```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

```
df = pd.read_csv('/content/conjunto_de_dados_de_funcionarios.csv')
```

Verificar a existência dos valores duplicados e efetuar a remoção

```
df.duplicated().sum() # número de linhas duplicadas
```

```
1889
```

```
df[df.duplicated()] #identificar e selecionar linhas duplicadas
```

	Education	JoiningYear	City	PaymentTier	Age	Gender	EverBenched	ExperienceInCurrentDomain	LeaveOrNot
111	Bachelors	2017	Pune	2	27	Female	No	5	1
130	Bachelors	2017	Bangalore	3	26	Female	No	4	0
138	Bachelors	2017	New Delhi	3	28	Male	No	2	0
160	Bachelors	2014	Bangalore	3	28	Female	No	3	0
167	Bachelors	2014	Bangalore	3	25	Male	No	3	0
...	...	...	...	...	...	...	...	...	...
4640	Bachelors	2015	Bangalore	3	35	Male	No	0	0
4642	Bachelors	2012	Bangalore	3	36	Female	No	4	0
4646	Bachelors	2013	Bangalore	3	25	Female	No	3	0
4648	Bachelors	2013	Bangalore	3	26	Female	No	4	0
4652	Bachelors	2015	Bangalore	3	33	Male	Yes	4	0

1889 rows × 9 columns

```
df.drop_duplicates(inplace=True)  # Removendo Duplicatas
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2764 entries, 0 to 4651
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Education                 2764 non-null   object
 1   JoiningYear               2764 non-null   int64
 2   City                      2764 non-null   object
 3   PaymentTier               2764 non-null   int64
```

```
 4   Age                       2764 non-null   int64
 5   Gender                    2764 non-null   object
 6   EverBenched               2764 non-null   object
 7   ExperienceInCurrentDomain  2764 non-null   int64
 8   LeaveOrNot                2764 non-null   int64
dtypes: int64(5), object(4)
memory usage: 215.9+ KB
```

```
df.duplicated().sum()
```

```
0
```

Mudar nome de coluna

```
df.rename(columns={'Education': 'Formacao'}, inplace=True) # Education para 'Formacao'
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2764 entries, 0 to 4651
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Formacao                  2764 non-null   object
 1   JoiningYear               2764 non-null   int64
 2   City                      2764 non-null   object
 3   PaymentTier               2764 non-null   int64
 4   Age                       2764 non-null   int64
 5   Gender                    2764 non-null   object
 6   EverBenched               2764 non-null   object
 7   ExperienceInCurrentDomain  2764 non-null   int64
 8   LeaveOrNot                2764 non-null   int64
dtypes: int64(5), object(4)
memory usage: 215.9+ KB
```

Realizar uma análise gráfica dos dados para verificar:

Como é o comportamento da distribuição na formação dos funcionários? (Pode-se usar o gráfico de colunas ou de pizza)

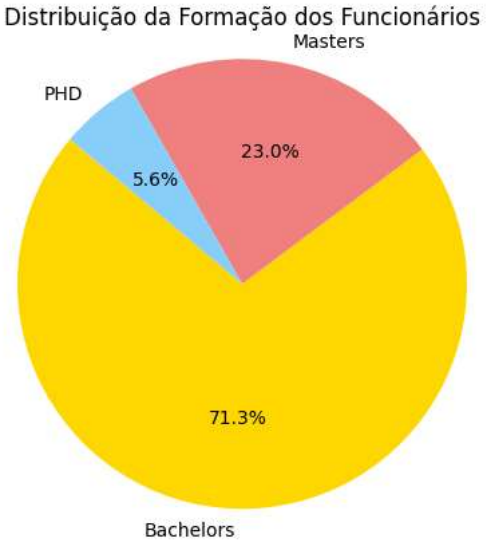
```
# Contar a distribuição das formações dos funcionários
formacao_distribuicao = df['Formacao'].value_counts()

# Preparar os dados para o gráfico de pizza
labels = formacao_distribuicao.index
sizes = formacao_distribuicao.values
colors = ['gold', 'lightcoral', 'lightskyblue', 'lightgreen']  # Cores para as fatias

# Criar um gráfico de pizza
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
```

```
plt.axis('equal')  # Assegura que o gráfico de pizza seja um círculo.

