VIÚVO(A):
0.84%
```

```
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)
```

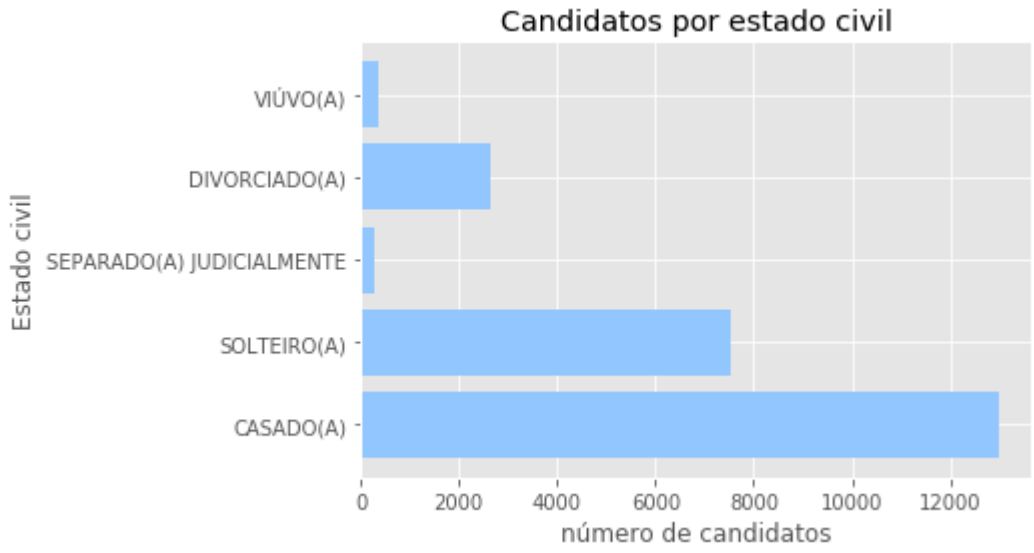
```
lista_estado_civil_candidatos_eleitos_labels
```

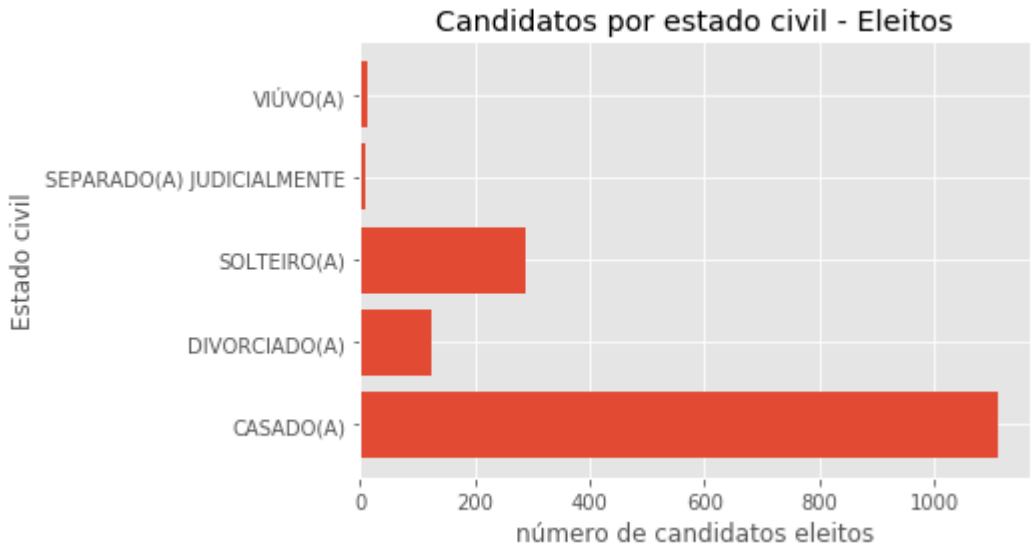
```
['CASADO(A)',
 'DIVORCIADO(A)',
 'SOLTEIRO(A)',
 'SEPARADO(A) JUDICIALMENTE',
 'VIÚVO(A)']
```

```
lista_estado_civil_candidatos_eleitos_valores
```

```
[1112, 125, 289, 9, 13]
```

```
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 candidatos eleitos')
plt.title('Candidatos por estado civil - Eleitos')
plt.show()
```





Percebe-se por meio dos gráficos e percentuais que a maioria dos candidatos é casado, com 54,53%, número que se eleva quando avaliamos os eleitos, com 71,83%. 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 percentual de casados apresenta uma alta de cerca de 17%.

Da análise combinada de todas as informações desse *dataframe*, percebe-se que os dados dos candidatos de todo o Brasil, eleitos ou não, se assemelham muito às informações dos deputados federais apresentadas anteriormente, o que indica um comportamento padrão em todo o país.

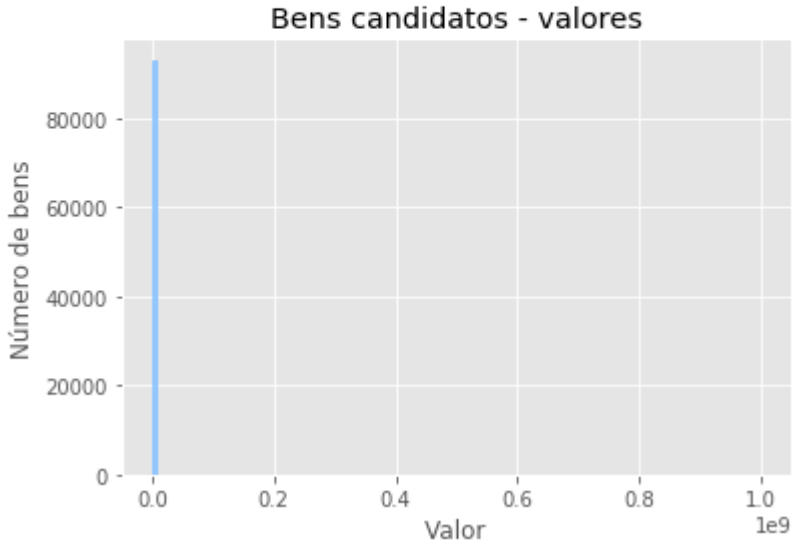
O próximo *dataframe* analisado é o que diz respeito aos bens declarados pelos candidatos. A primeira verificação será na análise estatística dos valores dos bens, o que será feito por meio da função “describe” e, para melhorar a visualização, os valores serão transformados para números inteiros usando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

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

```
count         93271
mean         257291
std         6875363
min               0
25%            8301
50%           37000
75%          122033
max      1000000000
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, já que 75% deles está abaixo de R$ 122.033,00. Em contrapartida o valor máximo identificado é de R$1.000.000.000,00 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, optou-se 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: 152 entries, 2411 to 92847
Data columns (total 4 columns):
SQ_CANDIDATO             152 non-null int64
DS_TIPO_BEM_CANDIDATO    152 non-null object
DS_BEM_CANDIDATO         152 non-null object
VR_BEM_CANDIDATO         152 non-null float64
dtypes: float64(1), int64(1), object(2)
memory usage: 5.9+ KB
```

Para que seja possível avaliar cada um dos 152 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'] > 10000000)].iterrows():
    print(index)
    print('Candidato: ' + str(row['SQ_CANDIDATO']))
    print('Tipo de bem: ' + row['DS_TIPO_BEM_CANDIDATO'])
    print('Descrição: ' + row['DS_BEM_CANDIDATO'])
    print('Valor: '+ str(row['VR_BEM_CANDIDATO']) + '\n')
```

O resultado desse comando imprimirá na tela os 152 registros da seguinte forma:

```
4509
Candidato: 250000600417
Tipo de bem: Veículo automotor terrestre: caminhão, automóvel, moto, etc.
Descrição: fiat siena. Valor R$ 14.000,00
Valor: 14000000.0
```

