IBM Attrition Analysis and Prediction

XGB : CV - Accuracy (5 folds) = .891

Enzo Rodriguez

December 2019

source : [http://thecontextofthings.com/2017/01/06/employee-attrition/ (http://thecontextofthings.com/2017/01/06/employee-attrition/)](http://thecontextofthings.com/2017/01/06/employee-attrition/)

- 1. Load libraries and read the data
    - 1.1. Load libraries
    - 1.2. Read the data
    - 1.3. Missing values
    - 1.4. Reassign target and drop useless features
- 2. Exploratory Data Analysis (EDA)
    - 2.1. Head and describe
    - 2.2. Target distribution (number and %)
    - 2.3. Features distribution and barplot (hue = Attrition)
    - 2.4. Pie plot and barplot
- 3. Feature engineering and selection
    - 3.1. New features (24)
    - 3.2. Drop some features
    - 3.3. Features encoding and scaling
    - 3.4. Correlation Matrix
    - 3.5. Remove collinear features
- 4. Define functions
    - 4.1. Define model performance plot
    - 4.2. Define feature importance plot
    - 4.3. Define cumulative gains curve
    - 4.4. Define cross validation metrics
- 5. Prepare dataset
    - 5.1. Define (X, y)
    - 5.2. Train test split
- 6. XGBoost - RandomizedSearchCV to optimize hyperparameters (800 comb)
- 7. XGBoost - With best hyperparameters = 89.11
    - 7.1. XGBoost - Modeling and performance plot
    - 7.2. XGBoost - Feature importance
    - 7.3. XGBoost - Cumulative gains curve
    - 7.4. XGBoost - Cross validation (5 folds)

1. Load libraries and read the data

1.1. Load libraries

In [1]:

```
# Python libraries
import pandas as pd
import numpy as np
from datetime import datetime
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import GridSearchCV, RandomizedSear
chCV, cross_val_score, learning_curve, train_test_split
from sklearn.metrics import precision_score, roc_auc_score, reca
ll_score, confusion_matrix, roc_curve, precision_recall_curve, a
ccuracy_score
import xgboost as xgb
import warnings
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import plotly.figure_factory as ff

warnings.filterwarnings('ignore')
```

```
Intel(R) Data Analytics Acceleration Library (Intel(
R) DAAL) solvers for sklearn enabled: https://intelp
ython.github.io/daal4py/sklearn.html
```

1.2. Read the data

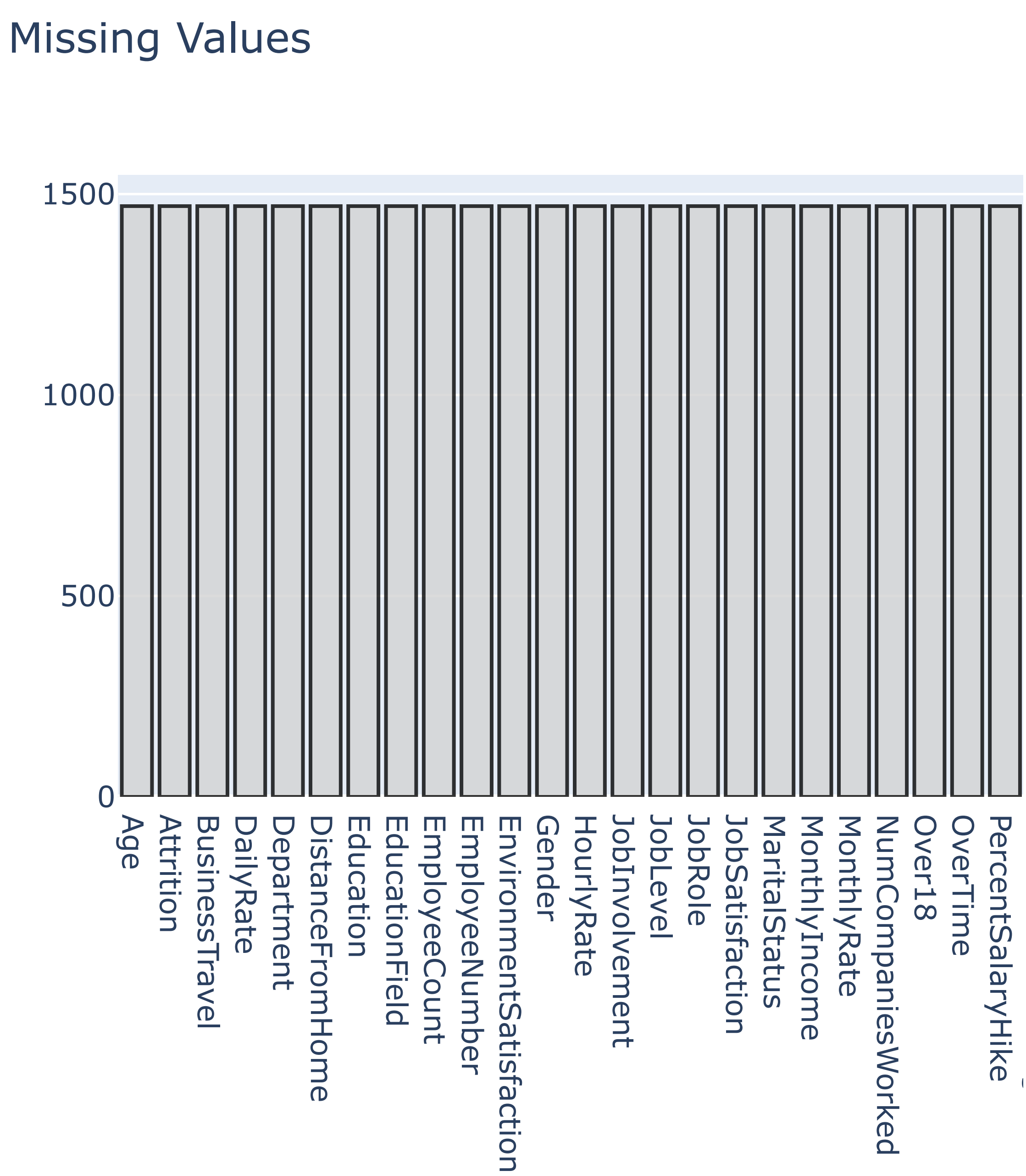
In [4]:

```
data = pd.read_csv('IBM.csv')
```

1.3. Missing values

In [5]:

```
null_feat = pd.DataFrame(len(data['Attrition']) - data.isnull().
sum(), columns = ['Count'])

trace = go.Bar(x = null_feat.index, y = null_feat['Count'] ,opac
ity = 0.8, marker=dict(color = 'lightgrey',
        line=dict(color='#000000',width=1.5)))

layout = dict(title =  "Missing Values")

fig = dict(data = [trace], layout=layout)
py.iplot(fig)
```



1.4. Reassign target and drop useless features

In [6]:

```
# Reassign target
data.Attrition.replace(to_replace = dict(Yes = 1, No = 0), inpla
ce = True)
# Drop useless feat
data = data.drop(columns=['StandardHours',
                          'EmployeeCount',
                          'Over18',
                        ])
```

2. Exploratory Data Analysis (EDA)

2.1. Head and describe

In [7]:

```
# head
data.head()
```

Out[7]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFrom
0	41	1	Travel_Rarely	1102	Sales	
1	49	0	Travel_Frequently	279	Research & Development	
2	37	1	Travel_Rarely	1373	Research & Development	
3	33	0	Travel_Frequently	1392	Research & Development	
4	27	0	Travel_Rarely	591	Research & Development	

5 rows × 32 columns

In [8]:

```
# describe
data.describe()
```

Out[8]:

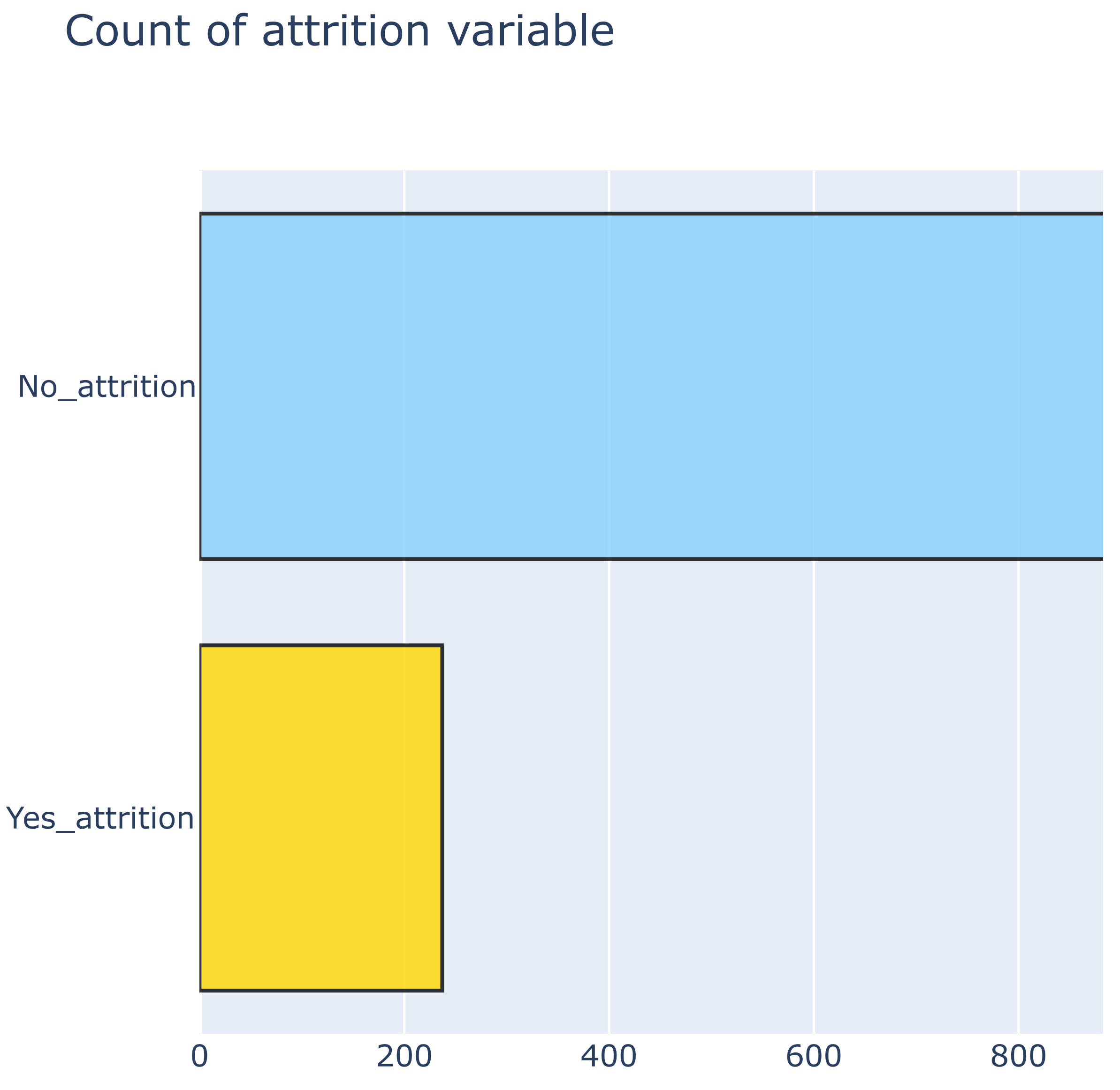
	Age	Attrition	DailyRate	DistanceFromHome	Ed
count	1470.000000	1470.000000	1470.000000	1470.000000	1470
mean	36.923810	0.161224	802.485714	9.192517	2
std	9.135373	0.367863	403.509100	8.106864	1
min	18.000000	0.000000	102.000000	1.000000	1
25%	30.000000	0.000000	465.000000	2.000000	2
50%	36.000000	0.000000	802.000000	7.000000	3
75%	43.000000	0.000000	1157.000000	14.000000	4
max	60.000000	1.000000	1499.000000	29.000000	5

8 rows × 25 columns

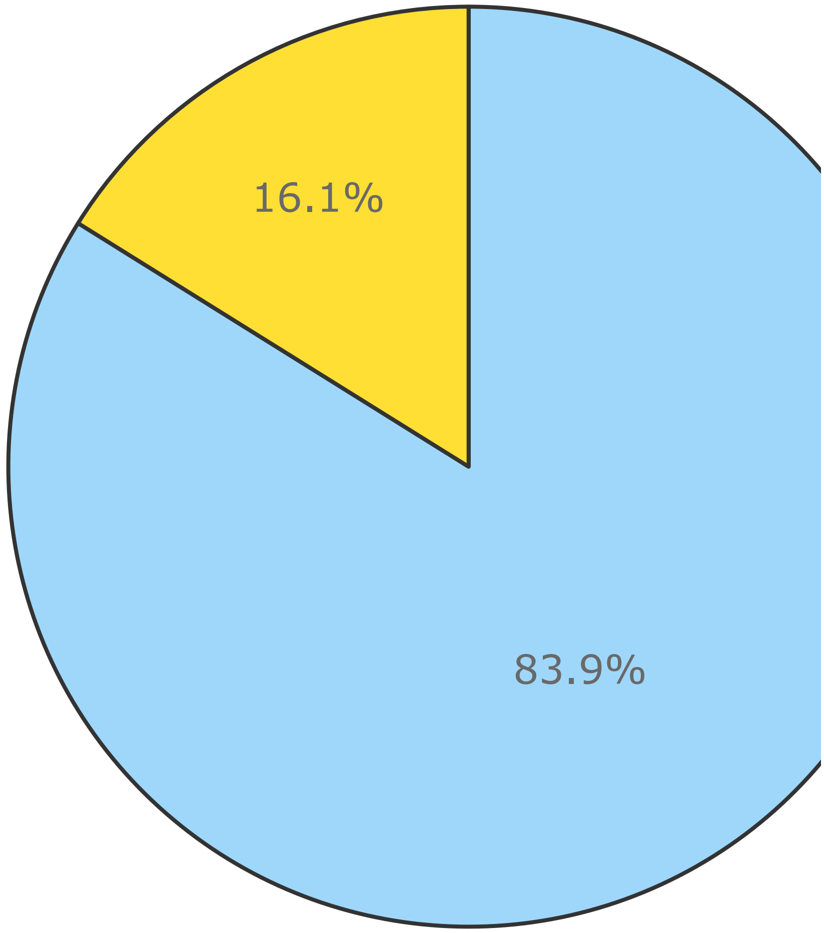
2.2. Target distribution (number and %)

In [9]:

```
attrition = data[(data['Attrition'] != 0)]
no_attrition = data[(data['Attrition'] == 0)]

#------------COUNT-----------------------
trace = go.Bar(x = (len(attrition), len(no_attrition)), y = ['Ye
s_attrition', 'No_attrition'], orientation = 'h', opacity = 0.8,
marker=dict(
        color=['gold', 'lightskyblue'],
        line=dict(color='#000000',width=1.5)))

layout = dict(title =  'Count of attrition variable')

fig = dict(data = [trace], layout=layout)
py.iplot(fig)

#------------PERCENTAGE-------------------
trace = go.Pie(labels = ['No_attrition', 'Yes_attrition'], value
s = data['Attrition'].value_counts(),
               textfont=dict(size=15), opacity = 0.8,
               marker=dict(colors=['lightskyblue','gold'],
                           line=dict(color='#000000', width=1.5)
))


layout = dict(title =  'Distribution of attrition variable')

fig = dict(data = [trace], layout=layout)
py.iplot(fig)
```


Count of attrition variable
No_attrition
Yes_attrition
0
200
400
600
800

Distribution of attrition variable



2.3. Features distribution and barplot (hue = Attrition)

In [10]:

```
def plot_distribution(var_select, bin_size) :
# Calculate the correlation coefficient between the new variable
and the target
    corr = data['Attrition'].corr(data[var_select])
    corr = np.round(corr,3)
    tmp1 = attrition[var_select]
    tmp2 = no_attrition[var_select]
    hist_data = [tmp1, tmp2]

    group_labels = ['Yes_attrition', 'No_attrition']
    colors = ['#FFD700', '#7EC0EE']

    fig = ff.create_distplot(hist_data, group_labels, colors = c
olors, show_hist = True, curve_type='kde', bin_size = bin_size)

    fig['layout'].update(title = var_select+' '+'(corr target ='
+ str(corr)+')')

    py.iplot(fig, filename = 'Density plot')
```

In [11]:

```
def barplot(var_select, x_no_numeric) :
    tmp1 = data[(data['Attrition'] != 0)]
    tmp2 = data[(data['Attrition'] == 0)]
    tmp3 = pd.DataFrame(pd.crosstab(data[var_select],data['Attri
tion']), )
    tmp3['Attr%'] = tmp3[1] / (tmp3[1] + tmp3[0]) * 100
    if x_no_numeric == True  :
        tmp3 = tmp3.sort_values(1, ascending = False)

    color=['lightskyblue','gold' ]
    trace1 = go.Bar(
        x=tmp1[var_select].value_counts().keys().tolist(),
        y=tmp1[var_select].value_counts().values.tolist(),
        name='Yes_Attrition',opacity = 0.8, marker=dict(
        color='gold',
        line=dict(color='#000000',width=1)))


    trace2 = go.Bar(
        x=tmp2[var_select].value_counts().keys().tolist(),
```

```
        y=tmp2[var_select].value_counts().values.tolist(),
        name='No_Attrition', opacity = 0.8, marker=dict(
        color='lightskyblue',
        line=dict(color='#000000',width=1)))

    trace3 =  go.Scatter(
        x=tmp3.index,
        y=tmp3['Attr%'],
        yaxis = 'y2',
        name='% Attrition', opacity = 0.6, marker=dict(
        color='black',
        line=dict(color='#000000',width=0.5
        )))

    layout = dict(title =  str(var_select),
              xaxis=dict(),
              yaxis=dict(title= 'Count'),
              yaxis2=dict(range= [-0, 75],
                          overlaying= 'y',
                          anchor= 'x',
                          side= 'right',
                          zeroline=False,
                          showgrid= False,
                          title= '% Attrition'
                         ))

    fig = go.Figure(data=[trace1, trace2, trace3], layout=layout
)
    py.iplot(fig)
```

plot_distribution('Age', False)

barplot('Age', False) plot_distribution('DailyRate', 100) plot_distribution('DistanceFromHome', False) barplot('DistanceFromHome', False) plot_distribution('HourlyRate', False) plot_distribution('MonthlyIncome', 100) plot_distribution('MonthlyRate', 100) plot_distribution('NumCompaniesWorked', False) barplot('NumCompaniesWorked',False) plot_distribution('PercentSalaryHike', False) barplot('PercentSalaryHike', False) plot_distribution('TotalWorkingYears', False) barplot('TotalWorkingYears', False) plot_distribution('TrainingTimesLastYear', False) barplot('TrainingTimesLastYear',False) plot_distribution('YearsAtCompany', False) barplot('YearsAtCompany', False) plot_distribution('YearsInCurrentRole', False) barplot('YearsInCurrentRole', False) plot_distribution('YearsSinceLastPromotion', False) barplot('YearsSinceLastPromotion', False) plot_distribution('YearsWithCurrManager', False) barplot('YearsWithCurrManager', False)

2.4. Pie plot and barplot

In [12]:

```
def plot_pie(var_select) :

    colors = ['gold', 'lightgreen', 'lightcoral', 'lightskyblue'
, 'lightgrey', 'orange', 'white', 'lightpink']
    trace1 = go.Pie(values  = attrition[var_select].value_counts
().values.tolist(),
                    labels  = attrition[var_select].value_counts
().keys().tolist(),
                    textfont=dict(size=15), opacity = 0.8,
                    hoverinfo = "label+percent+name",
                    domain  = dict(x = [0,.48]),
                    name    = "attrition employes",
                    marker  = dict(colors = colors, line = dict(
width = 1.5)))
    trace2 = go.Pie(values  = no_attrition[var_select].value_cou
nts().values.tolist(),
                    labels  = no_attrition[var_select].value_cou
nts().keys().tolist(),
                    textfont=dict(size=15), opacity = 0.8,
                    hoverinfo = "label+percent+name",
```

