Employee Performance Analysis

IABAC™ Project Submission

Following insights are expected from this project:

- Department wise performances.
- Top 3 Important Factors effecting employee performance.
- A trained model which can predict the employee performance based on factors as inputs.
- Recommendations to improve the employee performance based on insights from analysis.

```
#import important libraries
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import seaborn as sns
```

```
path =r"/content/drive/MyDrive/INX_Future_Inc_Employee_Performance_CDS_Project6.x
df=pd.read_excel(path) # read the data
```

1. Domain Analysis :

```
df.head()
```

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment
0	E1001000	32	Male	Marketing	Single	Sales
1	E1001006	47	Male	Marketing	Single	Sales
2	E1001007	40	Male	Life Sciences	Married	Sales
3	E1001009	41	Male	Human Resources	Divorced	Human Resources

```
#size/shaepof this dataset
df.shape
#1200row and 28 columns
```

```
(1200, 28)
```

```
df.columns
```

```
Index(['EmpNumber', 'Age', 'Gender', 'EducationBackground', 'MaritalStatus',
       'EmpDepartment', 'EmpJobRole', 'BusinessTravelFrequency',
       'DistanceFromHome', 'EmpEducationLevel',
'EmpEnvironmentSatisfaction',
       'EmpHourlyRate', 'EmpJobInvolvement', 'EmpJobLevel',
       'EmpJobSatisfaction', 'NumCompaniesWorked', 'OverTime',
       'EmpLastSalaryHikePercent', 'EmpRelationshipSatisfaction',
       'TotalWorkExperienceInYears', 'TrainingTimesLastYear',
       'EmpWorkLifeBalance', 'ExperienceYearsAtThisCompany',
       'ExperienceYearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager', 'Attrition', 'PerformanceRating'],
      dtype='object')
```

Double-click (or enter) to edit

```
df.info() #no missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 28 columns):
 #   Column                        Non-Null Count  Dtype 
---  ------                        --------------  ----- 
 0   EmpNumber                     1200 non-null   object
 1   Age                           1200 non-null   int64 
 2   Gender                        1200 non-null   object
 3   EducationBackground           1200 non-null   object
 4   MaritalStatus                 1200 non-null   object
 5   EmpDepartment                 1200 non-null   object
 6   EmpJobRole                    1200 non-null   object
 7   BusinessTravelFrequency       1200 non-null   object
 8   DistanceFromHome              1200 non-null   int64 
 9   EmpEducationLevel             1200 non-null   int64 
 10  EmpEnvironmentSatisfaction    1200 non-null   int64 
 11  EmpHourlyRate                 1200 non-null   int64 
 12  EmpJobInvolvement             1200 non-null   int64 
 13  EmpJobLevel                   1200 non-null   int64 
 14  EmpJobSatisfaction            1200 non-null   int64 
 15  NumCompaniesWorked            1200 non-null   int64 
 16  OverTime                      1200 non-null   object
 17  EmpLastSalaryHikePercent      1200 non-null   int64 
 18  EmpRelationshipSatisfaction   1200 non-null   int64 
 19  TotalWorkExperienceInYears    1200 non-null   int64 
```

```
 20  TrainingTimesLastYear         1200 non-null   int64
 21  EmpWorkLifeBalance            1200 non-null   int64
 22  ExperienceYearsAtThisCompany  1200 non-null   int64
 23  ExperienceYearsInCurrentRole  1200 non-null   int64
 24  YearsSinceLastPromotion       1200 non-null   int64
 25  YearsWithCurrManager          1200 non-null   int64
 26  Attrition                     1200 non-null   object
 27  PerformanceRating             1200 non-null   int64
dtypes: int64(19), object(9)
memory usage: 262.6+ KB
```

```
# stastics about numeric columns :
df.describe().T
```

	count	mean	std	min	25%	50%	7
Age	1200.0	36.918333	9.087289	18.0	30.0	36.0	43
DistanceFromHome	1200.0	9.165833	8.176636	1.0	2.0	7.0	14
EmpEducationLevel	1200.0	2.892500	1.044120	1.0	2.0	3.0	4
EmpEnvironmentSatisfaction	1200.0	2.715833	1.090599	1.0	2.0	3.0	4
EmpHourlyRate	1200.0	65.981667	20.211302	30.0	48.0	66.0	83
EmpJobInvolvement	1200.0	2.731667	0.707164	1.0	2.0	3.0	3
EmpJobLevel	1200.0	2.067500	1.107836	1.0	1.0	2.0	3
EmpJobSatisfaction	1200.0	2.732500	1.100888	1.0	2.0	3.0	4
NumCompaniesWorked	1200.0	2.665000	2.469384	0.0	1.0	2.0	4
EmpLastSalaryHikePercent	1200.0	15.222500	3.625918	11.0	12.0	14.0	18
EmpRelationshipSatisfaction	1200.0	2.725000	1.075642	1.0	2.0	3.0	4
TotalWorkExperienceInYears	1200.0	11.330000	7.797228	0.0	6.0	10.0	15
TrainingTimesLastYear	1200.0	2.785833	1.263446	0.0	2.0	3.0	3
EmpWorkLifeBalance	1200.0	2.744167	0.699374	1.0	2.0	3.0	3
ExperienceYearsAtThisCompany	1200.0	7.077500	6.236899	0.0	3.0	5.0	10
ExperienceYearsInCurrentRole	1200.0	4.291667	3.613744	0.0	2.0	3.0	7
YearsSinceLastPromotion	1200.0	2.194167	3.221560	0.0	0.0	1.0	3
YearsWithCurrManager	1200.0	4.105000	3.541576	0.0	2.0	3.0	7
PerformanceRating	1200.0	2.948333	0.518866	2.0	3.0	3.0	3

```
df.describe(include='O') #CATEGORICAL FEATURES STATISTICS
# 9 categoricalfeatures
```

	EmpNumber	Gender	EducationBackground	MaritalStatus	EmpDepartment
count	1200	1200	1200	1200	1200
unique	1200	2	6	3	6
top	E100998	Male	Life Sciences	Married	Sales

