Entendendo o problema de negócio

Esta base foi disponibilizada pela IBM e possuía diversas variáveis para realizarmos a previssão. Nosso objetivo é maximizar o precision score, tendo em vista que, o intuíto da empresa é identificar quais colaboradores pretendem pedir demissão. Dessa forma, podemos criar planos para reter esses funcionários que possam sair, caso seja uma pessoa chave o time, ou até mesmo evitar de demitir alguém que possivelmente já possa pedir demissão, assim economizando com encargos trabalhistas.

Agora que já sabemos o problema de negócio, vamos criar um modelo tentanto utilizar a menor quantidade de técnicas para atingir um resultado de pelo menos 85% no precision. Fica como desafio ou parte 2 desse case, utilizar mais técnicas para tentar um resultado ainda melhor. Ao longo da resolução, irei dando algumas sugestões caso queiram testar.

Importando as bibliotecas

In [1]:
```
# Lib para manipulação de dados
import pandas as pd
import numpy as np

# Lib para configuração do pandas
pd.set_option('display.max_columns', 200)
pd.set_option('display.max_rows', 200)

# Lib para configuração de warnings
import warnings
warnings.filterwarnings('ignore')
SEED = 123

# Lib para visualização de dados
import seaborn as sns
import matplotlib.pyplot as plt

# Lib para machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import (cross_validate,
                                     RandomizedSearchCV,
                                     train_test_split,
                                     StratifiedKFold,
                                     GridSearchCV)
from imblearn.over_sampling import RandomOverSampler
from sklearn.metrics import precision_score, recall_score, accuracy_score, confusi
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import (MinMaxScaler,
                                   StandardScaler,
                                   RobustScaler,
                                   OneHotEncoder)
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline, FeatureUnion
```

```
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler, SMOTENC
from imblearn.pipeline import Pipeline as imbpipeline

# Lib para importação/exportação do modelo
import joblib
```

Importando o dataset

In [2]:
```
df = pd.read_csv('WA_Fn-UseC_-HR-Employee-Attrition.csv')
df.head()
```

Out[2]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educa
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life
4	27	No	Travel_Rarely	591	Research & Development	2	1	

Análise Explorátoria dos Dados (EDA)

Verificando o tamanho do dataset

In [3]:
```
df.shape
```

Out[3]:
```
(1470, 35)
```

Conclusão: O dataframe possui 1470 linhas e 35 colunas.

Verificando os tipos das colunas

In [4]:
```
pd.DataFrame(df.dtypes, columns = ['Tipo'])
```

Out[4]:

	Tipo
Age	int64
Attrition	object
BusinessTravel	object
DailyRate	int64
Department	object
DistanceFromHome	int64
Education	int64
EducationField	object
EmployeeCount	int64
EmployeeNumber	int64
EnvironmentSatisfaction	int64
Gender	object
HourlyRate	int64
JobInvolvement	int64
JobLevel	int64
JobRole	object
JobSatisfaction	int64
MaritalStatus	object
MonthlyIncome	int64
MonthlyRate	int64
NumCompaniesWorked	int64
Over18	object
OverTime	object
PercentSalaryHike	int64
PerformanceRating	int64
RelationshipSatisfaction	int64
StandardHours	int64
StockOptionLevel	int64
TotalWorkingYears	int64
TrainingTimesLastYear	int64
WorkLifeBalance	int64
YearsAtCompany	int64
YearsInCurrentRole	int64
YearsSinceLastPromotion	int64
YearsWithCurrManager	int64

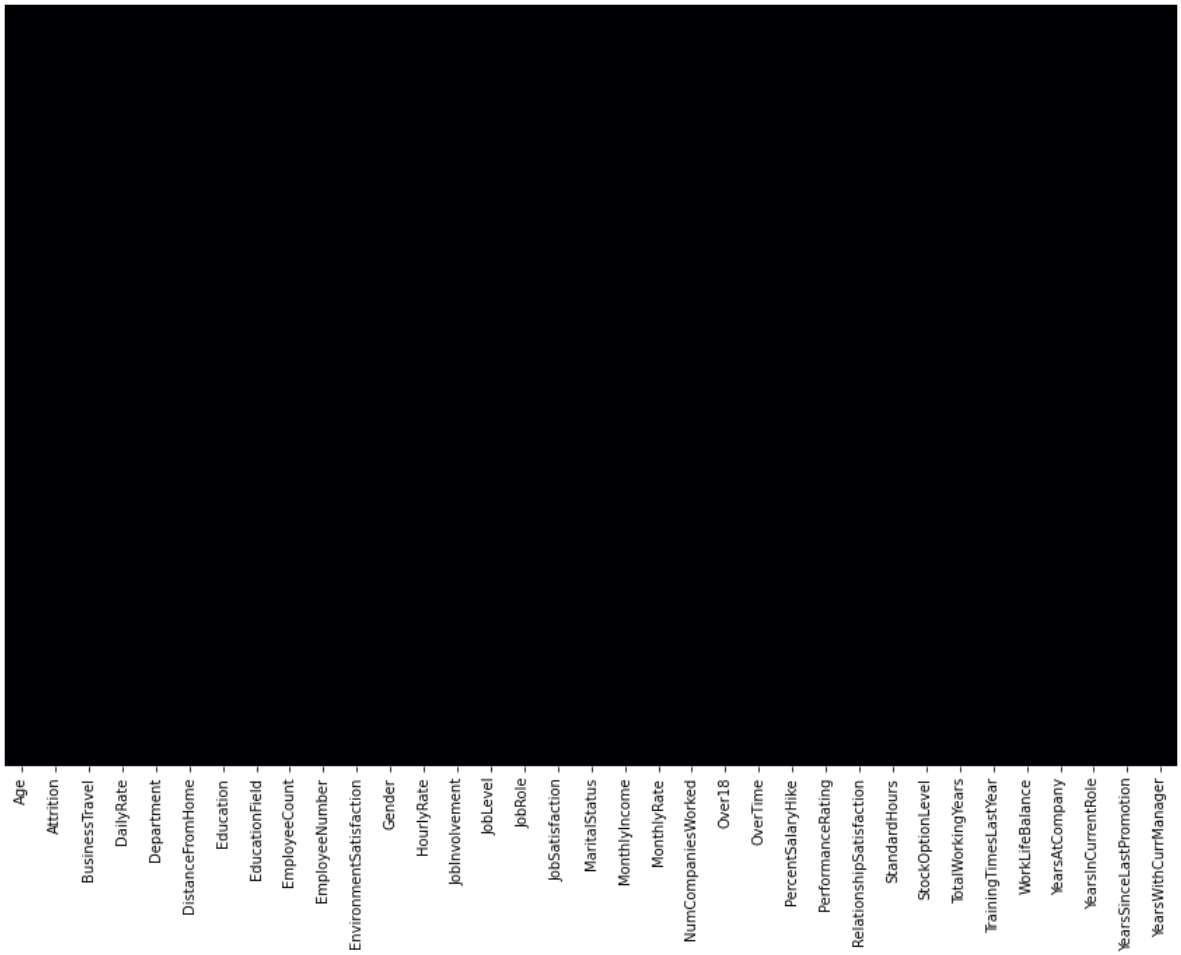
Conclusão: Não possuímos nenhuma variável com a classificação errada.

Verificando se há valores nulos

In [5]:
```
df.isna().sum()
```

Out[5]:
```
Age                       0
Attrition                 0
BusinessTravel            0
DailyRate                 0
Department                0
DistanceFromHome          0
Education                 0
EducationField            0
EmployeeCount             0
EmployeeNumber            0
EnvironmentSatisfaction   0
Gender                    0
HourlyRate                0
JobInvolvement            0
JobLevel                  0
JobRole                   0
JobSatisfaction           0
MaritalStatus             0
MonthlyIncome             0
MonthlyRate               0
NumCompaniesWorked        0
Over18                    0
OverTime                  0
PercentSalaryHike         0
PerformanceRating         0
RelationshipSatisfaction  0
StandardHours             0
StockOptionLevel          0
TotalWorkingYears         0
TrainingTimesLastYear     0
WorkLifeBalance           0
YearsAtCompany            0
YearsInCurrentRole        0
YearsSinceLastPromotion   0
YearsWithCurrManager      0
dtype: int64
```

In [6]:
```
plt.figure(figsize=(15, 10))

sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='magma');
```



Conclusão: Não possuímos valores nulos.

Verificando se há linhas duplicadas

In [7]:
```
df.duplicated().sum()
```

Out[7]:
```
0
```

Conclusão: Não há linhas duplicadas.

Separando as variáveis numéricas das categóricas

In [8]:
```
# Variáveis numéricas
df_num = df.select_dtypes(include = 'number').columns

# Variáveis categóricas
df_cat = df.select_dtypes(include = 'object').columns

print('Variáveis Numéricas\n')
print(df_num)
print('\n----------------------------------------------------------------------------
print('Variáveis Categóricas\n')
print(df_cat)
```

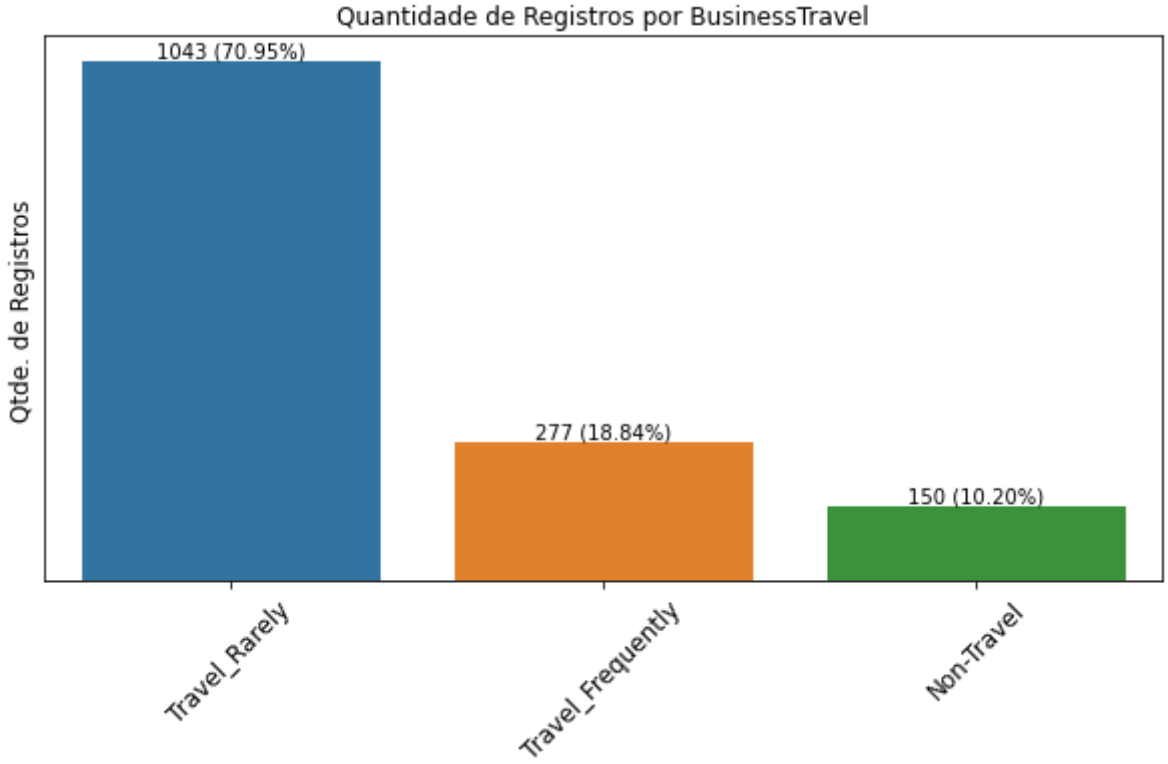
```
Variáveis Numéricas

Index(['Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobSatisfaction', 'MonthlyIncome',
       'MonthlyRate', 'NumCompaniesWorked', 'PercentSalaryHike',
       'PerformanceRating', 'RelationshipSatisfaction', 'StandardHours',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager'],
      dtype='object')

--------------------------------------------------------------------------------

Variáveis Categóricas

Index(['Attrition', 'BusinessTravel', 'Department', 'EducationField', 'Gender',
       'JobRole', 'MaritalStatus', 'Over18', 'OverTime'],
      dtype='object')
```