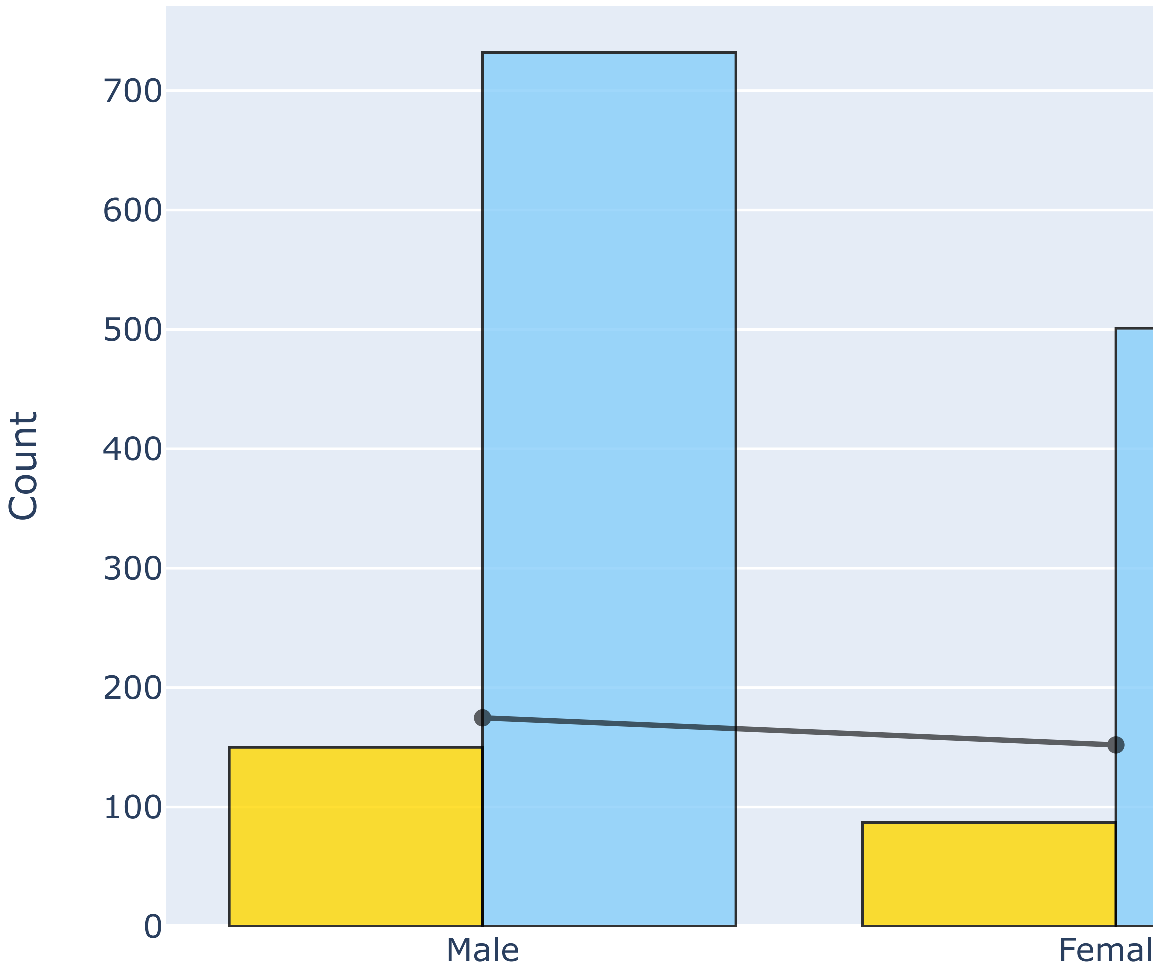
```
                    marker   = dict(colors = colors, line = dict(
width = 1.5)),
                    domain   = dict(x = [.52,1]),
                    name     = "Non attrition employes" )

    layout = go.Layout(dict(title = var_select + " distribution
in employes attrition ",
                            annotations = [dict(text = "Yes_attr
ition",
                                                font = dict(size
= 13),
                                                showarrow = Fals
e,
                                                x = .22, y = -0.
1),
                                            dict(text = "No_attr
ition",
                                                font = dict(size
= 13),
                                                showarrow = Fals
e,
                                                x = .8,y = -.1)]
))


    fig  = go.Figure(data = [trace1,trace2],layout = layout)
    py.iplot(fig)
```

In [13]:

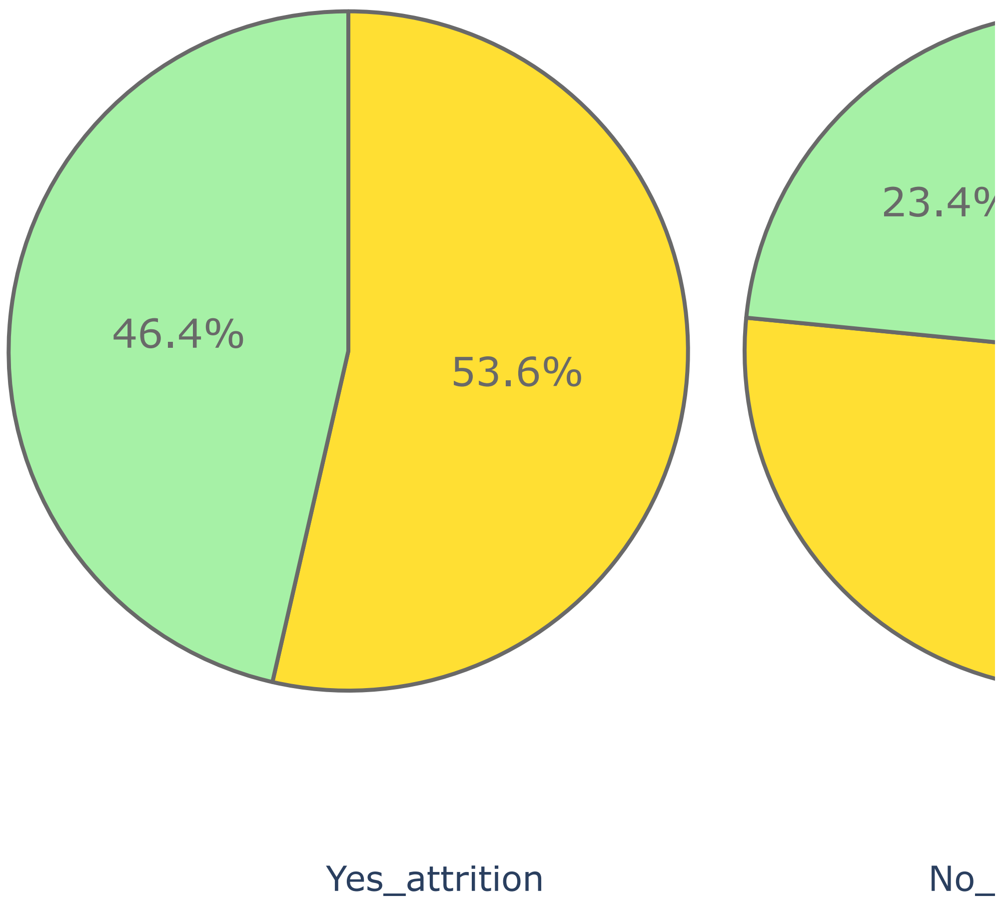
```
plot_pie("Gender")
barplot('Gender',True)
plot_pie('OverTime')
barplot('OverTime',True)
plot_pie('BusinessTravel')
barplot('BusinessTravel',True)
plot_pie('JobRole')
barplot('JobRole',True)
plot_pie('Department')
barplot('Department',True)
plot_pie('MaritalStatus')
barplot('MaritalStatus',True)
plot_pie('EducationField')
barplot('EducationField',True)
plot_pie('Education')
barplot('Education',False)
plot_pie('EnvironmentSatisfaction')
barplot('EnvironmentSatisfaction',False)
plot_pie('JobInvolvement')
barplot('JobInvolvement', False)
plot_pie('JobLevel')
barplot('JobLevel',False)
plot_pie('JobSatisfaction')
barplot('JobSatisfaction',False)
plot_pie('PerformanceRating')
barplot('PerformanceRating',False)
plot_pie('RelationshipSatisfaction')
barplot('RelationshipSatisfaction', False)
plot_pie('StockOptionLevel')
barplot('StockOptionLevel', False)
plot_pie('WorkLifeBalance')
barplot('WorkLifeBalance', False)
```

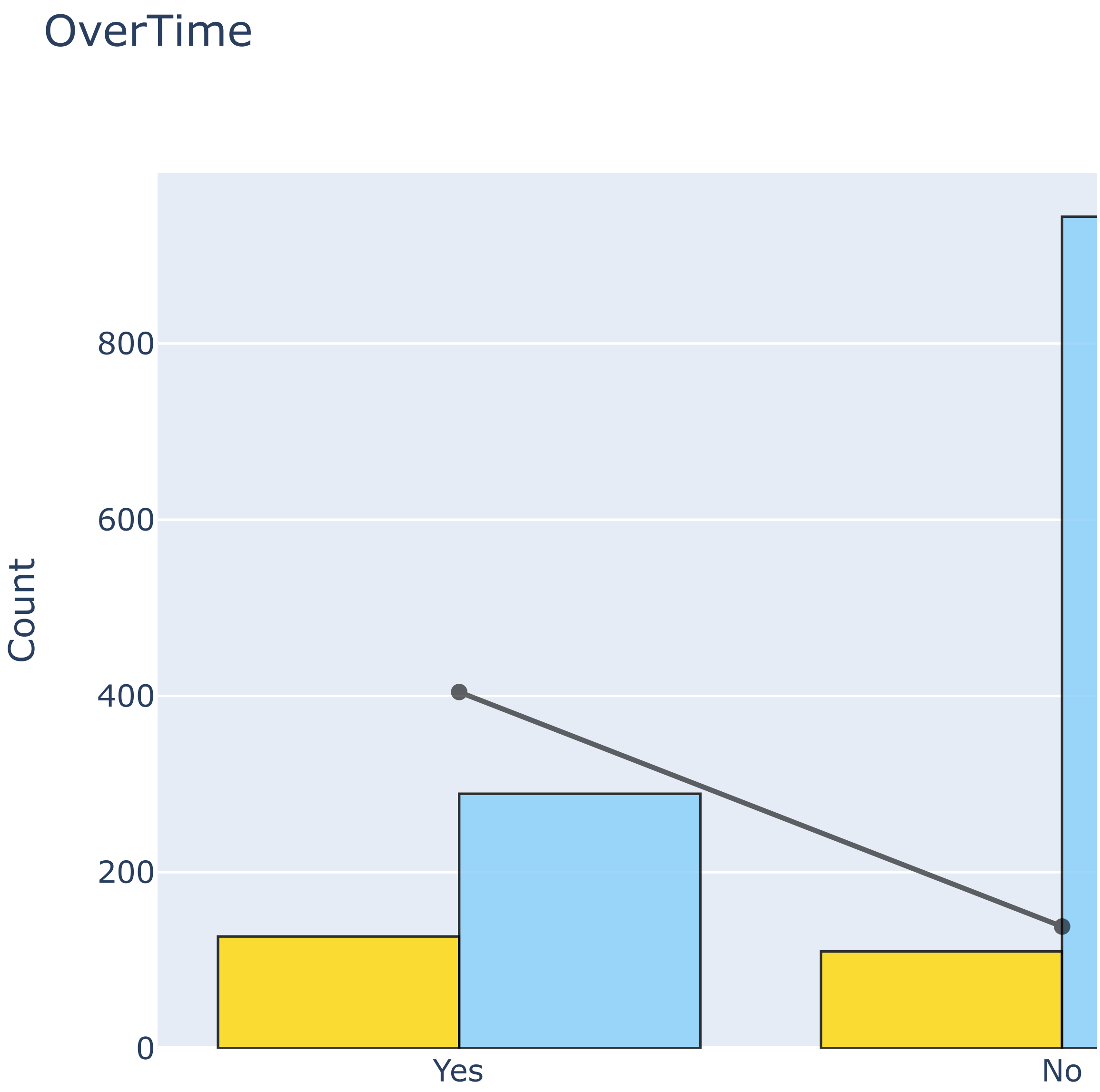
Gender distribution in employes attrition
36.7%
63.3%
40.6%
Yes_attrition
No_att

Gender

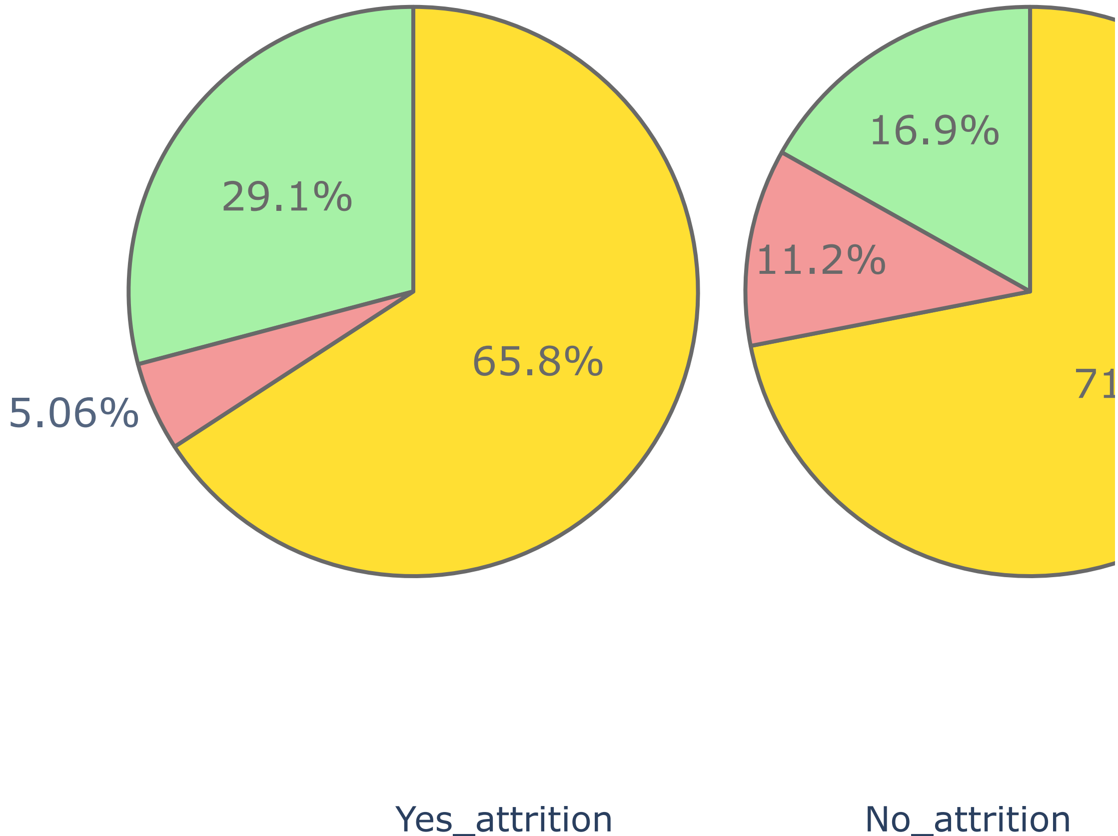


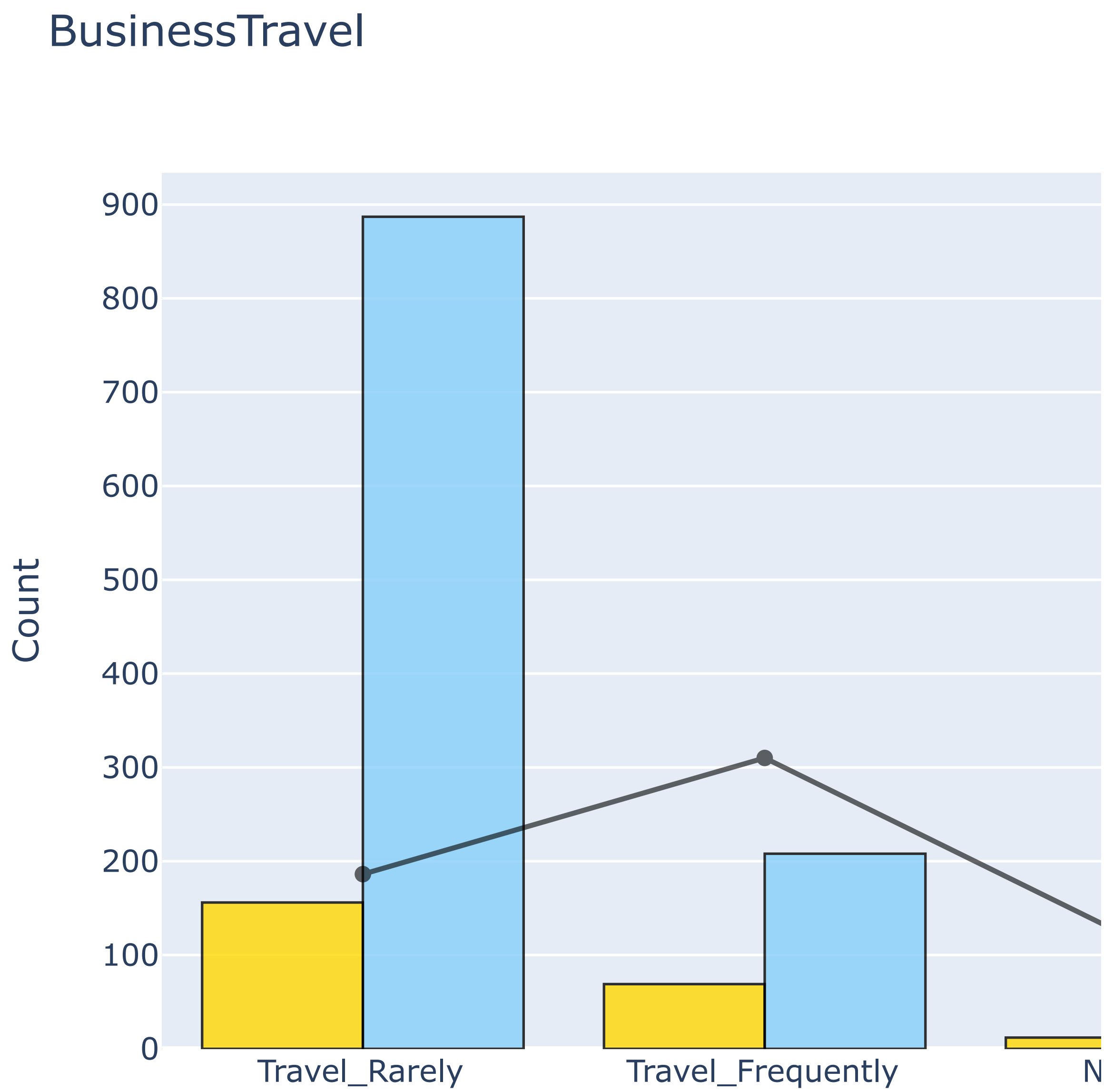
OverTime distribution in employes attrition



