Attrition Data - Analysis and Prediction

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, RandomizedSearchCV, cross_val_score, learning_curve, train_test_split
from sklearn.metrics import precision_score, roc_auc_score, recall_score, confusion_matrix, roc_curve
from sklearn.metrics import precision_recall_curve, accuracy_score, mean_squared_error, r2_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
import matplotlib.pyplot as plt
import seaborn as sns
warnings.filterwarnings('ignore')
```

1.2. Read the data

In [2]:

```
data = pd.read_csv('D:/unified mentor/um inside/Attrition data.csv')
```

1.3. Missing values

In [3]:

```
print('\n')
print(f'The sum of all null values in the Employee Attrition data is {data.isnull().sum().sum()}')
```

```
The sum of all null values in the Employee Attrition data is 111
```

1.4. Reassign target and drop useless features

In [4]:

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

2. Exploratory Data Analysis (EDA)

2.1. Head and describe

In [5]:

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

Out[5]:

	EmployeeID	Age	Attrition	BusinessTravel	Department	DistanceFromHome	Education	EducationField	Gender	JobLevel	...	TotalWorkingYears	TrainingTimesLastYear	YearsAtCompany
0	1	51	0	Travel_Rarely	Sales	6	2	Life Sciences	Female	1	...	1.0	6	1
1	2	31	1	Travel_Frequently	Research & Development	10	1	Life Sciences	Female	1	...	6.0	3	5
2	3	32	0	Travel_Frequently	Research & Development	17	4	Other	Male	4	...	5.0	2	5
3	4	38	0	Non-Travel	Research & Development	2	5	Life Sciences	Male	3	...	13.0	5	8
4	5	32	0	Travel_Rarely	Research & Development	10	1	Medical	Male	1	...	9.0	2	6

5 rows × 26 columns

In [6]:

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

Out[6]:

	EmployeeID	Age	Attrition	DistanceFromHome	Education	JobLevel	MonthlyIncome	NumCompaniesWorked	PercentSalaryHike	StockOptionLevel	TotalWorkingYears
count	4410.000000	4410.000000	4410.000000	4410.000000	4410.000000	4410.000000	4410.000000	4391.000000	4410.000000	4410.000000	4401.000000
mean	2205.500000	36.923810	0.161224	9.192517	2.912925	2.063946	65029.312925	2.694830	15.209524	0.793878	11.279936
std	1273.201673	9.133301	0.367780	8.105026	1.023933	1.106689	47068.888559	2.498887	3.659108	0.851883	7.782222
min	1.000000	18.000000	0.000000	1.000000	1.000000	1.000000	10090.000000	0.000000	11.000000	0.000000	0.000000
25%	1103.250000	30.000000	0.000000	2.000000	2.000000	1.000000	29110.000000	1.000000	12.000000	0.000000	6.000000
50%	2205.500000	36.000000	0.000000	7.000000	3.000000	2.000000	49190.000000	2.000000	14.000000	1.000000	10.000000
75%	3307.750000	43.000000	0.000000	14.000000	4.000000	3.000000	83800.000000	4.000000	18.000000	1.000000	15.000000
max	4410.000000	60.000000	1.000000	29.000000	5.000000	5.000000	199990.000000	9.000000	25.000000	3.000000	40.000000

In [7]:

```
data.shape
```

Out[7]:

```
(4410, 26)
```

In [8]:

```
data.info()
# There are six categoroical attributes and twenty numerical attributes.
```

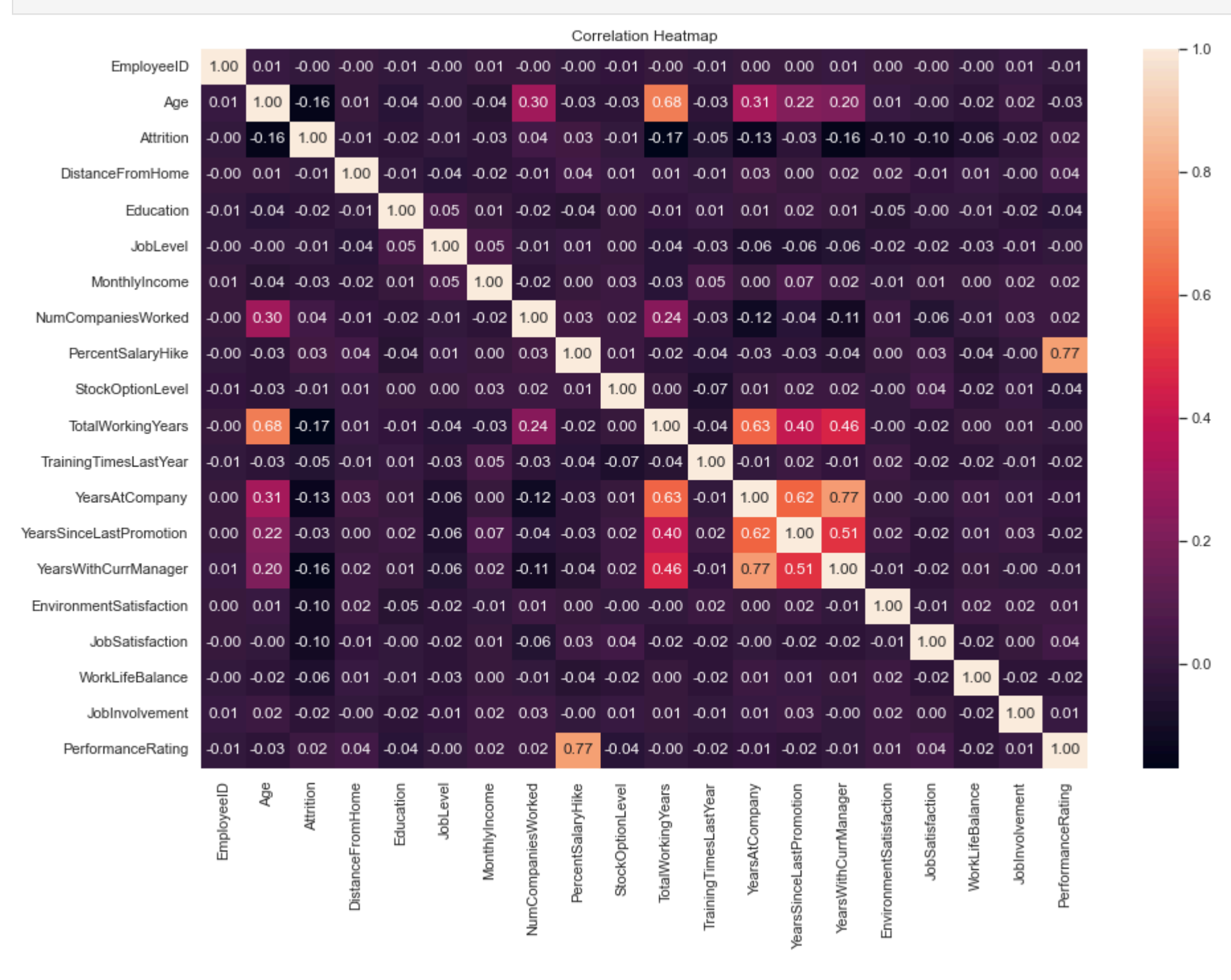
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4410 entries, 0 to 4409
Data columns (total 26 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   EmployeeID               4410 non-null   int64
 1   Age                      4410 non-null   int64
 2   Attrition                4410 non-null   int64
 3   BusinessTravel           4410 non-null   object
 4   Department               4410 non-null   object
 5   DistanceFromHome         4410 non-null   int64
 6   Education                4410 non-null   int64
 7   EducationField           4410 non-null   object
 8   Gender                   4410 non-null   object
 9   JobLevel                 4410 non-null   int64
 10  JobRole                  4410 non-null   object
 11  MaritalStatus            4410 non-null   object
 12  MonthlyIncome            4410 non-null   int64
 13  NumCompaniesWorked       4391 non-null   float64
 14  PercentSalaryHike        4410 non-null   int64
 15  StockOptionLevel         4410 non-null   int64
 16  TotalWorkingYears        4401 non-null   float64
 17  TrainingTimesLastYear    4410 non-null   int64
 18  YearsAtCompany           4410 non-null   int64
 19  YearsSinceLastPromotion  4410 non-null   int64
 20  YearsWithCurrManager     4410 non-null   int64
 21  EnvironmentSatisfaction  4385 non-null   float64
 22  JobSatisfaction          4390 non-null   float64
 23  WorkLifeBalance          4372 non-null   float64
 24  JobInvolvement           4410 non-null   int64
 25  PerformanceRating        4410 non-null   int64
dtypes: float64(5), int64(15), object(6)
memory usage: 895.9+ KB
```