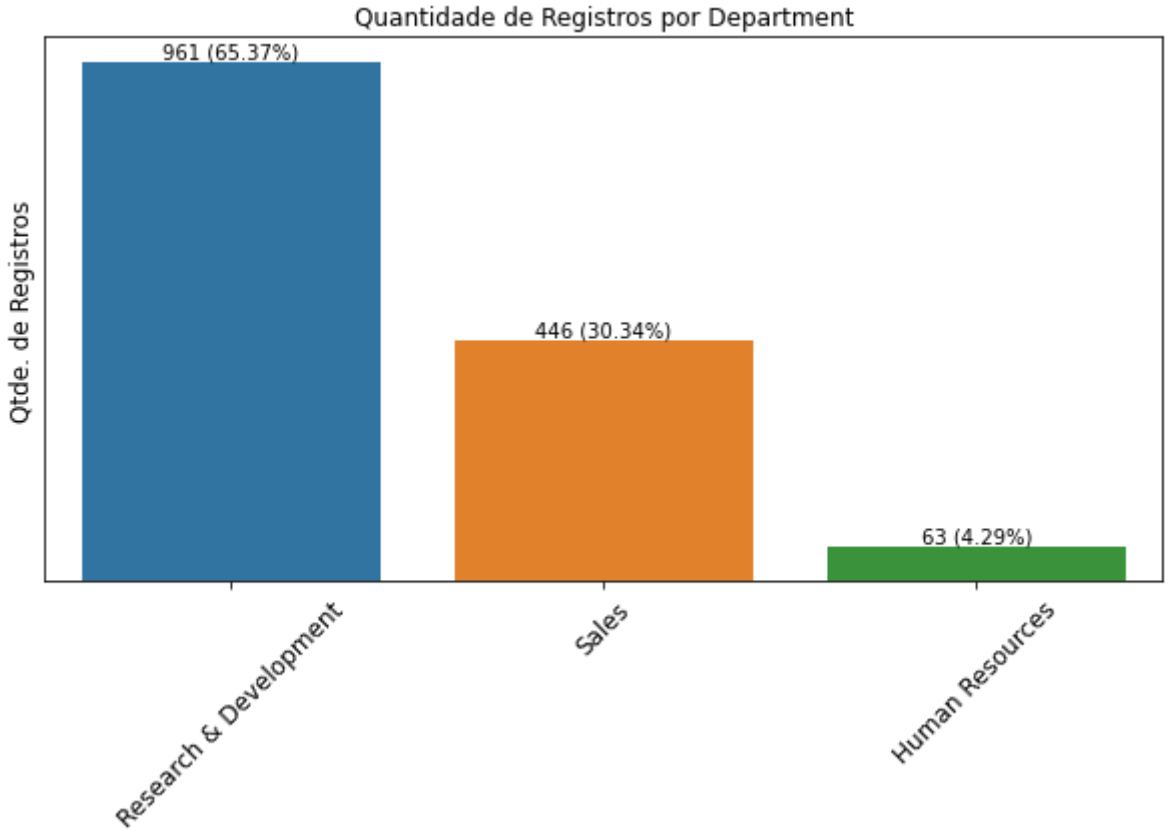
Visualizando as frenquências das variáveis categóricas

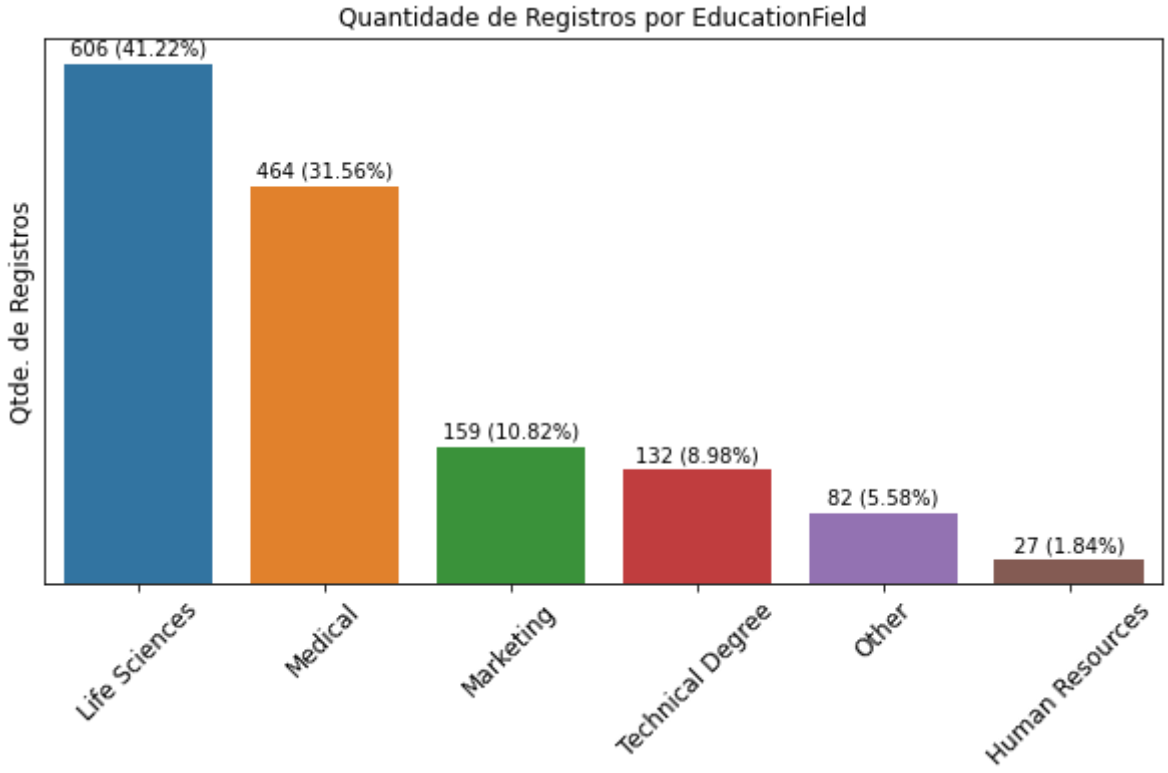
In [9]:

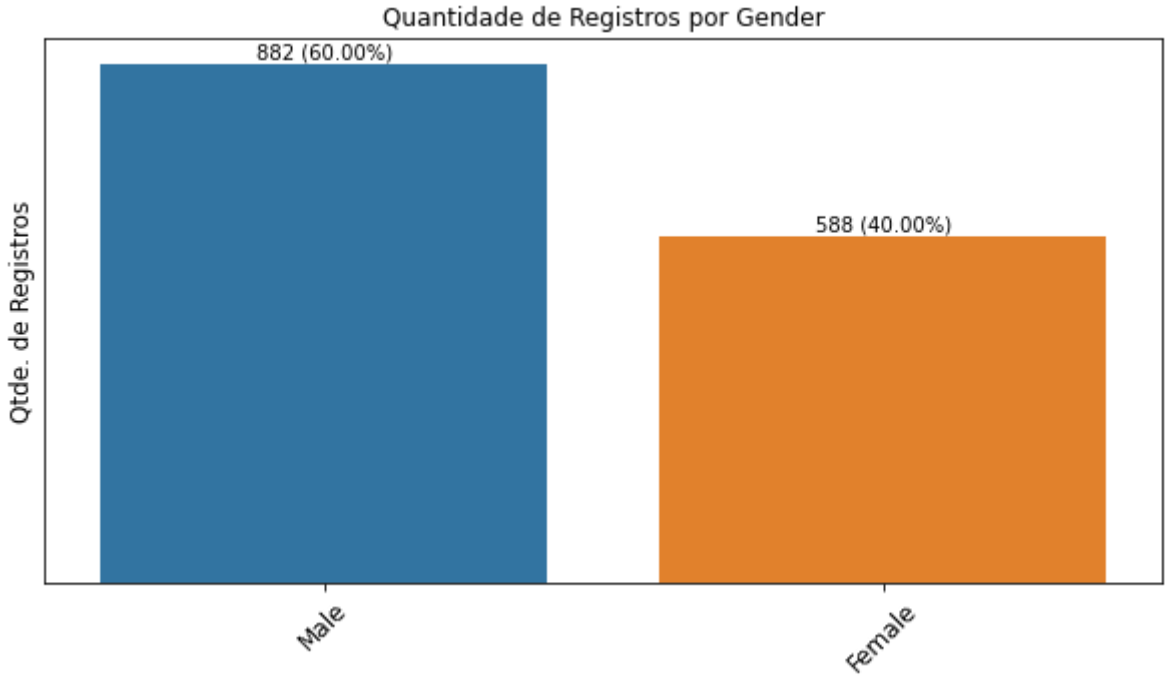
```
# Cria um looping percorrendo as variáveis categóricas
for coluna in df_cat:

    # Caso seja a coluna Attrition, ele fará nada
    if coluna == 'Attrition':
        pass
    else:
        # Cria uma figura com esses pixels
        plt.figure(figsize=(10, 5))

        # Pega o total de linhas
        total = df.shape[0]

        # Plota o gráfico de barras
        ax = sns.countplot(x=coluna, data=df, order=df[coluna].value_counts().inde

        # Plota os valores com os totais e porcentagens encima das barras
        for p in ax.patches:
            rotulo = '{:1d} ({:.2f}%)'.format(p.get_height(), 100 * p.get_height()
            x = (p.get_x() + (p.get_width() / 2))
            y = p.get_height() + 10
            ax.annotate(rotulo, (x, y), ha='center', size=10)

        # Personaliza os ticks do eixo X
        ax.set_xticklabels(df[coluna].value_counts().index, size=12)
        # Personaliza os ticks do eixo Y
        ax.set_yticklabels(ax.get_yticks().astype(int), size=10)

        # Escreve um título
        plt.title(f'Quantidade de Registros por {coluna}', size=12)

        # Escreve o título do eixo Y
        plt.ylabel('Qtde. de Registros', size=12)

        # Retira o título do eixo X
        plt.xlabel("")

        # Rotaciona os ticks do eixo X
        plt.xticks(rotation=45)

        # Retira os ticks do eixo Y
```

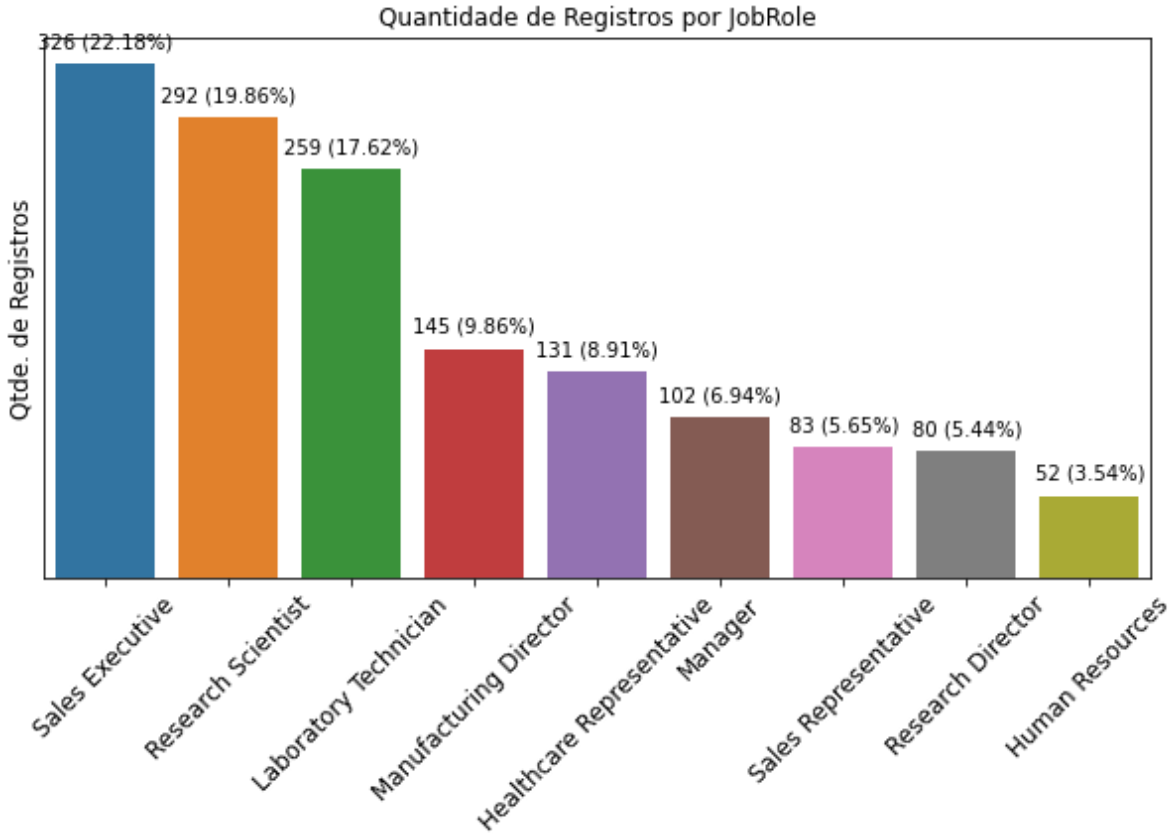
```
        plt.yticks([])

        # Plota os gráficos
        plt.show()
```