plt.title("Distribuição da Formação dos Funcionários")
plt.show()
```

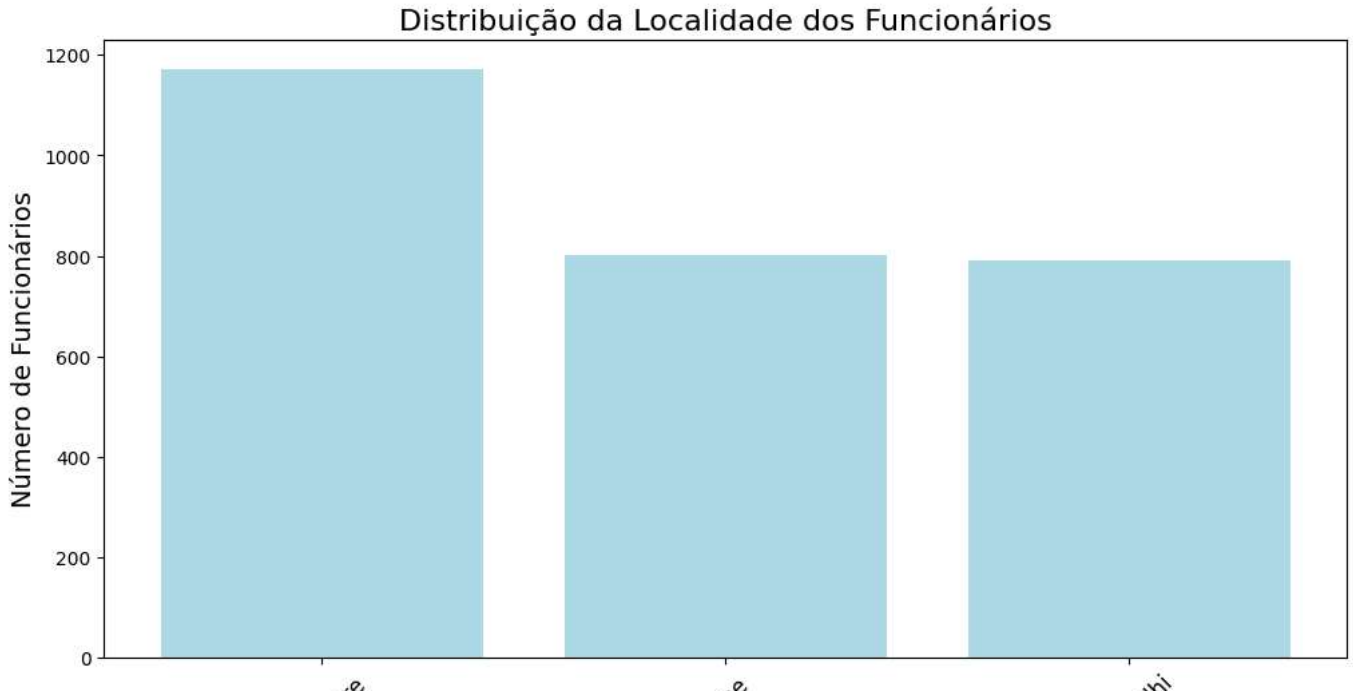


Como é a distribuição da localidade dos funcionários? (Pode-se usar o gráfico de colunas ou de pizza)

```
# Contar a distribuição das localidades dos funcionários
localidade_distribuicao = df['City'].value_counts()

# Preparar os dados para o gráfico de colunas
cidades = localidade_distribuicao.index
quantidades = localidade_distribuicao.values