2.DATA PREPROCESSING

```
#CHECK MISSIING VALUES
```

```
df.isnull().sum() #NO MISSING VALUES
```

```
EmpNumber                       0
Age                             0
Gender                          0
EducationBackground             0
MaritalStatus                   0
EmpDepartment                   0
EmpJobRole                      0
BusinessTravelFrequency         0
DistanceFromHome                0
EmpEducationLevel               0
EmpEnvironmentSatisfaction      0
EmpHourlyRate                   0
EmpJobInvolvement               0
EmpJobLevel                     0
EmpJobSatisfaction              0
NumCompaniesWorked              0
OverTime                        0
EmpLastSalaryHikePercent        0
EmpRelationshipSatisfaction     0
TotalWorkExperienceInYears      0
TrainingTimesLastYear           0
EmpWorkLifeBalance              0
ExperienceYearsAtThisCompany    0
ExperienceYearsInCurrentRole    0
YearsSinceLastPromotion         0
YearsWithCurrManager            0
Attrition                       0
PerformanceRating               0
dtype: int64
```

```
df.isna().values.any()
```

```
False
```

There is no NaN or Null values present in the Data Set

```
# check duplicates
df.duplicated().any()
```

```
False
```

3.EDA :

- DATA ANALYSIS WITH VISUALIZATION

DEPARTEMENT WISE PERFOMANCEANLYSIS

```
dept = df.iloc[:,[5,27]].copy()
dept_per = dept.copy()
```

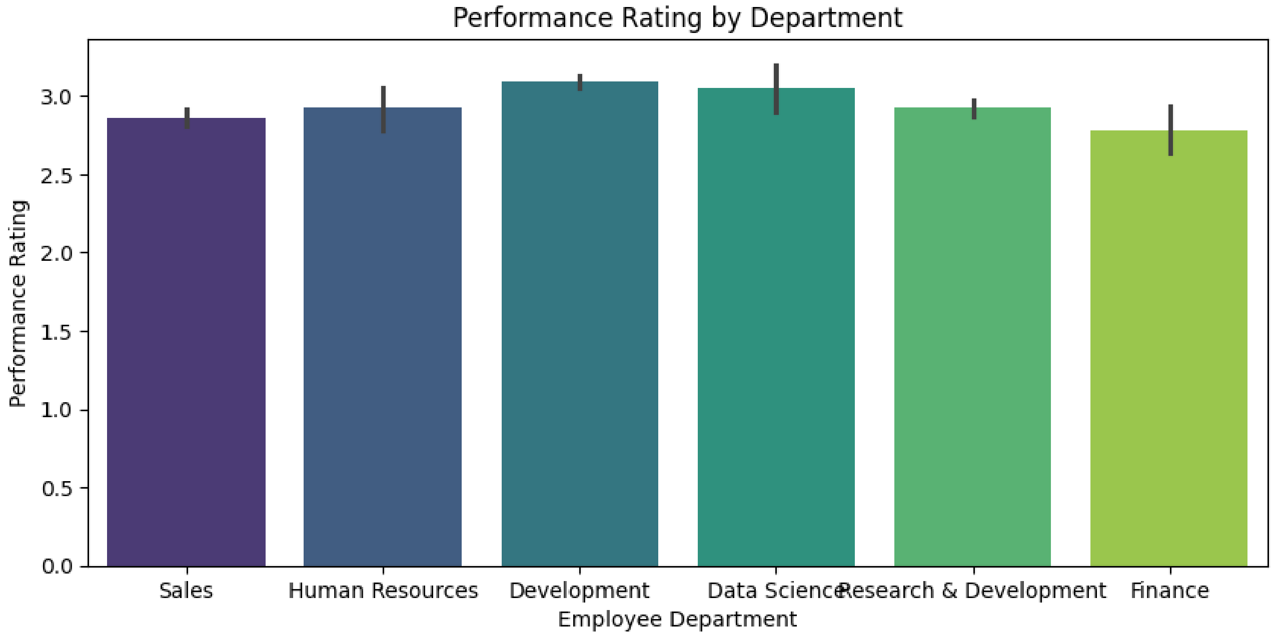
```
dept_per.groupby(by='EmpDepartment')['PerformanceRating'].mean()
```

```
EmpDepartment
Data Science              3.050000
Development               3.085873
Finance                   2.775510
Human Resources           2.925926
Research & Development    2.921283
Sales                     2.860590
Name: PerformanceRating, dtype: float64
```

```
''' using this analysis datascience,developement,HR departemnt gives more perfoma
```

```
' using this analysis datascience,developement,HR departemnt gives more perf
omance '
```

```
#
import matplotlib.pyplot as plt
plt.figure(figsize=(10,4.5))
sns.barplot(data=dept_per, x='EmpDepartment', y='PerformanceRating',palette='viri
plt.xlabel('Employee Department')
plt.ylabel('Performance Rating')
plt.title('Performance Rating by Department')
plt.show()
```




```
# Analyze each department separately
dept_per.groupby(by='EmpDepartment')['PerformanceRating'].value_counts()
```

```
EmpDepartment           PerformanceRating
Data Science            3                     17
                        4                      2
                        2                      1
Development             3                    304
                        4                     44
                        2                     13
Finance                 3                     30
                        2                     15
                        4                      4
Human Resources         3                     38
                        2                     10
                        4                      6
Research & Development  3                    234
                        2                     68
                        4                     41
Sales                   3                    251
                        2                     87
                        4                     35
Name: PerformanceRating, dtype: int64
```

```
# Creating a new dataframe to analyze each department separatel
department = pd.get_dummies(dept_per['EmpDepartment'])
```

```
performance = pd.DataFrame(dept_per['PerformanceRating'])
```

```
department.head()
```

	Data Science	Development	Finance	Human Resources	Research & Development	Sales
0	0	0	0	0	0	1
1	0	0	0	0	0	1
2	0	0	0	0	0	1
3	0	0	0	1	0	0
4	0	0	0	0	0	1