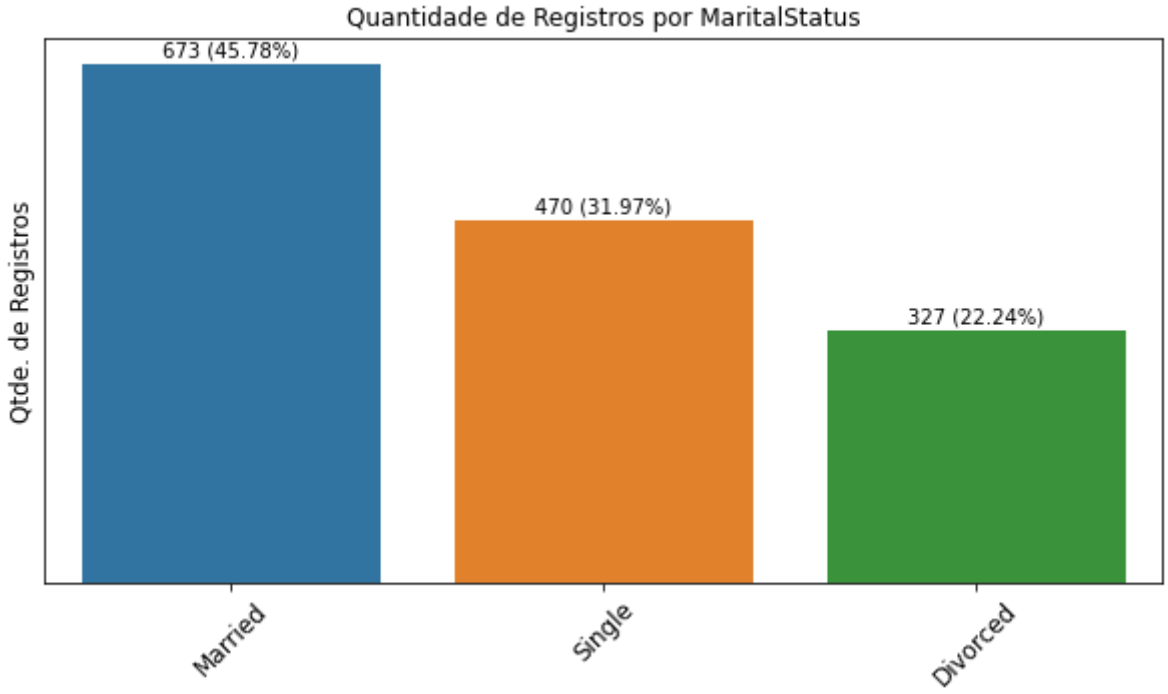


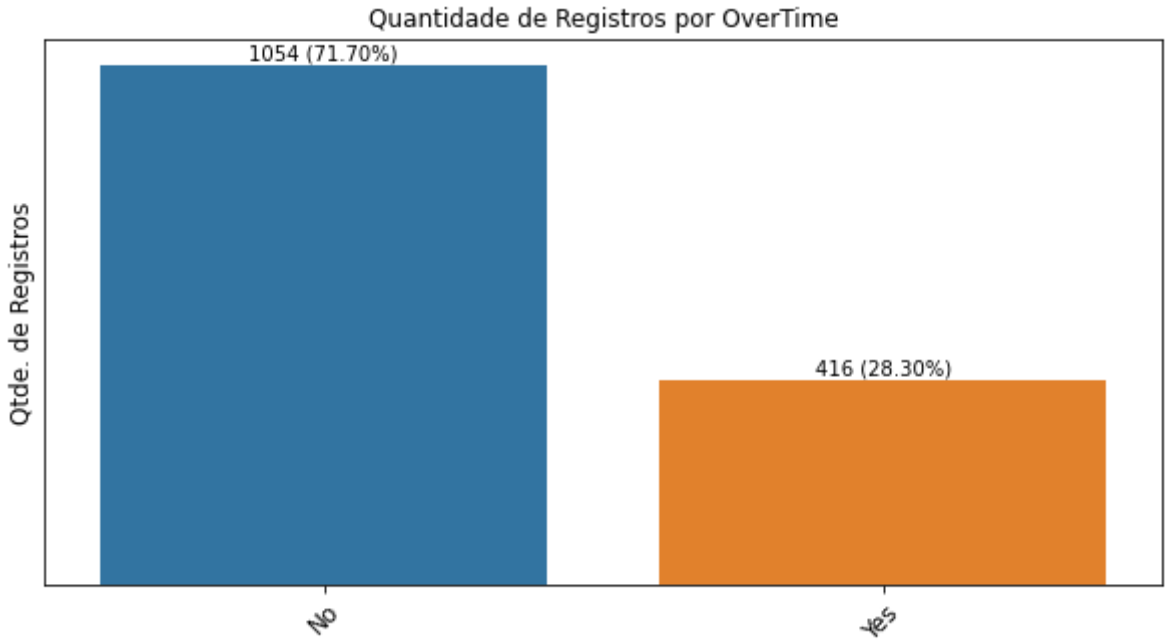



Quantidade de Registros por EducationField
Qtde. de Registros
606 (41.22%)
464 (31.56%)
159 (10.82%)
132 (8.98%)
82 (5.58%)
27 (1.84%)
Life Sciences
Medical
Marketing
Technical Degree
Other
Human Resources


Quantidade de Registros por Gender
Qtde. de Registros
882 (60.00%)
588 (40.00%)
Male
Female


Quantidade de Registros por JobRole
Qtde. de Registros
326 (22.18%)
292 (19.86%)
259 (17.62%)
145 (9.86%)
131 (8.91%)
102 (6.94%)
83 (5.65%)
80 (5.44%)
52 (3.54%)
Sales Executive
Research Scientist
Laboratory Technician
Manufacturing Director
Healthcare Representative
Manager
Sales Representative
Research Director
Human Resources


Quantidade de Registros por MaritalStatus
Qtde. de Registros
673 (45.78%)
470 (31.97%)
327 (22.24%)
Married
Single
Divorced



Conclusão: A variável Over18 só possui um único tipo de dados. Logo, podemos removê-la.

Visualizando as estatísticas das variáveis categóricas e numéricas

In [10]:
```
# Variáveis categóricas
df.describe(include = ['O']).T
```

Out[10]:

	count	unique	top	freq
Attrition	1470	2	No	1233
BusinessTravel	1470	3	Travel_Rarely	1043
Department	1470	3	Research & Development	961
EducationField	1470	6	Life Sciences	606
Gender	1470	2	Male	882
JobRole	1470	9	Sales Executive	326
MaritalStatus	1470	3	Married	673
Over18	1470	1	Y	1470
OverTime	1470	2	No	1054

Conclusão: Como já visto, a variável Over18 é constante. Logo, podemos excluir do nosso modelo.

In [11]:
```
# Variáveis categóricas
df.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	75%	
Age	1470.0	36.923810	9.135373	18.0	30.00	36.0	43.00	
DailyRate	1470.0	802.485714	403.509100	102.0	465.00	802.0	1157.00	1
DistanceFromHome	1470.0	9.192517	8.106864	1.0	2.00	7.0	14.00	
Education	1470.0	2.912925	1.024165	1.0	2.00	3.0	4.00	
EmployeeCount	1470.0	1.000000	0.000000	1.0	1.00	1.0	1.00	
EmployeeNumber	1470.0	1024.865306	602.024335	1.0	491.25	1020.5	1555.75	2
EnvironmentSatisfaction	1470.0	2.721769	1.093082	1.0	2.00	3.0	4.00	
HourlyRate	1470.0	65.891156	20.329428	30.0	48.00	66.0	83.75	
JobInvolvement	1470.0	2.729932	0.711561	1.0	2.00	3.0	3.00	
JobLevel	1470.0	2.063946	1.106940	1.0	1.00	2.0	3.00	
JobSatisfaction	1470.0	2.728571	1.102846	1.0	2.00	3.0	4.00	
MonthlyIncome	1470.0	6502.931293	4707.956783	1009.0	2911.00	4919.0	8379.00	19
MonthlyRate	1470.0	14313.103401	7117.786044	2094.0	8047.00	14235.5	20461.50	26
NumCompaniesWorked	1470.0	2.693197	2.498009	0.0	1.00	2.0	4.00	
PercentSalaryHike	1470.0	15.209524	3.659938	11.0	12.00	14.0	18.00	
PerformanceRating	1470.0	3.153741	0.360824	3.0	3.00	3.0	3.00	
RelationshipSatisfaction	1470.0	2.712245	1.081209	1.0	2.00	3.0	4.00	
StandardHours	1470.0	80.000000	0.000000	80.0	80.00	80.0	80.00	
StockOptionLevel	1470.0	0.793878	0.852077	0.0	0.00	1.0	1.00	
TotalWorkingYears	1470.0	11.279592	7.780782	0.0	6.00	10.0	15.00	
TrainingTimesLastYear	1470.0	2.799320	1.289271	0.0	2.00	3.0	3.00	
WorkLifeBalance	1470.0	2.761224	0.706476	1.0	2.00	3.0	3.00	
YearsAtCompany	1470.0	7.008163	6.126525	0.0	3.00	5.0	9.00	
YearsInCurrentRole	1470.0	4.229252	3.623137	0.0	2.00	3.0	7.00	
YearsSinceLastPromotion	1470.0	2.187755	3.222430	0.0	0.00	1.0	3.00	
YearsWithCurrManager	1470.0	4.123129	3.568136	0.0	2.00	3.0	7.00	

Conclusão: As variáveis EmployeeCount e StandardHours possuem desvio padrão 0, logo só possuem um único tipo de valor. Por conta disso, podemos excluir do nosso modelo.

Verificando variáveis que possuem baixa variância para serem removidas

In [12]:
```
def detect_low_variance(dataframe, threshold):
    """
    Esta função retorna uma lista com os nomes das colunas que possuem uma variânci
    """
```

```
    # Define um threshold para detectar variáveis com baixa variação
    threshold = threshold

    # Intância o normalizador
    scaler = MinMaxScaler()

    # Seleciona as vaiáveis numérias
    num_cols = dataframe.select_dtypes(include = [int, float])

    # Normaliza os dados das variáveis numéricas
    scaled_num_cols = scaler.fit_transform(num_cols)

    # Cria um dataframe com as variáveis normalizadas
    scaled_num_df = pd.DataFrame(scaled_num_cols,
                                 columns = num_cols.columns)

