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

%matplotlib inline
```

```
df = pd.read_csv('emp.csv')
df.head()
```

	Employee_Name	EmpID	MarriedID	MaritalStatusID	GenderID	EmpStatusID	DeptID
0	Adinolfi, Wilson K	10026	0	0	1	1	5
1	Ait Sidi, Karthikeyan	10084	1	1	1	5	3
2	Akinkuolie, Sarah	10196	1	1	0	5	5
3	Alagbe,Trina	10088	1	1	0	1	5
4	Anderson, Carol	10069	0	2	0	5	5

5 rows × 36 columns

```
df.describe().transpose()
```

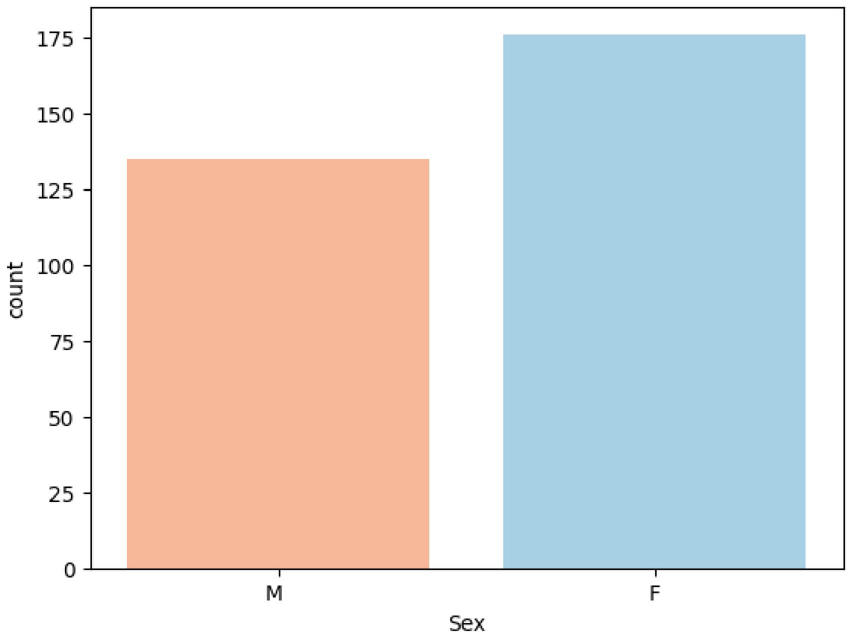
	count	mean	std	min	25%	50%
EmpID	311.0	10156.000000	89.922189	10001.00	10078.50	10156.00
MarriedID	311.0	0.398714	0.490423	0.00	0.00	0.00
MaritalStatusID	311.0	0.810289	0.943239	0.00	0.00	1.00
GenderID	311.0	0.434084	0.496435	0.00	0.00	0.00
EmpStatusID	311.0	2.392283	1.794383	1.00	1.00	1.00
DeptID	311.0	4.610932	1.083487	1.00	5.00	5.00
PerfScoreID	311.0	2.977492	0.587072	1.00	3.00	3.00
FromDiversityJobFairID	311.0	0.093248	0.291248	0.00	0.00	0.00
Salary	311.0	69020.684887	25156.636930	45046.00	55501.50	62810.00
Termd	311.0	0.334405	0.472542	0.00	0.00	0.00
PositionID	311.0	16.845659	6.223419	1.00	18.00	19.00
Zip	311.0	6555.482315	16908.396884	1013.00	1901.50	2132.00
ManagerID	303.0	14.570957	8.078306	1.00	10.00	15.00
EngagementSurvey	311.0	4.110000	0.789938	1.12	3.69	4.28
EmpSatisfaction	311.0	3.890675	0.909241	1.00	3.00	4.00
SpecialProjectsCount	311.0	1.218650	2.349421	0.00	0.00	0.00
DaysLateLast30	311.0	0.414791	1.294519	0.00	0.00	0.00
Absences	311.0	10.237942	5.852596	1.00	5.00	10.00

```
df.columns
```

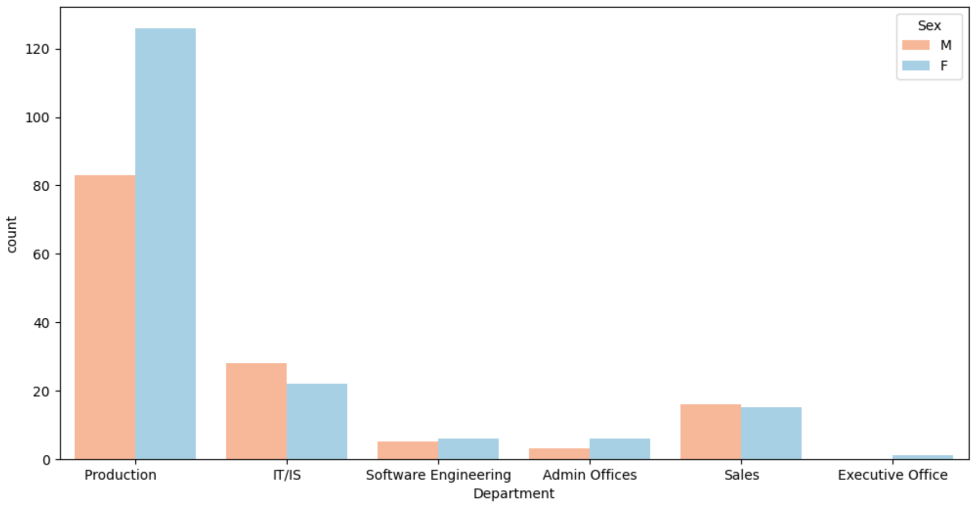
```
Index(['Employee_Name', 'EmpID', 'MarriedID', 'MaritalStatusID', 'GenderID',
       'EmpStatusID', 'DeptID', 'PerfScoreID', 'FromDiversityJobFairID',
       'Salary', 'Termd', 'PositionID', 'Position', 'State', 'Zip', 'DOB',
       'Sex', 'MaritalDesc', 'CitizenDesc', 'HispanicLatino', 'RaceDesc',
       'DateofHire', 'DateofTermination', 'TermReason', 'EmploymentStatus',
```