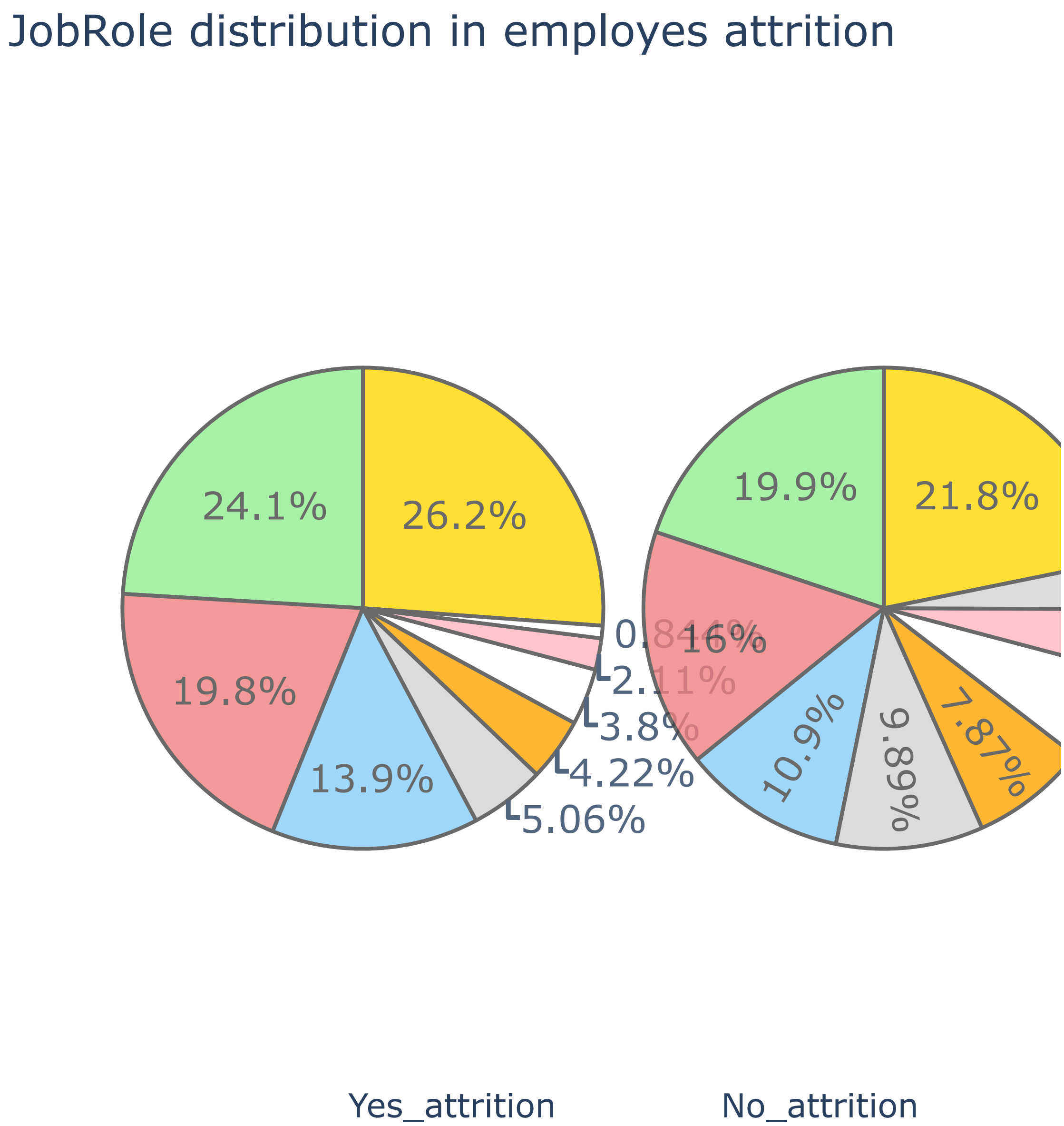

OverTime
Count
0
200
400
600
800
Yes
No

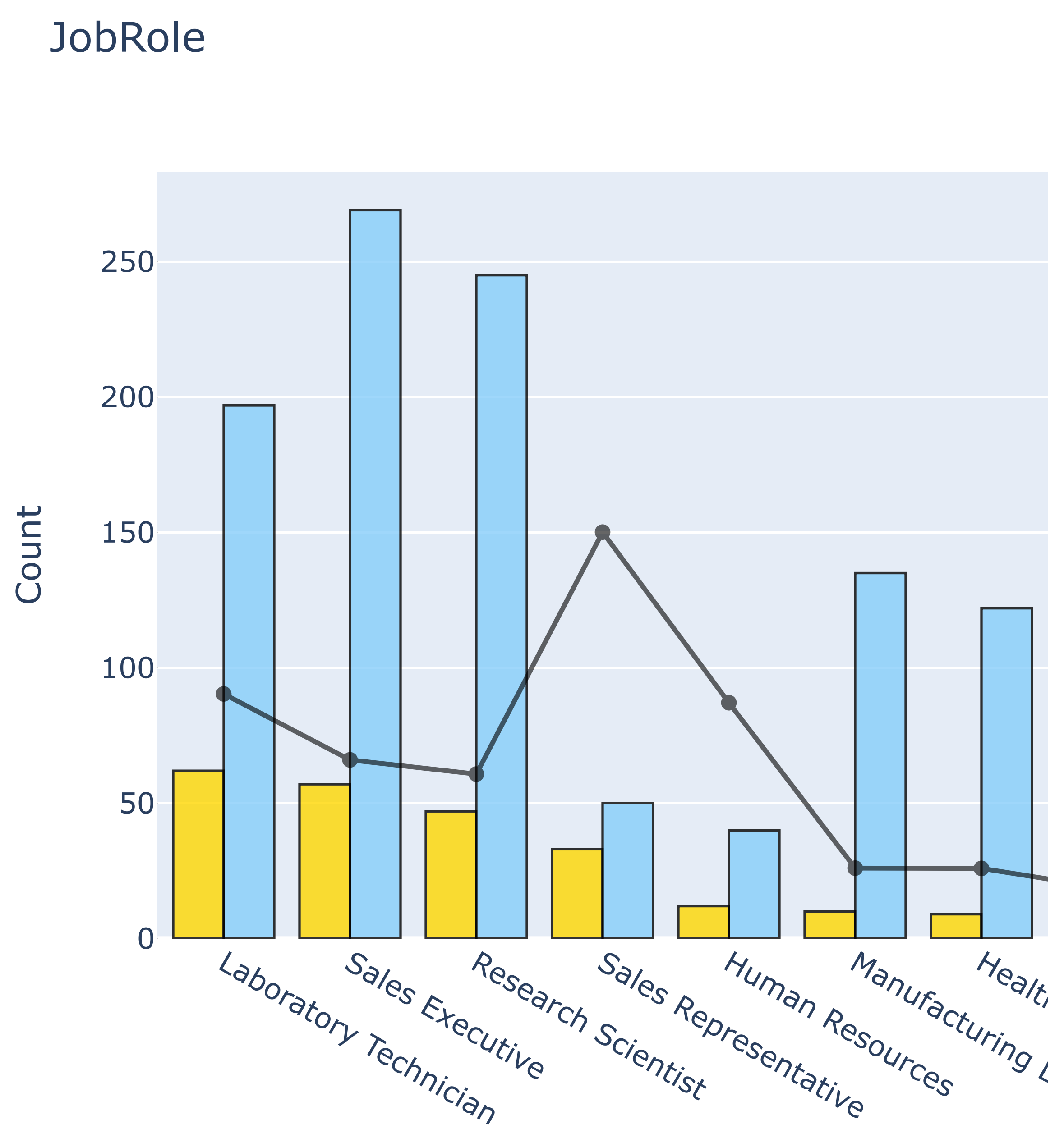
BusinessTravel distribution in employes attrition



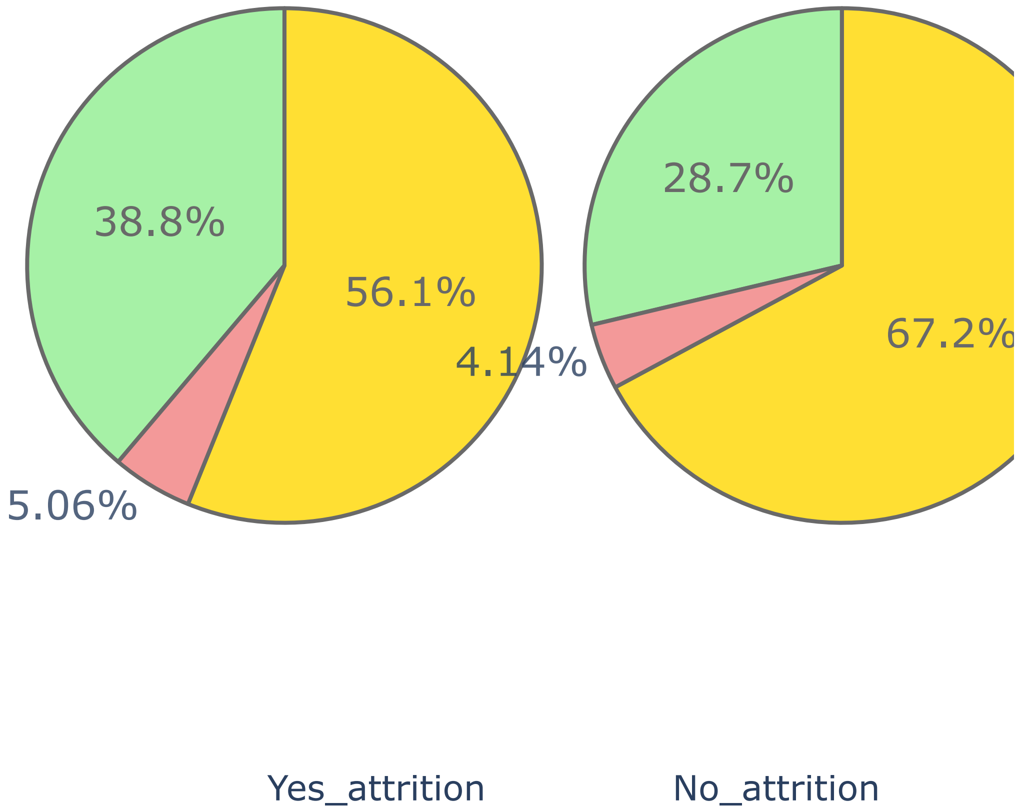


BusinessTravel
Count
0
100
200
300
400
500
600
700
800
900
Travel_Rarely
Travel_Frequently
N

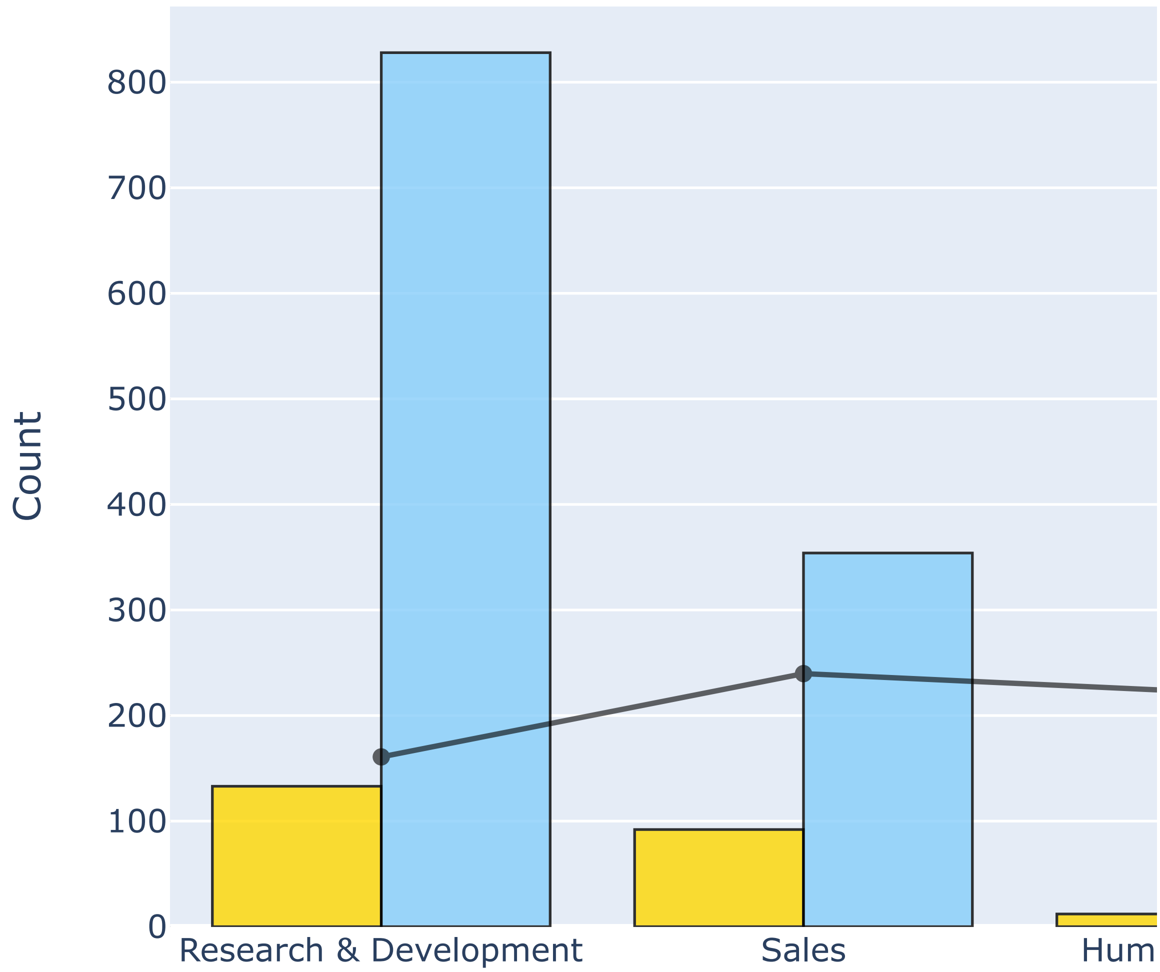

JobRole distribution in employes attrition
24.1%
26.2%
0.844%
2.11%
3.8%
4.22%
5.06%
19.8%
13.9%
19.9%
21.8%
16%
10.9%
9.89%
7.87%
Yes_attrition
No_attrition


JobRole
Count
0
50
100
150
200
250
Laboratory Technician
Sales Executive
Research Scientist
Sales Representative
Human Resources
Manufacturing
Healt

Department distribution in employes attrition

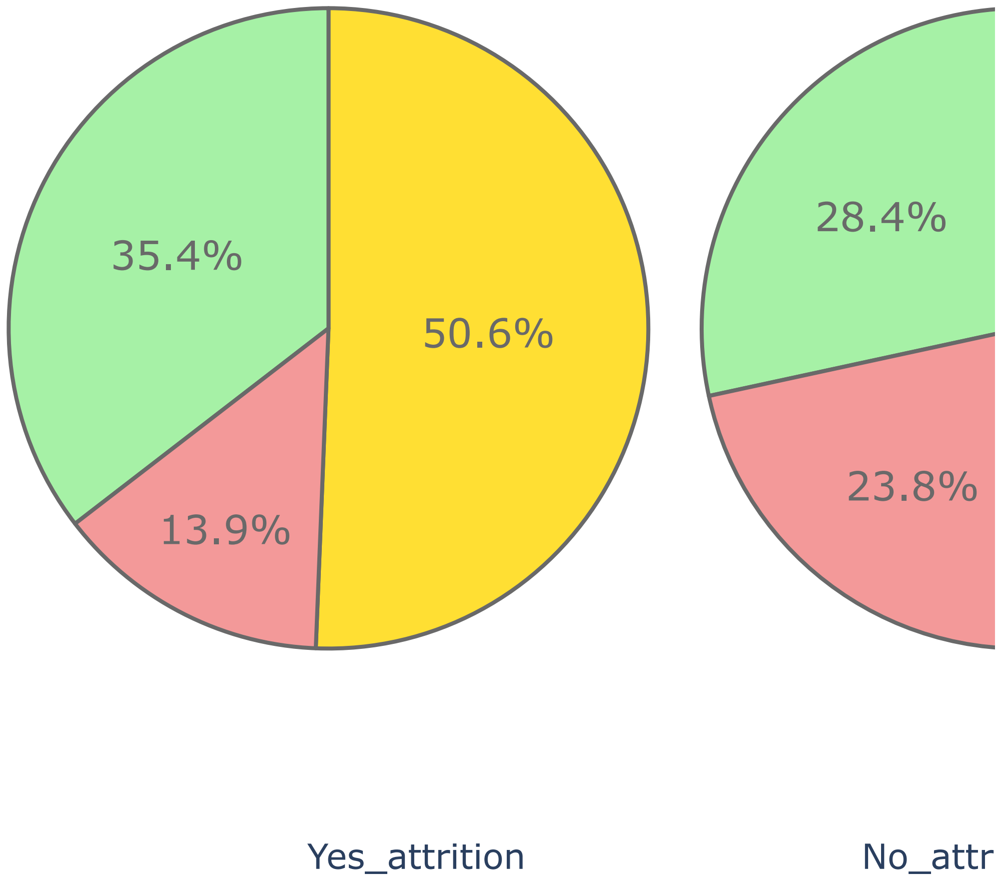


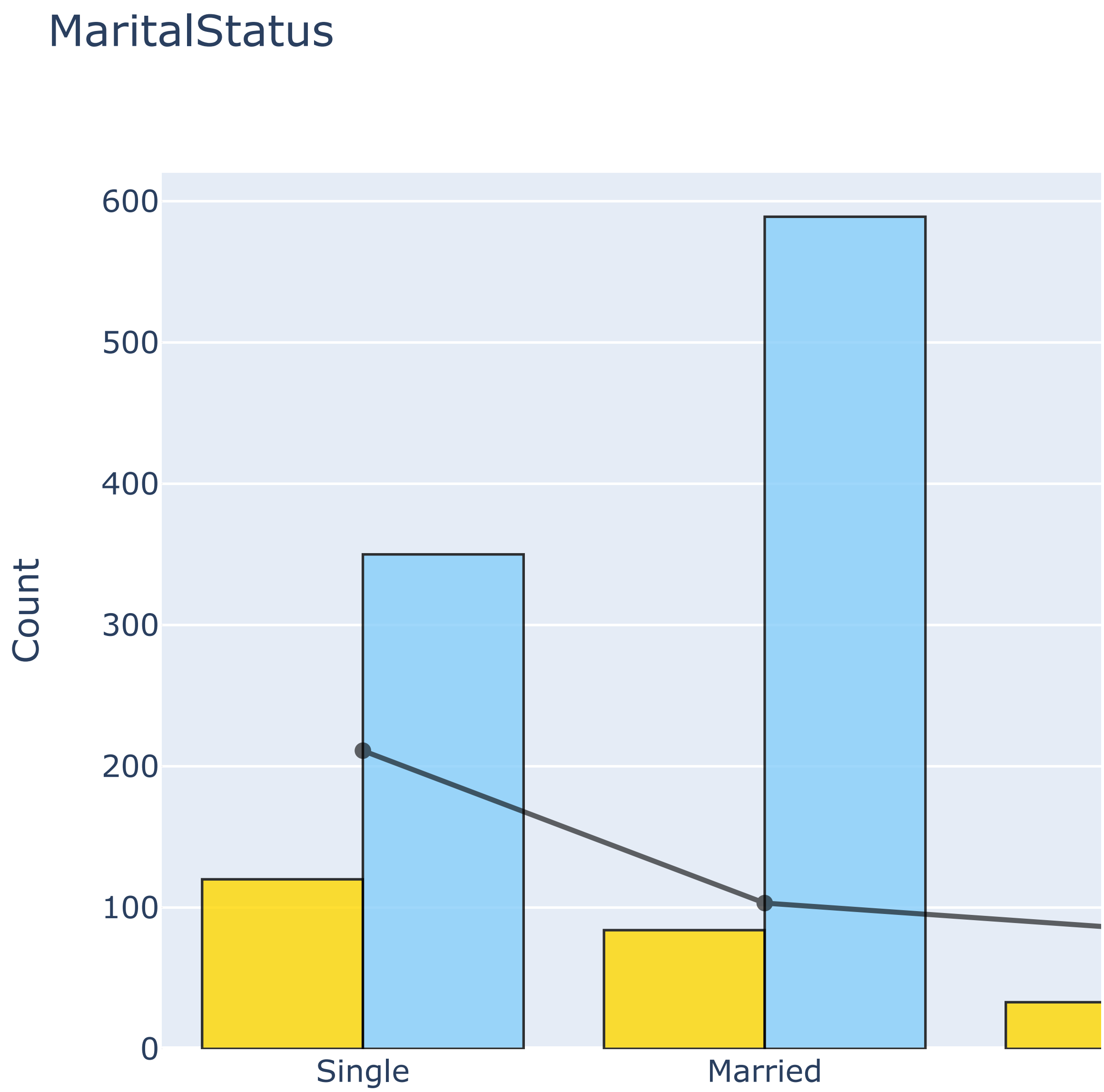
Department



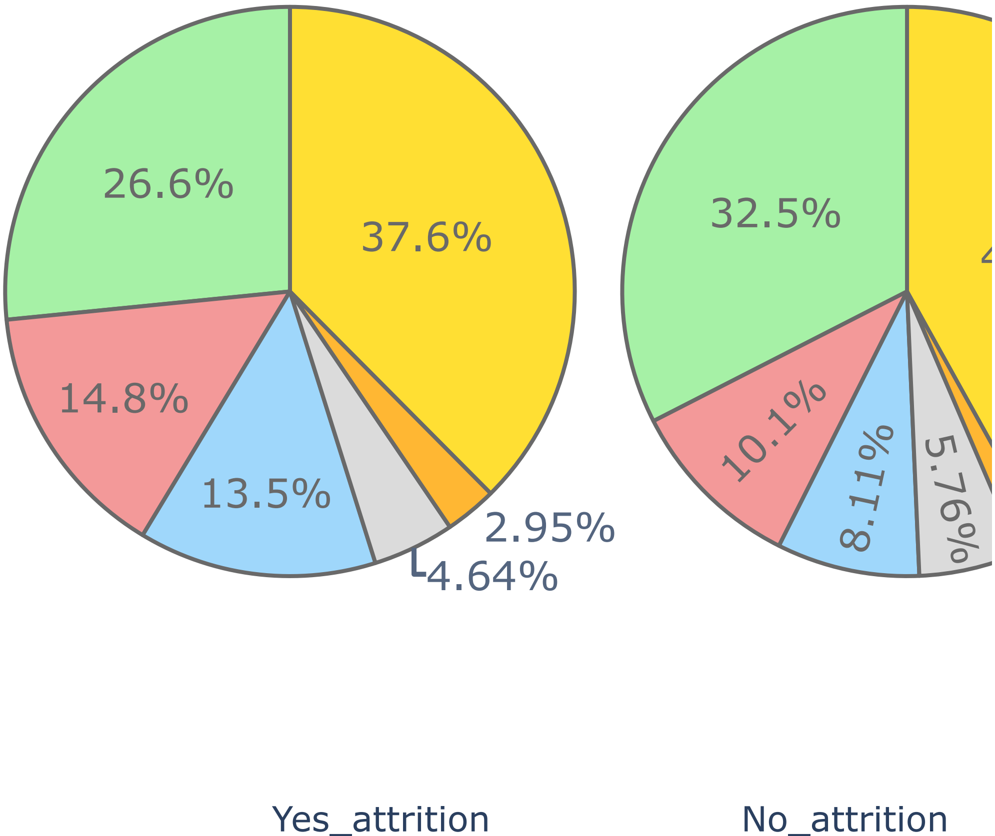
800
700
600
500
400
300
200
100
0
Count
Research & Development
Sales
Hum