    # Cria uma lista vazia
    low_variance_columns = list()

    # Cria um looping que irá percorre todas as colunas
    for column in scaled_num_df:

        # Calcula a variância das colunas
        column_variance = scaled_num_df[column].var()

        # Verifica se a variância é menor que o threshold, se sim, irá adicionar na
        if column_variance < threshold:
            low_variance_columns.append(column)

    print(low_variance_columns)

detect_low_variance(df, 0.001)
```

```
['EmployeeCount', 'StandardHours']
```

Conclusão: Como visto acima, devemos excluir as variáveis EmployeeCount e StandardHours.

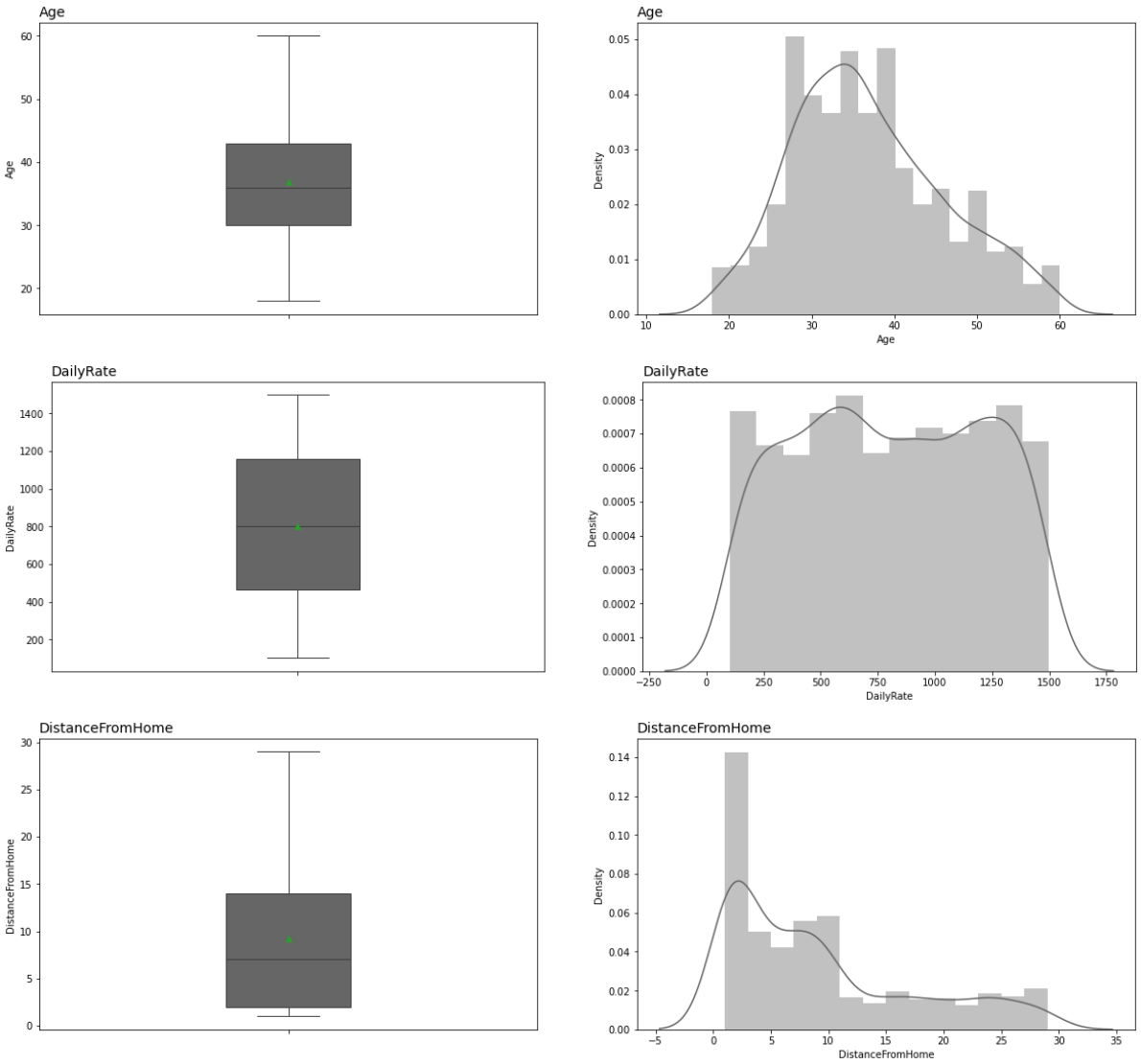
Visualizando a distribuição das variáveis numéricas

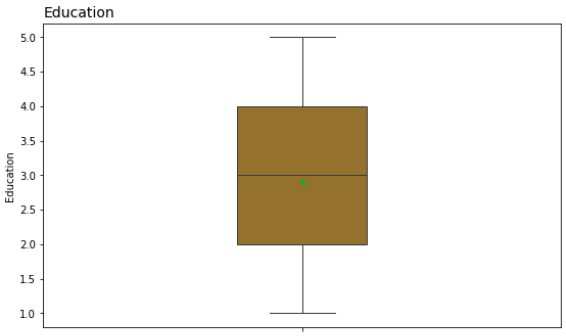
In [13]:

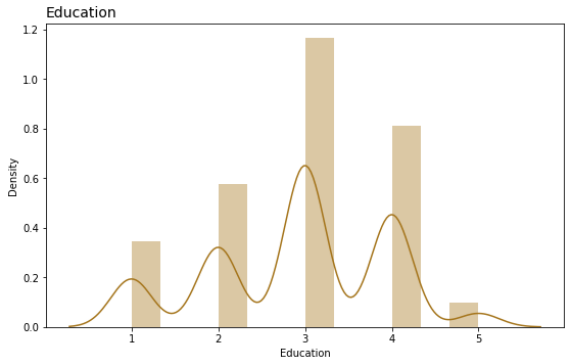
```
def plot_histbox(data, coluna):
    """
    Esta função retorno o plot de um boxplot e de um distplot de cada coluna
    """

    # Definindo o tamanho dos gráficos
    f, ax = plt.subplots ( figsize=(20, 5) )

    # Cor de fundo
    cor_fundo = '#FFFFFF'
    f.set_facecolor( cor_fundo )

    # Paleta de Cores
    paleta_cores = sns.color_palette( 'Dark2_r', len( data.columns ) * 2)

    # Plot no Grid -- Boxplot
    plt.subplot( 1, 2, 1 )

    # Titulo
    plt.title( f'{coluna}', loc='left', fontsize=14, fontweight=200 )
```

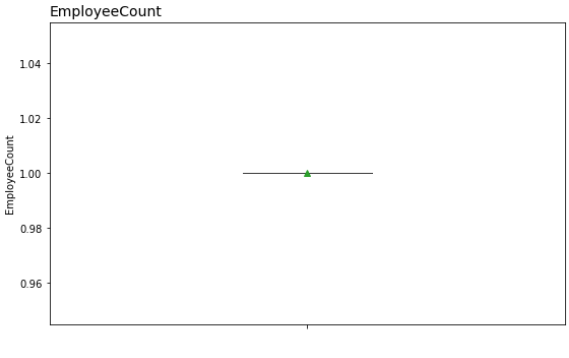
```
    # Plot
    sns.boxplot( data=data, y=coluna, showmeans=True, saturation=0.75,
                linewidth=1, width=0.25, color=paleta_cores[ list(df.columns).index