```
       'Department', 'ManagerName', 'ManagerID', 'RecruitmentSource',
       'PerformanceScore', 'EngagementSurvey', 'EmpSatisfaction',
       'SpecialProjectsCount', 'LastPerformanceReview_Date', 'DaysLateLast30',
       'Absences'],
      dtype='object')
```

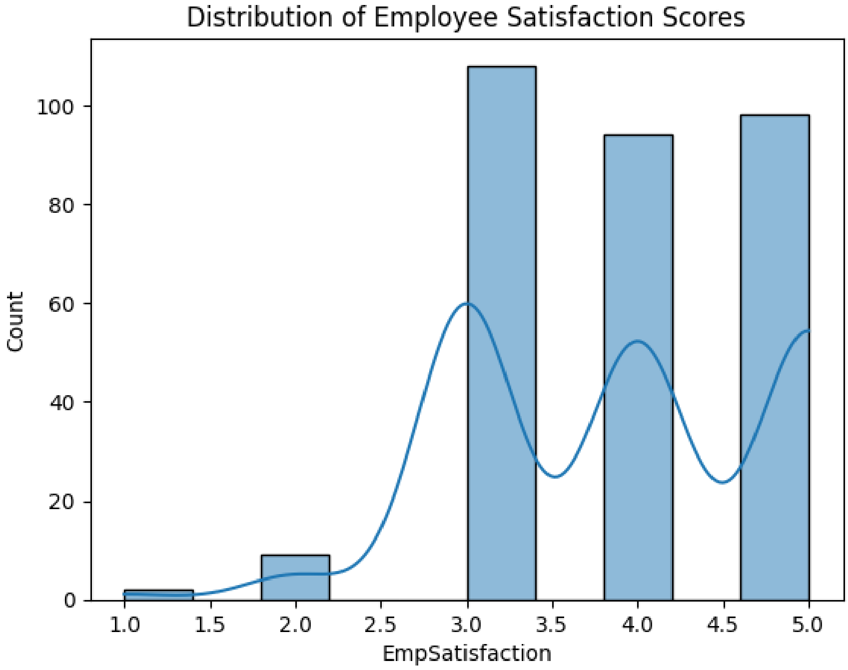
```
sns.countplot(x='Sex',data=df,palette='RdBu',saturation=1);
```

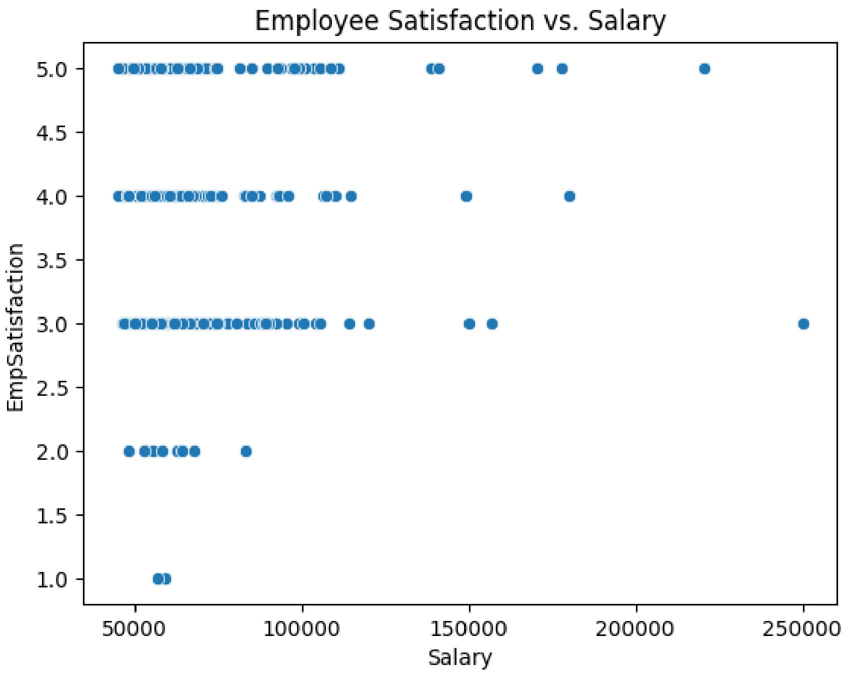



```
plt.figure(figsize=(12,6))
sns.countplot(x='Department',data=df,hue='Sex',palette='RdBu',saturation=1);
```



```
# Distribution of employee satisfaction scores
sns.histplot(df['EmpSatisfaction'],kde=True )
plt.title('Distribution of Employee Satisfaction Scores')
plt.show()