# Criar um gráfico de colunas
plt.figure(figsize=(12, 6))
plt.bar(cidades, quantidades, color='lightblue')
plt.xticks(rotation=45, fontsize=12)
plt.xlabel('Localidade (Cidade)', fontsize=14)
plt.ylabel('Número de Funcionários', fontsize=14)
plt.title('Distribuição da Localidade dos Funcionários', fontsize=16)
plt.show()
```

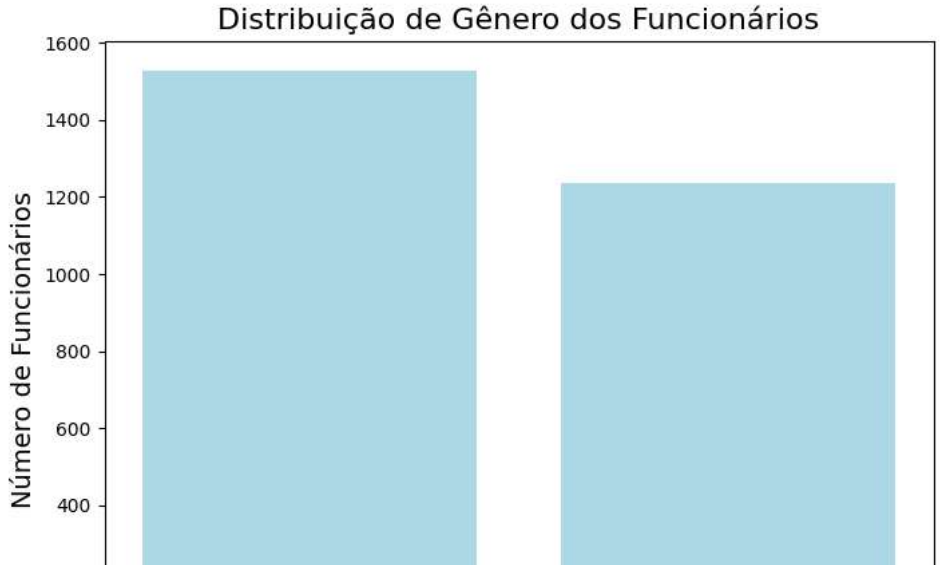


A idade dos funcionários, tem um comportamento próximo a uma distribuição normal? (Pode-se usar o histograma para verificação)

```
# Contar a distribuição dos gêneros dos funcionários
genero_distribuicao = df['Gender'].value_counts()

# Preparar os dados para o gráfico de colunas
generos = genero_distribuicao.index
quantidades = genero_distribuicao.values

# Criar um gráfico de colunas
plt.figure(figsize=(8, 6))
plt.bar(generos, quantidades, color='lightblue')
plt.xlabel('Gênero', fontsize=14)
plt.ylabel('Número de Funcionários', fontsize=14)
plt.title('Distribuição de Gênero dos Funcionários', fontsize=16)
plt.show()
```



Qual o gênero predominante? (Pode-se usar o gráfico de colunas ou de pizza)


```
# Criar um histograma da idade dos funcionários
plt.figure(figsize=(10, 6))
plt.hist(df['Age'], bins=20, color='lightblue', edgecolor='black')
plt.xlabel('Idade dos Funcionários', fontsize=14)
plt.ylabel('Número de Funcionários', fontsize=14)
plt.title('Distribuição da Idade dos Funcionários', fontsize=16)
plt.grid(True)
plt.show()
```

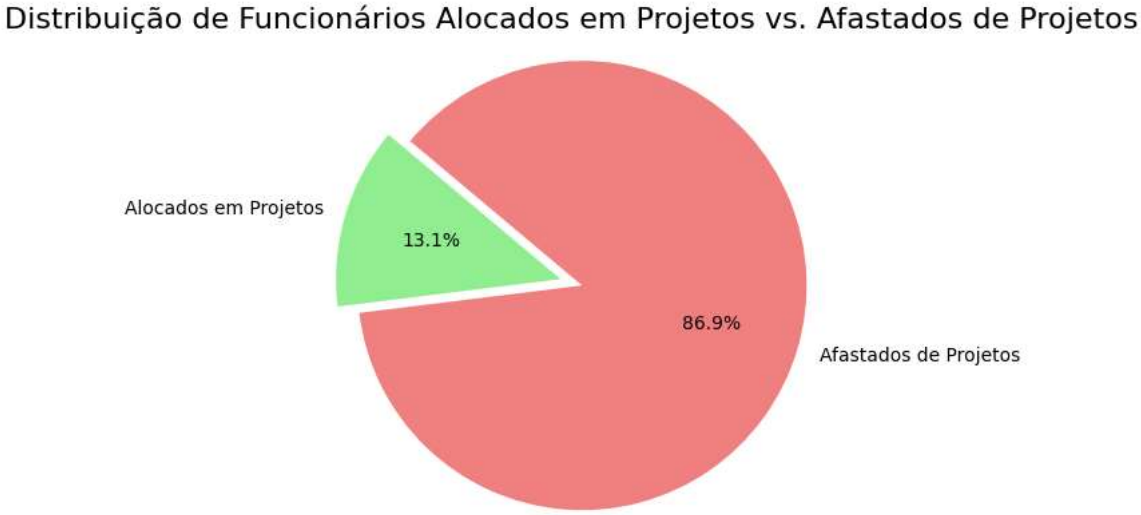
A quantidade de funcionários afastados dos projetos é relevante? (Pode-se usar o gráfico de colunas ou de pizza)