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$ 14.000,00 e não R$ 14.000.000,00. 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.

```
24456
Candidato: 130000620731
Tipo de bem: Terreno
Descrição: #NULO#
Valor: 20000000.0
```

Apesar de R$ 20.000.000,00 ser um valor elevado para um terreno, é um valor possível e não se pode desconsiderar. O mesmo não ocorre com o exemplo seguinte, em que um veículo é declarado com um valor de R$ 200.000.000,00. Nesse caso há claramente um erro.

```
17396
Candidato: 190000615521
Tipo de bem: Veículo automotor terrestre: caminhão, automóvel, moto, etc.
Descrição: #NULO#
Valor: 200000000.0
```

Com base nesses critérios, foram corrigidos 15 itens pelos motivos apresentados na tabela abaixo:

Índice	Tipo de bem	Descrição	Valor registrado	Novo Valor	Motivo
4509	Veículo automotor terrestre: caminhão, automóvel, moto, etc.	fiat siena. Valor R$ 14.000,00	R$ 14.000.000,00	R$ 14.000,00	Valor correto apresentado na descrição
17396	Veículo automotor terrestre: caminhão, automóvel, moto, etc.	#NULO#	R$ 200.000.000,00	R$ -	Valor descartado por ser muito discrepante e não ser possível corrigi-lo com certeza
22364	Veículo automotor terrestre: caminhão, automóvel, moto, etc.	gol no valor de R$ 16.000,00	R$ 16.000.000,00	R$ 16.000,00	Valor correto apresentado na descrição
23420	Casa	1) 50% (cinquenta por cento) de uma casa residencial adquirida em 2001/2002, por recursos próprios, localizada na Rua Iguará, n° 417, casa 6 - Vila Alpina. Valor de R$ 850.000,00	R$ 850.000.000,00	R$ 850.000,00	Valor correto apresentado na descrição
28756	Terreno	#NULO#	R$ 1.000.000.000,00	R$ -	Valor descartado por ser muito discrepante e não ser possível corrigi-lo com certeza

Índice	Tipo de bem	Descrição	Valor registrado	Novo Valor	Motivo
49940	OUTROS BENS E DIREITOS	50% (cinquenta por cento) de um imóvel urbano denominado lote 15, da quadra 135, do Loteamento Residencial Campo Belo, localizado na cidade de Monte Castelo - SP. Valor de R$50.642,00	R$ 50.642.000,00	R$ 50.642,00	Valor correto apresentado na descrição
50448	Outros bens imóveis	#NULO#	R$ 1.000.000.000,00	R$ -	Valor descartado por ser muito discrepante e não ser possível corrigi-lo com certeza
55349	Apartamento	50% de 1/6 de yn prédio em Tanabi. Valor de R$ 40.000,00	R$ 40.000.000,00	R$ 40.000,00	Valor correto apresentado na descrição
60840	Casa	localizado em Paraibuna no valor de R$ 600.000,00	R$ 600.000.000,00	R$ 600.000,00	Valor correto apresentado na descrição
65226	Veículo automotor terrestre: caminhão, automóvel, moto, etc.	veículo santa fé no valor de R$ 38.000,00	R$ 38.000.000,00	R$ 38.000,00	Valor correto apresentado na descrição
67475	OUTROS BENS E DIREITOS	3) 50% (cinquenta por cento) de um imóvel urbano denominado lote 14, da quadra 135, do Loteamento Residencial Campo Belo. Valor de R$50.642,00	R$ 50.642.000,00	R$ 50.642,00	Valor correto apresentado na descrição
77320	Casa	Imóvel Rua Professor Vasconcelos Sarmento no valor de R$ 340.000,00	R$ 340.000.000,00	R$ 340.000,00	Valor correto apresentado na descrição
77372	Casa	04 chácaras no valor de R$ 480.000,00	R$ 480.000.000,00	R$ 480.000,00	Valor correto apresentado na descrição
86521	Outros bens imóveis	01 barracão no valor de R$ 700.000,00	R$ 700.000.000,00	R$ 700.000,00	Valor correto apresentado na descrição

Índice	Tipo de bem	Descrição	Valor registrado	Novo Valor	Motivo
89495	Veículo automotor terrestre: caminhão, automóvel, moto, etc	5) 50% (cinquenta por cento) de um veículo da marca Honda, modelo CITY LX 1.5. Valor de R$ 58.000,00	R$ 58.000.000,00	R$ 58.000,00	Valor correto apresentado na descrição

Para aplicar essas correções, utilizou-se o código abaixo:

```
df_bem.at[4509, 'VR_BEM_CANDIDATO'] = 14000
df_bem.at[17396, 'VR_BEM_CANDIDATO'] = 0
df_bem.at[22364, 'VR_BEM_CANDIDATO'] = 16000
df_bem.at[23420, 'VR_BEM_CANDIDATO'] = 850000
df_bem.at[28756, 'VR_BEM_CANDIDATO'] = 0
df_bem.at[49940, 'VR_BEM_CANDIDATO'] = 50642
df_bem.at[50448, 'VR_BEM_CANDIDATO'] = 0
df_bem.at[55349, 'VR_BEM_CANDIDATO'] = 40000
df_bem.at[60840, 'VR_BEM_CANDIDATO'] = 600000
df_bem.at[65226, 'VR_BEM_CANDIDATO'] = 38000
df_bem.at[67475, 'VR_BEM_CANDIDATO'] = 50642
df_bem.at[77320, 'VR_BEM_CANDIDATO'] = 340000
df_bem.at[77372, 'VR_BEM_CANDIDATO'] = 480000
df_bem.at[86521, 'VR_BEM_CANDIDATO'] = 700000
df_bem.at[89495, 'VR_BEM_CANDIDATO'] = 58000
```

Após essas correções, os indicadores estatísticos do *dataframe* ficaram da seguinte forma:

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

```
count        93271
mean        199030
std        2174254
min              0
25%           8300
50%          37000
75%         122000
max      283176015
Name: VR_BEM_CANDIDATO, dtype: int32
```

Percebe-se que houve uma redução significativa no desvio padrão e no valor máximo, que apesar de ainda alto, é menor do que o anterior.

Após o tratamento dos valores individuais dos bens, eles foram agrupados por candidato, por meio de seu número sequencial, através dos comandos “groupby” e “sum()” que, em conjunto, somaram os valores de cada bem pelo número sequencial do candidato.

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

```
df_bem.info()
```

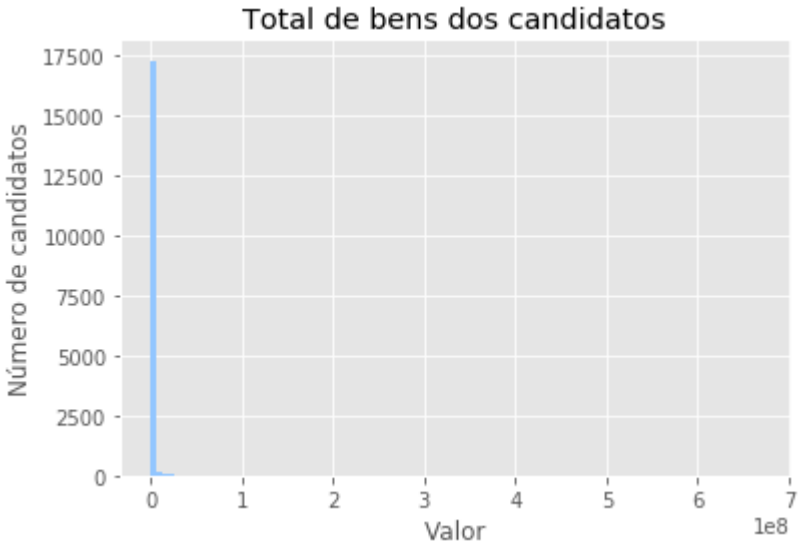
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17614 entries, 10000600001 to 280000629808
Data columns (total 1 columns):
VR_BEM_CANDIDATO    17614 non-null float64
dtypes: float64(1)
memory usage: 275.2 KB
```

Percebe-se que agora temos um *dataframe* com 17.614 entradas com apenas uma coluna, que agora é a soma dos bens do candidato, e que o índice agora é o número sequencial do candidato. Esse novo *dataframe* apresenta as seguintes características estatísticas:

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