# Employee satisfaction vs. salary
sns.scatterplot(x='Salary', y='EmpSatisfaction', data=df)
plt.title('Employee Satisfaction vs. Salary')
plt.show()
```

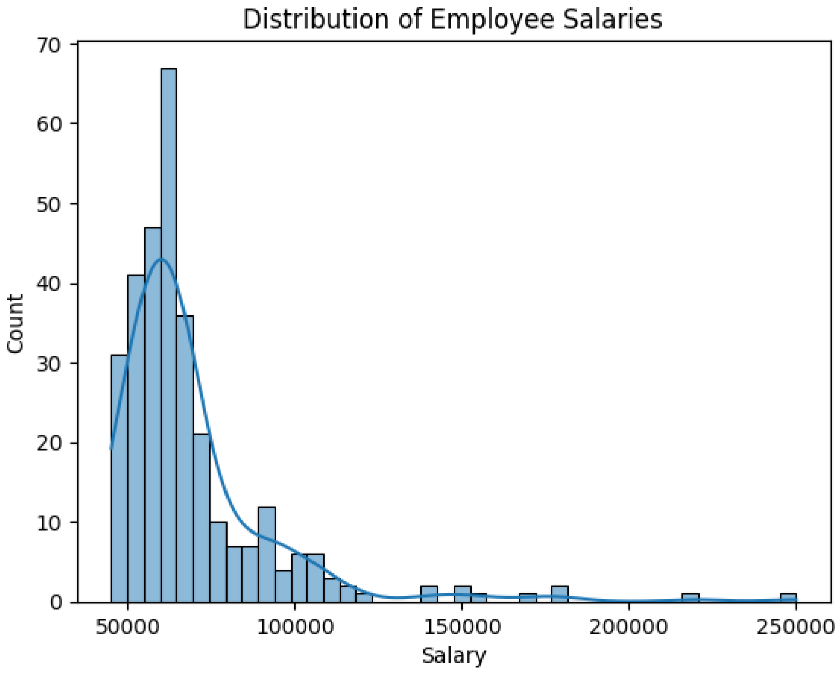


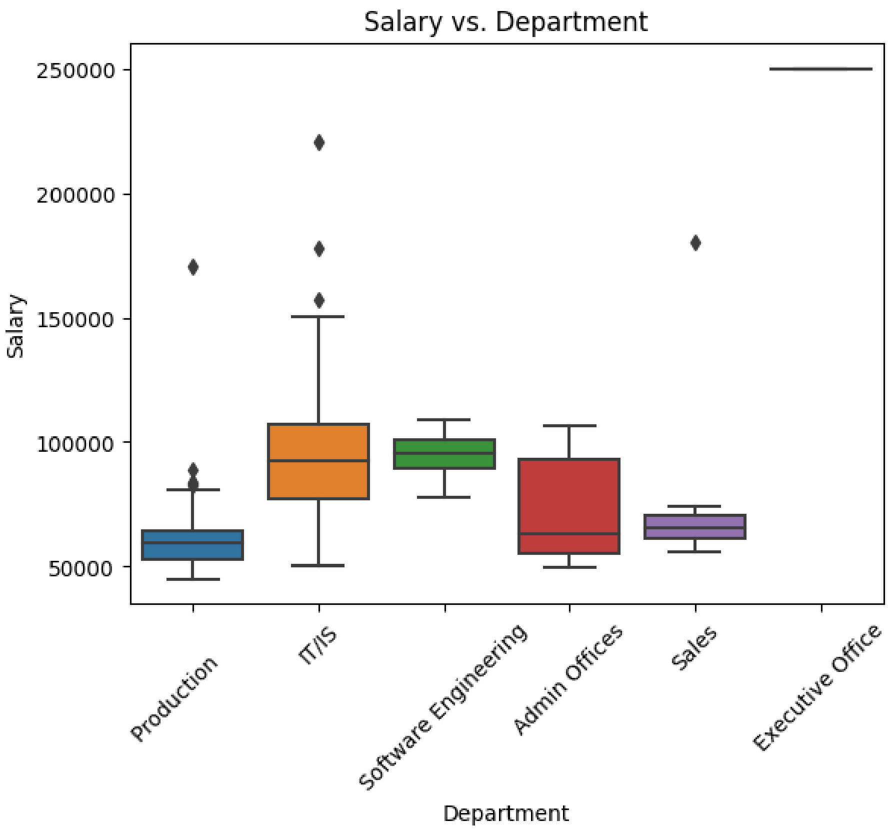


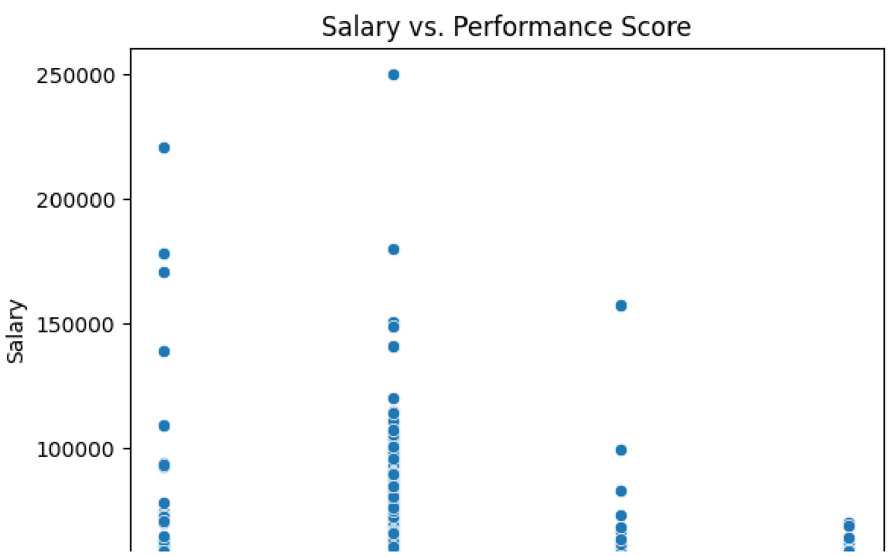