```
# Contar a distribuição de funcionários alocados em projetos e afastados
alocados_em_projetos = df[df['EverBenched'] == 'Yes']
afastados_de_projetos = df[df['EverBenched'] == 'No']

# Preparar os dados para o gráfico de pizza
labels = ['Alocados em Projetos', 'Afastados de Projetos']
sizes = [len(alocados_em_projetos), len(afastados_de_projetos)]
colors = ['lightgreen', 'lightcoral']
explode = (0.1, 0)  # Explodir a primeira fatia (Alocados em Projetos)

# Criar um gráfico de pizza
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
plt.axis('equal')  # Assegura que o gráfico de pizza seja um círculo.

plt.title('Distribuição de Funcionários Alocados em Projetos vs. Afastados de Projetos', fontsize=16)
plt.show()
```



Agrupar colunas para analisar dados(técnica do Groupby)

```
#Como agrupar colunas para analisar dados(técnica do Groupby)

cols_=['Formacao','PaymentTier']
filtro_cols=df.filter(items=cols_)
filtro_cols
```

	Formacao	PaymentTier
0	Bachelors	3
1	Bachelors	1
2	Bachelors	3
3	Masters	3
4	Masters	3
...	...	...
4645	Masters	2
4647	Bachelors	3
4649	Masters	2
4650	Masters	3
4651	Bachelors	3

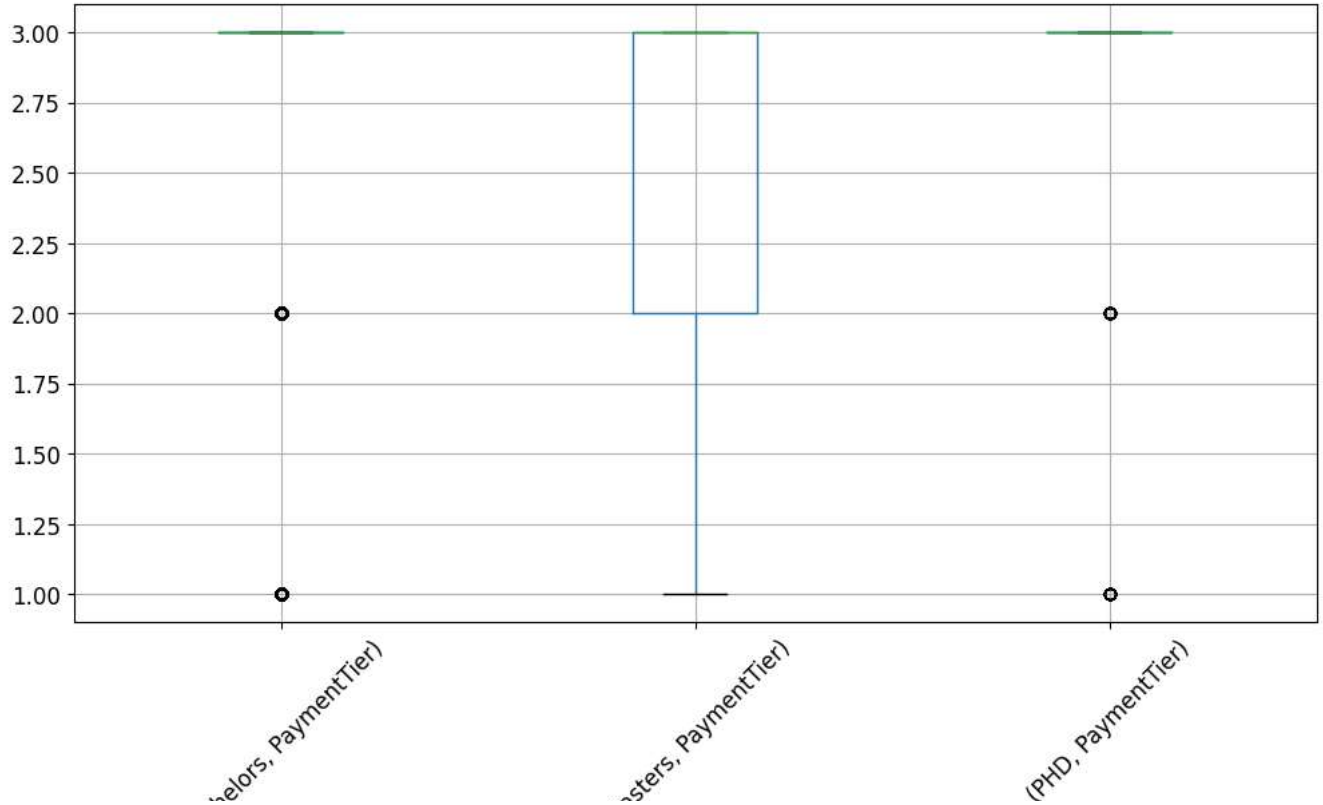
2764 rows × 2 columns