2.2. Correlation Heatmap

In [9]:

```
plt.figure(figsize=(15,10))
sns.set(font_scale=1)
sns.heatmap(data.corr(), fmt='.2f', annot=True)
plt.title('Correlation Heatmap')
plt.show()

# The variables like
# (1) 'TotalWorkingYears', 'Age' (correlation=0.68)
# (2) 'PercentSalaryHike', 'PerformanceRating' (correlation=0.77)
# (3) 'YearsAtCompany', 'YearsWithCurrManager' (correlation=0.77)
# have moderate to high correlation with each other.
# These variables should be removed from the dataset by analysing the Variation Inflation Factor
# of the variables or they might affect the accuracy and prediction of the ML model.
```



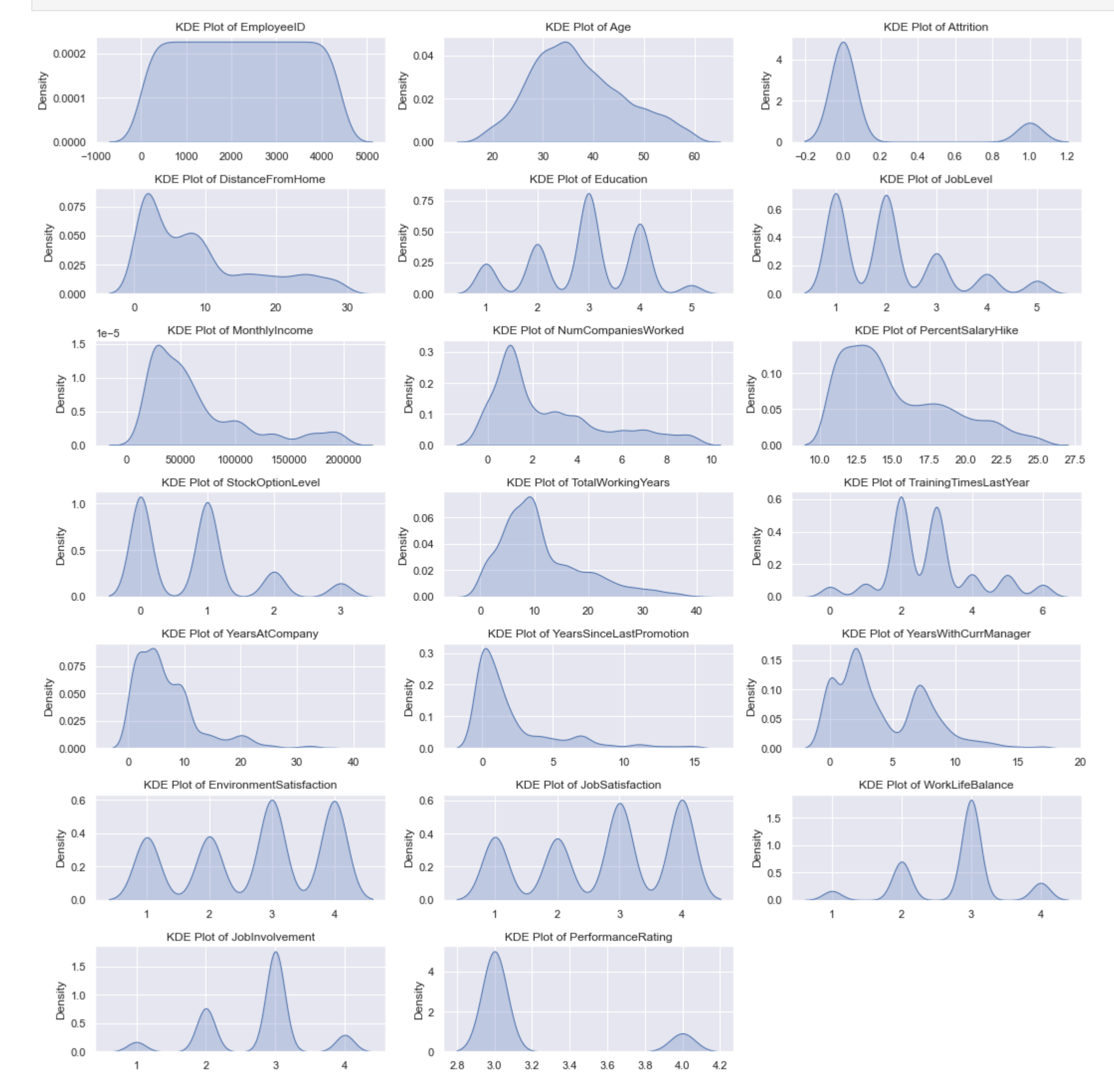
2.3. Kernel Density Plot

In [10]:

```
# Filter numeric columns
numeric_columns = data.select_dtypes(include=['int64', 'float64']).columns

plt.figure(figsize=(15,15))
for i, column in enumerate(numeric_columns):
    plt.subplot(int(len(numeric_columns)/3)+1, 3, i+1)
    sns.kdeplot(data[column], shade=True)
    plt.xlabel('')
    plt.title(f'KDE Plot of {column}')
    plt.tight_layout()

# KDE Plot of 'YearswithcurrManager' follows a bimodal distribution.
# KDE Plot of 'Age', 'DistanceFromHome', 'MonthlyIncome', 'NumComapniesWorked', 'PercentSaalaryHike',
# 'TotalWorkingYears', 'YearsAtCompany', 'YearsSinceLastPromotion' have a left skewed distribution.
```



3. Feature engineering and selection

3.1 Impute the null values

In [11]:

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

```
Missing values :   111
```

In [12]:

```
data.isnull().sum()
```

Out[12]:

```
EmployeeID                  0
Age                         0
Attrition                   0
BusinessTravel              0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
Gender                      0
JobLevel                    0
JobRole                     0
MaritalStatus               0
MonthlyIncome               0
NumCompaniesWorked         19
PercentSalaryHike           0
StockOptionLevel            0
TotalWorkingYears           9
TrainingTimesLastYear       0
YearsAtCompany              0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
EnvironmentSatisfaction    25
JobSatisfaction            20
WorkLifeBalance            38
JobInvolvement              0
PerformanceRating           0
dtype: int64
```

In [13]:

```
null_feat = ['NumCompaniesWorked', 'EnvironmentSatisfaction', 'JobSatisfaction', 'WorkLifeBalance', 'TotalWorkingYears']
```

In [14]:

```
data[null_feat] = data[null_feat].fillna(data[null_feat].median())
```

In [15]:

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

```
Missing values :   0
```

3.2. Features encoding and scaling

In [16]:

```
#customer id col
Id_col     = ['EmployeeID']
#Target columns
target_col = ["Attrition"]
```

In [17]:

```
#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]
```

In [18]:

```
#numerical columns
num_cols   = [x for x in data.columns if x not in cat_cols + target_col + Id_col]
```

In [19]:

```
#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]