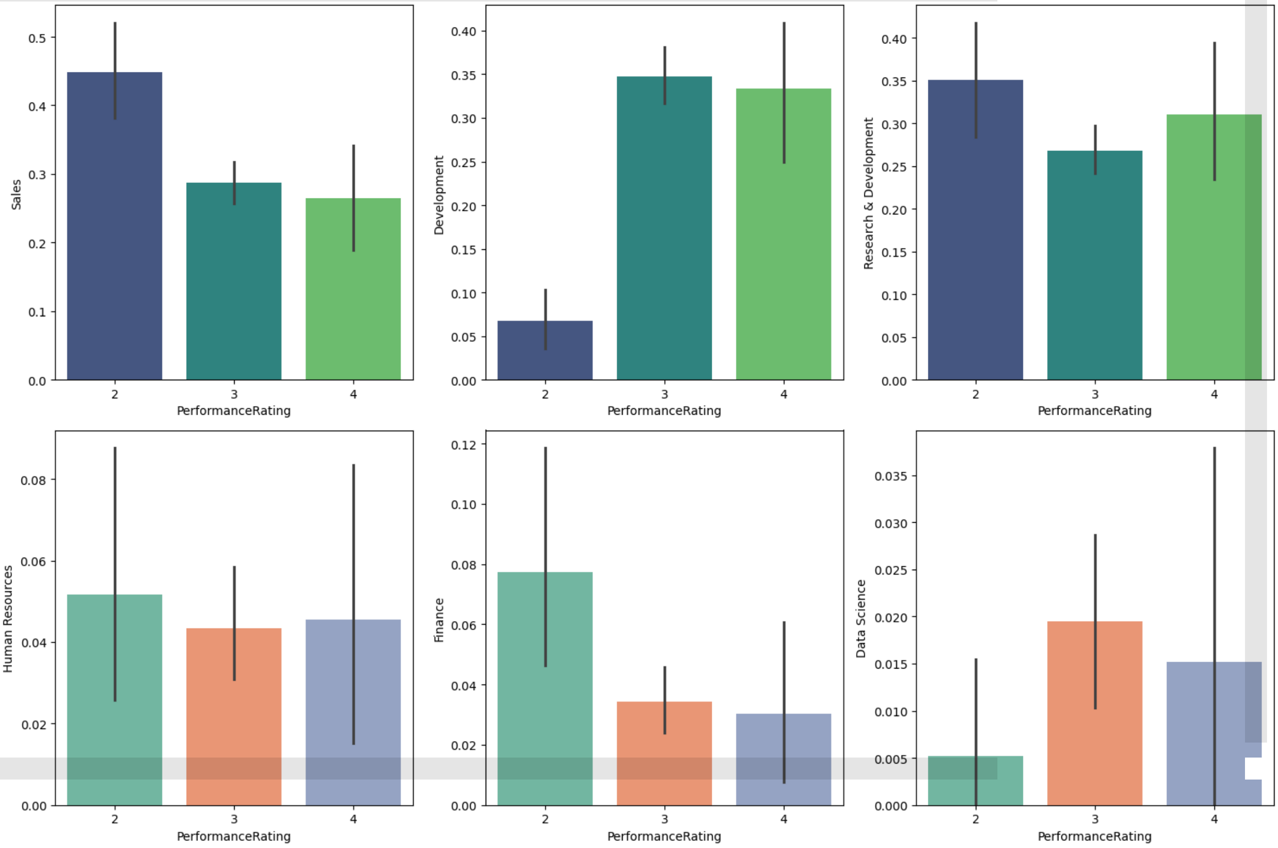
```
dept_rating.head()
```

	Data Science	Development	Finance	Human Resources	Research & Development	Sales	PerformanceRa
0	0	0	0	0	0	1	
1	0	0	0	0	0	1	
2	0	0	0	0	0	1	
3	0	0	0	1	0	0	

Next steps: Generate code with `dept_rating` View recommended plots

```
# Plotting a separate bar graph for performance of each department using seaborn
plt.figure(figsize=(15, 10))

plt.subplot(2, 3, 1)
sns.barplot(x='PerformanceRating', y='Sales', data=dept_rating,palette='viridis'

plt.subplot(2, 3, 2)
sns.barplot(x='PerformanceRating', y='Development', data=dept_rating,palette='vi

plt.subplot(2, 3, 3)
sns.barplot(x='PerformanceRating', y='Research & Development', data=dept_rating,

plt.subplot(2, 3, 4)
sns.barplot(x='PerformanceRating', y='Human Resources', data=dept_rating,palette

plt.subplot(2, 3, 5)
sns.barplot(x='PerformanceRating', y='Finance', data=dept_rating,palette='Set2')
```

```
plt.subplot(2, 3, 6)
sns.barplot(x='PerformanceRating', y='Data Science', data=dept_rating,palette='S

plt.tight_layout()
plt.show()
```



factors affecting the employee perfomance:

```
corr_matrix = df.corr()
```

```
correlation_with_target=corr_matrix['PerformanceRating'].abs().sort_values(ascend
```

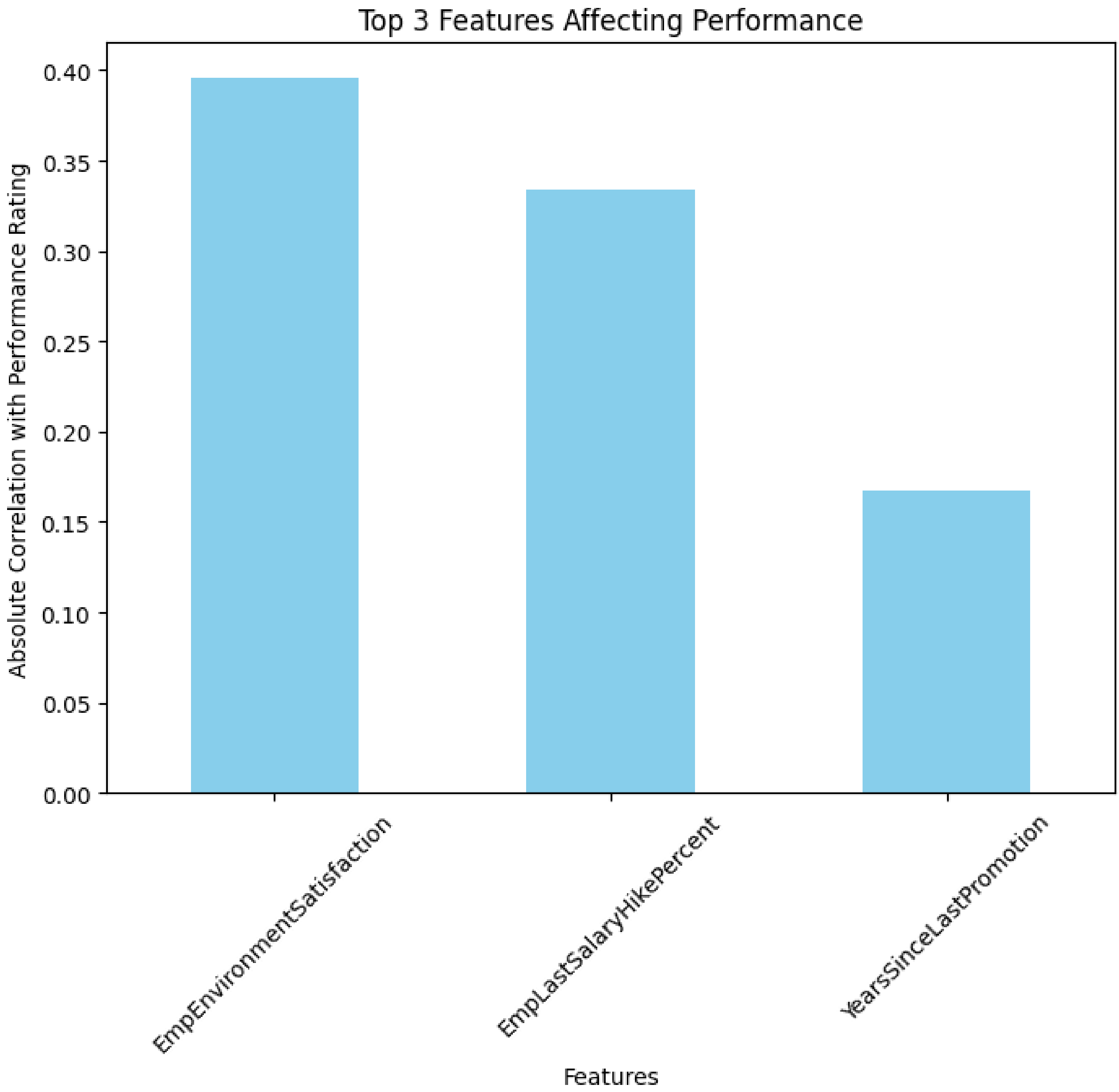
```
correlation_with_target=correlation_with_target.drop('PerformanceRating')
```

```
top_3_features = correlation_with_target.head(3)
```