MaritalStatus distribution in employes attrition

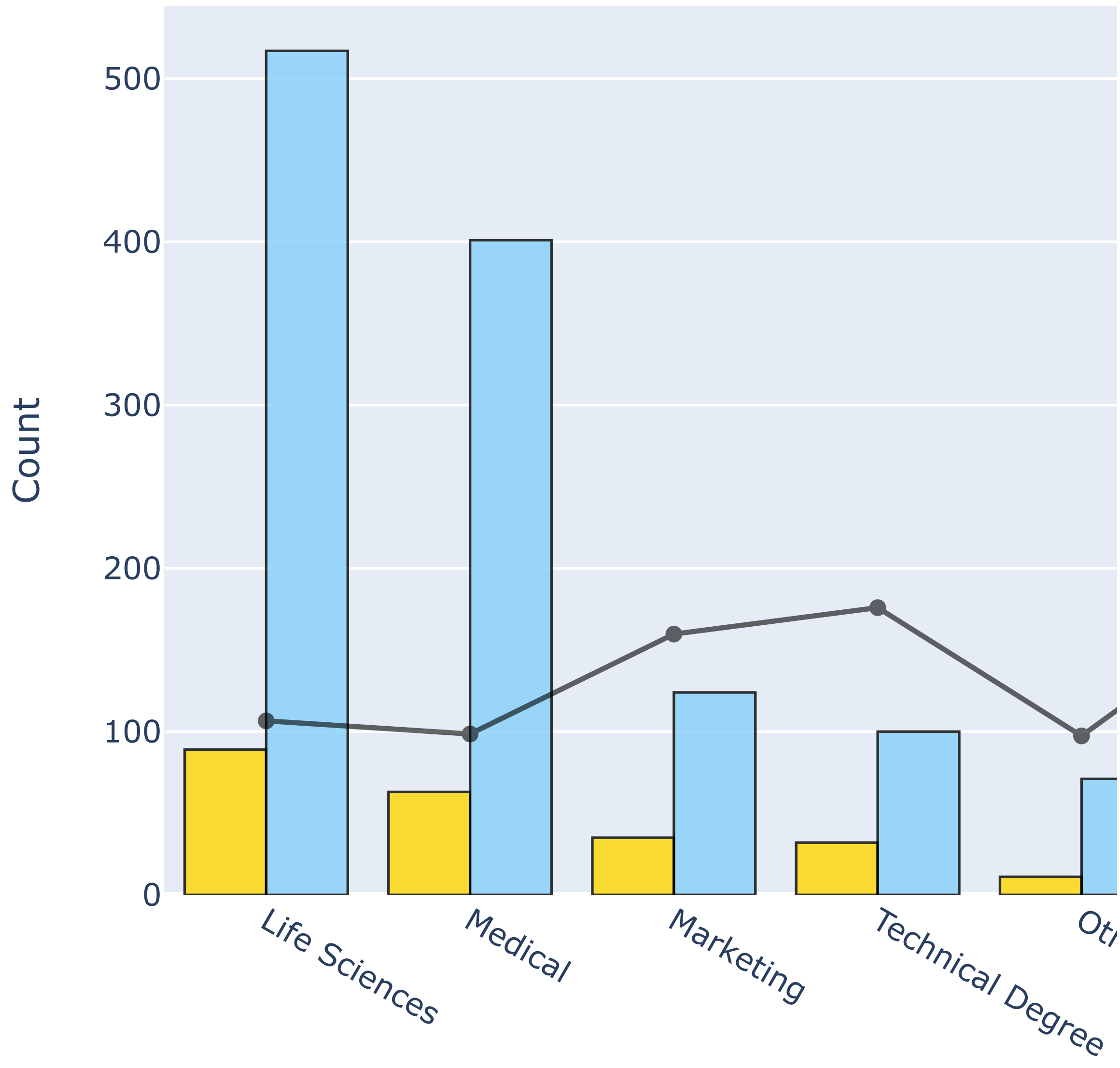



MaritalStatus
Count
600
500
400
300
200
100
0
Single
Married

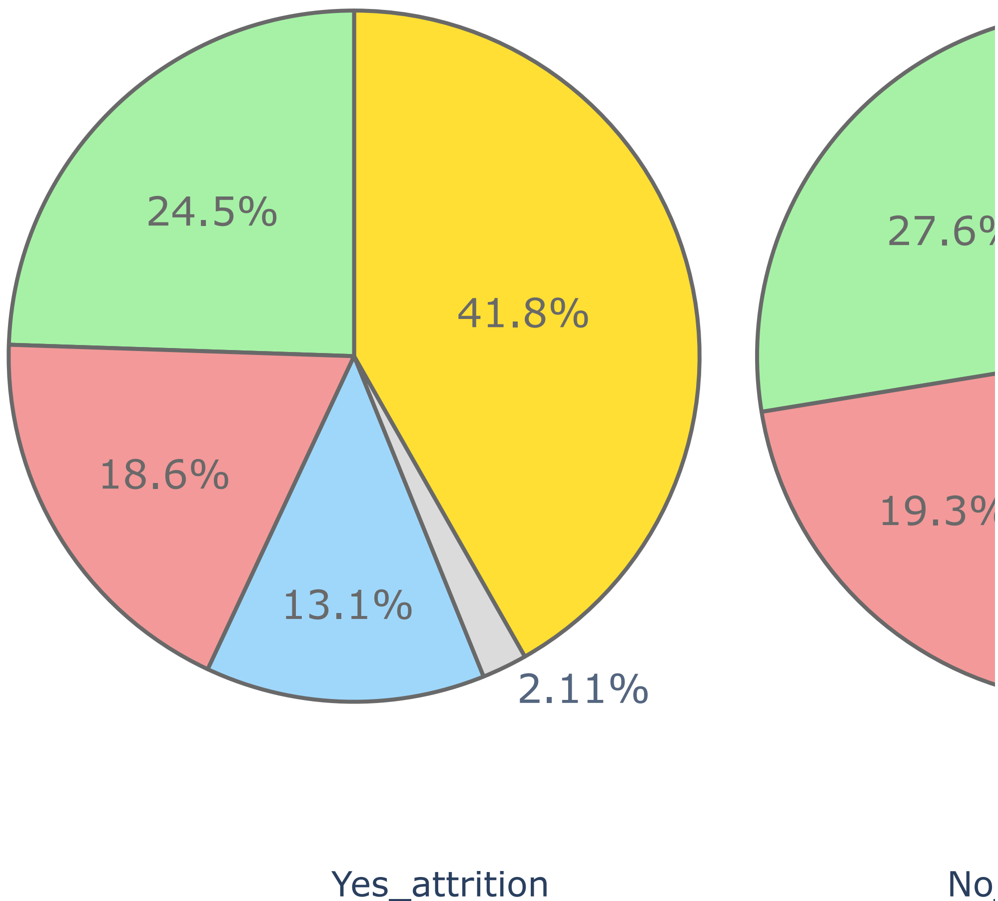
EducationField distribution in employes attrition



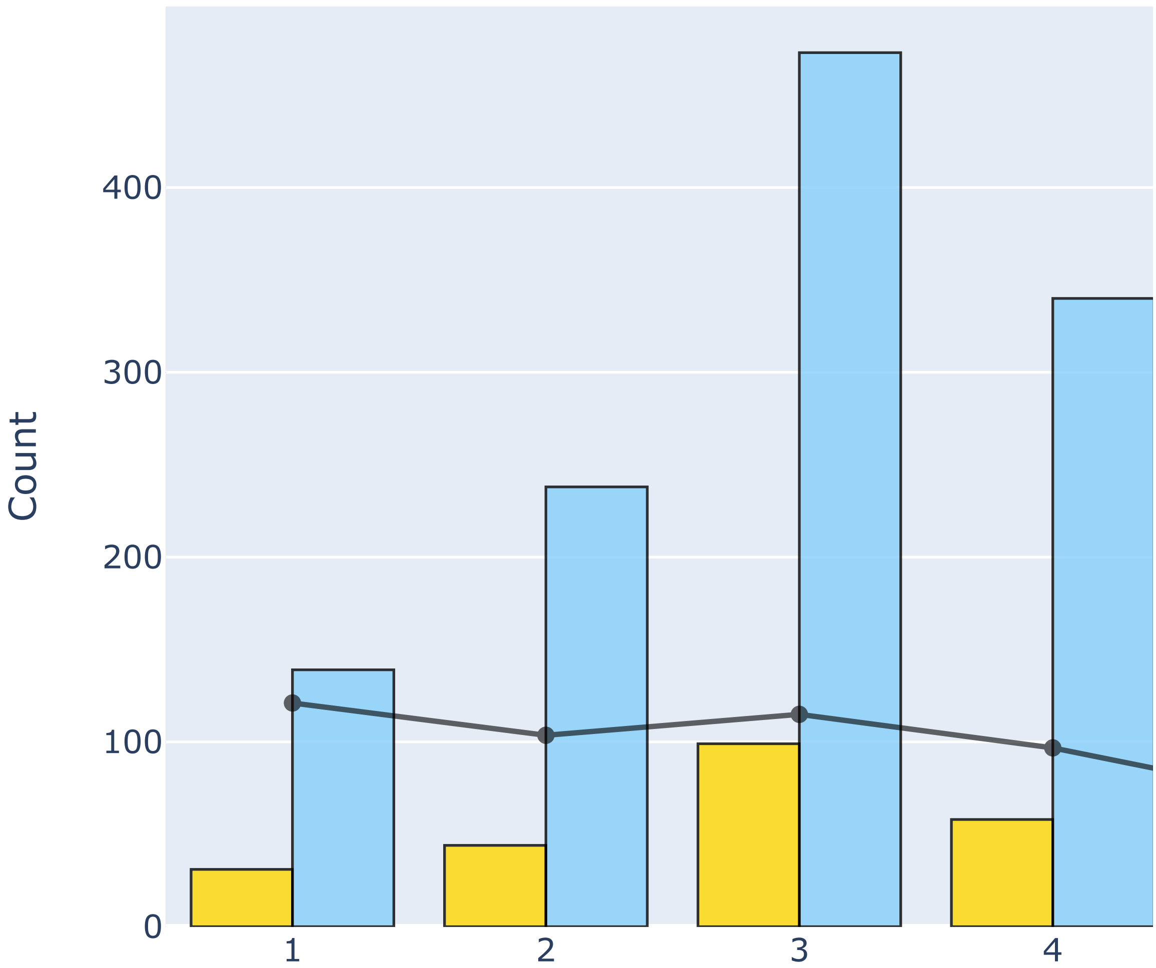
EducationField

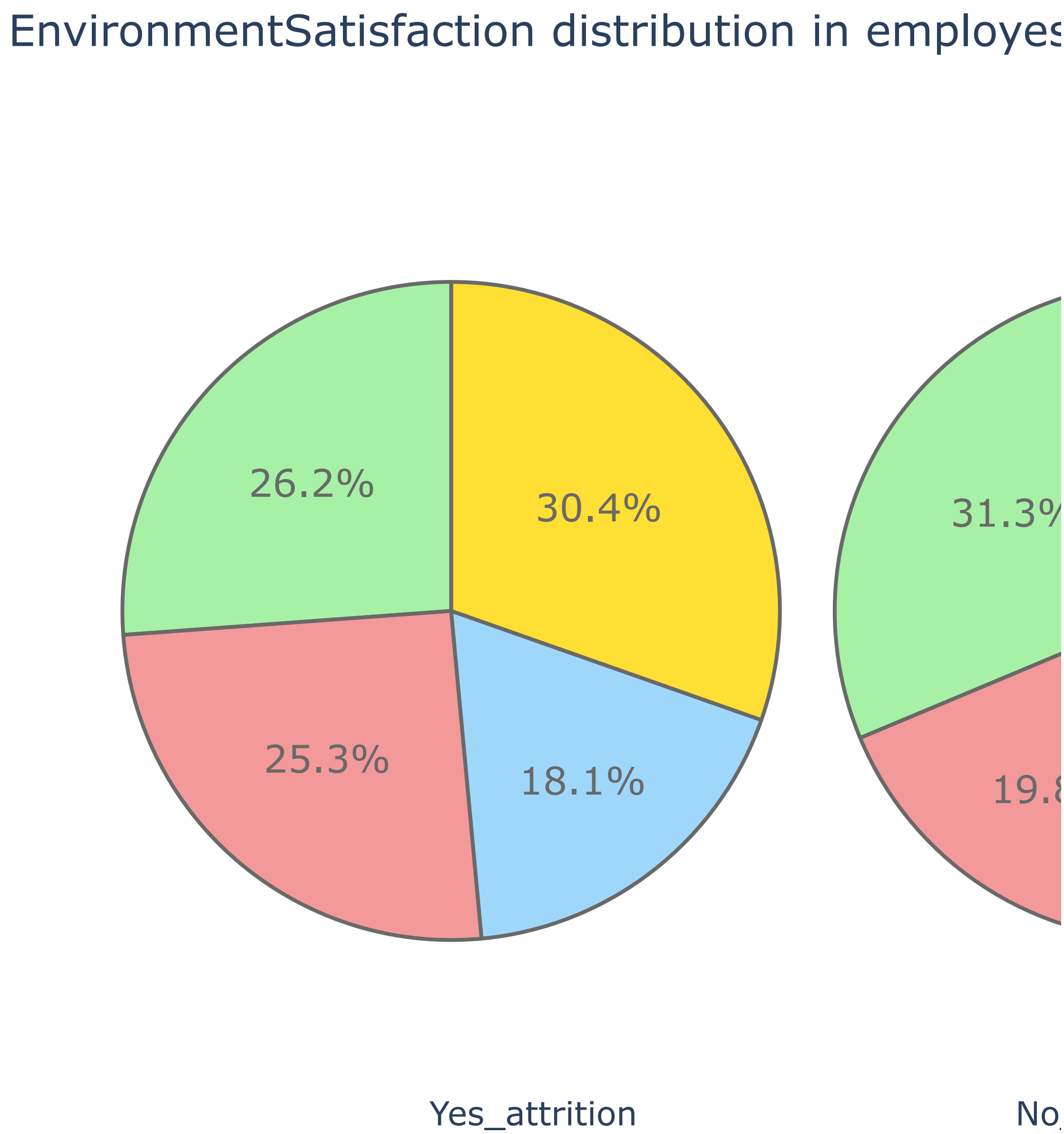


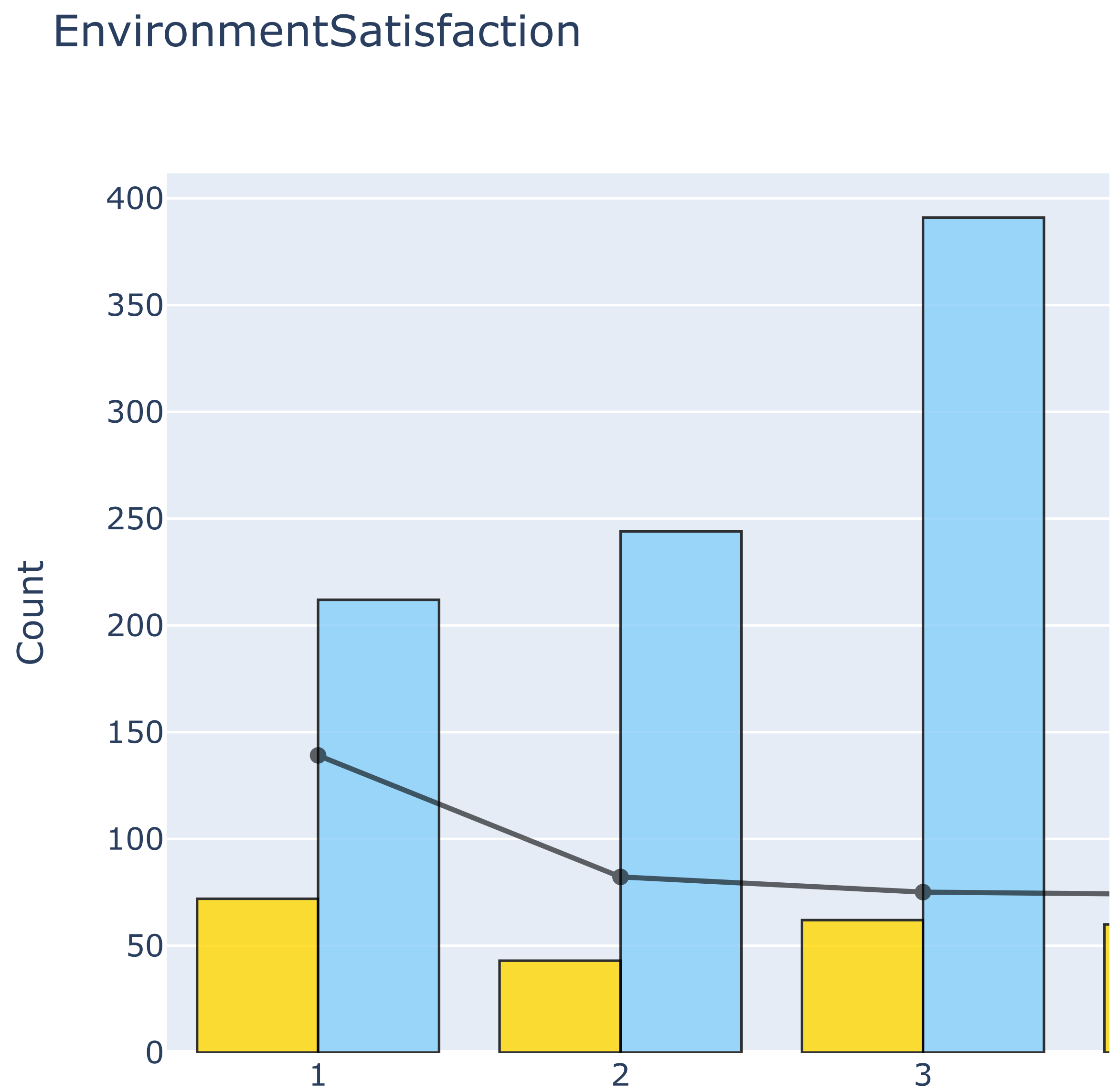
Education distribution in employes attrition



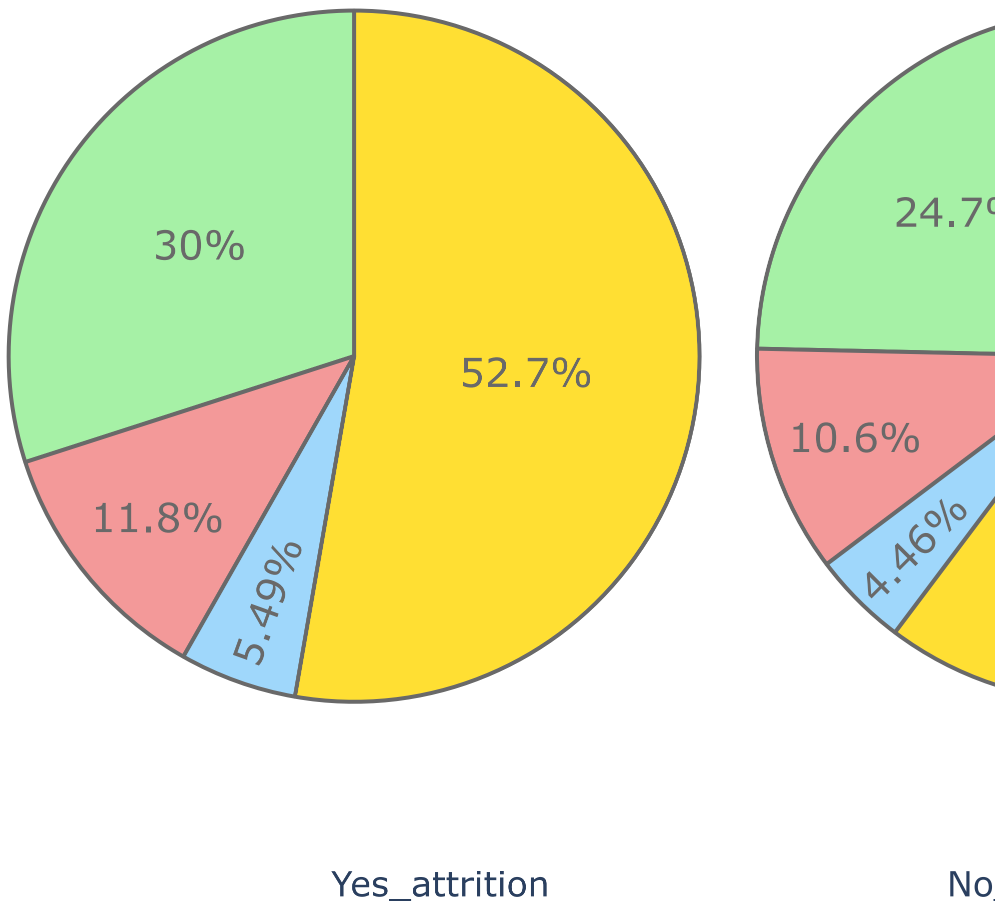
Education

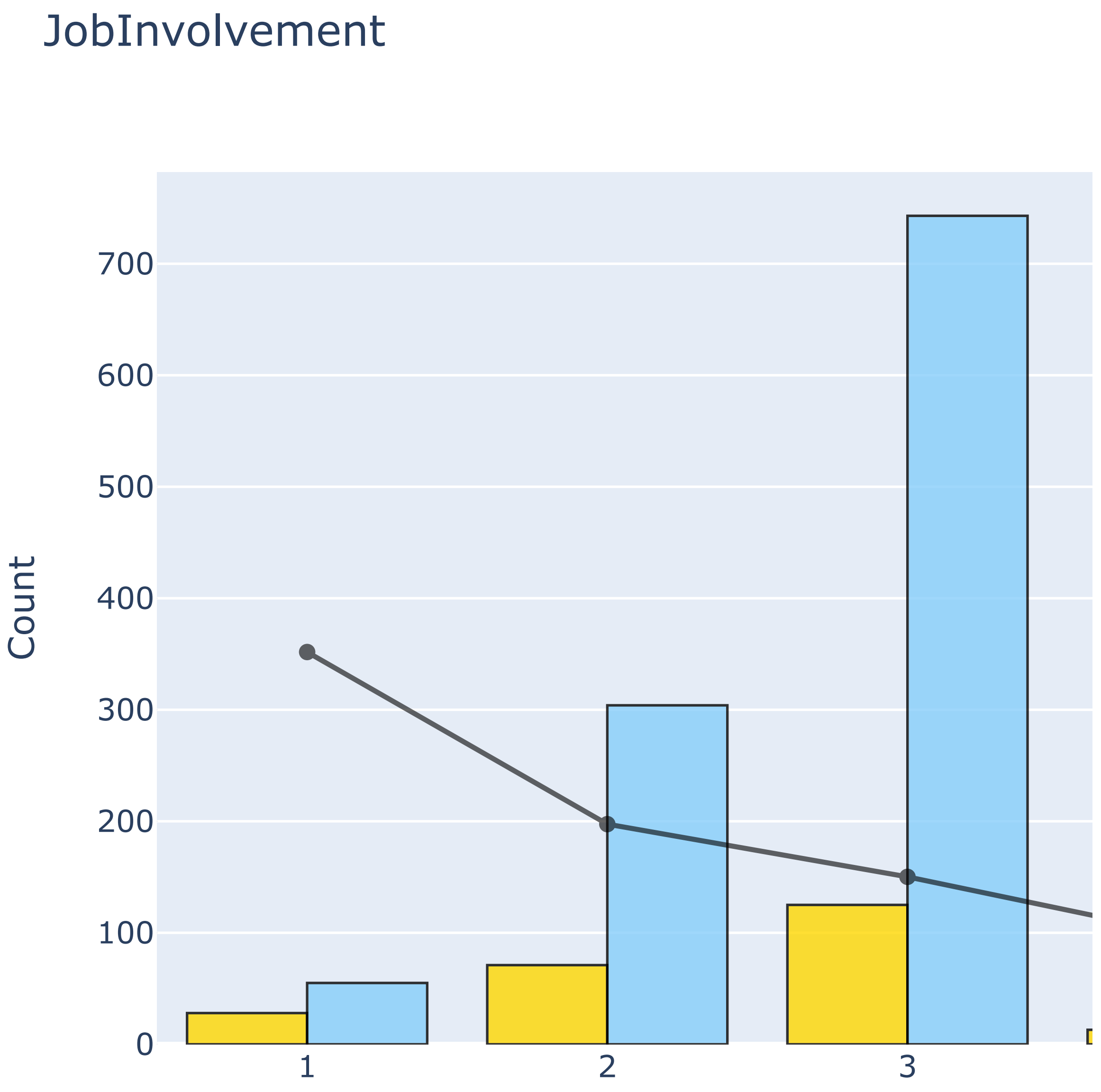



EnvironmentSatisfaction distribution in employes
26.2%
30.4%
25.3%
18.1%
31.3%
Yes_attrition