```
count        17614
mean       1053921
std        9255080
min              0
25%          60000
50%         227389
75%         650000
max      667953170
Name: VR_BEM_CANDIDATO, dtype: int32
```

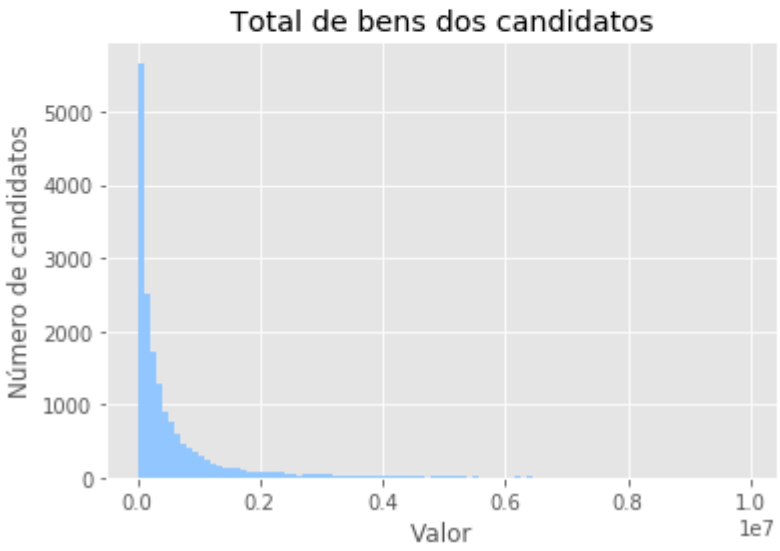
Novamente, percebe-se que os valores estão concentrados em valores baixos, o que é percebido pelo histograma da série de valores:



Com o objetivo de melhorar a visualização dessa informação, será criado um *dataframe* apenas com os valores inferiores a R$ 10.000.000,00, exclusivamente para plotagem em um histograma para se mostrar como a distribuição dos bens se comporta.

```
df_bem_10mi = df_bem.loc[(df_bem['VR_BEM_CANDIDATO'] < 10000000)]
```

```
df_bem_10mi.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel("Valor")
plt.ylabel("Número de candidatos")
plt.title("Total de bens dos candidatos")
plt.show()
```



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 ao cargo de deputado (estadual ou federal), 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'>
Int64Index: 23831 entries, 260000607562 to 90000607680
Data columns (total 6 columns):
NR_IDADE_DATA_POSSE    23831 non-null int64
DS_GENERO              23831 non-null object
DS_GRAU_INSTRUCAO      23831 non-null object
DS_ESTADO_CIVIL        23831 non-null object
DS_COR_RACA            23831 non-null object
DS_SIT_TOT_TURNO       23831 non-null object
dtypes: int64(1), object(5)
memory usage: 1.3+ 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'>
Int64Index: 23831 entries, 260000607562 to 90000607680
Data columns (total 7 columns):
NR_IDADE_DATA_POSSE    23831 non-null int64
DS_GENERO              23831 non-null object
DS_GRAU_INSTRUCAO      23831 non-null object
DS_ESTADO_CIVIL        23831 non-null object
DS_COR_RACA            23831 non-null object
DS_SIT_TOT_TURNO       23831 non-null object
VR_BEM_CANDIDATO       14677 non-null float64
dtypes: float64(1), int64(1), object(5)
memory usage: 2.1+ MB
```

Verifica-se que o novo *dataframe* “df_consolidado” possui 7 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       0
DS_GENERO                 0
DS_GRAU_INSTRUCAO         0
DS_ESTADO_CIVIL           0
DS_COR_RACA               0
DS_SIT_TOT_TURNO          0
VR_BEM_CANDIDATO       9154
dtype: int64
```

Pelo código, temos que 9.154 registros são nulos, ou seja, esses candidatos não apresentam bens declarados no *dataframe* “df_bem”. Nesse caso, esses valores serão preenchidos com o valor 0 (zero) por meio da função “fillna”.

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

```
df_consolidado.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23831 entries, 260000607562 to 90000607680
Data columns (total 7 columns):
NR_IDADE_DATA_POSSE    23831 non-null int64
DS_GENERO              23831 non-null object
DS_GRAU_INSTRUCAO      23831 non-null object
DS_ESTADO_CIVIL        23831 non-null object
DS_COR_RACA            23831 non-null object
DS_SIT_TOT_TURNO       23831 non-null object
VR_BEM_CANDIDATO       23831 non-null float64
dtypes: float64(1), int64(1), object(5)
memory usage: 2.1+ MB
```

Percebe-se que agora todos os 23.831 registros da coluna “VR_BEM_CANDIDATO” são não nulos e, com os dados consolidados, é possível filtrar as informações apenas para candidatos eleitos como feito anteriormente.

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

```
df_consolidado_eleitos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1548 entries, 240000600256 to 170000616969
Data columns (total 7 columns):
```

Com essa divisão é possível fazer a comparação de bens entre os candidatos eleitos e os de todos os candidatos.

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

```
count        23831
mean        454483
std        2797766
min              0
25%              0
50%          34646
75%         319267
max      255665376
Name: VR_BEM_CANDIDATO, dtype: int32
```

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

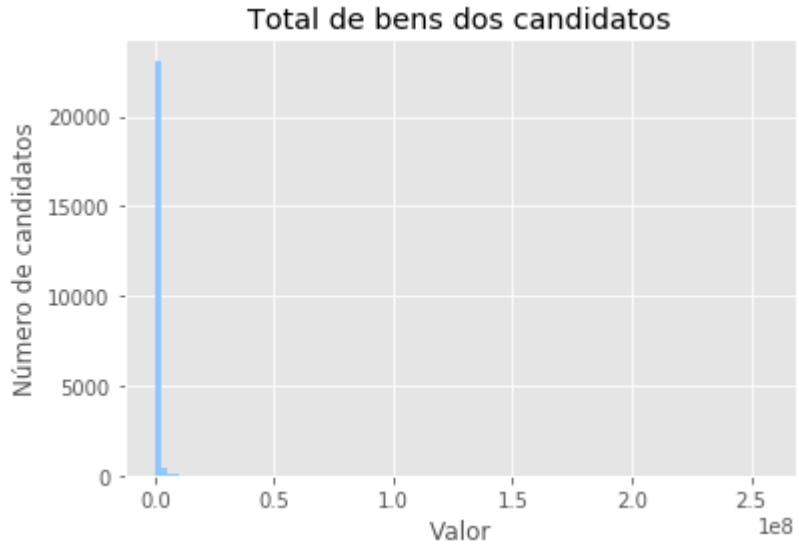
```
count         1548
mean       1641964
std        4531213
min              0
25%         285464
50%         768632
75%        1594491
max      119810503
Name: VR_BEM_CANDIDATO, dtype: int32
```

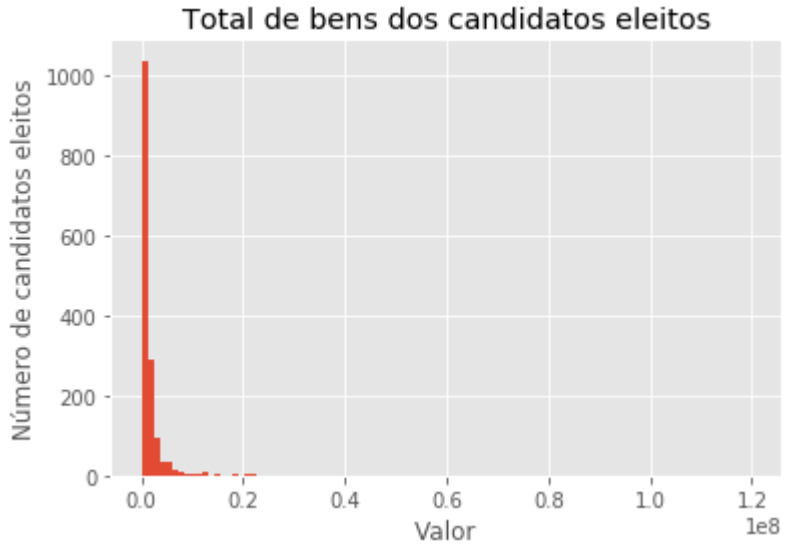
Analisando os dados, o que mais chama a atenção é que a média de bens dos candidatos eleitos é 261% superior a de todos os candidatos (R$ 1.641.964,00 contra R$ 454.483,00).

Colocando as informações no histograma depara-se com o mesmo problema anterior, ou seja, a grande distribuição de dados que prejudica a visualização, como pode-se verificar a seguir:

```
df_consolidado.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel("Valor")
plt.ylabel("Número de candidatos")
plt.title("Total de bens dos candidatos")
plt.show()
```

```
df_consolidado_eleitos.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('ggplot')
plt.xlabel("Valor")
plt.ylabel("Número de candidatos eleitos")
plt.title("Total de bens dos candidatos eleitos")
plt.show()
```





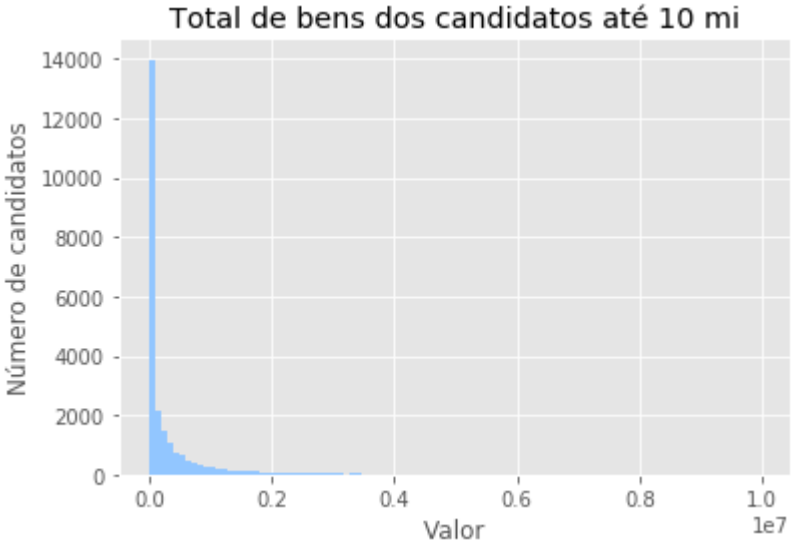
Apesar de já se perceber que os candidatos eleitos possuem um maior patrimônio do que os não eleitos, serão criados *dataframes* com patrimônios limitados à R$ 10.000.000,00 para melhor visualização. Esse processo será similar ao realizado anteriormente.

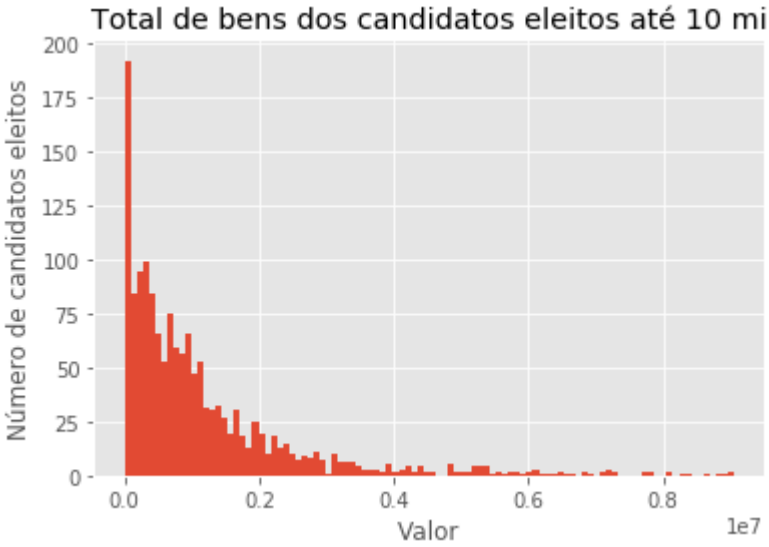
```
df_consolidado_10mi = df_consolidado.loc[(df_consolidado['VR_BEM_CANDIDATO'] < 10000000)]
```

```
df_consolidado_eleitos_10mi = df_consolidado_eleitos.loc[(df_consolidado_eleitos
                                                          ['VR_BEM_CANDIDATO'] < 10000000)]
```

```
df_consolidado_10mi.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel("Valor")
plt.ylabel("Número de candidatos")
plt.title("Total de bens dos candidatos até 10 mi")
plt.show()
```

```
df_consolidado_eleitos_10mi.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('ggplot')
plt.xlabel("Valor")
plt.ylabel("Número de candidatos eleitos")
plt.title("Total de bens dos candidatos eleitos até 10 mi")
plt.show()
```





Esses histogramas confirmam os dados estatísticos apresentados, mostrando que, aparentemente, o patrimônio impacta na eleição ou não de um candidato.

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.

```
df_despesas_pagas.head()
```

	SQ_DESPESA	VR_PAGTO_DESPESA
0	17599117	13.20
1	17609525	430.00
2	17885414	100.00
3	17885414	407.50
4	18136639	1258.81

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_pagas = df_despesas_pagas.groupby(['SQ_DESPESA']).sum()
```

```
df_despesas_pagas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1396351 entries, 17564736 to 25469128
Data columns (total 1 columns):
VR_PAGTO_DESPESA    1396351 non-null float64
dtypes: float64(1)
memory usage: 21.3 MB
```

Percebe-se a redução de entradas no *dataframe* após esse procedimento, de 1.647.553 para 1.306.351, 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: 1711062 entries, 22337526 to 23779597
Data columns (total 3 columns):
SQ_CANDIDATO             int64
DS_DESPESA               object
VR_DESPESA_CONTRATADA    float64
dtypes: float64(1), int64(1), object(1)
memory usage: 52.2+ 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_pagas)
```

```
df_despesas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1711062 entries, 17471857 to 25469131
Data columns (total 4 columns):
SQ_CANDIDATO             int64
DS_DESPESA               object
VR_DESPESA_CONTRATADA    float64
VR_PAGTO_DESPESA         float64
dtypes: float64(2), int64(1), object(1)
memory usage: 65.3+ 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        23484
dtype: int64
```

Verifica-se que há 23.484 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
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: 19292 entries, 10000600001 to 280000629808
Data columns (total 2 columns):
VR_DESPESA_CONTRATADA    19292 non-null float64
VR_PAGTO_DESPESA         19292 non-null float64
dtypes: float64(2)
memory usage: 452.2 KB
```