```
nivel=filtro_cols.groupby(['Formacao'])
nivel.agg({'PaymentTier':['mean','median','std','count']})
```

	PaymentTier			
	mean	median	std	count
Formacao				
Bachelors	2.677321	3.0	0.609059	1971
Masters	2.492936	3.0	0.648193	637
PHD	2.698718	3.0	0.626510	156

```
plt.figure(figsize=(12,6))
nivel.boxplot(subplots=False, rot=45, fontsize=12);
```

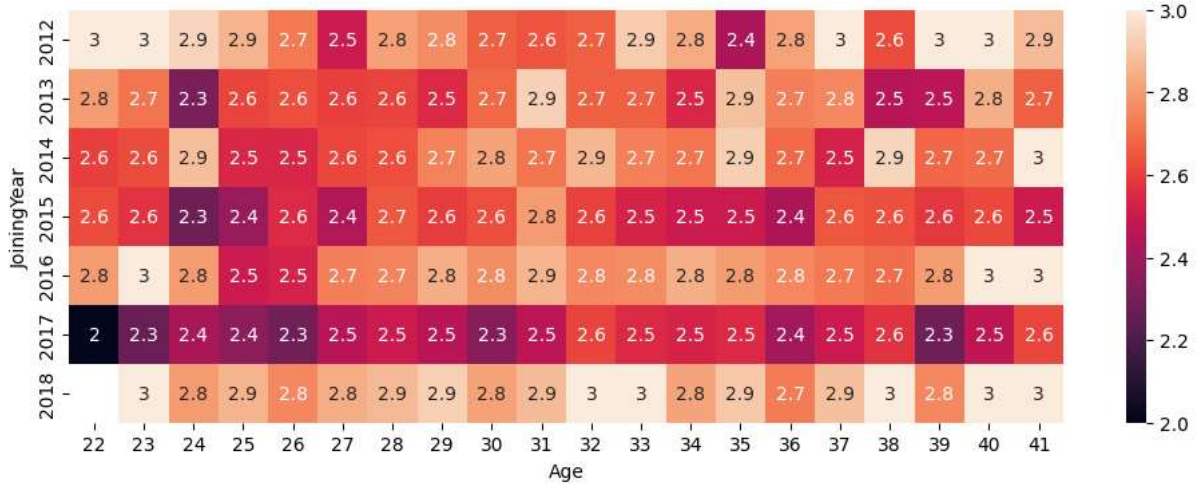
```
<ipython-input-18-f0670601079e>:2: FutureWarning: In a future version of pandas, a length 1 tuple will be returned when iterating over a groupby with a grouper equal to a list of length 1. Don't s
  nivel.boxplot(subplots=False, rot=45, fontsize=12);
```




```
anos_idade=df.groupby(['JoiningYear','Age'])
anos_idade
anos_idade['PaymentTier'].mean()
```

```
JoiningYear  Age
2012         22     3.000000
             23     3.000000
             24     2.941176
             25     2.857143
             26     2.722222
                       ...
2018         37     2.888889
             38     3.000000
             39     2.750000
             40     3.000000
             41     3.000000
Name: PaymentTier, Length: 139, dtype: float64
```