EnvironmentSatisfaction
Count
400
350
300
250
200
150
100
50
0
1
2
3

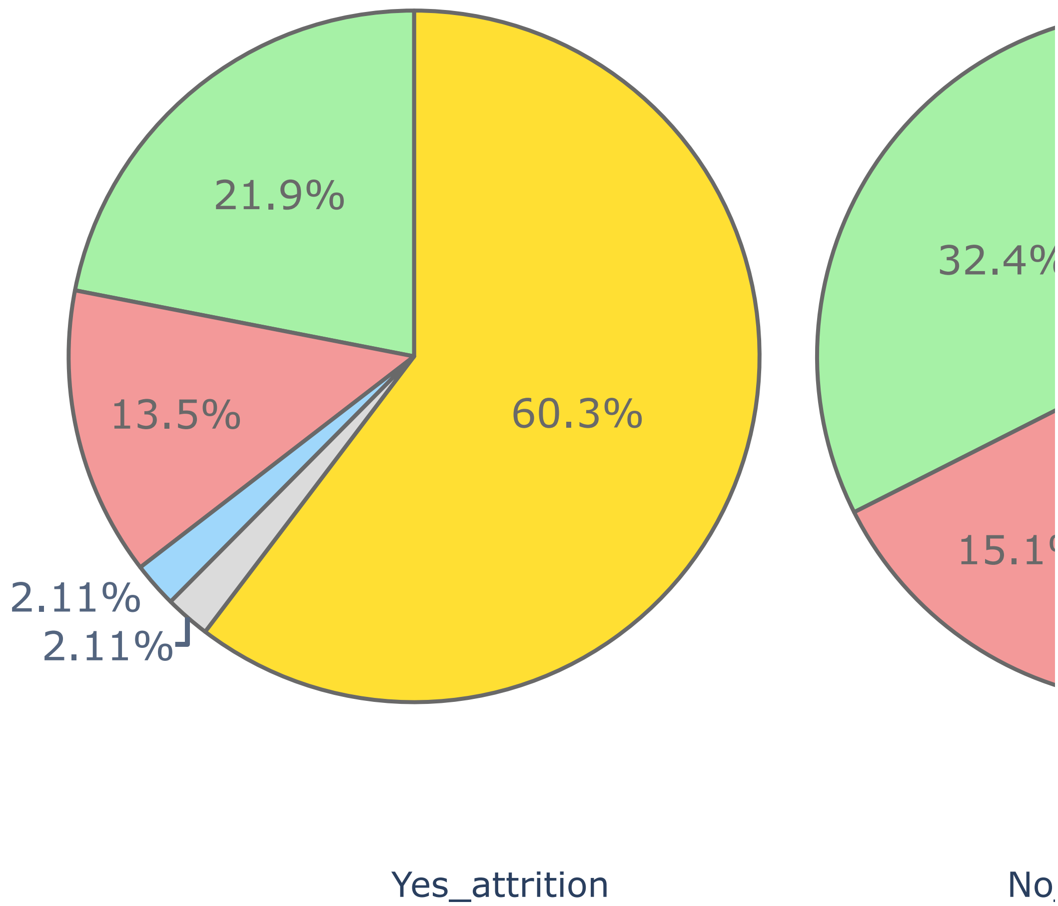
JobInvolvement distribution in employes attritior



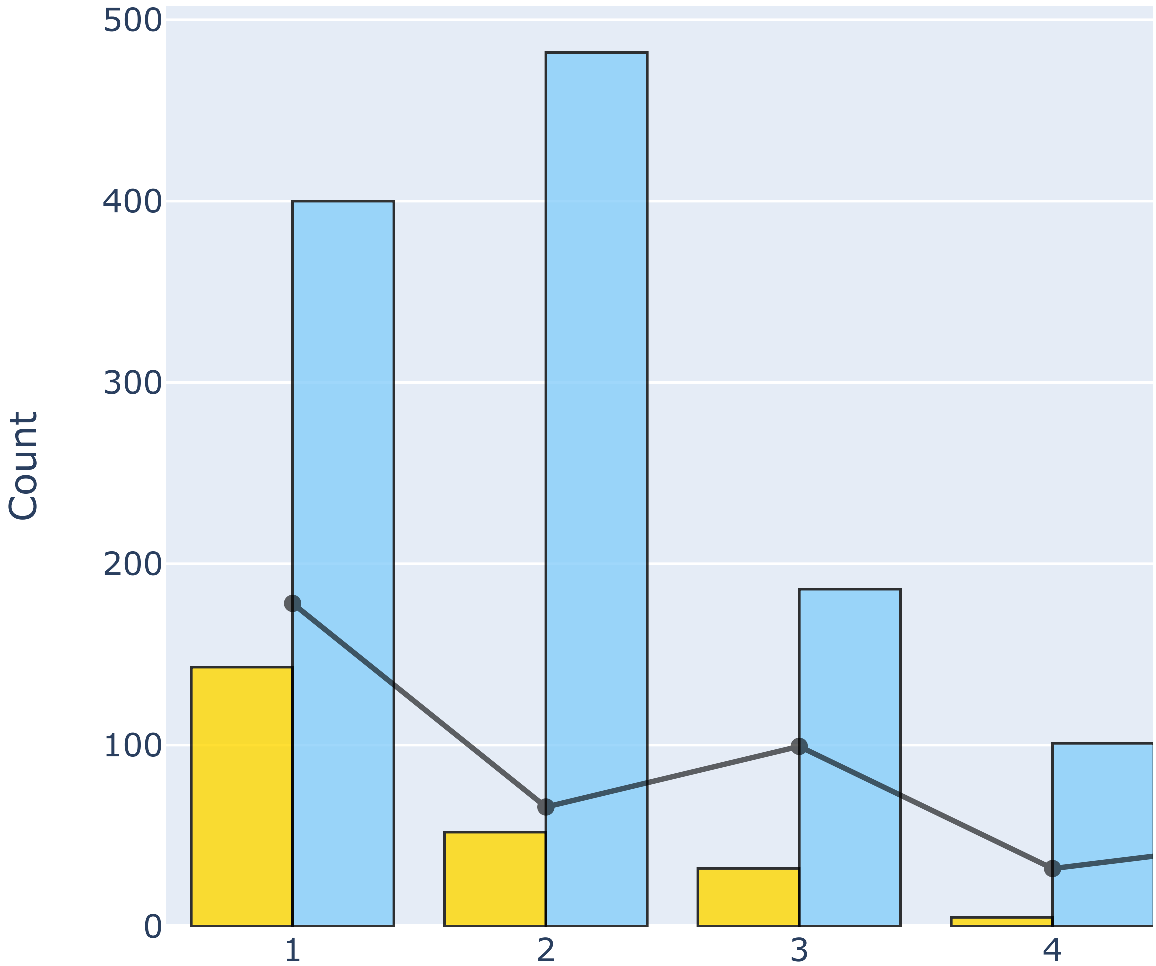


JobInvolvement
Count
0
100
200
300
400
500
600
700
1
2
3

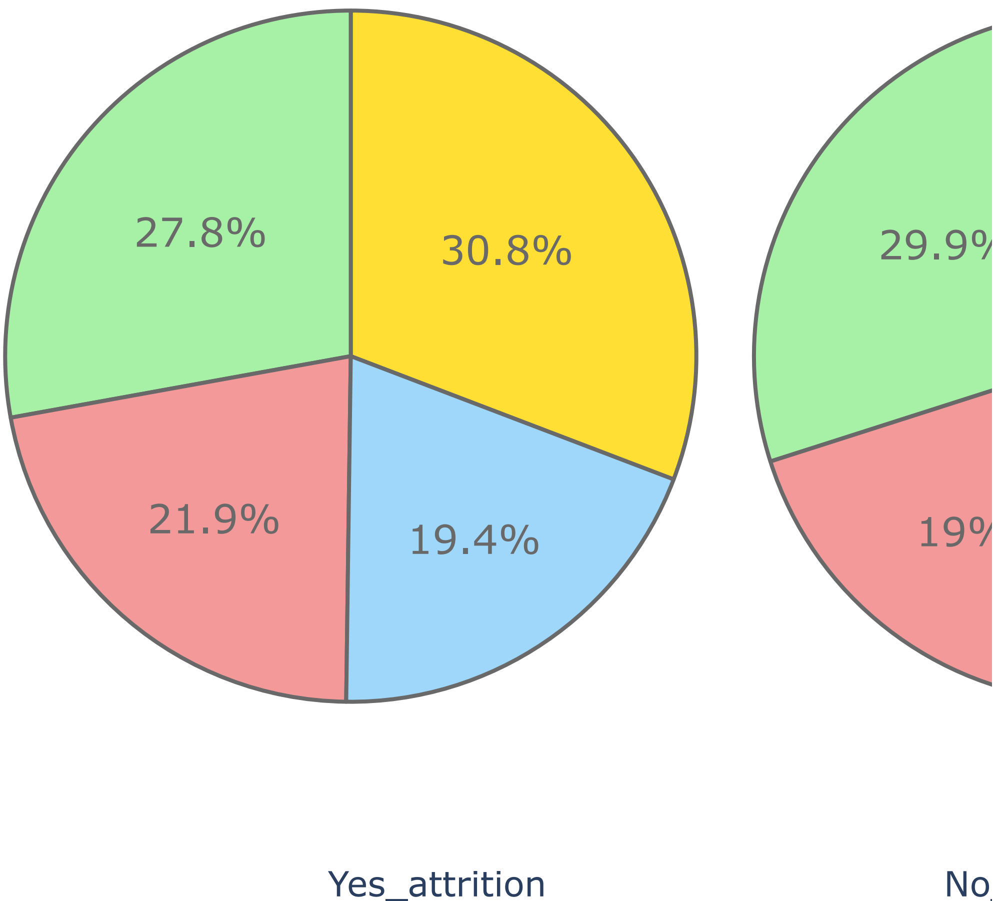
JobLevel distribution in employes attrition