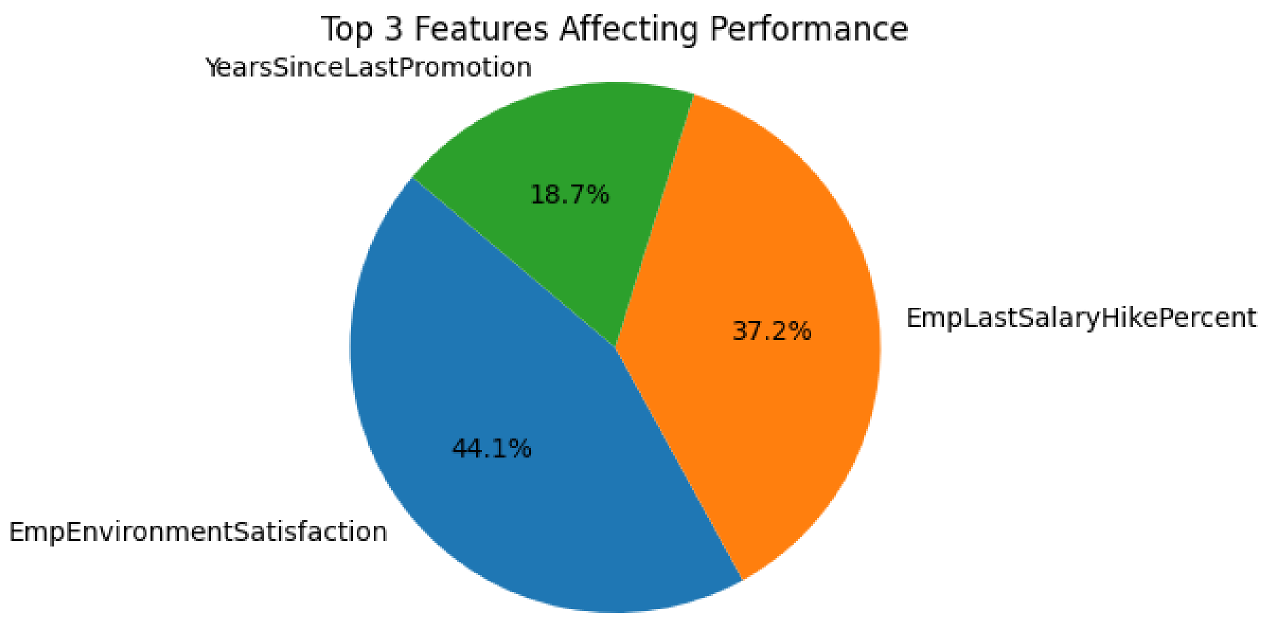
```
print(top_3_features)
```

```
EmpEnvironmentSatisfaction    0.395561
EmpLastSalaryHikePercent      0.333722
YearsSinceLastPromotion       0.167629
Name: PerformanceRating, dtype: float64
```

```
import matplotlib.pyplot as plt

# Plotting the top 3 features affecting performance
plt.figure(figsize=(8, 6))
top_3_features.plot(kind='bar', color='skyblue')
plt.xlabel('Features')
plt.ylabel('Absolute Correlation with Performance Rating')
plt.title('Top 3 Features Affecting Performance')
plt.xticks(rotation=45)
plt.show()
```




```
# Plotting the top 3 features affecting performance using a pie chart
plt.figure(figsize=(6, 4))
plt.pie(top_3_features, labels=top_3_features.index, autopct='%1.1f%%', startangle=
plt.title('Top 3 Features Affecting Performance')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.show()
```




```
''' Accoding to above visualization in piechart most affecting factorsare EMPLOYE
```

```
' Accoding to above visualization in piechart most affecting factorsare EMPL
OYEEENVIRONMENTSATISFACTION , EMPLOYEESALARY HIKE YEARSINCELAST PROMOTION'
```

```
top_5_features = correlation_with_target.head(5)
top_5_features
```

```
EmpEnvironmentSatisfaction      0.395561
EmpLastSalaryHikePercent        0.333722
YearsSinceLastPromotion         0.167629
ExperienceYearsInCurrentRole    0.147638
EmpWorkLifeBalance              0.124429
Name: PerformanceRating, dtype: float64
```