```
pip install plotly
```

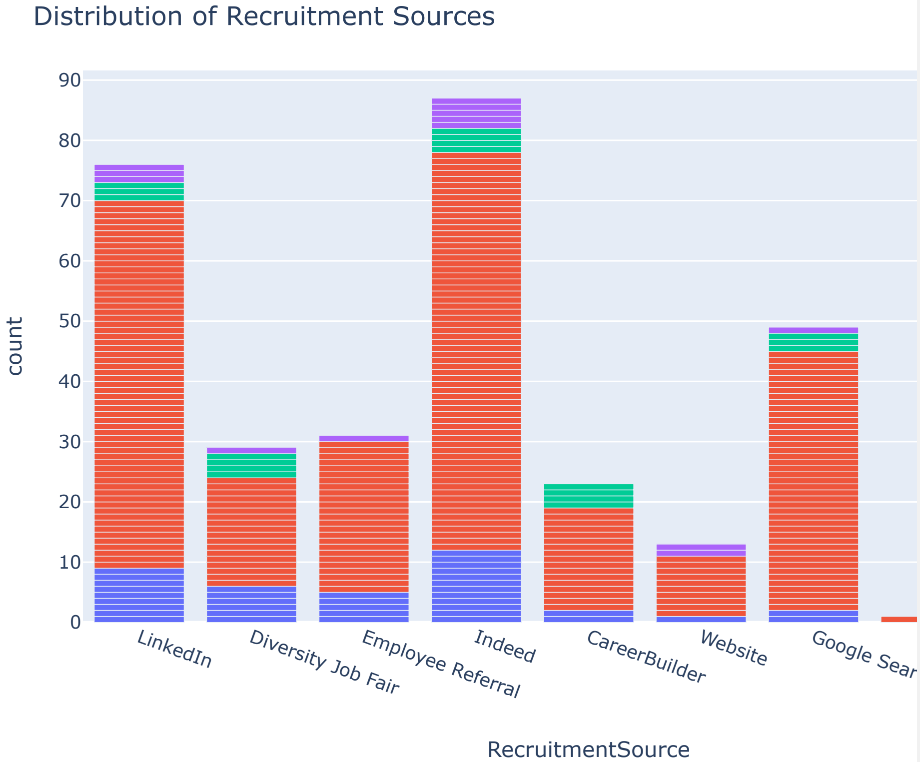
```
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly) (8.2.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly) (23.2)
```

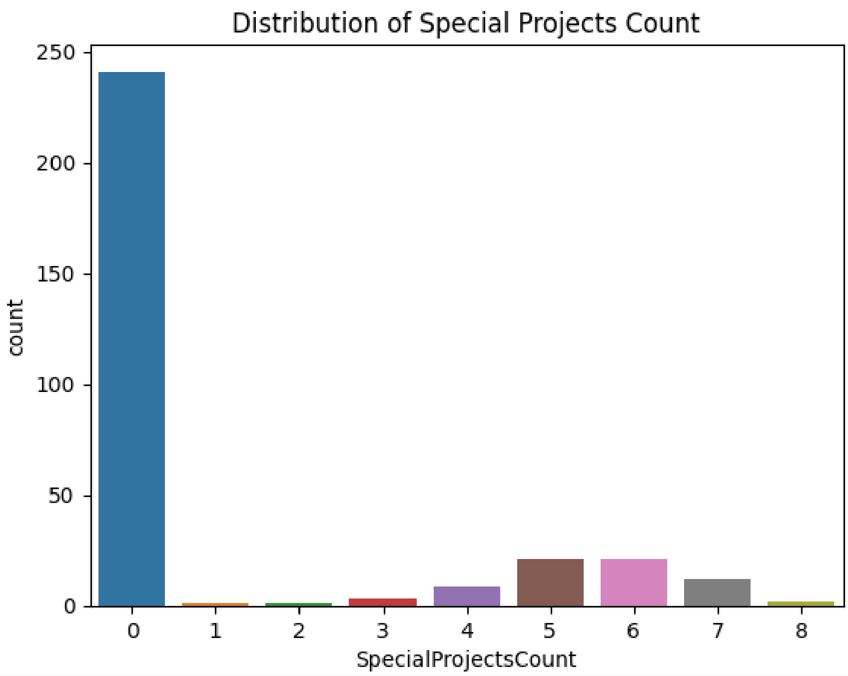
```
# Salary distribution
import plotly.express as px
sns.histplot(df['Salary'], kde=True)
plt.title('Distribution of Employee Salaries')
plt.show()

# Salary vs. department
sns.boxplot(x='Department', y='Salary', data=df)
plt.title('Salary vs. Department')
plt.xticks(rotation=45)
plt.show()

# Salary vs. performance
sns.scatterplot(x='PerformanceScore', y='Salary', data=df)
plt.title('Salary vs. Performance Score')
plt.show()

# Distribution of recruitment sources
import plotly.express as px

# Assuming 'df' is your DataFrame with 'RecruitmentSource' and 'PerformanceScore' columns
fig = px.bar(df, x="RecruitmentSource", color="PerformanceScore")