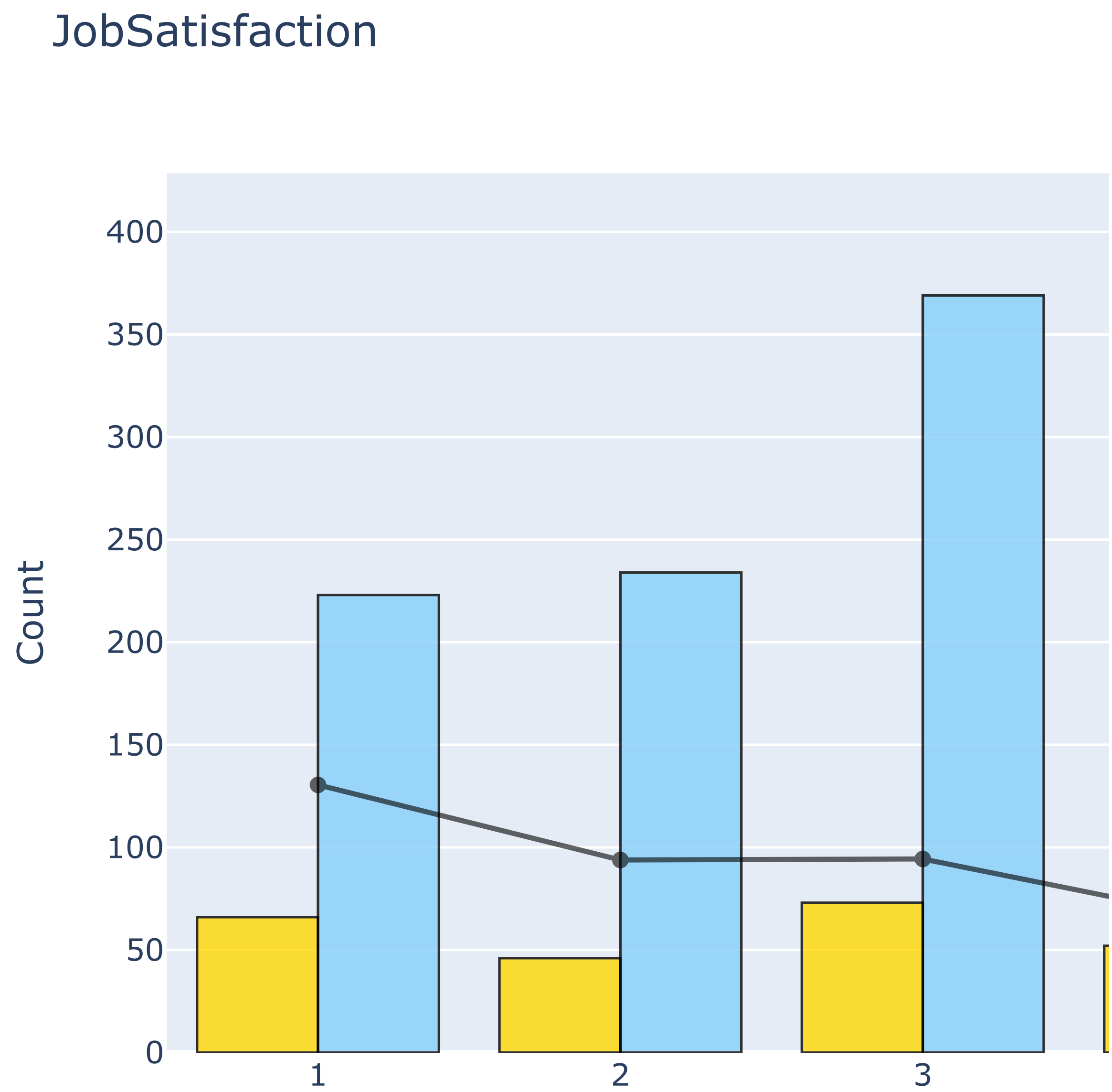


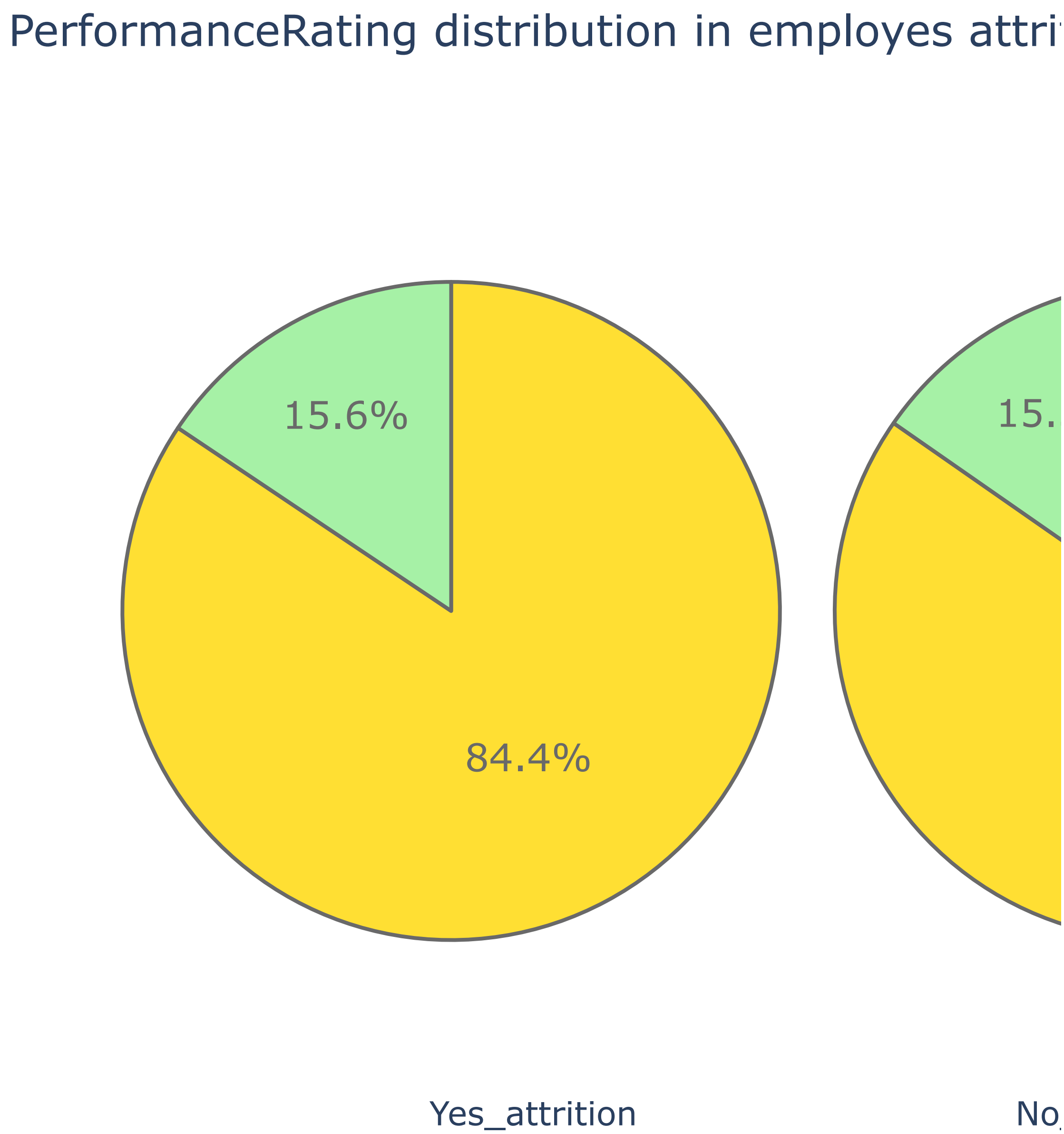
JobLevel

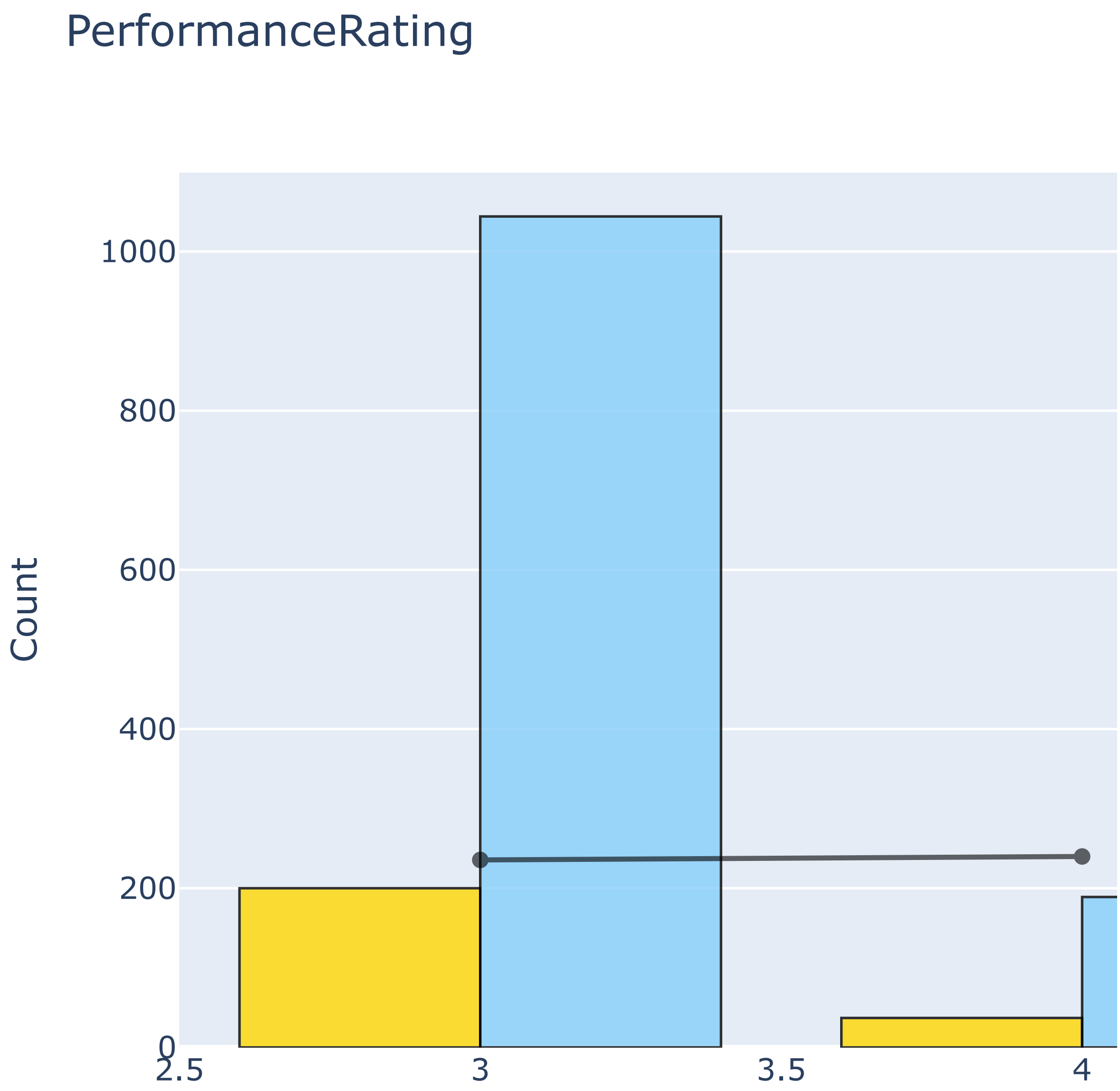


JobSatisfaction distribution in employes attrition

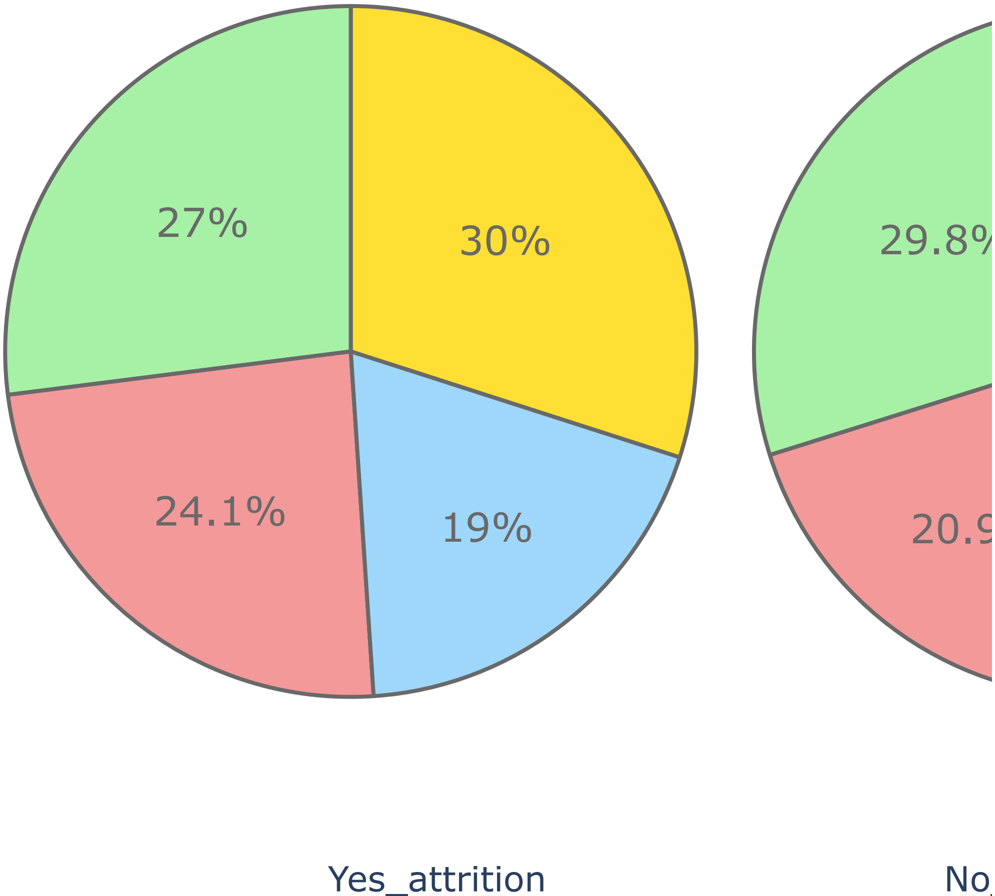



JobSatisfaction
Count
0
50
100
150
200
250
300
350
400
1
2
3

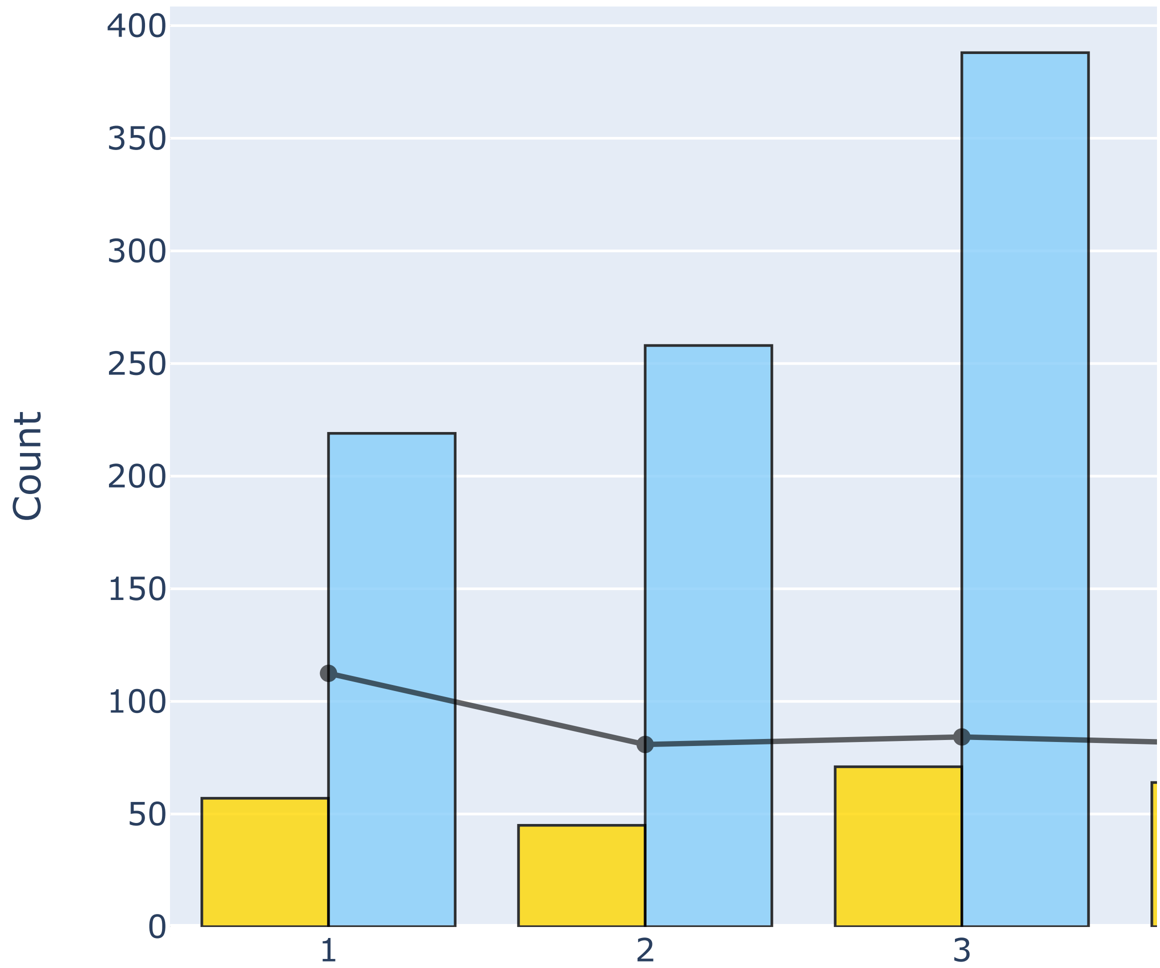

PerformanceRating distribution in employes attri
15.6%
84.4%
15.
Yes_attrition
No


PerformanceRating
Count
1000
800
600
400
200
0
2.5
3
3.5
4

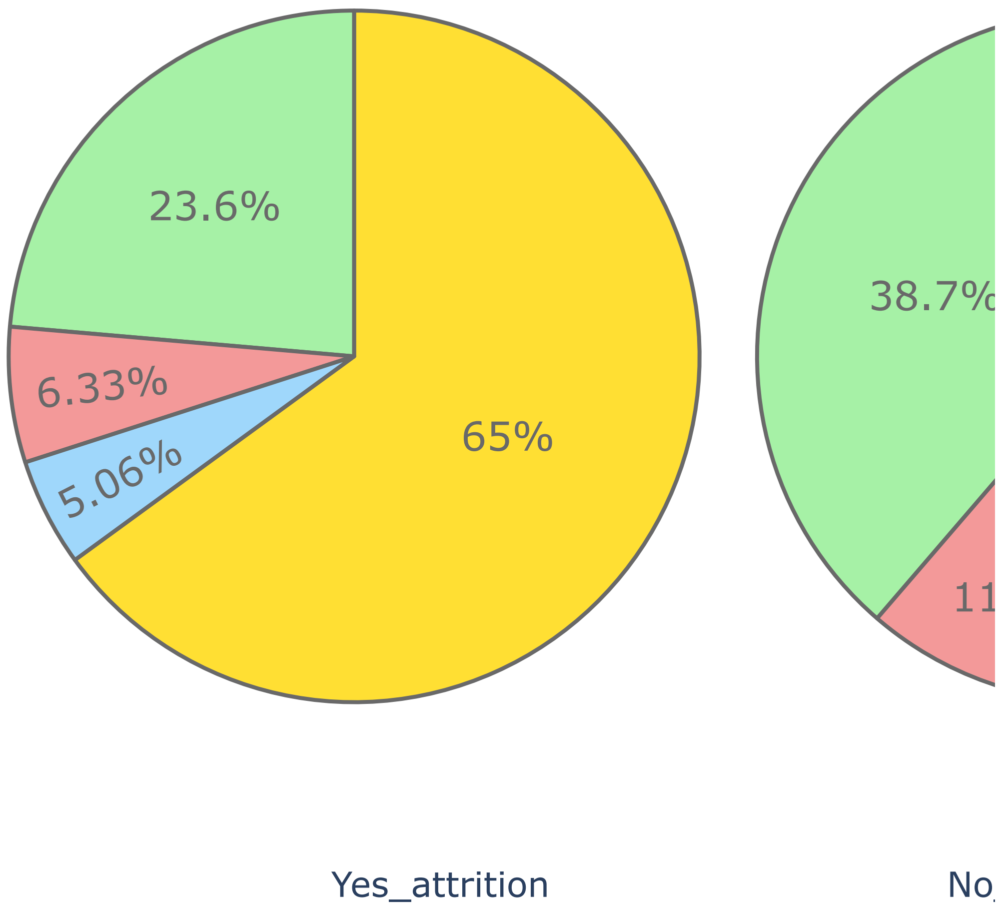
RelationshipSatisfaction distribution in employes



RelationshipSatisfaction

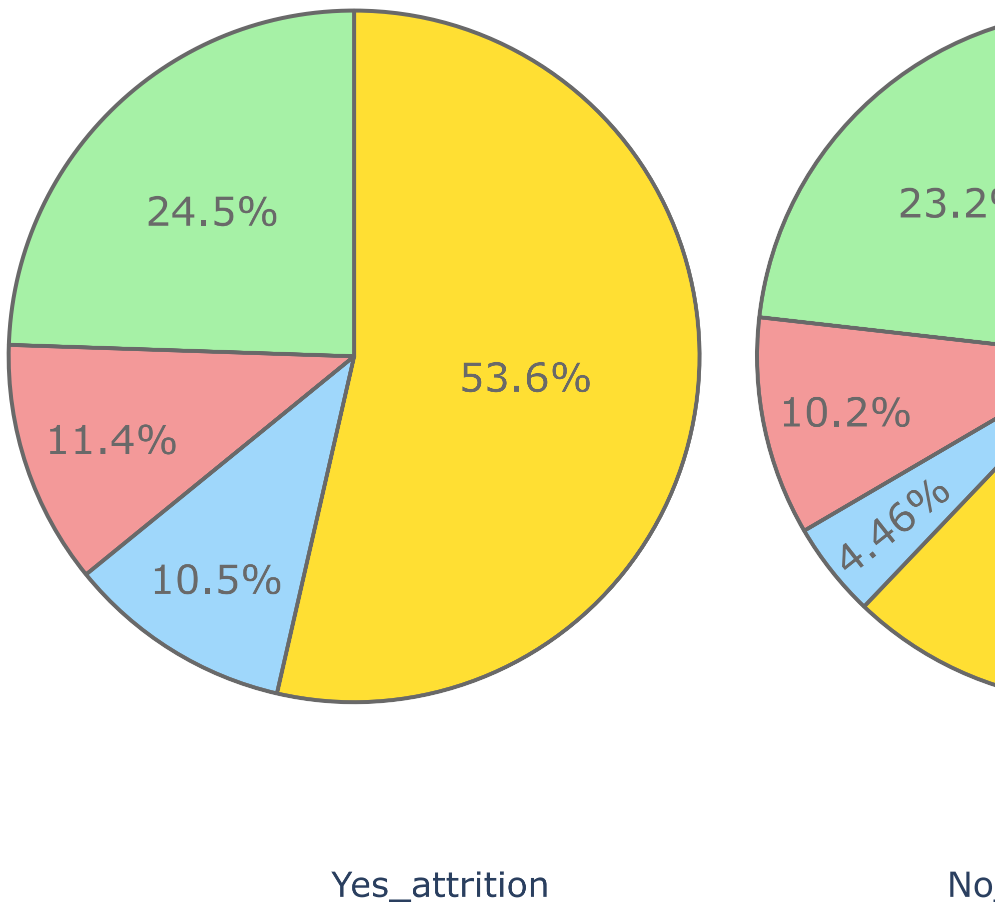


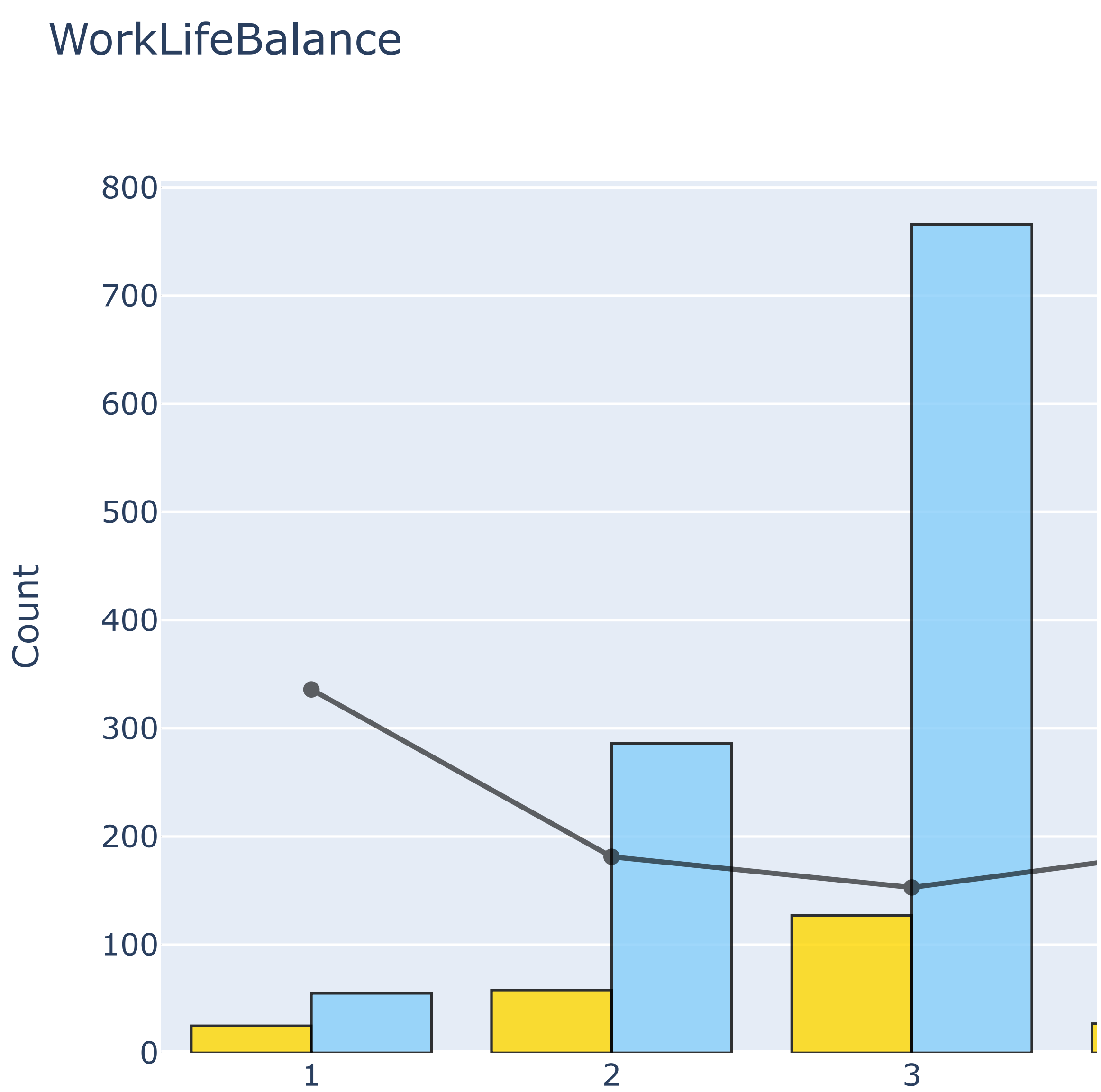
StockOptionLevel distribution in employes attritio




StockOptionLevel
Count
500
400
300
200
100
0
0
1
2

WorkLifeBalance distribution in employes attritio





3. Feature engineering and selection

3.1. New features : 24

In [14]:

```
def SalesDpt(data) :
    if data['Department'] == 'Sales':
        return 1
```

```
    else:

        return 0
data['SalesDpt'] = data.apply(lambda data:SalesDpt(data) ,axis = 1)

def JobInvCut(data) :
    if data['JobInvolvement'] < 2.5 :
        return 1
    else:
        return 0
data['JobInvCut'] = data.apply(lambda data:JobInvCut(data) ,axis = 1)

def MiddleTraining(data) :
    if data['TrainingTimesLastYear'] >= 3 and data['TrainingTimesLastYear'] <= 6:
        return 1
    else:
        return 0
data['MiddleTraining'] = data.apply(lambda data:MiddleTraining(data) ,axis = 1)

def MoovingPeople(data) :
    if data['NumCompaniesWorked'] > 4:
        return 1
    else:
        return 0
data['MoovingPeople'] = data.apply(lambda data:MoovingPeople(data), axis = 1)

data['TotalSatisfaction_mean'] = (data['RelationshipSatisfaction']  + data['EnvironmentSatisfaction'] + data['JobSatisfaction'] + data['JobInvolvement'] + data['WorkLifeBalance'])/5

def NotSatif(data) :
    if  data['TotalSatisfaction_mean'] < 2.35 :
        return 1
    else :
        return 0
data['NotSatif'] = data.apply(lambda data:NotSatif(data) ,axis = 1)

def LongDisWL1(data) :
    if  data['DistanceFromHome'] > 11 and data['WorkLifeBalance'
```