FEATURE ENCODING

Label encoding is a process of converting categorical variables into numerical format. It assigns a unique numerical label to each category in the categorical variable

```
categorical_columns = df.select_dtypes(include=['object']).columns
categorical_columns
```

```
Index(['EmpNumber', 'Gender', 'EducationBackground', 'MaritalStatus',
       'EmpDepartment', 'EmpJobRole', 'BusinessTravelFrequency', 'OverTime',
       'Attrition'],
      dtype='object')
```

```
# Dropping the first columns as it is of no use for analysis.
df.drop(['EmpNumber'],inplace=True,axis=1)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 27 columns):
 #   Column                        Non-Null Count  Dtype 
---  ------                        --------------  ----- 
 0   Age                           1200 non-null   int64 
 1   Gender                        1200 non-null   object
 2   EducationBackground           1200 non-null   object
 3   MaritalStatus                 1200 non-null   object
 4   EmpDepartment                 1200 non-null   object
 5   EmpJobRole                    1200 non-null   object
 6   BusinessTravelFrequency       1200 non-null   object
 7   DistanceFromHome              1200 non-null   int64 
 8   EmpEducationLevel             1200 non-null   int64 
 9   EmpEnvironmentSatisfaction    1200 non-null   int64 
 10  EmpHourlyRate                 1200 non-null   int64 
 11  EmpJobInvolvement             1200 non-null   int64 
 12  EmpJobLevel                   1200 non-null   int64 
 13  EmpJobSatisfaction            1200 non-null   int64 
 14  NumCompaniesWorked            1200 non-null   int64 
 15  OverTime                      1200 non-null   object
 16  EmpLastSalaryHikePercent      1200 non-null   int64 
 17  EmpRelationshipSatisfaction   1200 non-null   int64 
 18  TotalWorkExperienceInYears    1200 non-null   int64 
 19  TrainingTimesLastYear         1200 non-null   int64 
 20  EmpWorkLifeBalance            1200 non-null   int64 
 21  ExperienceYearsAtThisCompany  1200 non-null   int64 
 22  ExperienceYearsInCurrentRole  1200 non-null   int64 
 23  YearsSinceLastPromotion       1200 non-null   int64 
 24  YearsWithCurrManager          1200 non-null   int64 
 25  Attrition                     1200 non-null   object
 26  PerformanceRating             1200 non-null   int64 
dtypes: int64(19), object(8)
memory usage: 253.2+ KB
```

```
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder() # labelencoder
for i in (1,2,3,4,5,6,15,25):
    df.iloc[:,i] = enc.fit_transform(df.iloc[:,i])
df.head()
```

	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJobRole
0	32	1	2	2	5	13
1	47	1	2	2	5	13
2	40	1	1	1	5	13
3	41	1	0	0	3	8
4	60	1	2	2	5	13