```
df_1 = df.pivot_table(index='JoiningYear', columns = 'Age', values= 'PaymentTier')
plt.figure(figsize=(12,4))       # tabela dinâmica (pivot table)
sns.heatmap(df_1,annot=True);
```



df_1

Age	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	4
JoiningYear																				
2012	3.000	3.000000	2.941176	2.857143	2.722222	2.500000	2.810811	2.761905	2.666667	2.642857	2.666667	2.916667	2.833333	2.428571	2.812500	3.000000	2.625000	3.000000	3.000000	2.87500
2013	2.800	2.714286	2.310345	2.612903	2.629630	2.600000	2.612245	2.550000	2.685714	2.933333	2.666667	2.666667	2.545455	2.888889	2.733333	2.750000	2.454545	2.461538	2.846154	2.66666
2014	2.600	2.625000	2.882353	2.545455	2.541667	2.615385	2.629630	2.741935	2.818182	2.714286	2.866667	2.736842	2.714286	2.923077	2.684211	2.529412	2.944444	2.681818	2.692308	3.00000
2015	2.625	2.571429	2.280000	2.384615	2.551724	2.441176	2.652174	2.593750	2.629630	2.800000	2.608696	2.533333	2.500000	2.500000	2.428571	2.647059	2.636364	2.583333	2.619048	2.50000
2016	2.800	3.000000	2.800000	2.500000	2.529412	2.730769	2.743590	2.840000	2.761905	2.857143	2.750000	2.764706	2.846154	2.800000	2.750000	2.722222	2.692308	2.800000	3.000000	3.00000
2017	2.000	2.272727	2.418605	2.357143	2.295455	2.454545	2.500000	2.459459	2.340909	2.466667	2.611111	2.548387	2.531250	2.548387	2.407407	2.500000	2.571429	2.285714	2.470588	2.61538
2018	NaN	3.000000	2.800000	2.857143	2.750000	2.842105	2.870968	2.928571	2.812500	2.875000	3.000000	3.000000	2.818182	2.909091	2.727273	2.888889	3.000000	2.750000	3.000000	3.00000

Análise Exploratória dos Dados - Análise Univariada

Separando os grupos LeaveOrNot

```
#NÃO SAÍRAM DA EMPRESA
df_0 = df.loc[df.LeaveOrNot == 0]

#SAÍRAM DA EMPRESA
df_1 = df.loc[df.LeaveOrNot == 1]
```

```
# CRIANDO FUNÇÃO PROBABILIDADE DE OCORRER UM EVENTO
def probab (A, E):
  resultado = (A / E)*100
  print('{:.2f}'.format(resultado))


# CRIANDO FUNÇÃO PROBABILIDADE DE NÃO OCORRER UM EVENTO
def probab_not (A, E):
  resultado = (1- (A / E))*100
  print('{:.2f}'.format(resultado))


# PROBABILIDADE DO FUNCIONÁRIO SAIR DA EMPRESA
p_sair = probab (len(df_1), len(df))
```

```
39.36
```

```
# PROBABILIDADE DO FUNCIONÁRIO NÃO SAIR DA EMPRESA
probab_not (len(df_1), len(df))
```

```
60.64
```

Variavel Education

```
#CONTAR QUANTOS VALORES ESTÃO DISPONÍVEIS NA COLUNA 'Formação'
df['Formacao'].value_counts()
```

```
Bachelors    1971
Masters       637
PHD           156
Name: Formacao, dtype: int64
```

```
#SEPARAR EM GRUPOS DE FORMAÇÃO
bachelors = df.loc[df.Formacao == 'Bachelors']
masters = df.loc[df.Formacao == 'Masters']
PHD = df.loc[df.Formacao == 'PHD']


#PROBABILIDADE FUNCIONARIO SER BACHAREL
p_bachelors = probab (len(bachelors), len(df))
```

```
71.31
```

```
#PROBABILIDADE FUNCIONARIO SER MESTRE
p_masters = probab (len(masters), len(df))
```