    # Plot no grid -- Distplot
    plt.subplot( 1, 2, 2 )

    # Titulo
    plt.title( f'{coluna}', loc='left', fontsize=14, fontweight=200 )

    # Plot
    sns.distplot( df[ coluna ], color=paleta_cores[ list(df.columns).index(coluna)

    # Ajustando o Grid
    plt.subplots_adjust( top=0.95, hspace=0.3 )
```

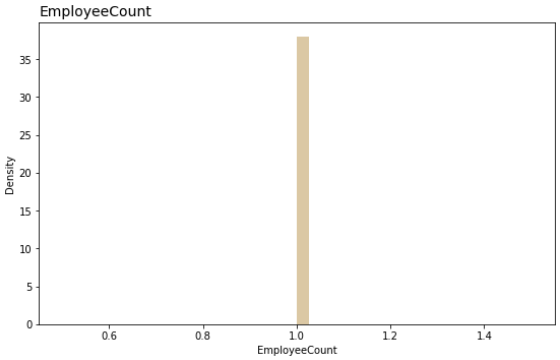
In [14]:
```
for coluna in df_num:
    plot_histbox(df[df_num], coluna)
```

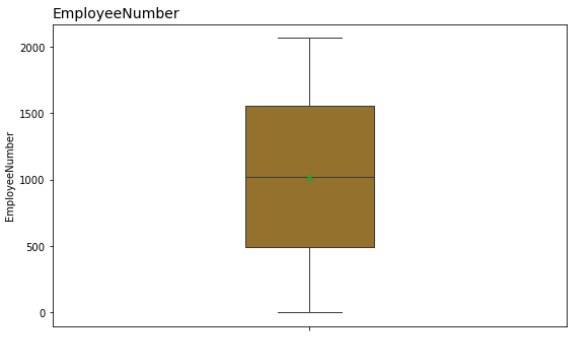


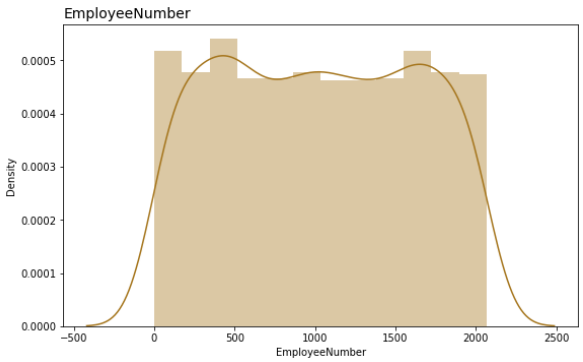

Education
Education

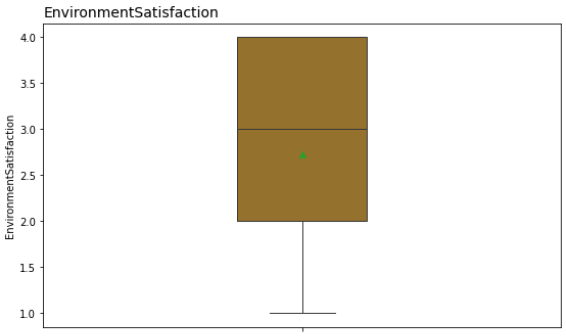

Education
Density
Education

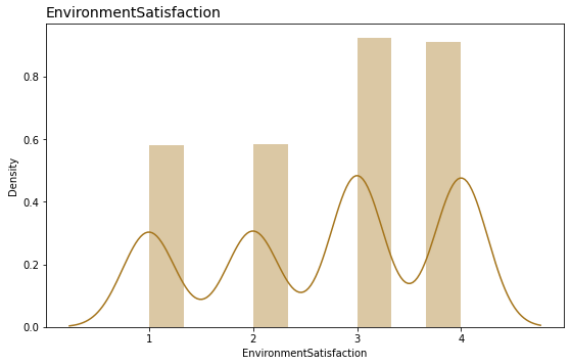

EmployeeCount
EmployeeCount

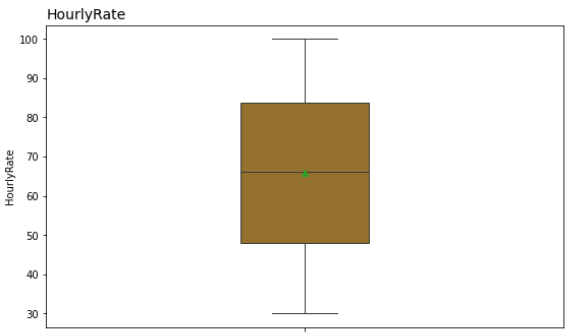

EmployeeCount
Density
EmployeeCount

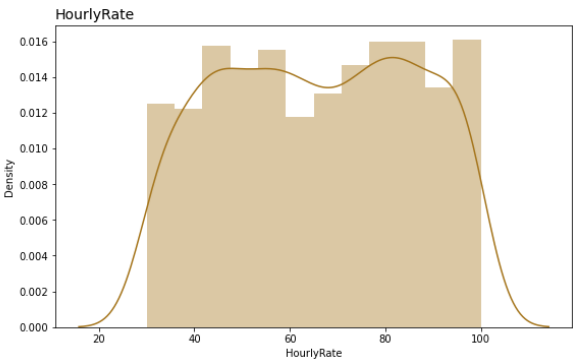

EmployeeNumber
EmployeeNumber

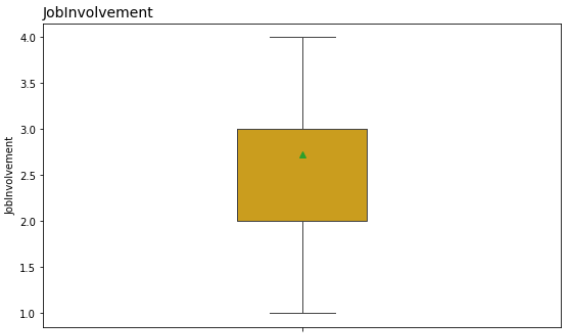

EmployeeNumber
Density
EmployeeNumber

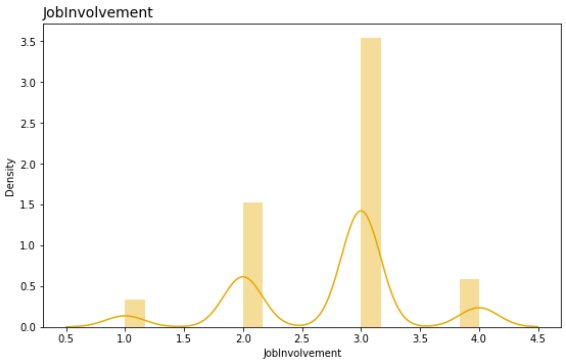

EnvironmentSatisfaction
EnvironmentSatisfaction

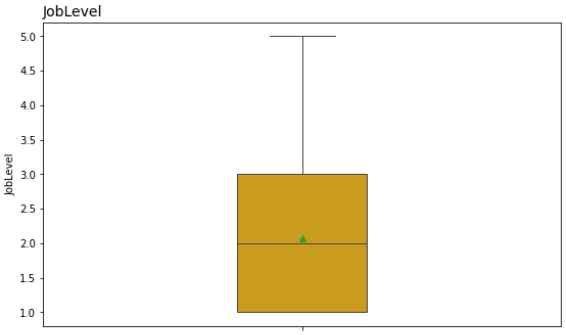

EnvironmentSatisfaction
Density
EnvironmentSatisfaction

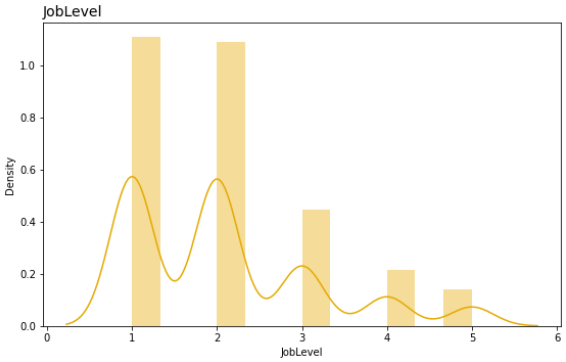

HourlyRate
HourlyRate

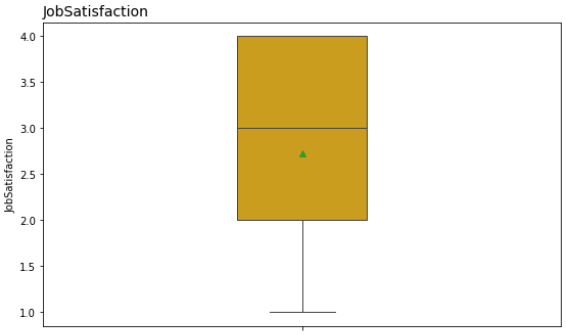

HourlyRate
Density
HourlyRate

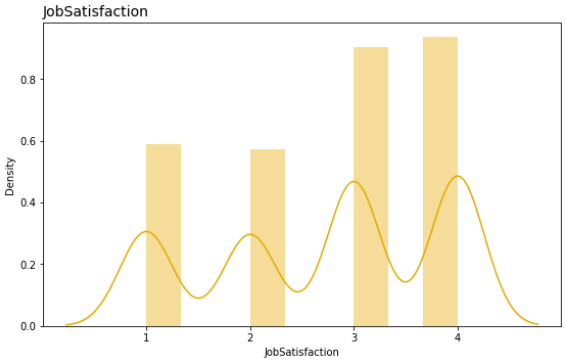

JobInvolvement
JobInvolvement

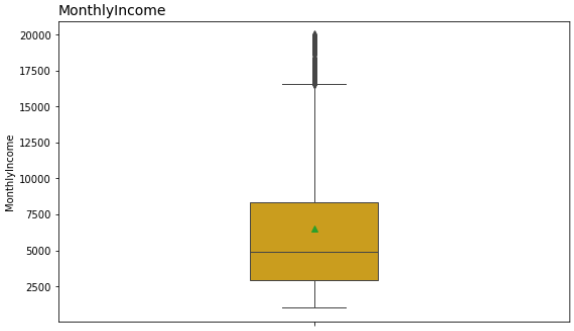

JobInvolvement
Density
JobInvolvement

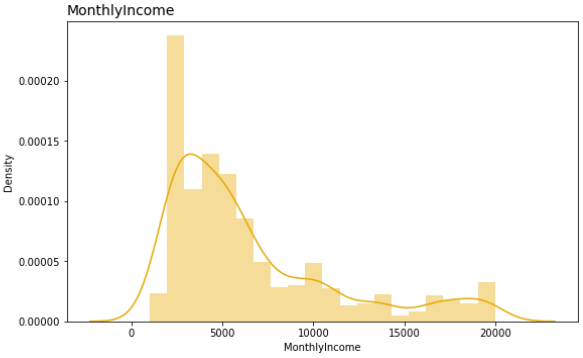

JobLevel
JobLevel

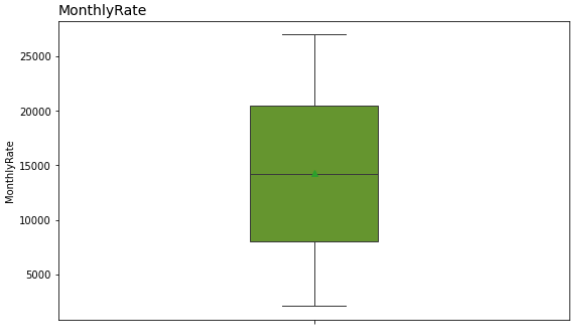

JobLevel
Density
JobLevel

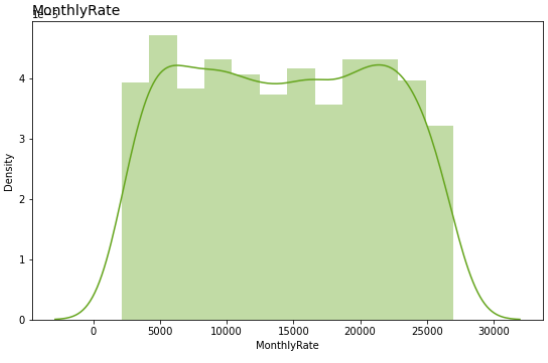

JobSatisfaction
JobSatisfaction

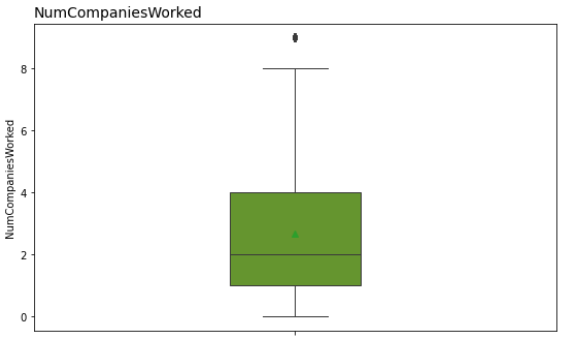

JobSatisfaction
Density
JobSatisfaction

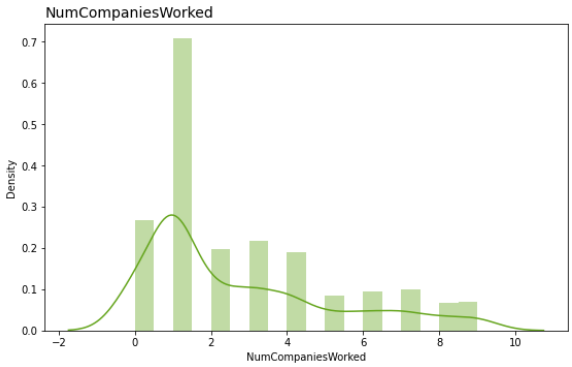

MonthlyIncome
MonthlyIncome

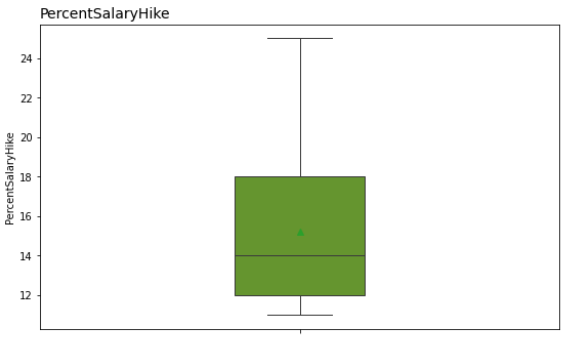

MonthlyIncome
Density
MonthlyIncome

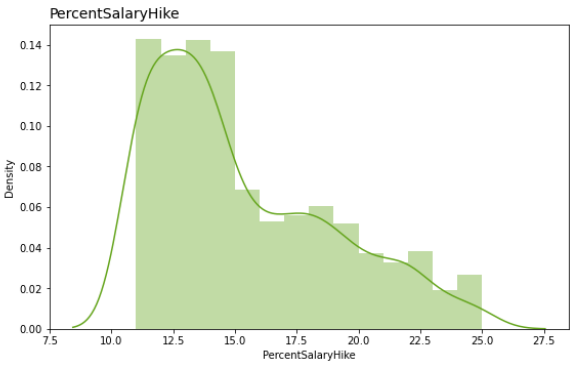

MonthlyRate
MonthlyRate

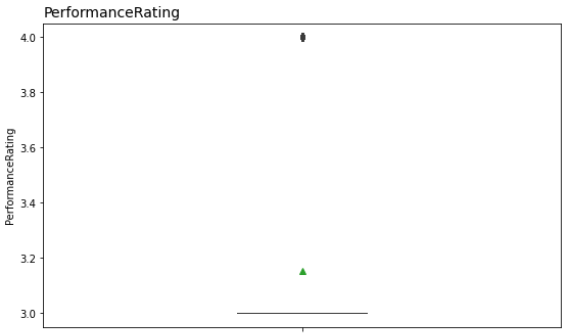

MonthlyRate
Density
MonthlyRate

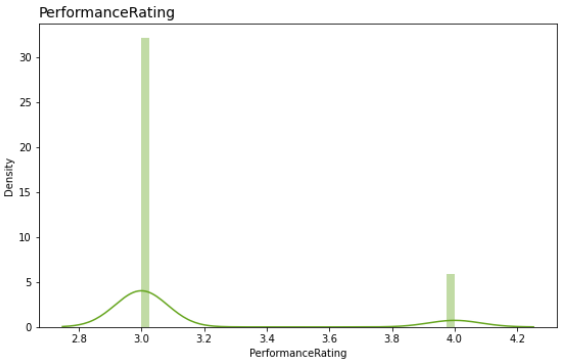

NumCompaniesWorked
NumCompaniesWorked

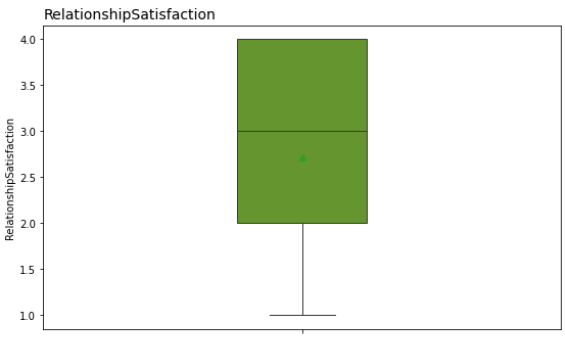

NumCompaniesWorked
Density
NumCompaniesWorked

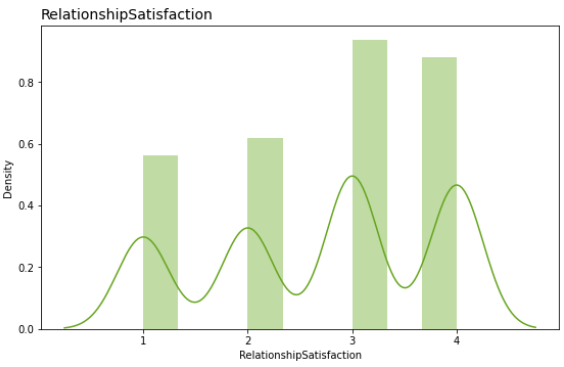

PercentSalaryHike
PercentSalaryHike

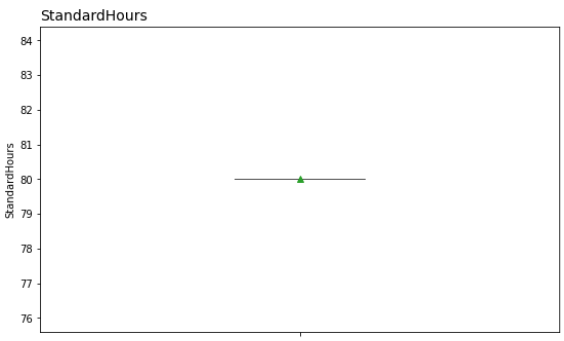