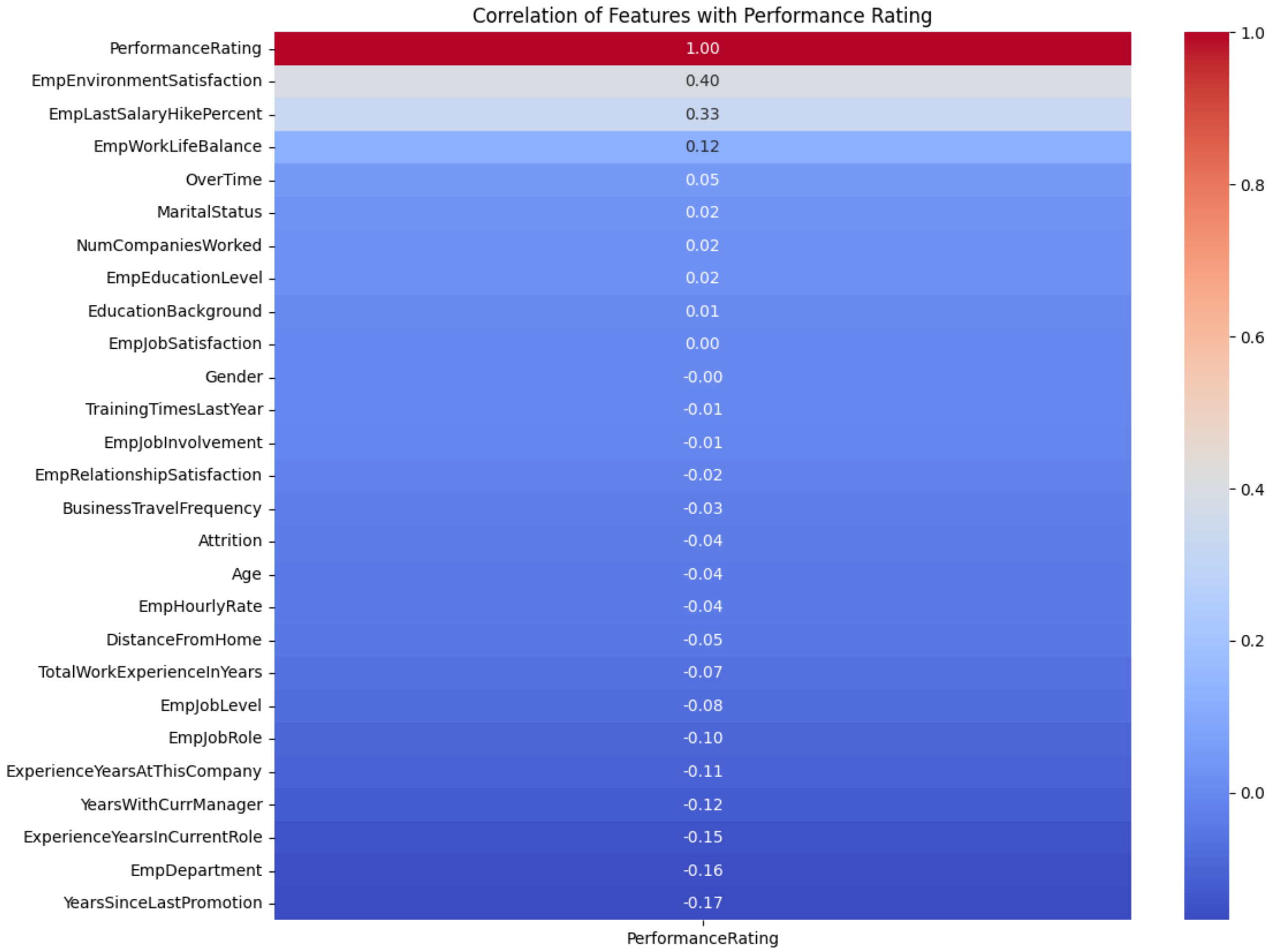
5 rows × 27 columns

FEATURE SELECTION:

```
corr=df.corr()
corr
```

Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJobR
1.000000	-0.040107	-0.055905	-0.098368	-0.000104	-0.037
-0.040107	1.000000	0.009922	-0.042169	-0.010925	0.011
-0.055905	0.009922	1.000000	-0.001097	-0.026874	-0.012
-0.098368	-0.042169	-0.001097	1.000000	0.067272	0.038
-0.000104	-0.010925	-0.026874	0.067272	1.000000	0.568
-0.037665	0.011332	-0.012325	0.038023	0.568973	1.000
0.040579	-0.043608	0.012382	0.028520	-0.045233	-0.086
0.020937	-0.001507	-0.013919	-0.019148	0.007707	0.022
0.207313	-0.022960	-0.047978	0.026737	0.019175	-0.016
0.013814	0.000033	0.045028	-0.032467	-0.019237	0.044
0.062867	0.002218	-0.030234	-0.013540	0.003957	-0.016
0.027216	0.010949	-0.025505	-0.043355	-0.076988	-0.008
0.509139	-0.050685	-0.056338	-0.087359	0.100526	0.004
-0.002436	0.024680	-0.030977	0.044593	0.007150	0.032
0.284408	-0.036675	-0.032879	-0.030095	-0.033950	-0.009
0.051910	-0.038410	0.007046	-0.022833	-0.026841	0.015
-0.006105	-0.005319	-0.009788	0.010128	-0.012661	0.005
0.049749	0.030707	0.005652	0.026410	-0.050286	-0.043
0.680886	-0.061055	-0.027929	-0.093537	0.016065	-0.049
-0.016053	-0.057654	0.051596	0.026045	0.016438	0.004
-0.019563	0.015793	0.022890	0.014154	0.068875	-0.007
0.318852	-0.030392	-0.009887	-0.075728	0.047677	-0.009
0.217163	-0.031823	-0.003215	-0.076663	0.069602	0.019
0.228199	-0.021575	0.014277	-0.052951	0.052315	0.012
0.205098	-0.036643	0.002767	-0.061908	0.033850	-0.004
-0.189317	0.035758	0.027161	0.162969	0.048006	0.037
-0.040164	-0.001780	0.005607	0.024172	-0.162615	-0.096

```
plt.figure(figsize=(12, 10))
sns.heatmap(corr[['PerformanceRating']].sort_values(by='PerformanceRating', ascendi
plt.title('Correlation of Features with Performance Rating')
plt.show()
```




```
# Taking only variables with correlation coeffecient greater than 0.1

corr_matrix = df.corr()
```

```
# Extract the correlation of each feature with the target column ('PerformanceRatir
correlation_with_target = corr_matrix['PerformanceRating'].drop('PerformanceRating'

# Select features with correlation coefficient greater than 0.1
selected_features = correlation_with_target[correlation_with_target.abs() > 0.1]

# Display the selected features
print("Selected features with correlation coefficient > 0.1 with PerformanceRating:
print(selected_features)
```

```
Selected features with correlation coefficient > 0.1 with PerformanceRating:
EmpDepartment                  -0.162615
EmpEnvironmentSatisfaction      0.395561
EmpLastSalaryHikePercent        0.333722
EmpWorkLifeBalance              0.124429
ExperienceYearsAtThisCompany   -0.111645
ExperienceYearsInCurrentRole   -0.147638
YearsSinceLastPromotion        -0.167629
YearsWithCurrManager           -0.122313
Name: PerformanceRating, dtype: float64
```

```
selected_feature_indices = selected_features.index.tolist()
```

```
X = df[selected_feature_indices]
X.head()
```

	EmpDepartment	EmpEnvironmentSatisfaction	EmpLastSalaryHikePercent	EmpWo
0	5	4	12	
1	5	4	12	
2	5	4	21	
3	3	2	15	
4	5	1	14	

Next steps: Generate code with X View recommended plots

```
y = df['PerformanceRating']
```

Splitting into train and test

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_sta
```

```
#The `random_state` parameter in `train_test_split` ensures reproducibility by s
```

SCALING :STANDARD SCALER:

- StandardScaler from scikit-learn is applied to scale the features in the training and testing sets, ensuring that they have a mean of 0 and a standard deviation of 1, which aids in model convergence and performance.

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()  # Initialize StandardScaler
X_train_scaled = sc.fit_transform(X_train)  # Scale training set
X_test_scaled = sc.transform(X_test)  # Scale test set
#scaling appliedonly for input features
```

```
X_train.shape
```

```
(840, 8)
```

```
X_test.shape
```

```
(360, 8)
```

5.MODEL TRAINING:

MODEL1. LOGISTICREGRESSION

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

model1 = LogisticRegression()
model1.fit(X_train, y_train)  #train the LR model
```

```
▾ LogisticRegression
LogisticRegression()
```

```
#make the prediction ontesting set
y_predict1 = model1.predict(X_test)
```

```
accuracy_model1 = accuracy_score(y_test, y_predict1)
print("Accuracy:", accuracy_model1)
```

```
Accuracy: 0.8361111111111111
```

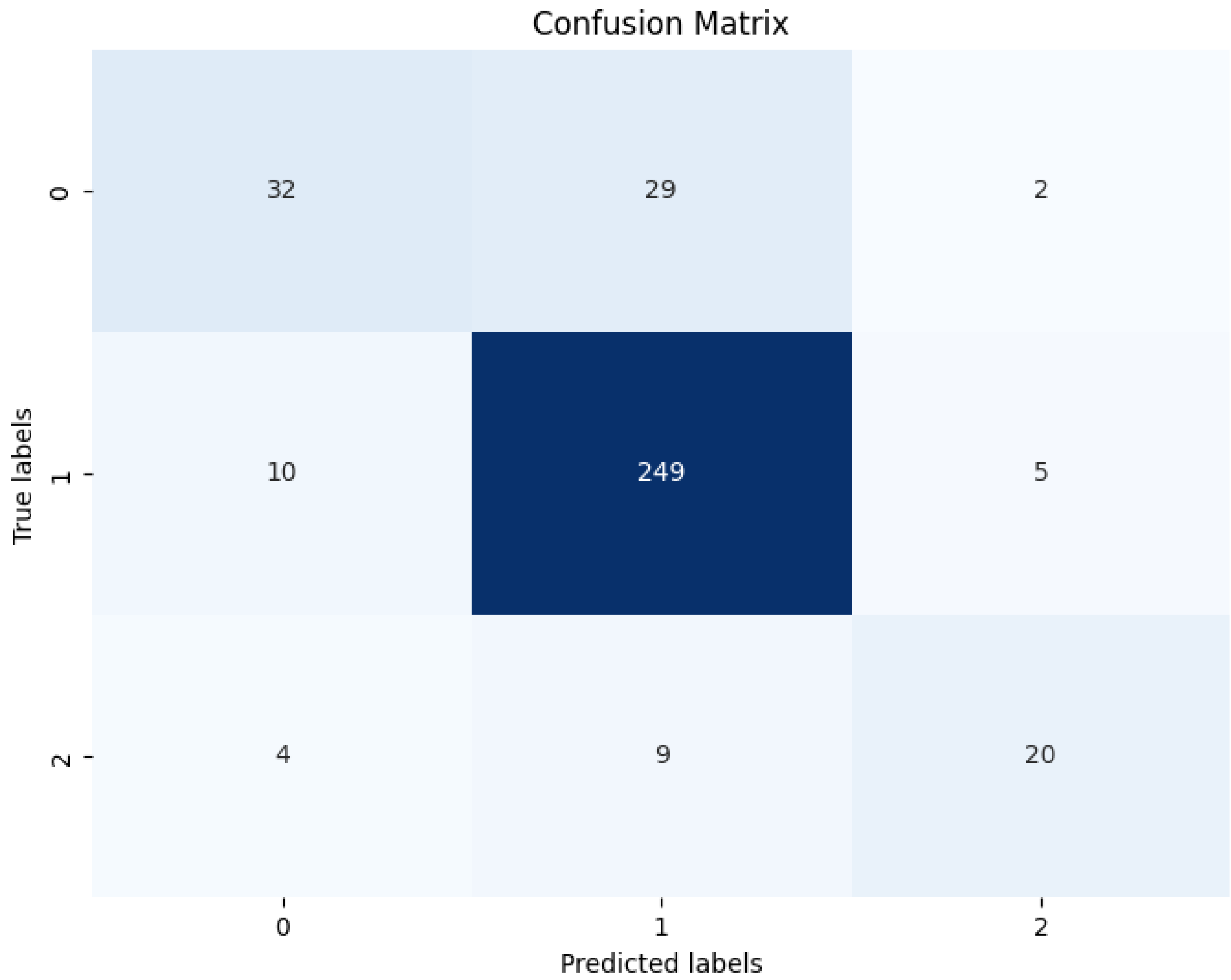
```
print("Classification Report:")
print(classification_report(y_test, y_predict1))
```