# Update layout to set the title and x-axis rotation
fig.update_layout(
    title='Distribution of Recruitment Sources',
    xaxis=dict(tickangle=20, tickmode='array', tickvals=list(df['RecruitmentSource'].unique()))
)

fig.show()

# Special projects count distribution
sns.countplot(x='SpecialProjectsCount', data=df)
plt.title('Distribution of Special Projects Count')
plt.show()
```


Distribution of Employee Salaries
Count
70
60
50
40
30
20
10
0
50000
100000
150000
200000
250000
Salary


Salary vs. Department
Salary
250000
200000
150000
100000
50000
Production
IT/IS
Software Engineering
Admin Offices
Sales
Executive Office
Department


Salary vs. Performance Score
Salary
250000
200000
150000
100000






```
df.isnull().sum()
```

```
Employee_Name               0
EmpID                       0
MarriedID                   0
MaritalStatusID             0
GenderID                    0
EmpStatusID                 0
DeptID                      0
PerfScoreID                 0
FromDiversityJobFairID      0
Salary                      0
Termd                       0
PositionID                  0
Position                    0
```

```
State                          0
Zip                            0
DOB                            0
Sex                            0
MaritalDesc                    0
CitizenDesc                    0
HispanicLatino                 0
RaceDesc                       0
DateofHire                     0
DateofTermination            207
TermReason                     0
EmploymentStatus               0
Department                     0
ManagerName                    0
ManagerID                      8
RecruitmentSource              0
PerformanceScore               0
EngagementSurvey               0
EmpSatisfaction                0
SpecialProjectsCount           0
LastPerformanceReview_Date     0
DaysLateLast30                 0
Absences                       0
dtype: int64
```

```
df = df.drop(columns=['DateofTermination'])
```

```
df[df['ManagerID'].isna()]
```

	Employee_Name	EmpID	MarriedID	MaritalStatusID	GenderID	EmpStatusID	DeptID
19	Becker, Scott	10277	0	0	1	3	5
30	Buccheri, Joseph	10184	0	0	1	1	5
44	Chang, Donovan E	10154	0	0	1	1	5
88	Fancett, Nicole	10136	0	0	0	1	5
135	Hutter, Rosalie	10214	0	3	0	2	5
177	Manchester, Robyn	10077	1	1	0	2	5
232	Rivera, Haley	10011	1	1	0	1	5
251	Sewkumar, Nori	10071	0	0	0	3	5

8 rows × 35 columns

```
df[df['ManagerName']=='Webster Butler'][['ManagerName','ManagerID']]
```

	ManagerName	ManagerID
4	Webster Butler	39.0
19	Webster Butler	NaN
30	Webster Butler	NaN
44	Webster Butler	NaN
65	Webster Butler	39.0
88	Webster Butler	NaN
89	Webster Butler	39.0
105	Webster Butler	39.0
124	Webster Butler	39.0
135	Webster Butler	NaN
151	Webster Butler	39.0
174	Webster Butler	39.0
177	Webster Butler	NaN
198	Webster Butler	39.0
206	Webster Butler	39.0
214	Webster Butler	39.0
232	Webster Butler	NaN
251	Webster Butler	NaN
276	Webster Butler	39.0
280	Webster Butler	39.0
300	Webster Butler	39.0

```
df['ManagerID'] = df['ManagerID'].replace(np.nan, 39.0)
```

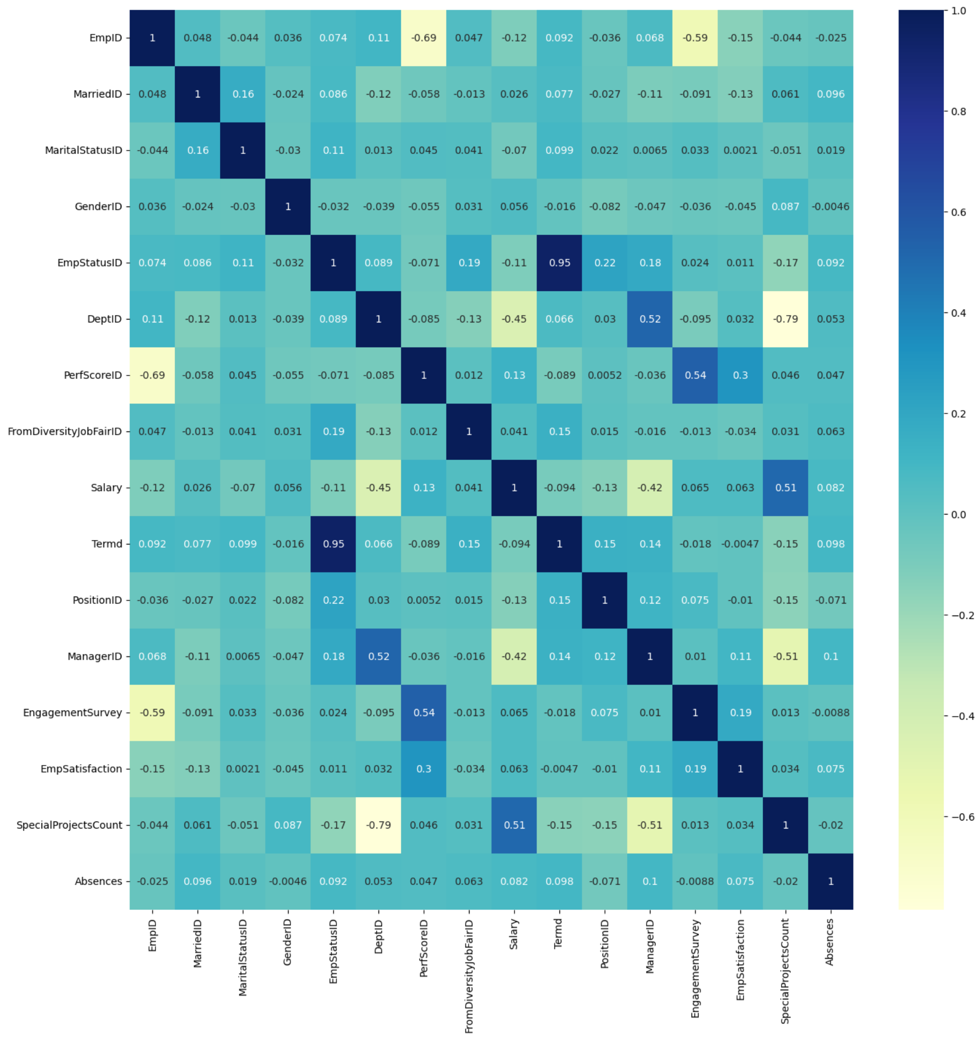
```
df[df['ManagerName']=='Webster Butler'][['ManagerName','ManagerID']]
```

	ManagerName	ManagerID
4	Webster Butler	39.0
19	Webster Butler	39.0
30	Webster Butler	39.0
44	Webster Butler	39.0
65	Webster Butler	39.0
88	Webster Butler	39.0
89	Webster Butler	39.0
105	Webster Butler	39.0
124	Webster Butler	39.0
135	Webster Butler	39.0
151	Webster Butler	39.0
174	Webster Butler	39.0
177	Webster Butler	39.0
198	Webster Butler	39.0
206	Webster Butler	39.0
214	Webster Butler	39.0
232	Webster Butler	39.0
251	Webster Butler	39.0
276	Webster Butler	39.0
280	Webster Butler	39.0
300	Webster Butler	39.0