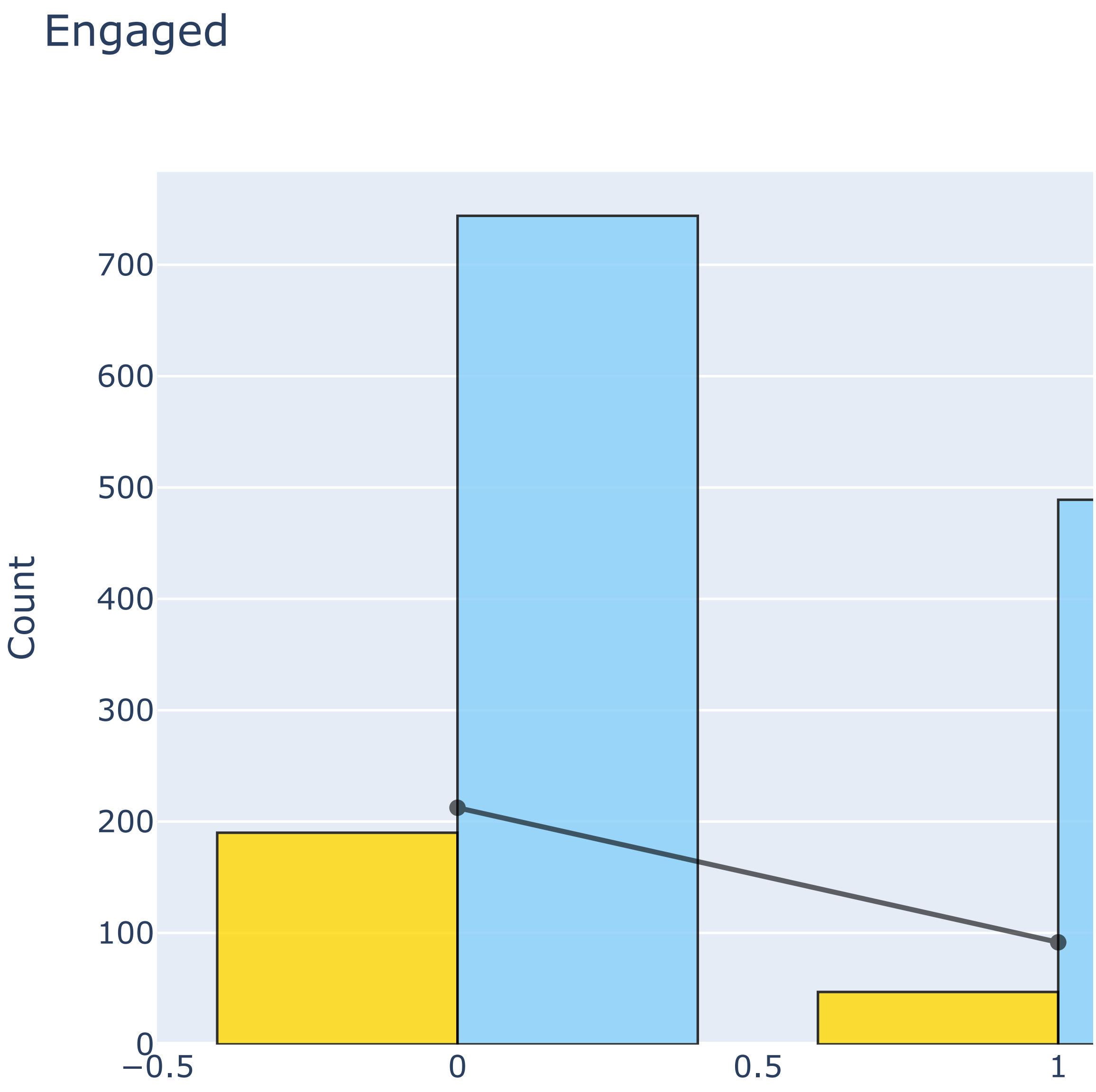
```
] == 1 :
        return 1
    else :
        return 0
data['LongDisWL1'] = data.apply(lambda data:LongDisWL1(data) ,ax
is = 1)

def LongDis(data) :
    if  data['DistanceFromHome'] > 11:
        return 1
    else :
        return 0
data['LongDis'] = data.apply(lambda data:LongDis(data) ,axis = 1
)

def LongDisJobS1(data) :
    if  data['DistanceFromHome'] > 11 and data['JobSatisfaction'
] == 1 :
        return 1
    else :
        return 0
data['LongDisJobS1'] = data.apply(lambda data:LongDisJobS1(data)
,axis = 1)

def LongDisJL1(data) :
    if  data['DistanceFromHome'] > 11 and data['JobLevel'] == 1
:
        return 1
    else :
        return 0
data['LongDisJL1'] = data.apply(lambda data:LongDisJL1(data) ,ax
is = 1)

def ShortDisNotSingle(data) :
    if  data['MaritalStatus'] != 'Single' and data['DistanceFrom
Home'] < 5:
        return 1
    else :
        return 0
data['ShortDisNotSingle'] = data.apply(lambda data:ShortDisNotSi
ngle(data) ,axis = 1)

def LongDisSingle(data) :
    if  data['MaritalStatus'] == 'Single' and data['DistanceFrom
Home'] > 11:
```

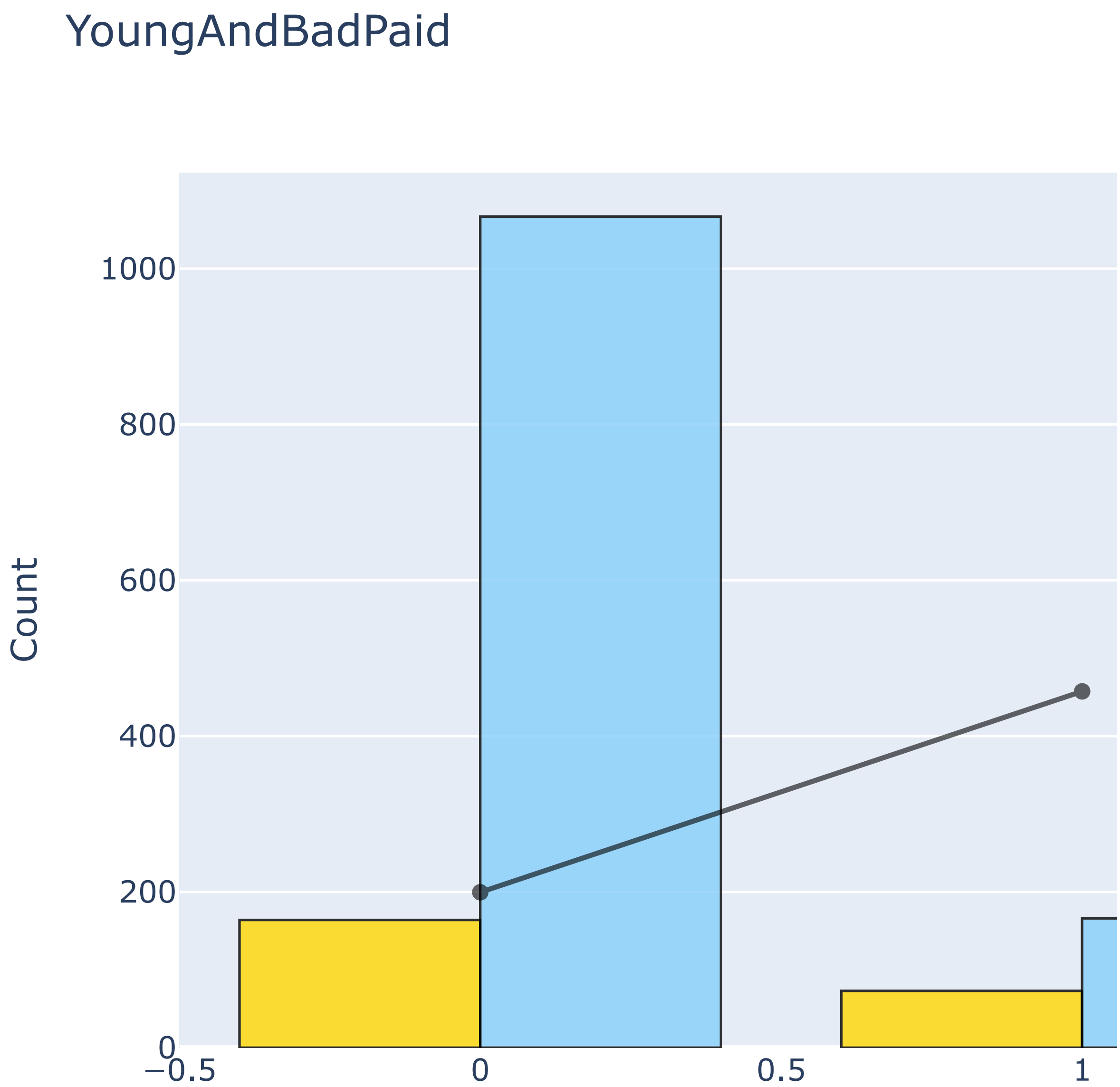
```
        return 1

    else :
        return 0
data['LongDisSingle'] = data.apply(lambda data:LongDisSingle(data) ,axis = 1)

def Engaged(data) :
    if data['Age'] > 35 and data['MaritalStatus'] != 'Single':
        return 1
    else :
        return 0
data['Engaged'] = data.apply(lambda data:Engaged(data) ,axis = 1)

def YoungAndBadPaid(data) :
    if data['Age'] < 35 and data['Age'] > 23 and (data['MonthlyIncome'] < 3500):
        return 1
    else :
        return 0
data['YoungAndBadPaid'] = data.apply(lambda data:YoungAndBadPaid(data) ,axis = 1)

def YoungNeverEngaged(data) :
    if data['Age'] < 24 and data['MaritalStatus'] == 'Single' :
        return 1
    else :
        return 0
data['YoungNeverEngaged'] = data.apply(lambda data:YoungNeverEngaged(data) ,axis = 1)

data['Time_in_each_comp'] = (data['Age'] - 20) / ((data)['NumCompaniesWorked'] + 1)
data['RelSatisf_mean'] = (data['RelationshipSatisfaction']  + data['EnvironmentSatisfaction']) / 2
data['JobSatisf_mean'] = (data['JobSatisfaction'] + data['JobInvolvement']) / 2
data['Income_Distance'] = data['MonthlyIncome'] / data['DistanceFromHome']
data['Hrate_Mrate'] = data['HourlyRate'] / data['MonthlyRate']
data['Stability'] = data['YearsInCurrentRole'] / data['YearsAtCompany']
data['Stability'].fillna((data['Stability'].mean()), inplace=True)
```

```
data['Income_YearsComp'] = data['MonthlyIncome'] / data['YearsAtCompany']
data['Income_YearsComp'] = data['Income_YearsComp'].replace(np.Inf, 0)
data['Fidelity'] = (data['NumCompaniesWorked']) / data['TotalWorkingYears']
data['Fidelity'] = data['Fidelity'].replace(np.Inf, 0)
```

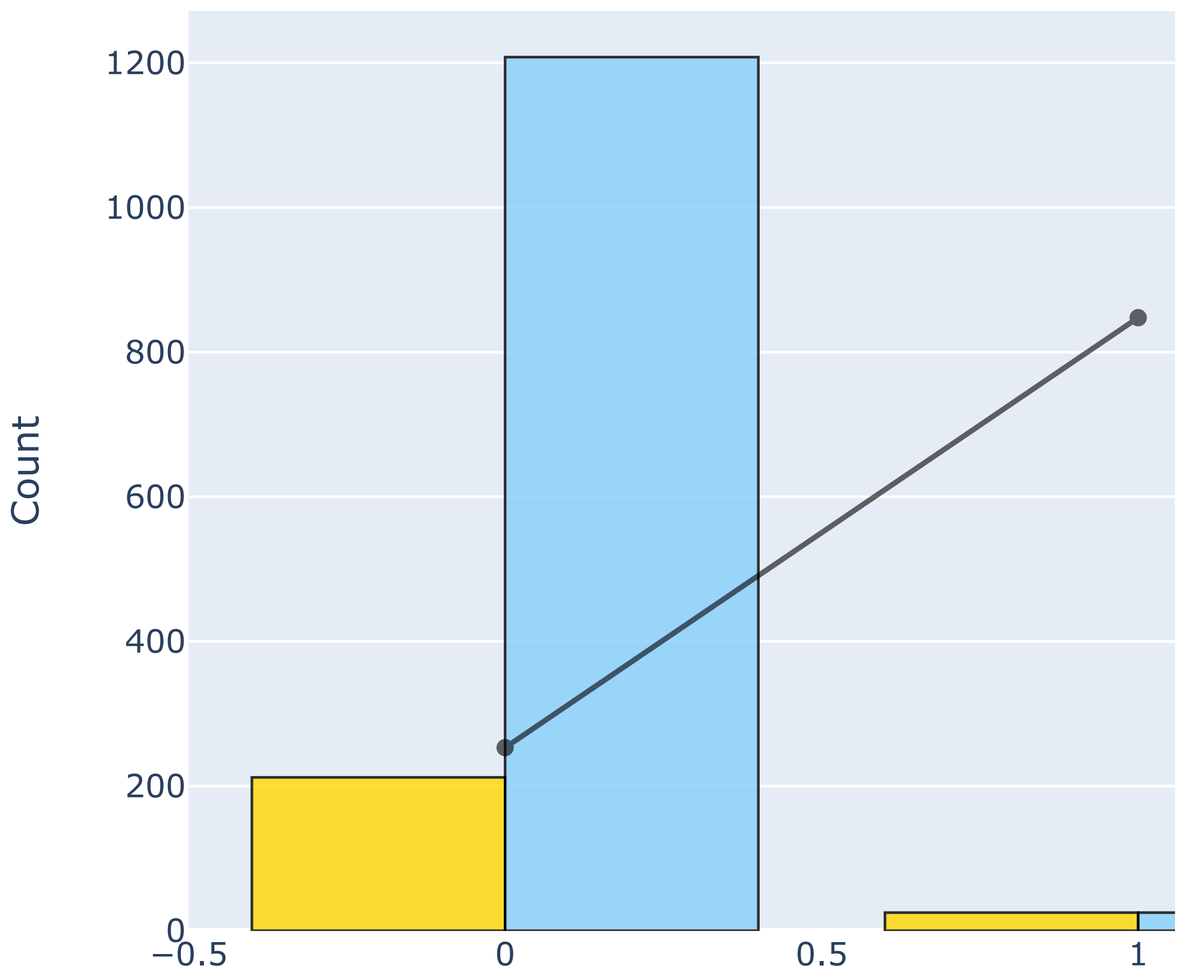
In [15]:

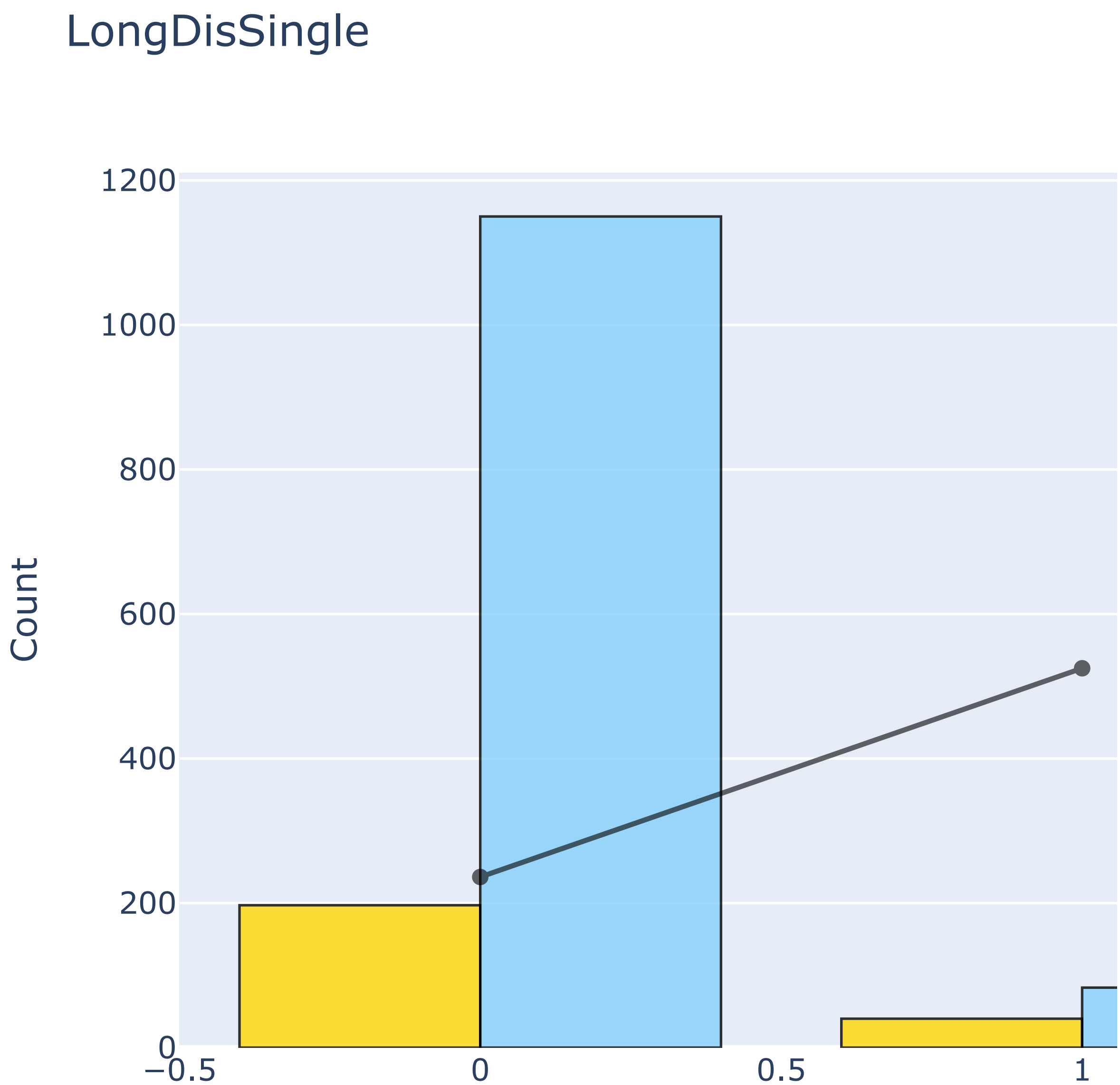
```
barplot('Engaged', False)
barplot('YoungAndBadPaid', False)
barplot('YoungNeverEngaged', False)
barplot('LongDisSingle', False)
barplot('LongDisJL1', False)
barplot('ShortDisNotSingle', False)
```