Tem-se agora um *dataframe* com 19.292 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: 19292 entries, 10000600001 to 280000629808
Data columns (total 1 columns):
VR_PAGTO_DESPESA    19292 non-null float64
dtypes: float64(1)
memory usage: 301.4 KB
```

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'>
Int64Index: 23831 entries, 260000607562 to 90000607680
Data columns (total 8 columns):
NR_IDADE_DATA_POSSE    23831 non-null int64
DS_GENERO              23831 non-null object
DS_GRAU_INSTRUCAO      23831 non-null object
DS_ESTADO_CIVIL        23831 non-null object
DS_COR_RACA            23831 non-null object
DS_SIT_TOT_TURNO       23831 non-null object
VR_BEM_CANDIDATO       23831 non-null float64
VR_PAGTO_DESPESA       17405 non-null float64
dtypes: float64(2), int64(1), object(5)
memory usage: 2.3+ 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_BEM_CANDIDATO          0
VR_PAGTO_DESPESA       6426
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_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'>
Int64Index: 1548 entries, 240000600256 to 170000616969
Data columns (total 8 columns):
NR_IDADE_DATA_POSSE    1548 non-null int64
DS_GENERO              1548 non-null object
DS_GRAU_INSTRUCAO      1548 non-null object
DS_ESTADO_CIVIL        1548 non-null object
DS_COR_RACA            1548 non-null object
DS_SIT_TOT_TURNO       1548 non-null object
VR_BEM_CANDIDATO       1548 non-null float64
VR_PAGTO_DESPESA       1548 non-null float64
dtypes: float64(2), int64(1), object(5)
memory usage: 108.8+ KB
```

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

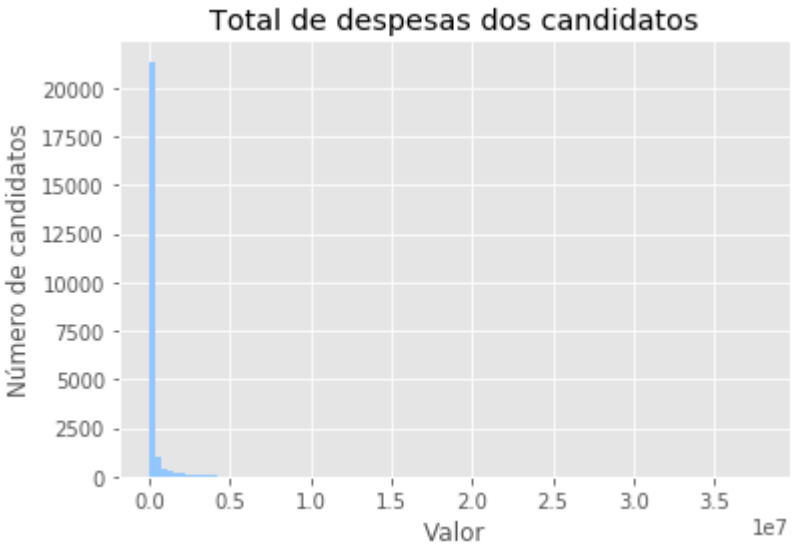
```
count       23831
mean       200365
std        835661
min             0
25%             0
50%          6864
75%         60945
max      37761372
Name: VR_PAGTO_DESPESA, dtype: int32
```

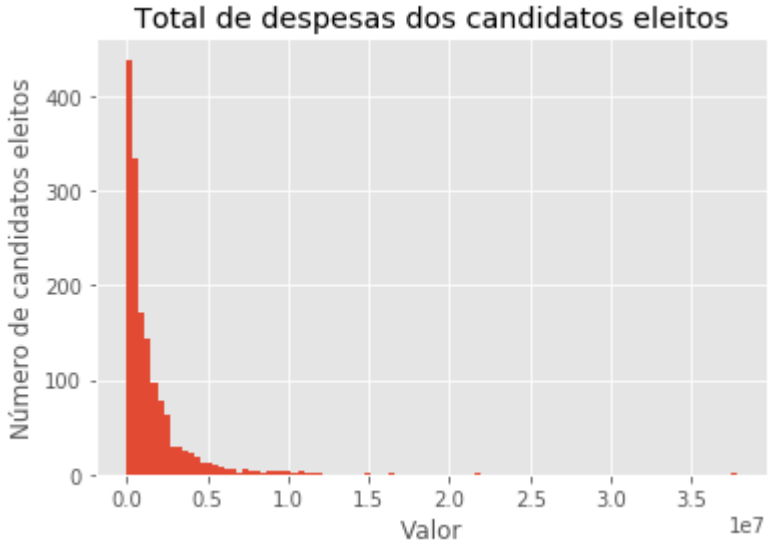
```
df_deputados_final_eleitos['VR_PAGTO_DESPESA'].describe().astype('int')
```

```
count        1548
mean      1453396
std       2124193
min             0
25%        323115
50%        755139
75%       1774531
max      37761372
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$ 200.365,00 para R$ 1.453.396,00, o que corresponde a um acréscimo de 625%.

5. Criação de Modelos de Machine Learning

O objetivo do estudo é identificar quais candidatos seriam eleitos dado um conjunto de características e, para isso, optou-se por utilizar algoritmos de classificação, que são cálculos preditivos usados para atribuir dados a categorias predefinidas, analisando um conjunto de dados de treinamento.

Classificação é o processo de reconhecer, compreender e agrupar ideias e objetos em categorias predefinidas ou "subpopulações". Usando conjuntos de dados de treinamento pré-categorizados, os programas de aprendizado de máquina usam uma variedade de algoritmos para classificar conjuntos de dados futuros em categorias.

Nesse estudo serão utilizados 6 tipos de algoritmos de classificação, são eles: Árvore de Decisão, Regressão Logística, Naïve Bayes, Gradiente Descendente, KNN (K - Nearest Neighbors) e Randon Forest.

A árvore de decisão é um algoritmo de aprendizado supervisionado adequado para problemas de classificação, pois é capaz de ordenar as classes em um nível preciso. Funciona como um fluxograma, separando os pontos de dados em duas categorias semelhantes ao mesmo tempo, do “tronco da árvore” aos “galhos” e às “folhas”, onde as categorias se tornam mais finitamente semelhantes. Isso cria categorias dentro das categorias, permitindo a classificação orgânica com supervisão humana limitada. A vantagem de se utilizar a árvore de decisão é que ela é de simples entendimento e visualização, requer pouca preparação de dados e suporta tanto dados numéricos quando categóricos. Em contrapartida, podem ser criadas árvores complexas que não possuem uma boa generalização e podem ser instáveis devido a pequenas variações dos dados que podem criar uma árvore completamente diferente da anterior.

A regressão logística é um recurso que nos permite estimar a probabilidade associada à ocorrência de determinado evento em face de um conjunto de variáveis explanatórias. É uma técnica recomendada para situações em que a variável dependente é de natureza dicotômica ou binária. Quanto às independentes, tanto podem ser categóricas ou não. Ela é mais útil no entendimento da influência de diversas variáveis independentes em uma saída única variável. A desvantagem é que funciona apenas quando a variável prevista é binária, assume que todos os preditores são independentes uns dos outros e assume que os dados estão livres de valores ausentes.

O algoritmo Naive Bayes é baseado no teorema de Bayes com a suposição de independência entre cada par de valores. A principal característica do algoritmo, e o motivo de receber “naive” (ingênuo) no nome, é que ele desconsidera completamente a correlação entre as variáveis. As principais vantagens deste algoritmo é que ele requer uma pequena quantidade de dados de treinamento para estimar os parâmetros necessários e é extremamente rápido em comparação com métodos mais sofisticados. A maior desvantagem é que ele é conhecido por não ser um bom avaliador.

O algoritmo gradiente descendente é um dos algoritmos de maior sucesso em problemas de Machine Learning. O método consiste em encontrar, de forma iterativa, os valores dos parâmetros que minimizam determinada função de

interesse. As vantagens desse algoritmo são sua eficiência e facilidade de implementação, mas ele requer vários hiperparâmetros e é sensível ao dimensionamento de recursos.

O algoritmo k-nearest neighbor, muitas vezes abreviado k-nn, é uma abordagem para classificação de dados que estima a probabilidade de um ponto de dados ser membro de um grupo ou de outro, dependendo de qual grupo os pontos de dados mais próximos a ele estão. Esse algoritmo é de fácil implementação, robusto a variações nos dados de treinamento e efetivo quando temos muitos dados de treinamento. O ponto negativo desse algoritmo é que exige muito recurso computacional.

Por fim, o algoritmo Random Forest ajusta árvores de decisão em várias subamostras de conjuntos de dados e usa a média para melhorar a precisão preditiva do modelo e controla o over-fitting. O tamanho da subamostra é sempre igual ao tamanho da amostra de entrada original, mas as amostras são retiradas com substituição. As dificuldades desse modelo são sua dificuldade de implementação e complexidade do algoritmo.

Para que se possa avaliar os algoritmos mencionados, optou-se por realizar alguns ajustes no *dataframe* “df_deputados_final”. Como boa prática, 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'], dtype=object)
```