```
df.drop(['DaysLateLast30','LastPerformanceReview_Date',
         'TermReason','DaysLateLast30','Zip'],axis=1,inplace=True)
```

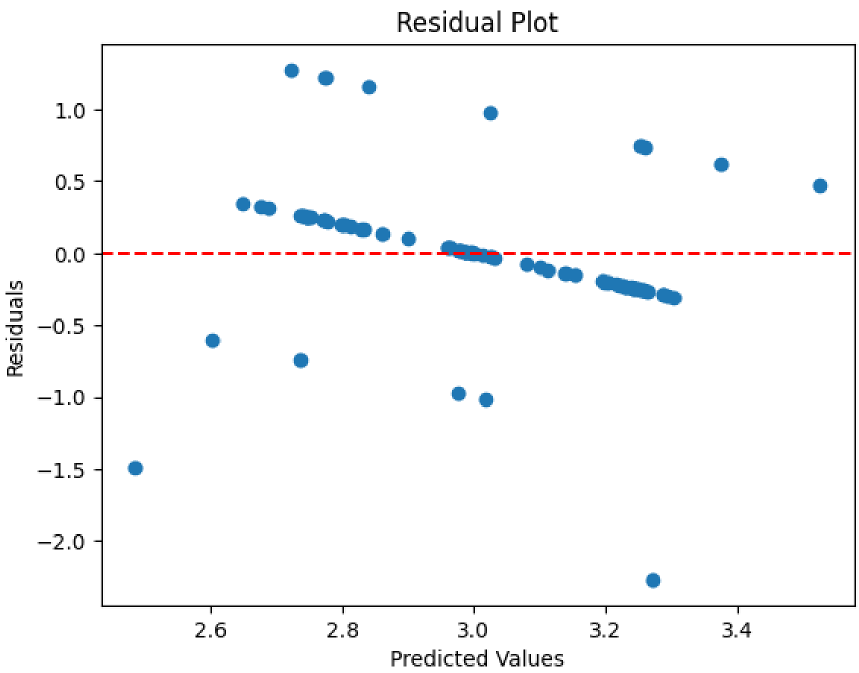
```
plt.figure(figsize=(16,16))
sns.heatmap(df.corr(),cmap="YlGnBu",annot=True);
```

```
<ipython-input-19-e31772ee4191>:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future versi
```

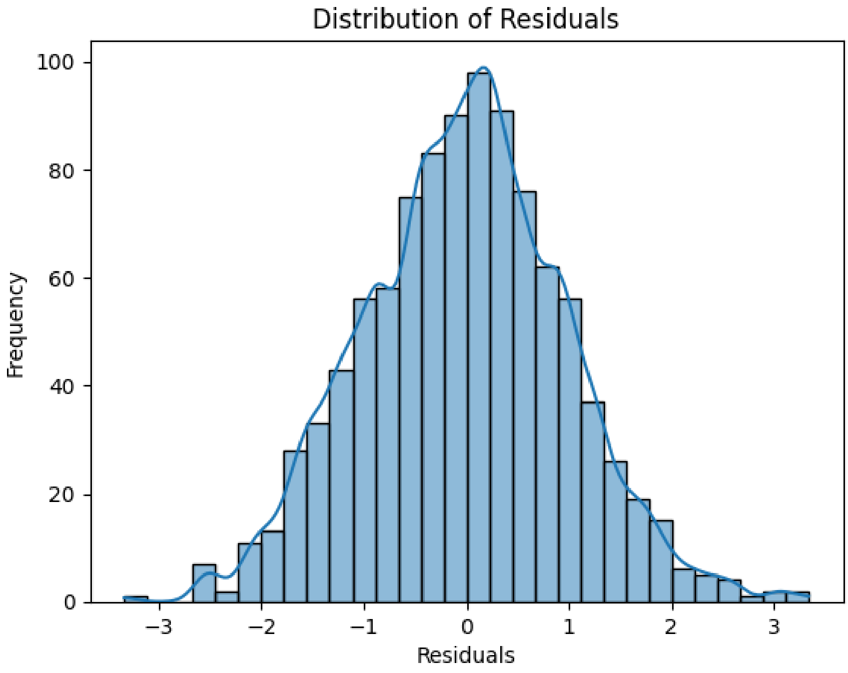