PercentSalaryHike
Density
PercentSalaryHike

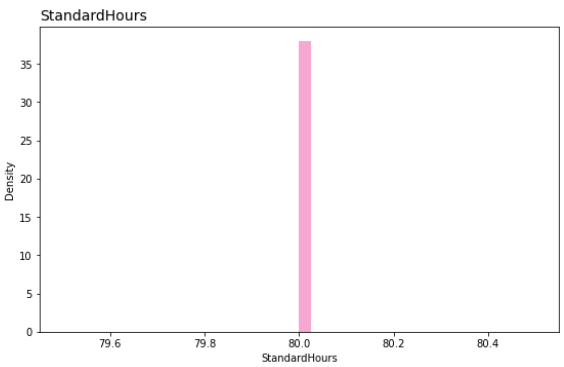

PerformanceRating
PerformanceRating

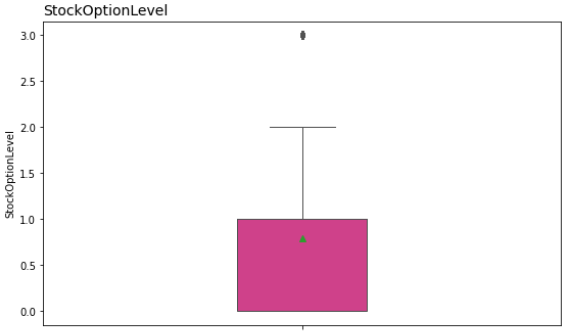

PerformanceRating
Density
PerformanceRating

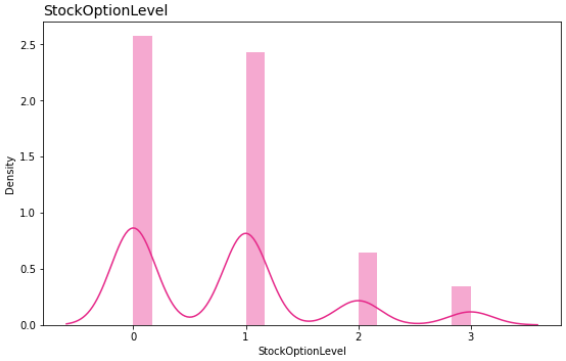

RelationshipSatisfaction
RelationshipSatisfaction

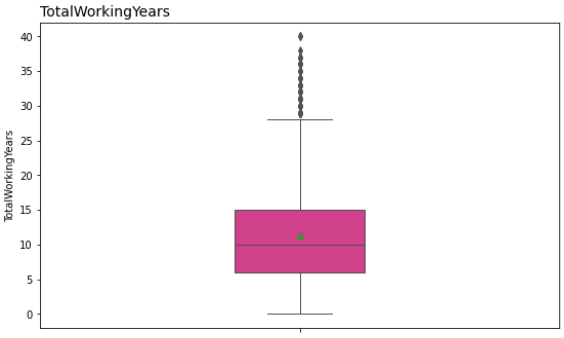

RelationshipSatisfaction
Density
RelationshipSatisfaction

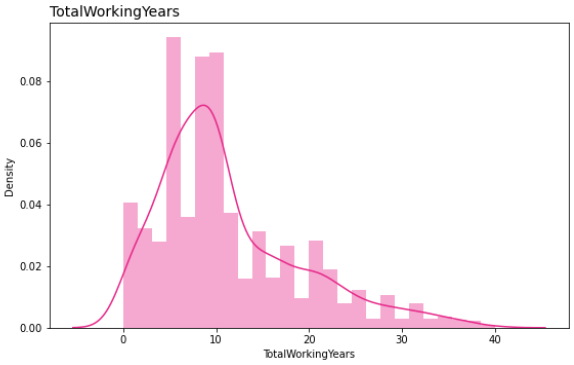

StandardHours
StandardHours

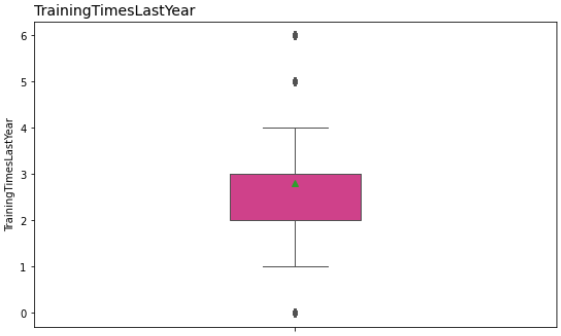

StandardHours
Density
StandardHours

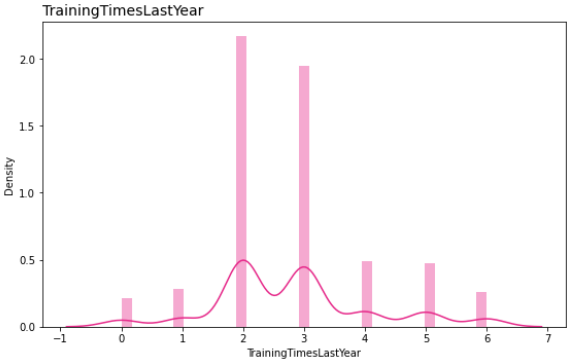

StockOptionLevel
StockOptionLevel

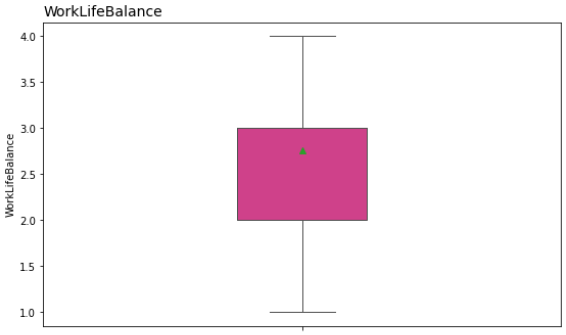

StockOptionLevel
Density
StockOptionLevel

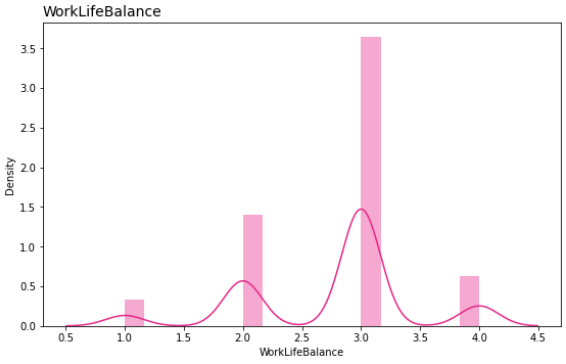

TotalWorkingYears
TotalWorkingYears

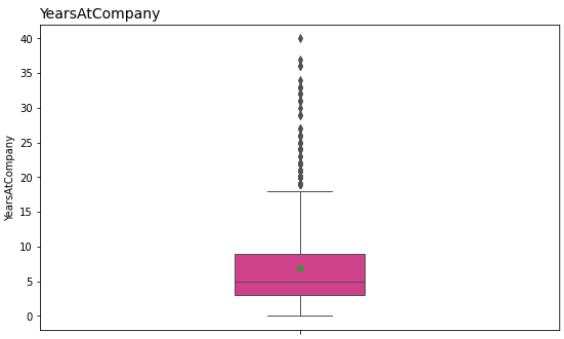

TotalWorkingYears
Density
TotalWorkingYears

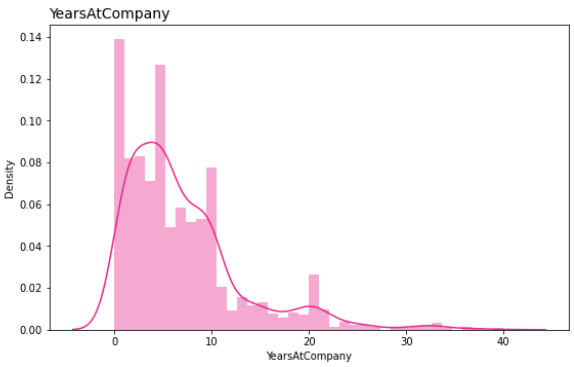

TrainingTimesLastYear
TrainingTimesLastYear

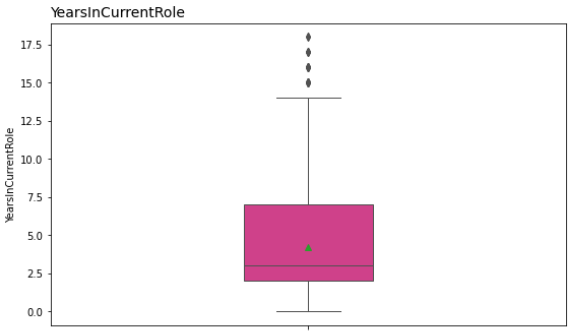

TrainingTimesLastYear
Density
TrainingTimesLastYear

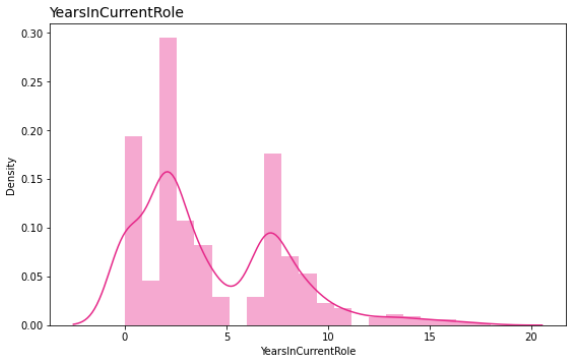

WorkLifeBalance
WorkLifeBalance

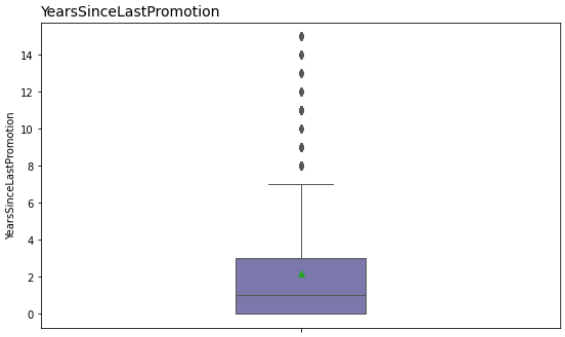

WorkLifeBalance
Density
WorkLifeBalance

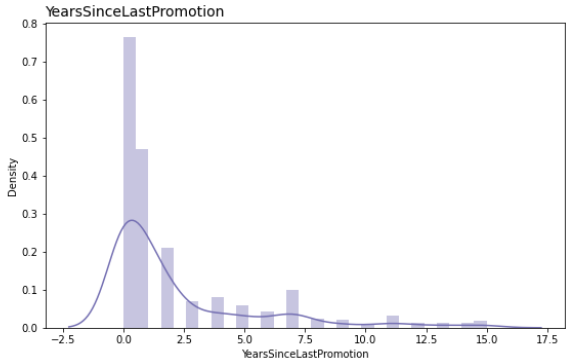

YearsAtCompany
YearsAtCompany

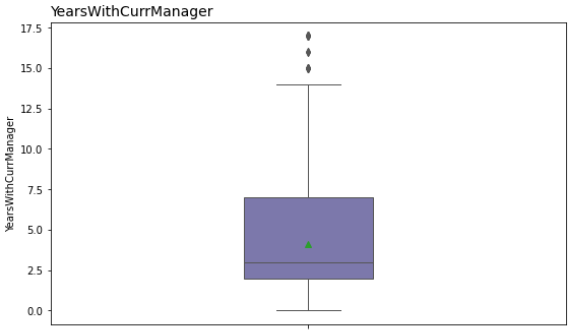

YearsAtCompany
Density
YearsAtCompany

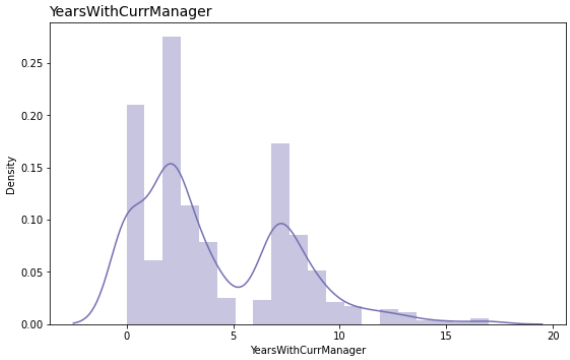












Conclusão: Detectamos que algumas variáveis possuem outliers, podemos substituir esses valores pela moda/média/mediana para tornar os dados mais simétricos ou utilizar a técnica de KNN para imputar esses dados. Antes, vamos realizar nosso modelo sem utilizar essas técnicas, para ver se mesmo assim conseguimos um desempenho satisfatório do nosso modelo.

Excluindo as variáveis

In [15]:
```
df.drop(columns = ['Over18', 'EmployeeCount', 'EmployeeNumber', 'StandardHours'],
        axis = 1,
        inplace = True)
```

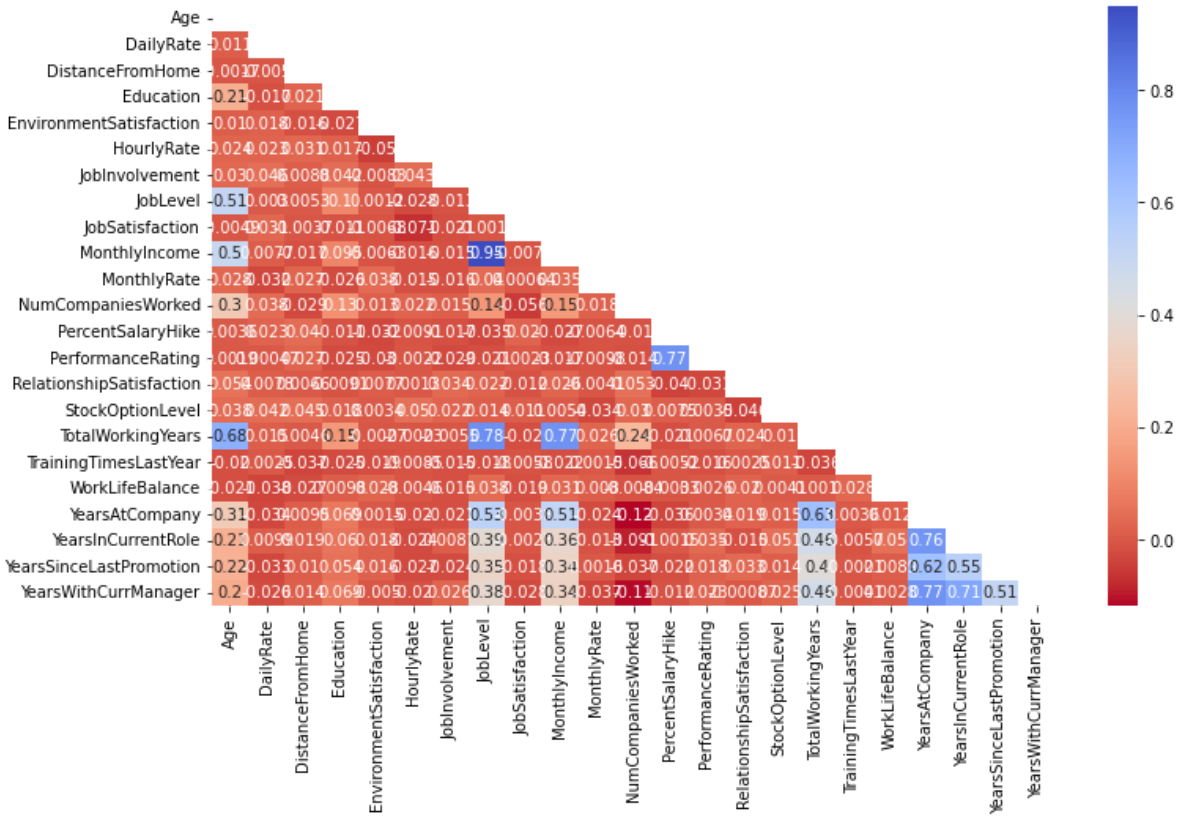
In [16]:
```
df.head()
```

Out[16]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educa
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life
4	27	No	Travel_Rarely	591	Research & Development	2	1	

Criando uma matriz de correlação

In [17]:
```
# Correlação