```
Classification Report:
              precision    recall  f1-score   support

           2       0.70      0.51      0.59        63
           3       0.87      0.94      0.90       264
           4       0.74      0.61      0.67        33

    accuracy                           0.84       360
   macro avg       0.77      0.69      0.72       360
weighted avg       0.83      0.84      0.83       360
```

```
#confusion matrix
print(confusion_matrix(y_test, y_predict1))
```

```
[[ 32  29   2]
 [ 10 249   5]
 [  4   9  20]]
```

```
# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_predict1)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



MODEL2 SVM:

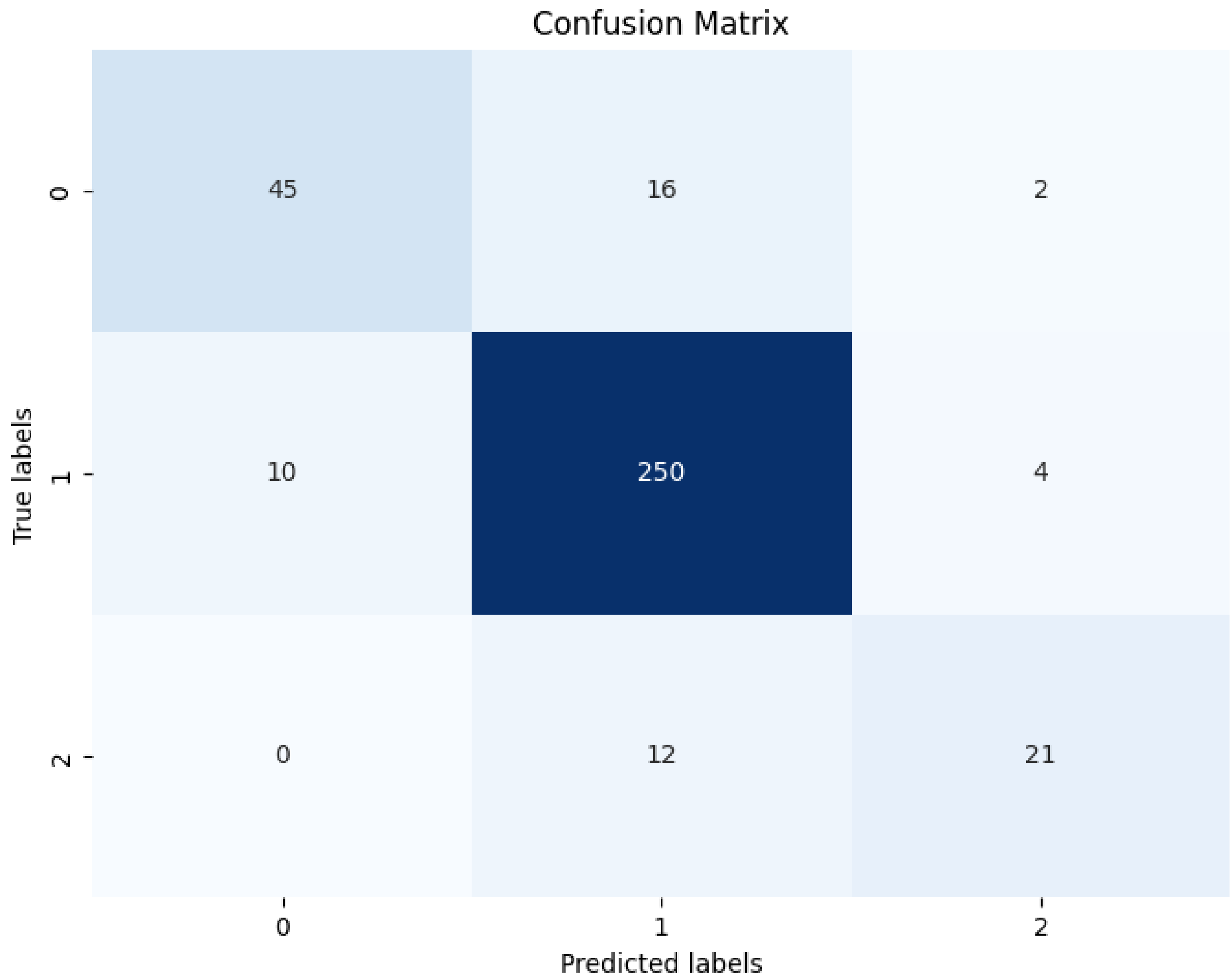
```
from sklearn.svm import SVC

model2=SVC(kernel='rbf', C=100, random_state=10)

model2.fit(X_train, y_train)

y_predict_model2 = model2.predict(X_test)

accuracy_model2 = accuracy_score(y_test, y_predict_model2)
print("Accuracy:", accuracy_model2)
```

```
Accuracy: 0.8777777777777778
```

```
# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_predict_model2))

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_predict_model2))
```

```
Classification Report:
              precision    recall  f1-score   support

           2       0.82      0.71      0.76        63
           3       0.90      0.95      0.92       264
           4       0.78      0.64      0.70        33

    accuracy                           0.88       360
   macro avg       0.83      0.77      0.80       360
weighted avg       0.87      0.88      0.87       360

Confusion Matrix:
[[ 45  16   2]
 [ 10 250   4]
 [  0  12  21]]
```

```
cm = confusion_matrix(y_test, y_predict_model2)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



MODEL3 Decision Tree with GridSearchCV:

- Decision Tree with GridSearchCV involves training a Decision Tree model while tuning its hyperparameters using a grid search technique called GridSearchCV. This method systematically explores different combinations of hyperparameters to find the best ones for the model.
- By doing so, it helps improve the model's performance.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```
#  the Decision Tree classifier
model3 = DecisionTreeClassifier(random_state=10)

# Instantiate GridSearchCV
grid_search = GridSearchCV(estimator=model3, param_grid=param_grid, cv=5)

# Fit GridSearchCV to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and best estimator
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Make predictions
y_pred_model3 = best_model.predict(X_test)
```

```
best_params
```

```
{'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