```

In [20]:

```
# 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 )
```

In [21]:

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

In [22]:

```
#dropping original values merging scaled values for numerical columns
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(['EmployeeID'],axis = 1)
```

3.3. Remove collinear features

In [23]:

```
# 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[column] > threshold)]

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

data = data.drop(columns = to_drop)

to_drop
```

```
There are 1 columns to remove :
```

Out[23]:

```
['Department_Sales']
```

4. Define functions

4.1. Define model performance plot

In [24]:

```
def model_performance_plot(model) :
    #conf matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    annot_text = [[str(conf_matrix[j, i]) for i in range(len(["0 (pred)", "1 (pred)"]))]
                  for j in range(len(["0 (true)", "1 (true)"]))]
    trace1 = go.Heatmap(z = conf_matrix  ,x = ["0 (pred)","1 (pred)"],
                        y = ["0 (true)","1 (true)"],xgap = 2, ygap = 2,
                        colorscale = 'Viridis', showscale  = True,
                        text=annot_text, hoverinfo='text')

    #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, Recall, 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_score'], 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_test, 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'+" "+ '('+ str(model_roc_auc)+')',
                                          'Precision - Recall curve'))

    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 performance</b><br>'+str(model),
                        autosize = False, height = 900,width = 830,
                        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 rate"))
    fig["layout"]["xaxis4"].update(dict(title = "recall"), range = [0,1.05])
    fig["layout"]["yaxis4"].update(dict(title = "precision"), range = [0,1.05])
    fig.layout.titlefont.size = 14

    py.iplot(fig)
```

4.2. Define feature importance plot

In [25]:

```
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_index= True,
                              right_index= True, how = "left"))
    coef_sumry.columns = ["coefficients","features"]
    coef_sumry    = coef_sumry.sort_values(by = "coefficients",ascending = False)
    coef_sumry = coef_sumry[coef_sumry["coefficients"] !=0]
    trace = go.Bar(x = coef_sumry["features"],y = coef_sumry["coefficients"],
                    name = "coefficients",
                    marker = dict(color = coef_sumry["coefficients"],
                                  colorscale = "Viridis",
                                  line = dict(width = .6,color = "black")))
    layout = dict(title =  'Feature Importances xgb_cfl')

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

5. Prepare dataset

5.1. Define (X, y)

In [26]:

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

5.2. Train test split

In [27]:

```
# Train_test split
random_state = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = random_state)
```

6. XGBoost - Modeling with hyperparameters = 91.84

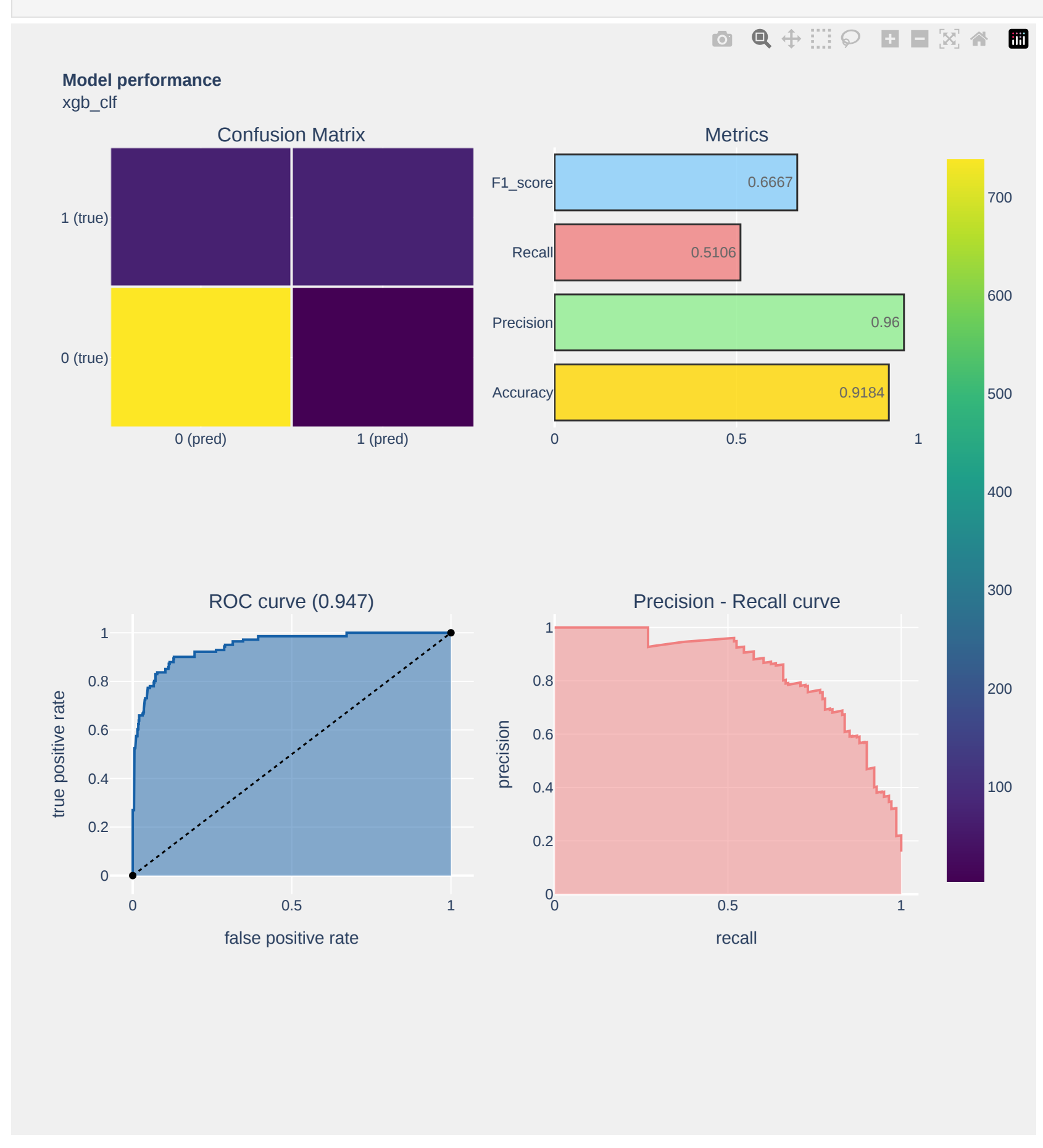
6.1. XGBoost - Modeling and performance plot

In [29]:

```
# xgb
xgb_clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                            colsample_bytree=0.8, gamma=1.5, learning_rate=0.01,
                            max_delta_step=0, max_depth=7, min_child_weight=7, missing=float(),
                            n_estimators=800, n_jobs=-1, nthread=None,
                            objective='binary:logistic', random_state=0, reg_alpha=0,
                            reg_lambda=1, scale_pos_weight=1, seed=None,
                            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 [30]:

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