```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

```
from sklearn.linear_model import LinearRegression

lm = LinearRegression()

X = df[['EmpSatisfaction','Salary','SpecialProjectsCount','ManagerID']]
y = df['PerfScoreID']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
lm.fit(X_train,y_train)
predictions = lm.predict(X_test)



residuals = y_test - predictions
plt.scatter(predictions, residuals)
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.axhline(y=0, color='r', linestyle='--')
plt.title("Residual Plot")
plt.show()
```

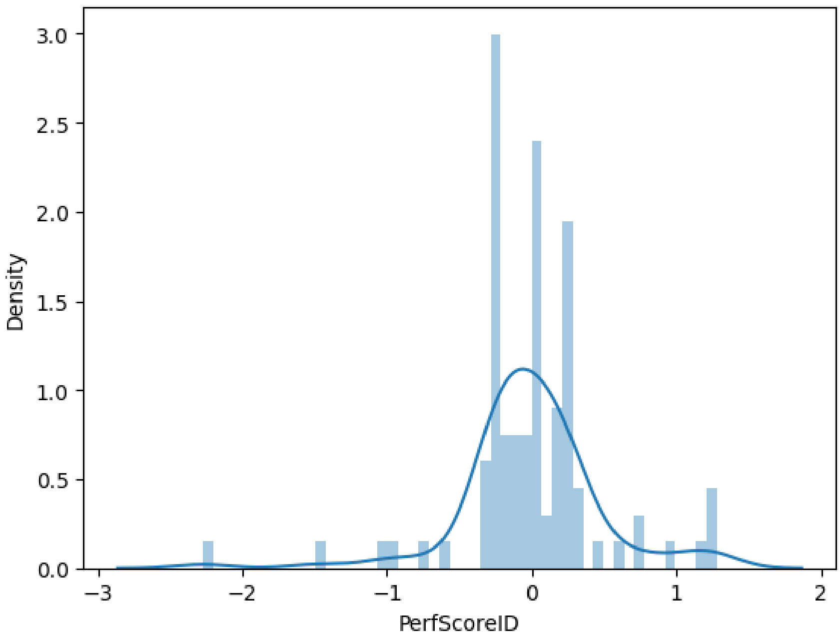



```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Assuming 'residuals' is your array of residuals
residuals = np.random.normal(size=1000)

# Plot histogram with KDE
sns.histplot(residuals, bins=30, kde=True, kde_kws={'bw_adjust': 0.5})

plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Distribution of Residuals")
plt.show()
```




```
sns.distplot((y_test-predictions),bins=50);
```

```
<ipython-input-24-5f2bc21c0ef7>:1: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```




```
coeffecients = pd.DataFrame(lm.coef_,X.columns)
coeffecients.columns = ['Coeffecient']
coeffecients
```

	Coeffecient
EmpSatisfaction	0.245674
Salary	0.000002
SpecialProjectsCount	-0.010926
ManagerID	-0.003805

Double-click (or enter) to edit

```
from sklearn import metrics

#print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
#print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MSE: 0.2347468097647208
```

```
import pandas as pd

# Read CSV file into a DataFrame
df2 = pd.read_csv('emp.csv')

# Convert 'Position' column to numeric (if it contains non-numeric values)
df2['PerfScoreID'] = pd.to_numeric(df2['PerfScoreID'], errors='coerce')

# Calculate mean skill level for "Position"
mean_position = df2['PerfScoreID'].mean()
print(mean_position)
# Calculate skill gap for "Position"
df2['Position_Gap'] = mean_position - df2['PerfScoreID']