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

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

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

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

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

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

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

```
array(['SUPLENTE', 'NÃO ELEITO', 'ELEITO POR MÉDIA', 'ELEITO POR QP'],
      dtype=object)
```

A primeira informação transformada será a que trata da escolaridade dos candidatos, a coluna DS_GRAU_INSTRUCAO. Como pode-se verificar, 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, 7, 6, 3, 2, 4, 1], dtype=int64)
```

O próximo ajuste a ser feito será na coluna “DS_SIT_TOT_TURNO” que indica se o candidato foi ou não eleito. Nesse 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], dtype=int64)
```

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'>
Int64Index: 23831 entries, 260000607562 to 90000607680
Data columns (total 17 columns):
NR_IDADE_DATA_POSSE                        23831 non-null int64
DS_GRAU_INSTRUCAO                          23831 non-null int64
DS_SIT_TOT_TURNO                           23831 non-null int64
VR_BEM_CANDIDATO                           23831 non-null float64
VR_PAGTO_DESPESA                           23831 non-null float64
DS_GENERO_FEMININO                         23831 non-null uint8
DS_GENERO_MASCULINO                        23831 non-null uint8
DS_ESTADO_CIVIL_CASADO(A)                  23831 non-null uint8
DS_ESTADO_CIVIL_DIVORCIADO(A)              23831 non-null uint8
DS_ESTADO_CIVIL_SEPARADO(A) JUDICIALMENTE  23831 non-null uint8
DS_ESTADO_CIVIL_SOLTEIRO(A)                23831 non-null uint8
DS_ESTADO_CIVIL_VIÚVO(A)                   23831 non-null uint8
DS_COR_RACA_AMARELA                        23831 non-null uint8
DS_COR_RACA_BRANCA                         23831 non-null uint8
DS_COR_RACA_INDÍGENA                       23831 non-null uint8
DS_COR_RACA_PARDA                          23831 non-null uint8
DS_COR_RACA_PRETA                          23831 non-null uint8
dtypes: float64(2), int64(3), uint8(12)
memory usage: 2.0 MB
```

É possível perceber que 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=uint64)
```

O caso em análise busca avaliar a previsibilidade de um candidato ser eleito ou não com base em alguns parâmetros. Dessa forma, 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 22.283 e 1.548.

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

```
0    22283
1     1548
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.

O primeiro passo é importar a função por meio do comando:

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

Em seguida 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'>
Int64Index: 22283 entries, 260000607562 to 90000607680
Data columns (total 17 columns):
```

```
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'>
Int64Index: 1548 entries, 240000600256 to 170000616969
Data columns (total 17 columns):
```

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'>
Int64Index: 22283 entries, 50000608012 to 250000615370
Data columns (total 17 columns):
```

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'>
Int64Index: 44566 entries, 260000607562 to 250000615370
Data columns (total 17 columns):
```

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

```
1    22283
0    22283
Name: DS_SIT_TOT_TURNO, dtype: int64
```

Percebe-se que o novo *dataframe* possui 44.566 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()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 44566 entries, 260000607562 to 250000615370
Data columns (total 16 columns):
```

```
type(y_train)
```

```
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'>
Int64Index: 33424 entries, 150000612361 to 210000605946
Data columns (total 16 columns):
```

```
xteste.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11142 entries, 210000604677 to 210000604696
Data columns (total 16 columns):
```

```
ytreinamento.count()
```

```
33424
```

```
yteste.count()
```

```
11142
```

Antes de se aplicar os algoritmos de classificação, é 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(random_state=0)
deputados_tree = deputados_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))
```

```
Acurácia:  1.0
Acurácia de previsão:  0.9744211093161013
             precision    recall  f1-score   support

          0       1.00      0.95      0.97      5495
          1       0.95      1.00      0.98      5647

avg / total       0.98      0.97      0.97     11142
```

```
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))
```

```
Acurácia:  0.5857467687888942
Acurácia de previsão:  0.5953150242326333
             precision    recall  f1-score   support

          0       1.00      0.18      0.30      5495
          1       0.56      1.00      0.71      5647

avg / total       0.77      0.60      0.51     11142
```

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb = nb.fit(xtreinamento, ytreinamento)
print("Acurácia: ", nb.score(xtreinamento, ytreinamento))
tp_nb = nb.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_nb))
print(classification_report(yteste, tp_nb))
```

```
Acurácia:  0.6956677836285304
Acurácia de previsão:  0.6850655178603482
             precision    recall  f1-score   support

          0       0.61      0.97      0.75      5495
          1       0.93      0.41      0.57      5647

avg / total       0.78      0.69      0.66     11142
```

```
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))
```

```
Acurácia:  0.5878709909047392
Acurácia de previsão:  0.5977382875605816
             precision    recall  f1-score   support

          0       1.00      0.18      0.31      5495
          1       0.56      1.00      0.72      5647

avg / total       0.78      0.60      0.52     11142
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn = knn.fit(xtreinamento, ytreinamento)
print("Acurácia: ", knn.score(xtreinamento, ytreinamento))
tp_knn = knn.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_knn))
print(classification_report(yteste, tp_knn))
```

```
Acurácia:  0.9576053135471517
Acurácia de previsão:  0.9506372285047567
             precision    recall  f1-score   support

          0       1.00      0.90      0.95      5495
          1       0.91      1.00      0.95      5647

avg / total       0.96      0.95      0.95     11142
```

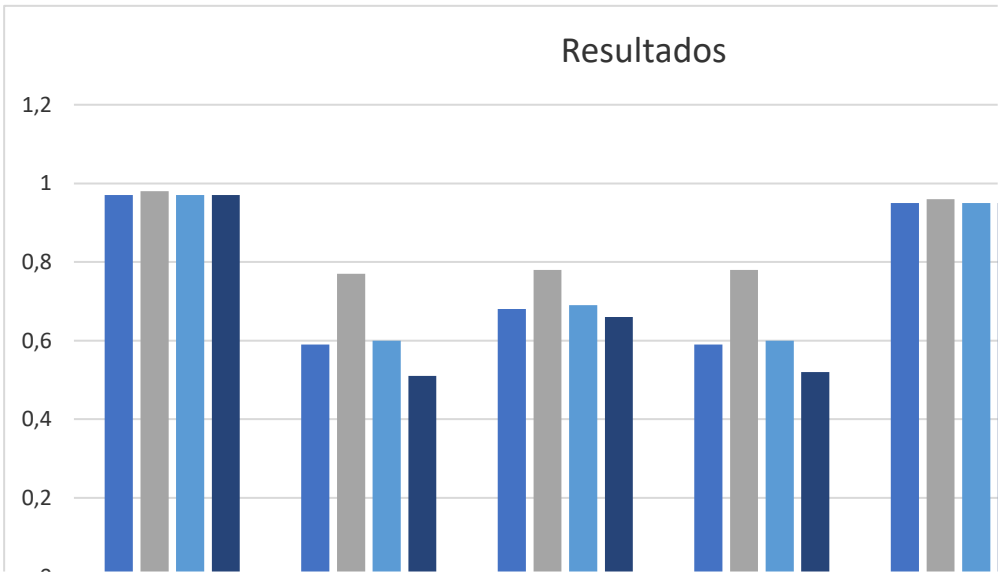
```
from sklearn.ensemble import RandomForestClassifier
rfm = RandomForestClassifier()
rfm = rfm.fit(xtreinamento, ytreinamento)
print("Acurácia: ", rfm.score(xtreinamento, ytreinamento))
tp_rfm = rfm.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_rfm))
print(classification_report(yteste,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

```
Acurácia:  0.9992221158449018
Acurácia de previsão:  0.9810626458445522
             precision    recall  f1-score   support

          0       1.00      0.96      0.98      5495
          1       0.96      1.00      0.98      5647

avg / total       0.98      0.98      0.98     11142
```