```
23.05
```

```
#PROBABILIDADE FUNCIONARIO SER PHD
p_PHD = probab (len(PHD), len(df))
```

```
5.64
```

```
#SEPARAR EM GRUPOS DE FORMAÇÃO QUE SAÍRAM DA EMPRESA
bachelors_1 = df_1.loc[df.Formacao == 'Bachelors']
masters_1 = df_1.loc[df.Formacao == 'Masters']
PHD_1 = df_1.loc[df.Formacao == 'PHD']


#CRIANDO UMA FUNÇÃO PARA PROBABILIDADE CONDICIONAL
def probab_cond (AB, B):
  resultado = (AB / B)*100
  print('{:.2f}'.format(resultado))


bachelors_1["Formacao"].value_counts()
#PROBABILIDADE FUNCIONARIO SER BACHAREL E SAIR DA EMPRESA
probab_cond(len(bachelors_1),len(df_1))
```

```
     67.92
```

```
masters_1["Formacao"].value_counts()
#PROBABILIDADE FUNCIONARIO SER MESTRE E SAIR DA EMPRESA
probab_cond(len(masters_1),len(df_1))
```

```
     28.40
```

primeira conclusão: a probabilidade de saída de bacharéis (67.92) supera a saída de mestres e PHDs (28.40 + 3.68)

Realizar a separação dos outros grupos que podem ser considerados como categóricos e analisar cada uma das probabilidades

Variavel Cidade

```
#CONTAR QUANTOS VALORES ESTÃO DISPONÍVEIS NA COLUNA Cidade
df["City"].value_counts()

#SEPARAR EM GRUPOS DE Cidade
pune = df.loc[df.City == 'Pune']
newDelhi = df.loc[df.City == 'New Delhi']
bangalore = df.loc[df.City == 'Bangalore']


#PROBABILIDADE DO FUNCIONARIO SER DE PUNE
p_pune = probab (len(pune), len(df))

#PROBABILIDADE DO FUNCIONARIO SER DE NEW DELHI
p_newDelhi  = probab (len(newDelhi), len(df))

#PROBABILIDADE DO FUNCIONARIO SER DE BANGALORE
p_bangalore= probab (len(bangalore), len(df))
```

```
     28.98
     28.65
```

```
42.37
```

```
#SEPARAR EM GRUPOS DE CIDADE QUE SAÍRAM DA EMPRESA
pune_1 = df_1.loc[df.City == 'Pune']
newDelhi_1 = df_1.loc[df.City == 'New Delhi']
bangalore_1 = df_1.loc[df.City == 'Bangalore']
```

```
pune_1["City"].value_counts()
#PROBABILIDADE DO FUNCIONARIO SER PUNE E SAIR DA EMPRESA
probab_cond(len(pune_1),len(df_1))
```

```
37.50
```

```
bangalore_1["City"].value_counts()
#PROBABILIDADE DO FUNCIONARIO SER DE BANGALORE E SAIR DA EMPRESA
probab_cond(len(bangalore_1),len(df_1))
```

```
37.68
```

```
newDelhi_1["City"].value_counts()
#PROBABILIDADE DO FUNCIONARIO SER DE NEW DELHI E SAIR DA EMPRESA
probab_cond(len(newDelhi_1),len(df_1))
```

```
24.82
```

A probabilidade de um funcionario que mora em New Delhi sair é menor que funcionarios das cidades de Bangalore e Pune. Curioso notar que Funcionarios de Pune e Bangalore tem praticamente as mesmas chances de sair, mesmo Bangalore tendo bem mais funcioarios que Pune.

```
df.rename(columns={'Niveis salariais': 'PaymentTier'}, inplace=True)
```

```
#Variável Nivel Salarial

#CONTAR QUANTOS VALORES ESTÃO DISPONÍVEIS NA COLUNA NivelSalarial
df["PaymentTier"].value_counts()

#SEPARAR EM GRUPOS DE Nivel Salarial
um = df.loc[df.PaymentTier ==  1]
dois = df.loc[df.PaymentTier == 2]
tres= df.loc[df.PaymentTier == 3]
```

```
#PROBABILIDADE DE NIVEL SALARIAL DO FUNCIONARIO SER 1
p_um  = probab (len(um ), len(df))

#PROBABILIDADEDE NIVEL SALARIAL DO FUNCIONARIO SER 2
p_dois  = probab (len(dois), len(df))