Engaged
Count
700
600
500
400
300
200
100
0
−0.5
0
0.5
1

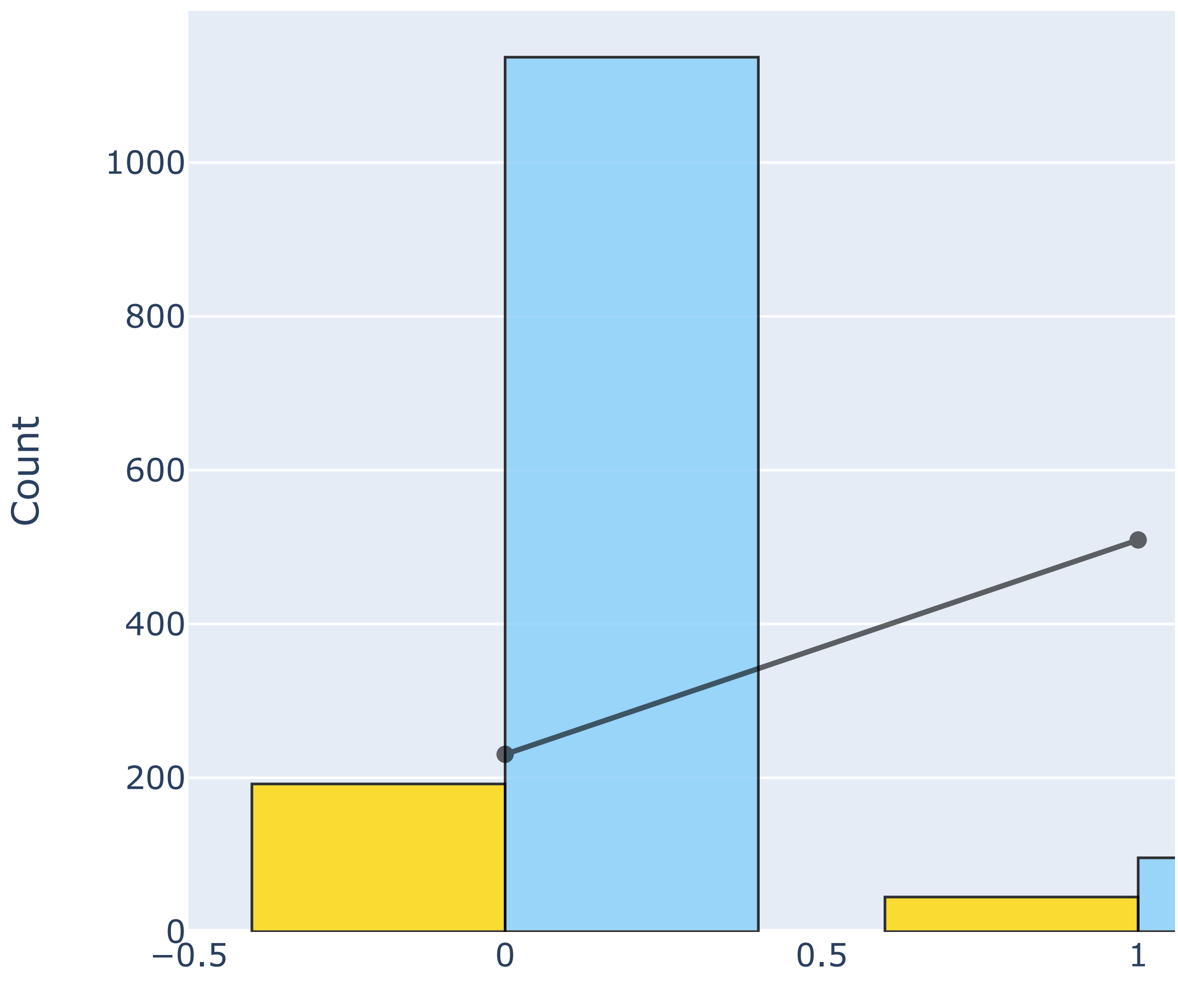

YoungAndBadPaid
Count
1000
800
600
400
200
0
−0.5
0
0.5
1

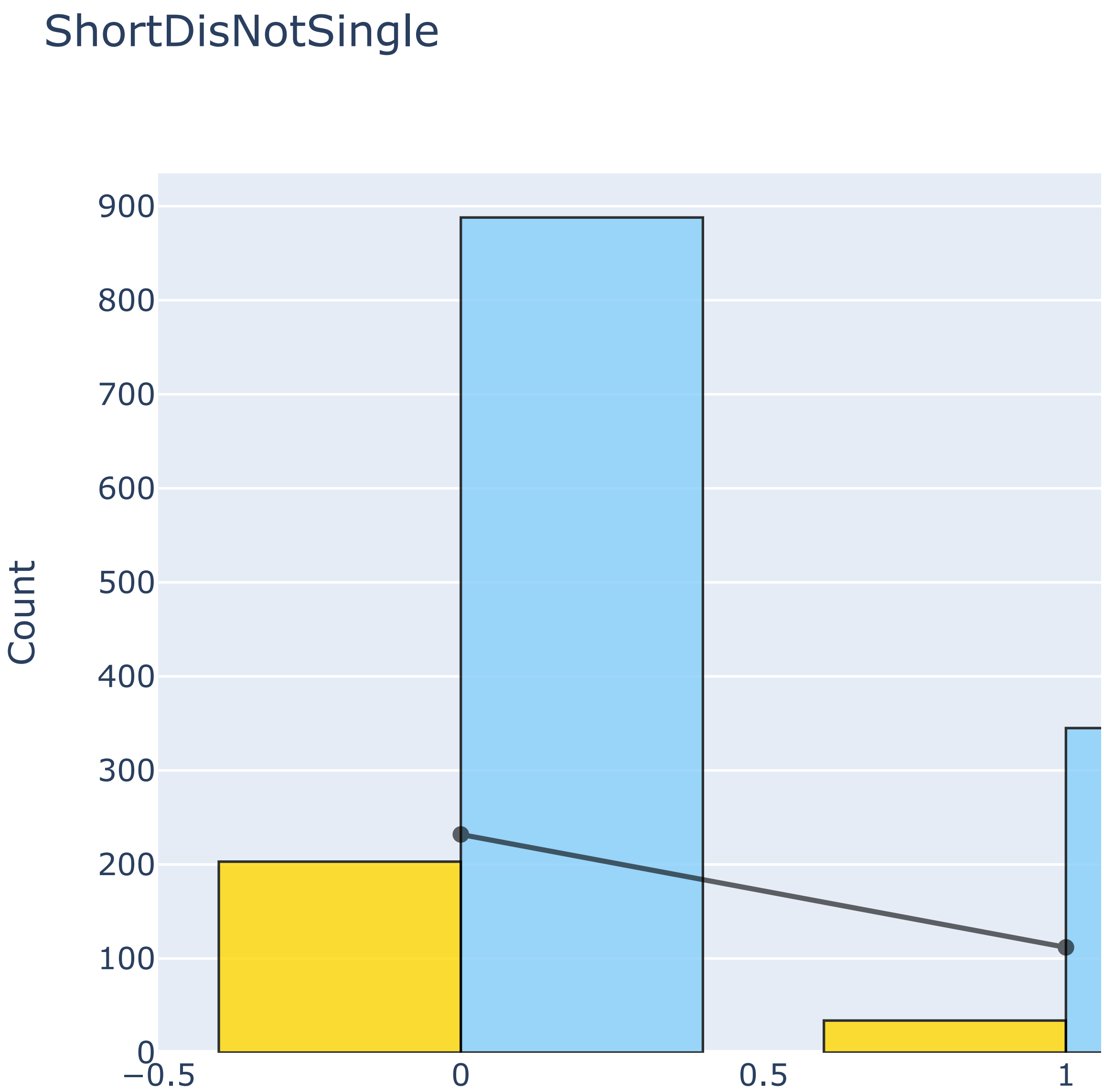
YoungNeverEngaged




LongDisSingle
Count
1200
1000
800
600
400
200
0
−0.5
0
0.5
1

LongDisJL1





3.2. Drop some features

In [16]:

```
data = data.drop(columns=[
                        'Age',
                        'MonthlyIncome',
                        'YearsAtCompany',
                        'DistanceFromHome',
                        'PerformanceRating',
                        'NumCompaniesWorked'
                     ])

print ("\nMissing values :  ", data.isnull().sum().values.sum())
```

```
('\nMissing values :  ', 0)
```

3.3. Features encoding and scaling

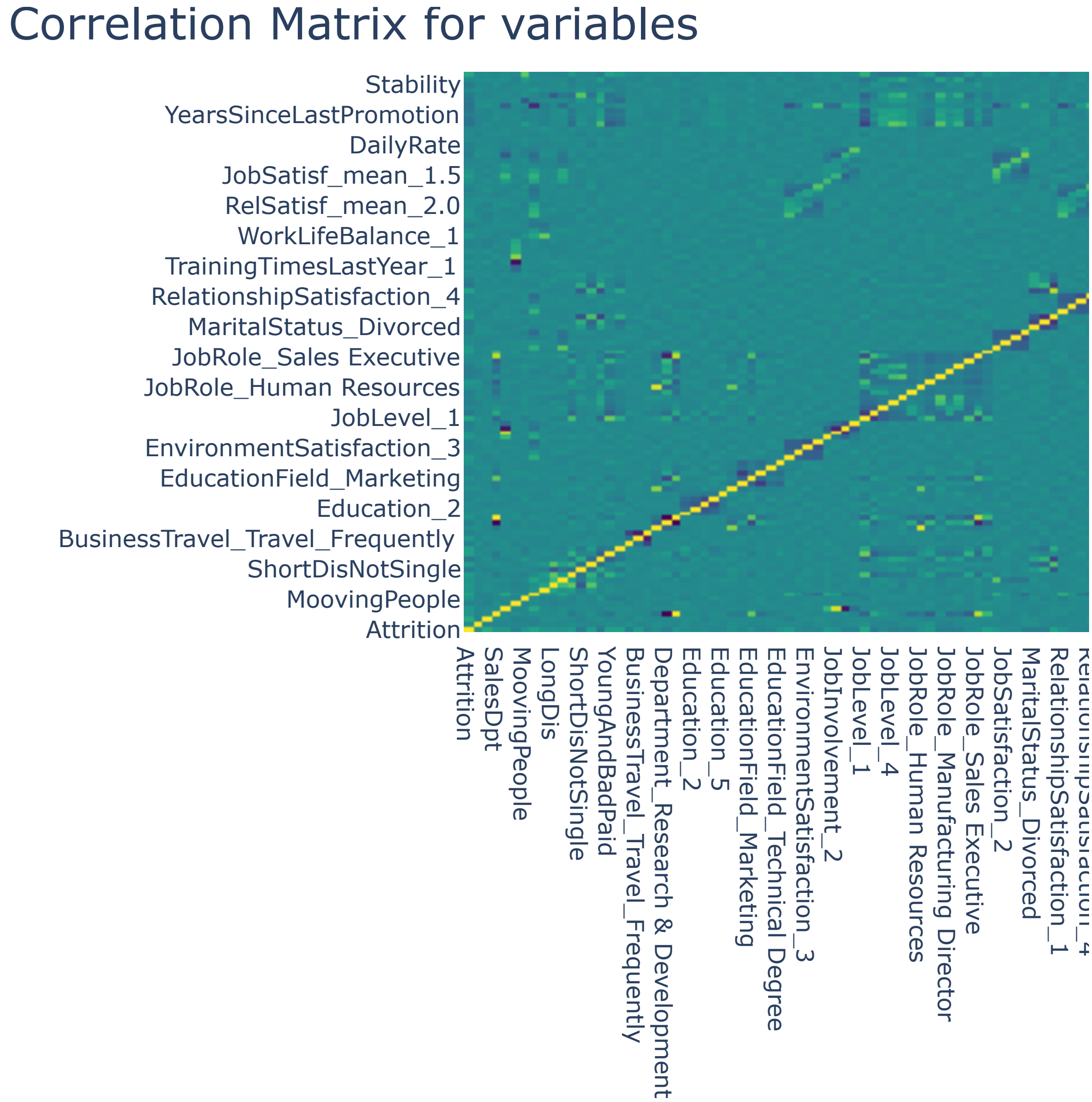
In [17]:

```
#customer id col
Id_col     = ['EmployeeNumber']
#Target columns
target_col = ["Attrition"]
#categorical columns
cat_cols   = data.nunique()[data.nunique() < 10].keys().tolist()
cat_cols   = [x for x in cat_cols if x not in target_col]
#numerical columns
num_cols   = [x for x in data.columns if x not in cat_cols + tar
get_col + Id_col]
#Binary columns with 2 values
bin_cols   = data.nunique()[data.nunique() == 2].keys().tolist()
#Columns more than 2 values
multi_cols = [i for i in cat_cols if i not in bin_cols]

#Label encoding Binary columns
le = LabelEncoder()
for i in bin_cols :
    data[i] = le.fit_transform(data[i])

#Duplicating columns for multi value columns
data = pd.get_dummies(data = data,columns = multi_cols )

#Scaling Numerical columns
std = StandardScaler()
scaled = std.fit_transform(data[num_cols])
scaled = pd.DataFrame(scaled,columns=num_cols)

#dropping original values merging scaled values for numerical co
lumns
df_data_og = data.copy()
data = data.drop(columns = num_cols,axis = 1)
data = data.merge(scaled,left_index=True,right_index=True,how =
"left")
data = data.drop(['EmployeeNumber'],axis = 1)
```

3.4. Correlation Matrix

In [18]:

```
#correlation
correlation = data.corr()
#tick labels
matrix_cols = correlation.columns.tolist()
#convert to array
corr_array  = np.array(correlation)

#Plotting
trace = go.Heatmap(z = corr_array,
                   x = matrix_cols,
                   y = matrix_cols,
                   colorscale='Viridis',
                   colorbar   = dict() ,
                  )
layout = go.Layout(dict(title = 'Correlation Matrix for variable
s',
                        autosize = False,
                        #height  = 1400,
                        #width   = 1600,
                        margin  = dict(r = 0 ,l = 210,
                                       t = 25,b = 210,
                                     ),
                        yaxis   = dict(tickfont = dict(size = 9)
),
                        xaxis   = dict(tickfont = dict(size = 9)
),
                       )
                  )
fig = go.Figure(data = [trace],layout = layout)
py.iplot(fig)
```



3.5. Remove collinear features

In [19]:

```
# Threshold for removing correlated variables
threshold = 0.8

# Absolute value correlation matrix
corr_matrix = data.corr().abs()
corr_matrix.head()

# Upper triangle of correlations
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=
1).astype(np.bool))
upper.head()

# Select columns with correlations above threshold
to_drop = [column for column in upper.columns if any(upper[colum
n] > threshold)]

print('There are %d columns to remove :' % (len(to_drop)))

data = data.drop(columns = to_drop)

to_drop
```

```
There are 7 columns to remove :
```

Out[19]:

```
['Department_Research & Development',
 'Department_Sales',
 'JobInvolvement_2',
 'JobInvolvement_3',
 'JobRole_Human Resources',
 'JobRole_Sales Executive',
 'TrainingTimesLastYear_2']
```

4. Define functions

4.1. Define model performance plot

In [20]:

In [20]:

```
def model_performance_plot(model) :
    #conf matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    trace1 = go.Heatmap(z = conf_matrix  ,x = ["0 (pred)","1 (pr
ed)"],
                        y = ["0 (true)","1 (true)"],xgap = 2, yg
ap = 2,
                        colorscale = 'Viridis', showscale  = Fal
se)

    #show metrics
    tp = conf_matrix[1,1]
    fn = conf_matrix[1,0]
    fp = conf_matrix[0,1]
    tn = conf_matrix[0,0]
    Accuracy  =  ((tp+tn)/(tp+tn+fp+fn))
    Precision =  (tp/(tp+fp))
    Recall    =  (tp/(tp+fn))
    F1_score  =  (2*(((tp/(tp+fp))*(tp/(tp+fn)))/((tp/(tp+fp))+(
tp/(tp+fn)))))

    show_metrics = pd.DataFrame(data=[[Accuracy , Precision, Rec
all, F1_score]])
    show_metrics = show_metrics.T

    colors = ['gold', 'lightgreen', 'lightcoral', 'lightskyblue'
]
    trace2 = go.Bar(x = (show_metrics[0].values),
                   y = ['Accuracy', 'Precision', 'Recall', 'F1_s
core'], text = np.round_(show_metrics[0].values,4),
                    textposition = 'auto',
                   orientation = 'h', opacity = 0.8,marker=dict(
            color=colors,
            line=dict(color='#000000',width=1.5)))

    #plot roc curve
    model_roc_auc = round(roc_auc_score(y_test, y_score) , 3)
    fpr, tpr, t = roc_curve(y_test, y_score)
    trace3 = go.Scatter(x = fpr,y = tpr,
                        name = "Roc : ",
                        line = dict(color = ('rgb(22, 96, 167)')
,width = 2), fill='tozeroy')
    trace4 = go.Scatter(x = [0,1],y = [0,1],
                        line = dict(color = ('black'),width = 1.
```

```
5,
                        dash = 'dot'))

    # Precision-recall curve
    precision, recall, thresholds = precision_recall_curve(y_tes
t, y_score)
    trace5 = go.Scatter(x = recall, y = precision,
                        name = "Precision" + str(precision),
                        line = dict(color = ('lightcoral'),width
= 2), fill='tozeroy')

    #subplots
    fig = tls.make_subplots(rows=2, cols=2, print_grid=False,
                        subplot_titles=('Confusion Matrix',
                                        'Metrics',
                                        'ROC curve'+" "+ '('+ st
r(model_roc_auc)+')',
                                        'Precision - Recall curv
e'))

    fig.append_trace(trace1,1,1)
    fig.append_trace(trace2,1,2)
    fig.append_trace(trace3,2,1)
    fig.append_trace(trace4,2,1)
    fig.append_trace(trace5,2,2)

    fig['layout'].update(showlegend = False, title = '<b>Model p
erformance</b><br>'+str(model),
                        autosize = False, height = 900,width = 8
30,
                        plot_bgcolor = 'rgba(240,240,240, 0.95)'
,
                        paper_bgcolor = 'rgba(240,240,240, 0.95)
',
                        margin = dict(b = 195))
    fig["layout"]["xaxis2"].update((dict(range=[0, 1])))
    fig["layout"]["xaxis3"].update(dict(title = "false positive
rate"))
    fig["layout"]["yaxis3"].update(dict(title = "true positive r
ate"))
    fig["layout"]["xaxis4"].update(dict(title = "recall"), range
= [0,1.05])
    fig["layout"]["yaxis4"].update(dict(title = "precision"), ra
nge = [0,1.05])
```

```
    fig.layout.titlefont.size = 14

    py.iplot(fig)
```

4.2. Define feature importance plot

In [21]:

```
def features_imp(model, cf) :

    coefficients  = pd.DataFrame(model.feature_importances_)
    column_data     = pd.DataFrame(list(data))
    coef_sumry    = (pd.merge(coefficients,column_data,left_inde
x= True,
                              right_index= True, how = "left"))
    coef_sumry.columns = ["coefficients","features"]
    coef_sumry    = coef_sumry.sort_values(by = "coefficients",a
scending = False)
    coef_sumry = coef_sumry[coef_sumry["coefficients"] !=0]
    trace = go.Bar(x = coef_sumry["features"],y = coef_sumry["co
efficients"],
                    name = "coefficients",
                    marker = dict(color = coef_sumry["coefficien
ts"],
                                  colorscale = "Viridis",
                                  line = dict(width = .6,color =
"black")))
    layout = dict(title =  'Feature Importances xgb_cfl')

    fig = dict(data = [trace], layout=layout)
    py.iplot(fig)
```

4.3. Define cumulative gains curve

In [22]:

```
#cumulative gain curve
def cum_gains_curve(model):
    pos = pd.get_dummies(y_test).as_matrix()
    pos = pos[:,1]
    npos = np.sum(pos)
    index = np.argsort(y_score)
    index = index[::-1]
    sort_pos = pos[index]
    #cumulative sum
    cpos = np.cumsum(sort_pos)
    #recall
    recall = cpos/npos
    #size obs test
    n = y_test.shape[0]
    size = np.arange(start=1,stop=369,step=1)
    #proportion
    size = size / n
    #plots
    model = 'xgb_cfl'
    trace1 = go.Scatter(x = size,y = recall,
                        name = "Lift curve",
                        line = dict(color = ('rgb(22, 96, 167)')
,width = 2))
    trace2 = go.Scatter(x = size,y = size,
                        name = "Baseline",
                        showlegend=False,
                        line = dict(color = ('black'),width = 1.
5,
                        dash = 'dot'))

    layout = dict(title = 'Cumulative gains curve'+' '+str(model
),
                  yaxis = dict(title = 'Percentage positive targ
eted',zeroline = False),
                  xaxis = dict(title = 'Percentage contacted', z
eroline = False)
                 )

    fig  = go.Figure(data = [trace1,trace2], layout = layout)
    py.iplot(fig)
```

4.4. Define cross validation metrics

In [23]:

```
# Cross val metric
def cross_val_metrics(model) :
    scores = ['accuracy', 'precision', 'recall']
    for sc in scores:
        scores = cross_val_score(model, X, y, cv = 5, scoring =
sc)
        print('[%s] : %0.5f (+/- %0.5f)'%(sc, scores.mean(), sco
res.std()))
```

5. Prepare dataset

5.1. Define (X, y)

In [24]:

```
# Def X and Y
y = np.array(data.Attrition.tolist())
data = data.drop('Attrition', 1)
X = np.array(data.as_matrix())
```

5.2. Train test split

In [25]:

```
# Train_test split
random_state = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_s
ize = 0.25, random_state = random_state)
```

6. XGBoost - RandomizedSearchCV to optimize hyperparameters

In [26]:

```
def timer(start_time=None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        thour, temp_sec = divmod((datetime.now() - start_time).t
otal_seconds(), 3600)
        tmin, tsec = divmod(temp_sec, 60)
        print('\n Time taken: %i hours %i minutes and %s
seconds.' % (thour, tmin, round(tsec, 2)))


xgb_cfl = xgb.XGBClassifier(n_jobs = -1)


# A parameter grid for XGBoost
params = {
        'n_estimators' : [100, 200, 500, 750],
        'learning_rate' : [0.01, 0.02, 0.05, 0.1, 0.25],
        'min_child_weight': [1, 5, 7, 10],
        'gamma': [0.1, 0.5, 1, 1.5, 5],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5, 10, 12]
        }

folds = 5
param_comb = 800

random_search = RandomizedSearchCV(xgb_cfl, param_distributions=
params, n_iter=param_comb, scoring='accuracy', n_jobs=-1, cv=5,
verbose=3, random_state=42)

# Here we go
start_time = timer(None) # timing starts from this point for "st
art_time" variable
#----------------------------# random_search.fit(X, y)
timer(start_time) # timing ends here for "start_time" variable
```

```
 Time taken: 0 hours 0 minutes and 0.0 seconds.
```

Remove "#--------------------------#" to lunch random_search

In [27]:

```
#print('\n All results:')
#print(random_search.cv_results_)
#print('\n Best estimator:')
#print(random_search.best_estimator_)
#print('\n Best accuracy for %d-fold search with %d parameter co
mbinations:' % (folds, param_comb))
#print(random_search.best_score_ )
#print('\n Best hyperparameters:')
#print(random_search.best_params_)
#results = pd.DataFrame(random_search.cv_results_)
#results.to_csv('xgb-random-grid-search-results-01.csv', index=F
alse)
```

RESULT :

Best estimator: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bytree=0.8, gamma=1.5, learning_rate=0.05, max_delta_step=0, max_depth=3, min_child_weight=7, missing=None, n_estimators=200, n_jobs=-1, nthread=None, objective='binary:logistic', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=True, subsample=0.6)

Best accuracy for 5-fold search with 800 parameter combinations:
0.891156462585034

Best hyperparameters: {'subsample': 0.6, 'n_estimators': 200, 'min_child_weight': 7, 'max_depth': 3, 'learning_rate': 0.05, 'gamma': 1.5, 'colsample_bytree': 0.8}

7. XGBoost - Modeling with best hyperparameters = 89.11

7.1. XGBoost - Modeling and performance plot

In [ ]:

```
# xgb
xgb_clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree', co
lsample_bylevel=1,
                            colsample_bytree=0.8, gamma=1.5, lear
ning_rate=0.05,
                            max_delta_step=0, max_depth=3, min_ch
ild_weight=7, missing=None,
                            n_estimators=200, n_jobs=-1, nthread=
None,
                            objective='binary:logistic', random_s
tate=0, reg_alpha=0,
                            reg_lambda=1, scale_pos_weight=1, see
d=None, silent=True,
                            subsample=0.6)

xgb_clf.fit(X_train, y_train)
y_pred = xgb_clf.predict(X_test)
y_score = xgb_clf.predict_proba(X_test)[:,1]

model_performance_plot('xgb_clf')
```

7.2. XGBoost - Feature importance

In [ ]:

```
features_imp(xgb_clf, 'features')
```

In [ ]:

```
#feature importance plot TOP 40
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
def plot_feature_importance(model):
    tmp = pd.DataFrame({'Feature': list(data), 'Feature importan
ce': model.feature_importances_})
    tmp = tmp.sort_values(by='Feature importance',ascending=Fals
e).head(30)
    plt.figure(figsize = (10,12))
    plt.title('Top 30 - Features importance - XGBoost',fontsize=
14)
    s = sns.barplot(y='Feature',x='Feature importance',data=tmp,
orient='h')
    s.set_xticklabels(s.get_xticklabels(),rotation=90)
    plt.show()
```

In [ ]:

```
plot_feature_importance(xgb_clf)
```

7.3. XGBoost - Cumulative gain curve

In [ ]:

```
cum_gains_curve(xgb_clf)
```

7.4. XGBoost - Cross validation (5 folds)

In [ ]:

```
# Cross val score
cross_val_metrics(xgb_clf)
```