```
#prediction model3
y_pred_model3 = best_model.predict(X_test)
```

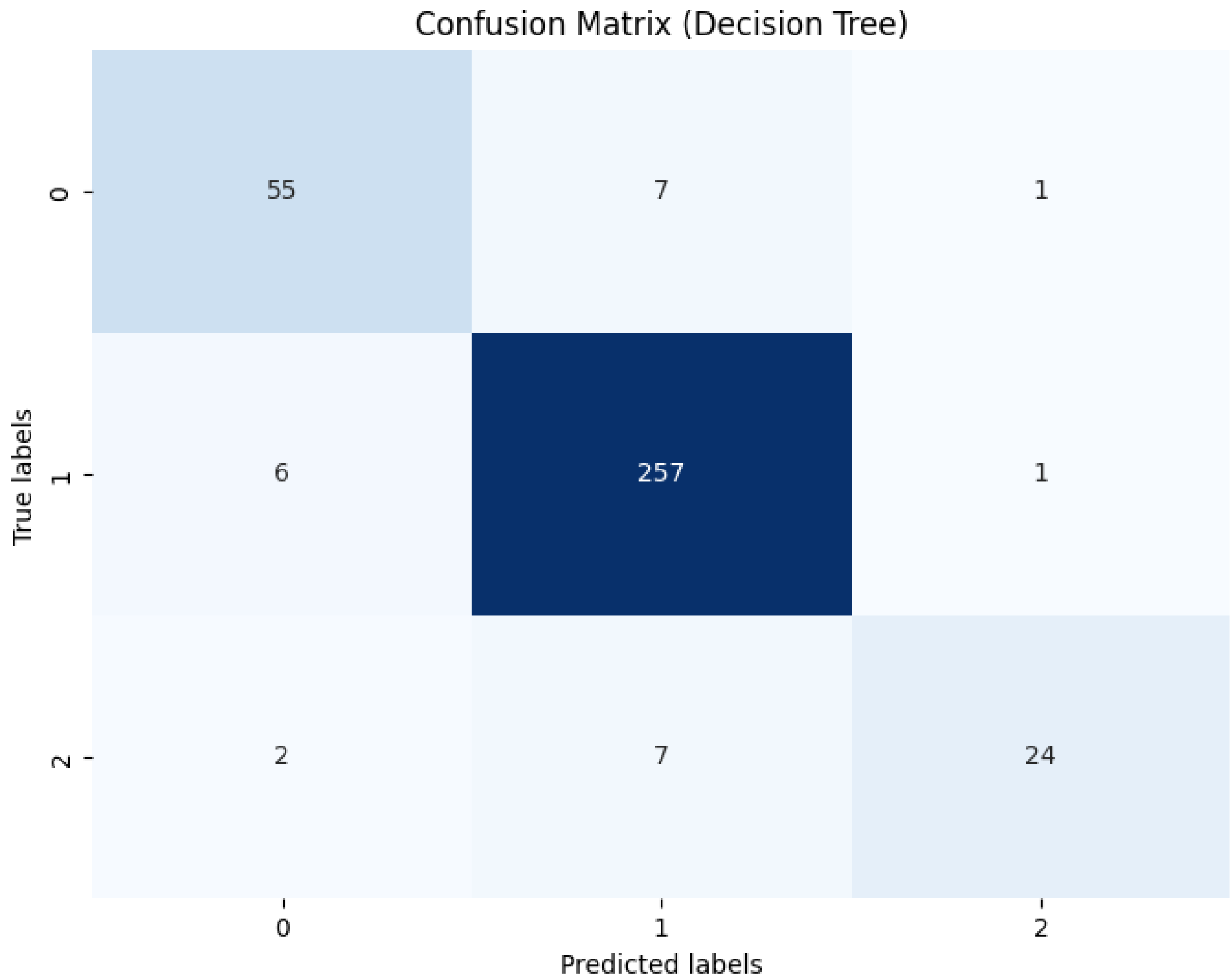
```
accuracy_model3 = accuracy_score(y_test, y_pred_model3)
print("Accuracy:", accuracy_model3)
```

```
Accuracy: 0.9333333333333333
```

```
print(classification_report(y_test, y_pred_model3))
print("Confusion Matrix:")
cm_model3 = confusion_matrix(y_test, y_pred_model3)
print(cm_model3)
```

```
              precision    recall  f1-score   support

           2       0.87      0.87      0.87        63
           3       0.95      0.97      0.96       264
           4       0.92      0.73      0.81        33

    accuracy                           0.93       360
   macro avg       0.91      0.86      0.88       360
weighted avg       0.93      0.93      0.93       360

Confusion Matrix:
[[ 55   7   1]
 [  6 257   1]
 [  2   7  24]]
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(cm_model3, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix (Decision Tree)')
plt.show()
```



MODEL 4:Randomforest girdsearch cv ⁉️

```
from sklearn.ensemble import RandomForestClassifier
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
model4 = RandomForestClassifier(random_state=10)
grid_search = GridSearchCV(estimator=model4, param_grid=param_grid, cv=5)
```

```
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
```

```
best_params
```

```
{'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 10,
 'n_estimators': 100}
```

```
y_pred_model4 = best_model.predict(X_test)
```

```
accuracy_model4 = accuracy_score(y_test, y_pred_model4)
print("Accuracy:", accuracy_model4)
```