6. Apresentação dos Resultados

Os resultados médios dos algoritmos podem ser verificados na tabela e gráfico a seguir:

Algoritmo	Acurácia de previsão	Precisão	Revocação	F1-Score
Árvore de Decisão	0,97	0,98	0,97	0,97
Regressão Logística	0,59	0,77	0,60	0,51
Naïve Bayes	0,68	0,78	0,69	0,66
Gradiente Descendente	0,59	0,78	0,60	0,52
KNN	0,95	0,96	0,95	0,95
Randon Forest	0,98	0,98	0,98	0,98



Analisando as informações da tabela e do gráfico, percebe-se que três algoritmos se destacam: a árvore de decisão, o KNN e o Random Forest. Eles apresentam resultados muito bons em todas as medidas de avaliação, com uma pequena vantagem para o algoritmo Random Forest, que apresenta um resultado de 0,98 para todas elas.

A análise mais detalhada será feita nos resultados do algoritmo que apresentou melhores resultados, ou seja, Random Forest. Ao se analisar a precisão dos resultados individuais, o valor obtido para a previsão de candidatos que não seriam eleitos chama muito a atenção, já que apresenta o valor 1,00, ou seja, o algoritmo não identificou nenhum falso positivo no momento de identificar candidatos que não seriam eleitos. O valor apresentado para os candidatos que seriam eleitos também chama a atenção, já traz o valor de 0,96, ou seja, a cada 100 previsões de que o candidato seria eleito, em apenas 4 delas se teria um falso positivo.

A interpretação dos resultados de revocação são similares, com a diferença de que o item que apresenta resultado 1,00 é o de candidatos eleitos, ou seja, não se identificou nenhum falso negativo. Quando a análise é feita sobre os candidatos que não seriam eleitos, a cada 100 previsões, 4 indicam um falso negativo, fazendo com que o seu valor seja de 0,96.

Por fim, analisando a acurácia de previsão, temos o valor de 0,98, ou seja, de todas as amostras, o algoritmo acerta em 98% das vezes, afirmando que um candidato será eleito ou não.

Esses resultados demonstram que há uma grande previsibilidade de resultado com os dados apresentados. Em outras palavras, em se mantendo o cenário das eleições de 2018, sabendo as características do candidato, seu patrimônio e o valor que se pretende gastar na campanha, é possível prever, com elevado grau de acurácia, quem será ou não eleito na próxima eleição de deputado, seja federal ou estadual.

A conclusão do trabalho é que para se ter uma melhor representatividade do eleitorado nas casas legislativas, que são as que representam a população, é necessário incentivar, principalmente, o público feminino e não branco a participar das disputas eleitorais e dar iguais condições competitivas a eles. Quando se fala

em grau de instrução, o ideal é que sejam proporcionadas melhores condições de ensino a todos, pois o interesse não é que a escolaridades dos deputados seja menor, mas sim que a escolaridade da população seja maior. Por fim, viu-se, também, a relevância do poder aquisitivo e financeiro no resultado das eleições e, assim como na escolaridade, deve-se buscar iniciativas que diminuam a desigualdade e que todos os candidatos tenham iguais condições de competição.

Em resumo, se não existirem ações concretas para mudar o perfil dos candidatos, para melhorar a escolaridade de toda a população e para se ter uma melhor distribuição de renda, o resultado da eleição é extremamente previsível e o perfil dos grupos que compõem as casas legislativas em todo o país continuará sendo o mesmo de 500 anos atrás, ou seja, homem branco pertencente à elite da sociedade.

7. Links

Link para o vídeo:
https://youtu.be/6oqSz9r77Ks

Link para o repositório Github (sem datasets):
https://github.com/henrique-m-oliveira/TCC---Big-Data

Link para o repositório Google Drive (com datasets):
https://drive.google.com/drive/folders/1VBsUZESYTF7MO9Kwi21cAWHSMzALmbIM?usp=sharing

APÊNDICE

Programação/Scripts

```
#Importação da biblioteca pandas
import pandas as pd

#leitura do arquivo consulta_cand_2018_BRASIL.csv
df_consulta = pd.read_csv("consulta_cand_2018_BRASIL.csv",
                          encoding = "Latin 1", sep = ";", decimal = ',')

#informações do dataframe df_consulta
df_consulta.info()

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

#Seleção das opções de interesse na coluna DS_CARGO
df_consulta = df_consulta[df_consulta.DS_CARGO.isin(['DEPUTADO ESTADUAL',
                                                     'DEPUTADO FEDERAL'])]

#Informações do dataframe após o filtro aplicado
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()

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

#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']]

#Informações do dataframe após o filtro aplicado
df_consulta.info()

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

#leitura do arquivo bem_candidato_2018_BRASIL.csv
df_bem = pd.read_csv("bem_candidato_2018_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_2018_BRASIL.csv
df_despesas_contratadas = pd.read_csv("despesas_contratadas_candidatos_2018_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_2018_BRASIL.csv
df_despesas_pagas = pd.read_csv("despesas_pagas_candidatos_2018_BRASIL.csv",
                                encoding = "Latin 1", sep = ";", decimal = ',')

#informações do dataframe df_despesas_pagas
df_despesas_pagas.info()

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

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

#Verificação dos valores nulos
df_despesas_pagas.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('seaborn-pastel')
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.show()

#Contagem das opções da coluna DS_GENERO de candidatos eleitos
genero_candidatos_eleitos = Counter(df_consulta_eleitos['DS_GENERO'])
genero_candidatos_eleitos

#Plotagem das informações de gênero dos candidatos eleitos
plt.style.use('ggplot')
plt.pie(genero_candidatos_eleitos.values(), labels = genero_candidatos_eleitos.keys(),
        autopct = '%1.1f%%', textprops={'fontsize': 16})
plt.title("Gênero dos candidatos eleitos", fontsize=18)
plt.axis("image")
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("Número de candidatos")
plt.title("Idade na posse")
plt.show()

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

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

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

```
#Verificação da lista criada
lista_escolaridade_candidatos_labels

#Verificação da lista criada
lista_escolaridade_candidatos_valores

#Plotagem do gráfico de escolaridade dos candidatos
plt.style.use('seaborn-pastel')
plt.barh(lista_escolaridade_candidatos_labels, 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_labels = []
lista_escolaridade_candidatos_eleitos_valores = []
for x, y in escolaridade_candidatos_eleitos.items():
    lista_escolaridade_candidatos_eleitos_labels.append(x)
    lista_escolaridade_candidatos_eleitos_valores.append(y)

#Verificação da lista criada
lista_escolaridade_candidatos_eleitos_labels

#Verificação da lista criada
lista_escolaridade_candidatos_eleitos_valores

#Plotagem do gráfico de escolaridade dos candidatos eleitos
plt.style.use('ggplot')
plt.barh(lista_escolaridade_candidatos_eleitos_labels,
         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')
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')
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)

#Verificação da lista criada
lista_estado_civil_candidatos_labels

#Verificação da lista criada
lista_estado_civil_candidatos_valores

#Plotagem do gráfico de estado civil dos candidatos
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 candidatos')
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)