#PROBABILIDADEDE NIVEL SALARIAL DO FUNCIONARIO SER 3
p_tres= probab (len(tres), len(df))
```

```
7.89
20.62
```

```
71.49
```

```
#SEPARAR EM GRUPOS DE NIVEL DE PAGAMENTO QUE SAÍRAM DA EMPRESA
um_1 = df_1.loc[df.PaymentTier == 1]
dois_1 = df_1.loc[df.PaymentTier == 2]
tres_1 = df_1.loc[df.PaymentTier == 3]
```

```
df_1.rename(columns={'Niveis salariais': 'PaymentTier'}, inplace=True)
```

```
<ipython-input-85-1bae4e764c23>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_1.rename(columns={'PaymentTier': 'NivelSalarial'}, inplace=True)
```

```
um_1["PaymentTier"].value_counts()
#PROBABILIDADE FUNCIONARIO TER NIVEL SALARIAL 1 E SAIR DA EMPRESA
probab_cond(len(um_1),len(df_1))
```

```
7.08
```

```
dois_1["PaymentTier"].value_counts()
#PROBABILIDADE FUNCIONARIO TER NIVEL SALARIAL 2 E SAIR DA EMPRESA
probab_cond(len(dois_1),len(df_1))
```

```
31.53
```

```
tres_1["PaymentTier"].value_counts()
#PROBABILIDADE FUNCIONARIO TER NIVEL SALARIAL 3 E SAIR DA EMPRESA
probab_cond(len(tres_1),len(df_1))
```

```
61.40
```

Funcionarios com nivel salarial 3 tem muito mais chances de sair da empresa que funcionarios com nivel salarial 1 e 2. Com os funcionarios de nivel salarial 1 tendo menos probilidade de sair.

Separação e Analise de da Probablilidade da coluna Genero

```
#CONTAR QUANTOS VALORES ESTÃO DISPONÍVEIS NA COLUNA Genero
df["Gender"].value_counts()

#SEPARAR EM GRUPOS DE Genero
male = df.loc[df.Gender == 'Male']
female = df.loc[df.Gender == 'Female']
```

```
#PROBABILIDADE DO FUNCIONARIO DE FUNCIONARIO SER HOMEM
p_male = probab (len(male), len(df))
```

```
#PROBABILIDADE DO FUNCIONARIO SER MULHER
p_female = probab (len(female), len(df))
```

```
55.32
44.68
```

```
#SEPARAR EM GRUPOS DE GENERO QUE SAÍRAM DA EMPRESA
male_1 = df_1.loc[df.Gender == 'Male']
female_1 = df_1.loc[df.Gender == 'Female']
```

```
male_1["Gender"].value_counts()
#PROBABILIDADE FUNCIONARIO SER HOMEM E SAIR DA EMPRESA
probab_cond(len(male_1),len(df_1))
```

```
43.57
```

```
female_1["Gender"].value_counts()
#PROBABILIDADE FUNCIONARIO SER MULHER E SAIR DA EMPRESA
probab_cond(len(female_1),len(df_1))
```

```
56.43
```

A probabidade de uma mulher sair da empresa é maior que a de um homem.

Separação e Analise de da Probablilidade da coluna EverBenched

```
df.rename(columns={'Sem Projeto': 'EverBenched'}, inplace=True)
```

```
#CONTAR QUANTOS VALORES ESTÃO DISPONÍVEIS NA COLUNA EverBenched
df["EverBenched"].value_counts()

#SEPARAR EM GRUPOS DE QUE JA FICARAM SEM PROJETO OU NÃO
sim = df.loc[df.EverBenched == 1]
nao = df.loc[df.EverBenched == 0]
```

```
#PROBABILIDADE DO FUNCIONARIO TER FICADO SEM PROJETO
p_sim = probab (len(sim), len(df))

##PROBABILIDADE DO FUNCIONARIO NÃO TER FICADO SEM PROJETO
p_nao  = probab (len(nao), len(df))
```

```
0.00
0.00
```

```
#SEPARAR EM GRUPOS QUE FICARAM SEM PROJETO QUE SAÍRAM DA EMPRESA
sim_1 = df_1.loc[df.EverBenched == 1]
nao_1 = df_1.loc[df.EverBenched == 0]
```