# Tamanho
plt.figure( figsize=(12, 7) )

# Ajustar a matriz
correlacao = df.corr()
matriz = np.triu( correlacao )

# Plot
sns.heatmap( df.corr(), mask=matriz, annot=True, cmap='coolwarm_r');
```



Neste caso, podemos remover as variáveis que possuam correlação acima de 0,7 em valores absolutos. Porém, vamos testar se conseguimos bons resultados antes de aplicar técnicas para aumentar o score.

Engenharia dos Dados

Alterando a variável Gender para binário

In [18]:
```
df['Gender'].unique()
```

Out[18]:
```
array(['Female', 'Male'], dtype=object)
```

In [19]:
```
df['Gender'] = df['Gender'].map({'Male': 1, 'Female': 0})
```

Alterando a variável OverTime para binário

In [20]:
```
df['OverTime'].unique()
```

Out[20]:
```
array(['Yes', 'No'], dtype=object)
```

In [21]:
```
df['OverTime'] = df['OverTime'].map({'Yes': 1, 'No': 0})
```

Alterando a variável Attrition para binário

In [22]:
```
df['Attrition'].unique()
```

Out[22]:
```
array(['Yes', 'No'], dtype=object)
```

In [23]:
```
df['Attrition'] = df['Attrition'].map({'Yes': 1, 'No': 0})
```

Caso nosso score de um valor muito baixo, podemos criar algumas flags ou novas variáveis baseando nessas que já temos e assim, testar se há uma melhora no nosso modelo.

Antes de tratar os dados precisamos separar nossa base em treino em teste. Caso contrário, podemos incorrer ao erro mais conhecido como datalakage, que é quando usamos dados de teste para aplicar transformações no nosso dados de treino.

Separando as Variáveis em X e y

In [24]:
```
X = df.drop(columns = ['Attrition'], axis = 1)
y = df['Attrition']
```

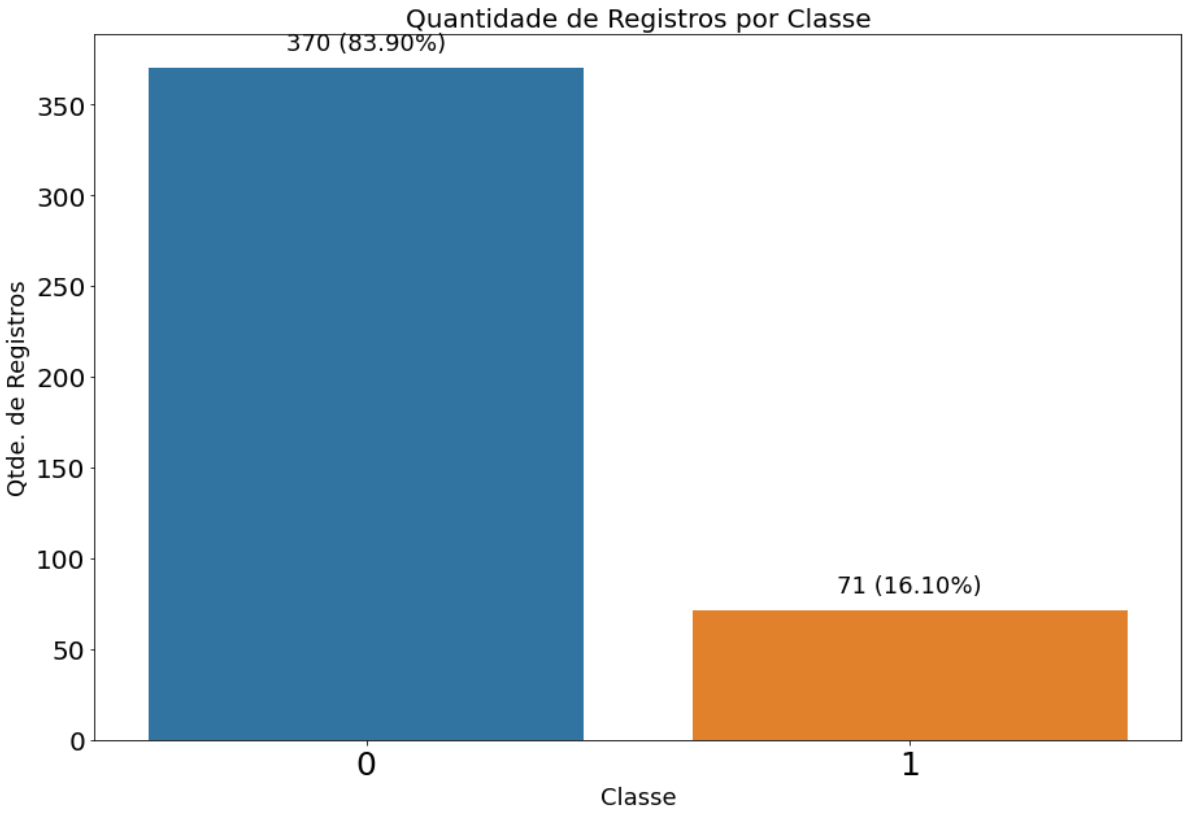
Separando em treino e teste

In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
                                                    shuffle = True,
```

```
                                                    stratify=y,
                                                    random_state = SEED)
```

Verificando se a variável dependente é balanceada

In [26]:
```
plt.figure(figsize=(15, 10))

total = y_test.shape[0]

ax = sns.countplot(x=y_test)

for p in ax.patches:
    rotulo = '{:1d} ({:.2f}%)'.format(p.get_height(), 100 * p.get_height() / total
    x = (p.get_x() + (p.get_width() / 2))
    y = p.get_height() + 10
    ax.annotate(rotulo, (x, y), ha='center', size=18)

ax.set_xticklabels(y_test.value_counts().index, size=25)
ax.set_yticklabels(ax.get_yticks().astype(int), size=20)

plt.title('Quantidade de Registros por Classe', size=20)
plt.ylabel('Qtde. de Registros', size=18)
plt.xlabel('Classe', size=18)
plt.show()
```



Conclusão: Não é balanceada.