#Verificação da lista criada
lista_estado_civil_candidatos_eleitos_labels

#Verificação da lista criada
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 candidatos eleitos')
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: ' + row['DS_TIPO_BEM_CANDIDATO'])
    print('Descrição: ' + row['DS_BEM_CANDIDATO'])
    print('Valor: '+ str(row['VR_BEM_CANDIDATO']) + '\n')

#Correções possíveis dos valores nos bens
df_bem.at[4509, 'VR_BEM_CANDIDATO'] = 14000
df_bem.at[17396, 'VR_BEM_CANDIDATO'] = 0
df_bem.at[22364, 'VR_BEM_CANDIDATO'] = 16000
df_bem.at[23420, 'VR_BEM_CANDIDATO'] = 850000
df_bem.at[28756, 'VR_BEM_CANDIDATO'] = 0
df_bem.at[49940, 'VR_BEM_CANDIDATO'] = 50642
df_bem.at[50448, 'VR_BEM_CANDIDATO'] = 0
df_bem.at[55349, 'VR_BEM_CANDIDATO'] = 40000
df_bem.at[60840, 'VR_BEM_CANDIDATO'] = 600000
df_bem.at[65226, 'VR_BEM_CANDIDATO'] = 38000
df_bem.at[67475, 'VR_BEM_CANDIDATO'] = 50642
```

```
df_bem.at[77320, 'VR_BEM_CANDIDATO'] = 340000
df_bem.at[77372, 'VR_BEM_CANDIDATO'] = 480000
df_bem.at[86521, 'VR_BEM_CANDIDATO'] = 700000
df_bem.at[89495, 'VR_BEM_CANDIDATO'] = 58000

#Descrição estatística dos valores dos bens após as correções
df_bem['VR_BEM_CANDIDATO'].describe().astype('int')

#Agrupamento dos bens por candidato
df_bem = df_bem.groupby(['SQ_CANDIDATO']).sum()

#Informações do dataframe df_bem após o agrupamento
df_bem.info()

#Descrição estatística do dataframe df_bem após agrupamento
df_bem['VR_BEM_CANDIDATO'].describe().astype('int')

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

#Criação de um dataframe com bens abaixo de R$ 10.000.000,00
df_bem_10mi = df_bem.loc[(df_bem['VR_BEM_CANDIDATO'] < 10000000)]

#Plotagem do histograma dos valores dos bens dos candidatos até R$ 10.000.000,00
df_bem_10mi.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel("Valor")
plt.ylabel("Número de candidatos")
plt.title("Total de bens dos candidatos")
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 do histograma dos valores dos bens dos candidatos do dataframe consolidado
df_consolidado.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel("Valor")
plt.ylabel("Número de candidatos")
plt.title("Total de bens dos candidatos")
plt.show()

#Plotagem do histograma dos valores dos bens dos candidatos eleitos do dataframe consolidado
df_consolidado_eleitos.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('ggplot')
plt.xlabel("Valor")
plt.ylabel("Número de candidatos eleitos")
plt.title("Total de bens dos candidatos eleitos")
plt.show()

#Criação de um dataframe com bens dos candidatos até R$ 10.000.000,00
df_consolidado_10mi = df_consolidado.loc[(df_consolidado['VR_BEM_CANDIDATO'] < 10000000)]

#Criação de um dataframe com bens dos candidatos eleitos até R$ 10.000.000,00
df_consolidado_eleitos_10mi = df_consolidado_eleitos.loc[(df_consolidado_eleitos
                                        ['VR_BEM_CANDIDATO'] < 10000000)]

#Plotagem dos bens de candidatos até R$ 10.000.000,00
df_consolidado_10mi.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('seaborn-pastel')
plt.xlabel("Valor")
plt.ylabel("Número de candidatos")
plt.title("Total de bens dos candidatos até 10 mi")
plt.show()

#Plotagem dos bens de candidatos eleitos até R$ 10.000.000,00
df_consolidado_eleitos_10mi.VR_BEM_CANDIDATO.hist(bins=100)
plt.style.use('ggplot')
plt.xlabel("Valor")
plt.ylabel("Número de candidatos eleitos")
plt.title("Total de bens dos candidatos eleitos até 10 mi")
plt.show()

#Descrição estatística das despesas contratadas
df_despesas_contratadas['VR_DESPESA_CONTRATADA'].describe().astype('int')

#Primeiras linhas do dataframe df_despesas_pagas
df_despesas_pagas.head()

#Agrupamento do dataframe df_despesas_pagas por código de despesa
df_despesas_pagas = df_despesas_pagas.groupby(['SQ_DESPESA']).sum()

#Informações do dataframe df_despesas_pagas após agrupamento
df_despesas_pagas.info()
```

```
#Definição de índice do dataframe df_despesas_contratadas
df_despesas_contratadas = df_despesas_contratadas.set_index('SQ_DESPESA')

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

#criação do dataframe df_despesas com a junção de df_despesas_contratadas e df_despesas_pagas
df_despesas = df_despesas_contratadas.join(df_despesas_pagas)

#Informações do dataframe df_despesas
df_despesas.info()

#identificação de valores nulos no dataframe df_despesas
df_despesas.isnull().sum()

#Preenchimento dos valores nulos no dataframe df_despesas
df_despesas.fillna(0, inplace = True)

#identificação de valores nulos no dataframe df_despesas
df_despesas.isnull().sum()

#Agrupamento das despesas por candidato
df_despesas = df_despesas.groupby(['SQ_CANDIDATO']).sum()

#Informações do dataframe df_despesas após agrupamento
df_despesas.info()

#Definiçãa do dataframe df_despesas apenas com a coluna VR_PAGTO_DESPESA
df_despesas = df_despesas[['VR_PAGTO_DESPESA']]

#Informações do dataframe df_despesas após seleção da coluna
df_despesas.info()

#criação do dataframe df_deputados_final com a junção de df_consolidado e df_despesas
df_deputados_final = df_consolidado.join(df_despesas)

#Informações do dataframe df_deputados_final
df_deputados_final.info()

#identificação de valores nulos no dataframe df_deputados_final
df_deputados_final.isnull().sum()

#Preenchimento dos valores nulos no dataframe df_deputados_final
df_deputados_final.fillna(0, inplace = True)

#Informações do dataframe df_deputados_final após preenchimento dos valores nulos
df_deputados_final.isnull().sum()

#Criação de um dataframe final com candidatos eleitos
df_deputados_final_eleitos = df_deputados_final[df_deputados_final.DS_SIT_TOT_TURNO.isin([
    'ELEITO POR MÉDIA', 'ELEITO POR QP'])]

#Informações do dataframe df_deputados_final_eleitos
df_deputados_final_eleitos.info()

#Descrição estatística das despesas pagas dos candidatos
df_deputados_final['VR_PAGTO_DESPESA'].describe().astype('int')

#Descrição estatística das despesas pagas dos candidatos eleitos
```

```
df_deputados_final_eleitos['VR_PAGTO_DESPESA'].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()

#Primeiras linhas do dataframe df_deputados_final
df_deputados_final.head()

#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()

#Primeiras linhas do dataframe df_deputados_final
df_deputados_final.head()

#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=22283,
                                         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

#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(random_state=0)
deputados_tree = deputados_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 Naïve Bayes
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb = nb.fit(xtreinamento, ytreinamento)
print("Acurácia: ", nb.score(xtreinamento, ytreinamento))
tp_nb = nb.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_nb))
print(classification_report(yteste, tp_nb))

#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 KNN (K - Nearest Neighbors)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn = knn.fit(xtreinamento, ytreinamento)
print("Acurácia: ", knn.score(xtreinamento, ytreinamento))
tp_knn = knn.predict(xteste)
print("Acurácia de previsão: ", accuracy_score(yteste, tp_knn))
print(classification_report(yteste, tp_knn))
```

```
#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(classification_report(yteste, tp_rfm))
```