Criando um pipeline para nosso modelo

In [27]:
```
#Cria uma lista com os nomes das colunas numéricas
X_train_num = X_train.select_dtypes(include = 'number').columns.to_list()

#Cria uma lista com os nomes das colunas categóricas
X_train_cat = X_train.select_dtypes(include = 'object').columns.to_list()
```

```
# Pipeline dos dados categóricos
steps_cat = Pipeline(steps = [
    ('imputer_cat', SimpleImputer(strategy = 'most_frequent')),     #Preenche as v
    ('OHE', OneHotEncoder(handle_unknown='ignore', drop = 'first')) #Instanciando
])

# Pipeline dos dados numéricos
steps_num = Pipeline(steps = [
    ('imputer_num', SimpleImputer(strategy = 'median')), #Preenche as variáveis nu
    ('scaler', RobustScaler())                           #Instanciando o robustsca
])
```

In [28]:
```
features_pipeline = ColumnTransformer(transformers = [
    ('num_pipe', steps_num, X_train_num), #Informando quais variáveis cada step se
    ('cat_pipe', steps_cat, X_train_cat)  #Informando quais variáveis cada step se
])
```

In [29]:
```
def func_pipeline(col_trans, modelo):
    """
    Essa função recebe o ColumTransformer e o modelo.
    A partir disso, cria uma novo pipeline com um balanceador (SMOTE) e o modelo.
    """

    # Criando o novo pipe
    pipe = imbpipeline(steps = [
        ('feature', col_trans),                #Incluindo os dois steps em um pipe
        ('smote', SMOTE(random_state = SEED)), #Balanceando o dataset
        ('model', modelo)                      #Isntanciando um modelo
    ])

    return pipe
```

In [30]:
```
# Criando uma lista de modelos
modelos = [
    ('KNN', KNeighborsClassifier()),
    ('LogReg', LogisticRegression(max_iter = 3000, random_state = SEED)),
    ('RF', RandomForestClassifier(max_depth = 3, random_state = SEED)),
    ('XGB', XGBClassifier(max_depth = 3, random_state = SEED)),
    ('GBC', GradientBoostingClassifier(max_depth = 3, random_state = SEED)),
    ('LGBM', LGBMClassifier(max_depth = 3, random_state = SEED))
]

# Assegura que em cada fold a seleção será aleatória, mas a proporção das classes
cv = StratifiedKFold(n_splits = 5, random_state = SEED, shuffle = True)

# Criando um data frame para armazenar os resultados
result_df = pd.DataFrame(columns = ['Modelo', 'Precision_Treino', 'Precision_Teste

for i, modelo in enumerate(modelos):
    # Criando nosso pipeline com o modelo
    pipe = func_pipeline(features_pipeline, modelo[1])

    # Treinando o modelo
    pipe.fit(X_train, y_train)

    # Prevendo valores para os dados de treino
    y_pred_treino = pipe.predict(X_train)

    # Prevendo valores para os dados de teste
    y_pred_teste = pipe.predict(X_test)

    # Precision para os dados de treino
```

```
    precision_treino = precision_score(y_train, y_pred_treino, average='macro')

    # Precision para os dados de treino
    precision_teste = precision_score(y_test, y_pred_teste, average='macro')

    # Criando a validação cruzada
    cross_val = cross_validate(
    estimator = pipe,           # Passando nosso pipeline
    X = X_train,                # Passando os dados de treino
    y = y_train,                # Passando os dados de treino
    scoring = 'precision',      # Definindo a métrica a ser analisada
    return_train_score = True,  # Retorna o score da validação cruzada
    cv = cv)

    # Armazenando os resultados no data frame
    result_df.loc[i] = [modelo[0],                        # Nome do modelo
                        precision_treino,                 # Resultado do Precision
                        precision_teste,                  # Resultado do Precision
                        cross_val['train_score'].mean(),  # Resultado da validação
                        cross_val['test_score'].mean()]   # Resultado da validação


result_df = result_df.sort_values(by = ['Cross_Teste', 'Precision_Teste'], ascendi
```

In [31]:
```
result_df
```

Out[31]:

	Modelo	Precision_Treino	Precision_Teste	Cross_Treino	Cross_Teste
5	LGBM	0.944798	0.843868	0.966689	0.633204
4	GBC	0.959111	0.854939	0.971345	0.609430
3	XGB	0.996548	0.855808	1.000000	0.527392
2	RF	0.692694	0.682418	0.536302	0.455610
1	LogReg	0.672541	0.662628	0.429538	0.377739
0	KNN	0.691286	0.588366	0.392199	0.242460

Analisando os resultado chegamos a conclusão que o melhor modelo é o LGBM pois, teve o melhor cross_teste sem perde muito do precision_teste.

Criando uma pipeline para otimizar o nosso melhor modelo

In [32]:
```
pipe = imbpipeline(steps = [
    ('feature', features_pipeline),                                  # Incluindo os
    ('smote', SMOTE(random_state = SEED)),                           # Balanceando
    ('model', LGBMClassifier(max_depth = 3, random_state = SEED))    # Isntanciando
])
```

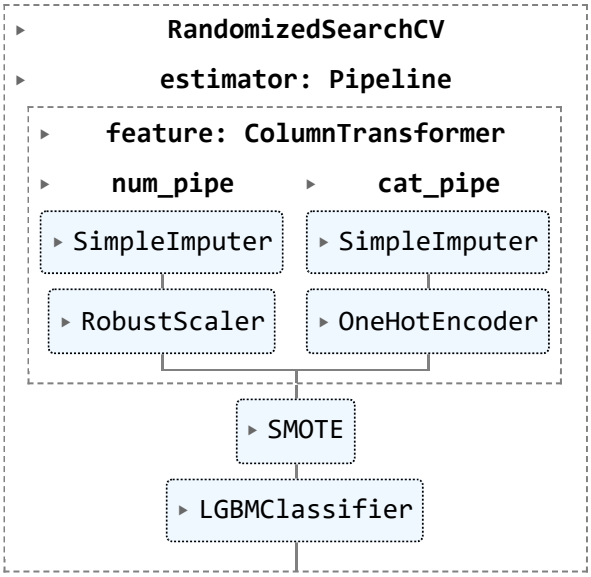
In [33]:
```
# Criando uma lista de normalizadodres/padronizadores para escolher o melhor
scaler = [
    MinMaxScaler(),
    StandardScaler(),
    RobustScaler(),
    None
]

valores_C = np.array([0.01, 0.1, 0.5, 1, 2, 3, 5, 10, 20, 50, 100])
regularizacao = ['l1', 'l2']
```

```
param_strat = {
    'feature__num_pipe__scaler': scaler, # Informando que o scaler a ser otimizado
    'model__max_depth': range(3, 5),    # Passando os níveis de profundidade a ser
    'model__min_child_samples': range(1, 30), # Número mínimo de dados em uma folh
    'model__min_child_weight': range(1, 30), # Soma mínima do peso da instância a s
    'model__num_leaves': range(1, 50) # Máximo de folhas em uma árvore
}

# Assegura que em cada fold a seleção será aleatória, mas a proporção das classes s
cv = StratifiedKFold(n_splits = 5, random_state = SEED, shuffle = True)

random_search = RandomizedSearchCV(
    estimator = pipe,                   # Pipe a ser otimizado
    param_distributions = param_strat, # Parâmetros para serem testados
    cv = cv,                            # Validação cruzada
    random_state = SEED,                # Travando a aleatoriedade
    n_iter = 10,                        # Número de interações
    scoring = 'precision')              # Métrica a ser otimizada
```

Treinando o modelo

In [34]:
```
random_search.fit(X_train, y_train)
```

Out[34]:



Visualizando os melhores parâmetros

In [35]:
```
random_search.best_params_
```

Out[35]:
```
{'model__num_leaves': 5,
 'model__min_child_weight': 14,
 'model__min_child_samples': 20,
 'model__max_depth': 3,
 'feature__num_pipe__scaler': RobustScaler()}
```

Logo, a combinação de parâmetros que resultou em um melhor precision foram:

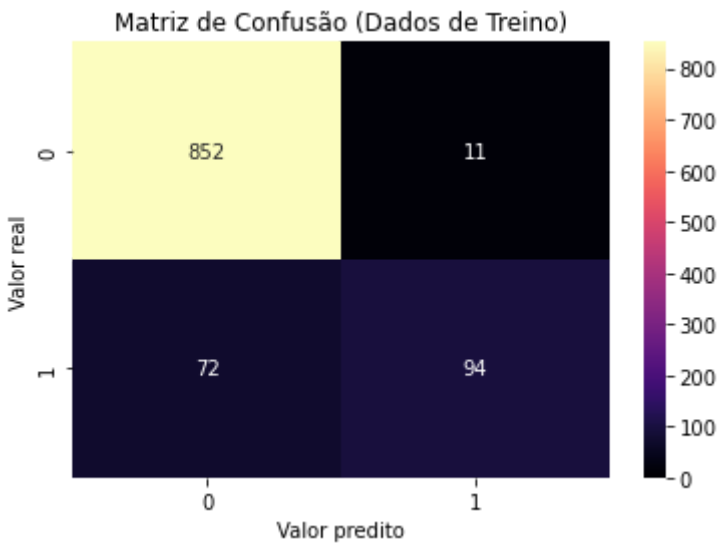
1. Num leaves: 5
2. Min child weight: 14
3. Min child samples: 20
4. Max depth: 3
5. Scaler: RobustScaler

Prevendo os valores

In [36]:
```
# Prevendo valores para os dados de treino
y_pred_treino = random_search.predict(X_train)

# Prevendo valores para os dados de teste
y_pred_teste = random_search.predict(X_test)
```

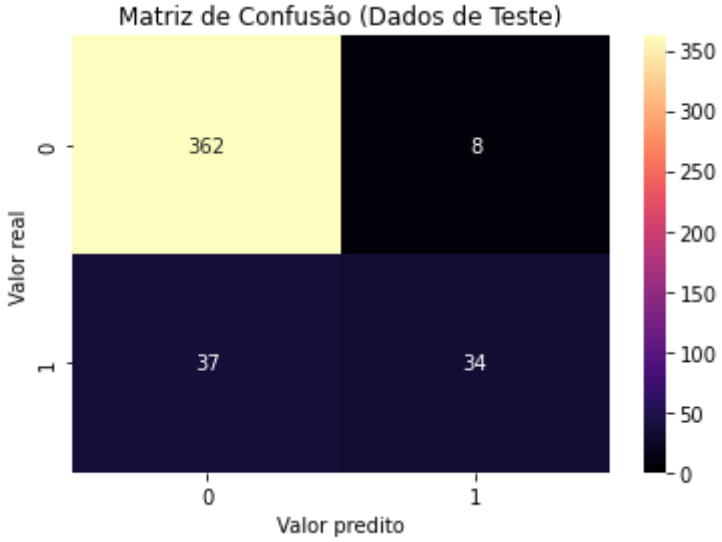
Matriz de confusão

In [37]:
```
# Criando a matriz de confusão
matriz = confusion_matrix(y_train, y_pred_treino)

# Plotando a matriz de confusão
sns.heatmap(
    matriz,          # Passando a matriz
    vmin = 0,        # Passando o valor mínimo
    annot = True,    # Inserindo os valores
    cmap = 'magma',  # Passando a paleta de cores
    fmt='.5g'        # Definindo o formato dos valores
)

# Inserindo um título
plt.title(f'Matriz de Confusão (Dados de Treino)', size=12)

# Inserindo um título para o eixo Y
plt.ylabel('Valor real')

# Inserindo um título para o eixo X
plt.xlabel('Valor predito');
```



In [38]:
```
# Criando a matriz de confusão
matriz = confusion_matrix(y_test, y_pred_teste)

# Plotando a matriz de confusão
sns.heatmap(
    matriz,          # Passando a matriz
    vmin = 0,        # Passando o valor mínimo
    annot = True,    # Inserindo os valores
    cmap = 'magma',  # Passando a paleta de cores
    fmt='.5g'        # Definindo o formato dos valores
)
```

```
# Inserindo um título
plt.title(f'Matriz de Confusão (Dados de Teste)', size=12)

# Inserindo um título para o eixo Y
plt.ylabel('Valor real')

# Inserindo um título para o eixo X
plt.xlabel('Valor predito');
```



Visualizando o precision

In [39]:
```
# Recall para os dados de treino
precision_treino = precision_score(y_train, y_pred_treino, average='macro')

# Recall para os dados de treino
precision_teste = precision_score(y_test, y_pred_teste, average='macro')
```

In [40]:
```
print(f'Precision de Treino: {round(precision_treino, 2)}')
print(f'Precision de Teste: {round(precision_teste, 2)}')
```

```
Precision de Treino: 0.91
Precision de Teste: 0.86
```

Como temos apenas 0.05 de diferença entre os dados de treino e teste, podemos considerar que não temos overfitting.

Analisando algumas métricas

In [41]:
```
metricas = classification_report(y_test, y_pred_teste)
```

In [42]:
```
print('Métricas\n')
print(metricas)
```

```
Métricas

              precision    recall  f1-score   support

           0       0.91      0.98      0.94       370
           1       0.81      0.48      0.60        71

    accuracy                           0.90       441
   macro avg       0.86      0.73      0.77       441
weighted avg       0.89      0.90      0.89       441
```

Baixando nosso modelo em pickle

In [43]:
```
nome_arquivo = 'attrition.pkl'
joblib.dump(random_search, nome_arquivo)
```

Out[43]:
```
['attrition.pkl']
```