# Display the DataFrame with the skill gap for "Position"
print(df2[['EmpID', 'PerfScoreID', 'Position_Gap']])
```

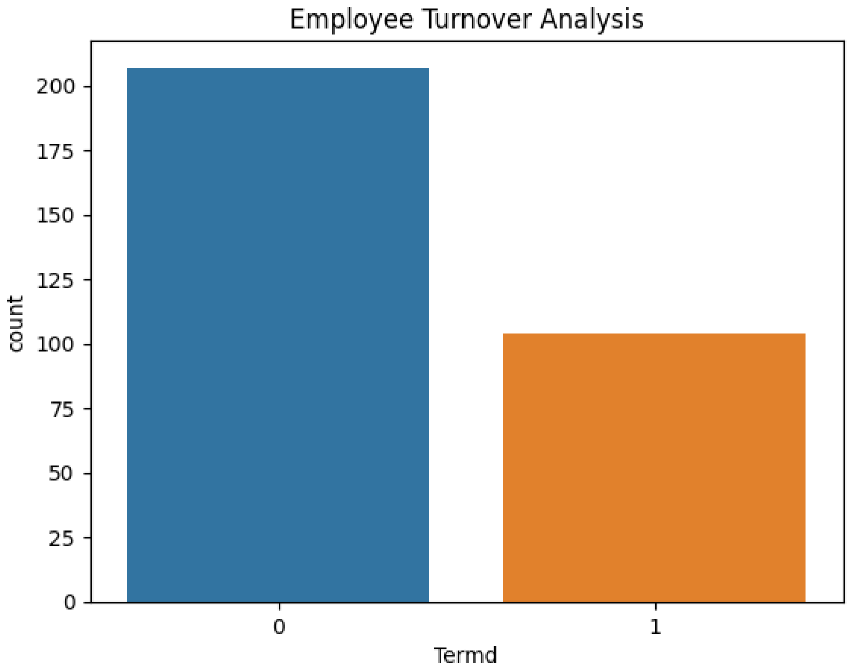
```
2.977491961414791
     EmpID  PerfScoreID  Position_Gap
0    10026            4     -1.022508
1    10084            3     -0.022508
2    10196            3     -0.022508
3    10088            3     -0.022508
4    10069            3     -0.022508
..     ...          ...           ...
306  10135            3     -0.022508
307  10301            1      1.977492
308  10010            4     -1.022508
309  10043            3     -0.022508
310  10271            3     -0.022508

[311 rows x 3 columns]
```

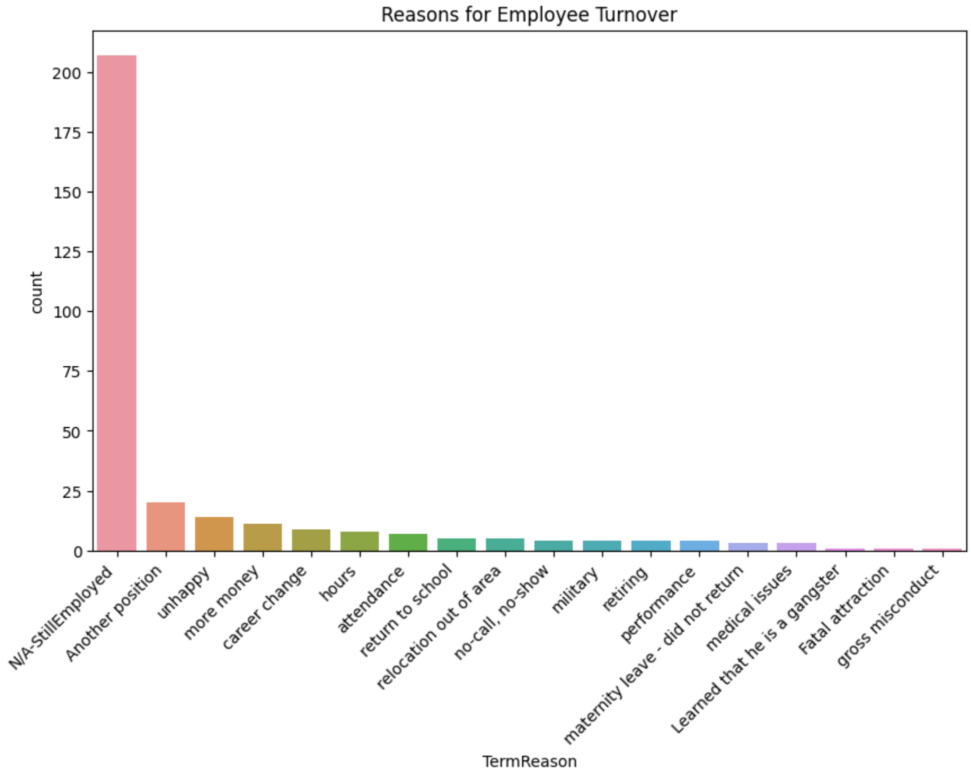
```
'''-------------------------------------------------------------------------------------------------------------------
```

```
'-----------------------------------------------------------------------------------
-------------------------------------------'
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
df=pd.read_csv('emp.csv')
```

```
# Turnover analysis
sns.countplot(x='Termd', data=df)
plt.title('Employee Turnover Analysis')
plt.show()

# Reasons for turnover
plt.figure(figsize=(10, 6))
sns.countplot(x='TermReason', data=df, order=df['TermReason'].value_counts().index)
plt.title('Reasons for Employee Turnover')
plt.xticks(rotation=45, ha='right')
plt.show()
```






```
df['ManagerID'] = df['ManagerID'].replace(np.nan, 39.0)
```

```
'''-------------------------------------------------------------------------------------------------------------------

     '------------------------------------------------------------------------------------
     ------------------------------------------------------------------------------------
     -'
```

```
pip install scikit-optimize
```

```
Collecting scikit-optimize
  Downloading scikit_optimize-0.9.0-py2.py3-none-any.whl (100 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100.3/100.3 kB 2.1 MB/s eta 0:00:00
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.3
Collecting pyaml>=16.9 (from scikit-optimize)
  Downloading pyaml-23.12.0-py3-none-any.whl (23 kB)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from scikit-optimi
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from pyaml>=16.9->scikit-optimiz
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>
Installing collected packages: pyaml, scikit-optimize
Successfully installed pyaml-23.12.0 scikit-optimize-0.9.0
```

```
pip install lightgbm
```

```
Requirement already satisfied: lightgbm in /usr/local/lib/python3.10/dist-packages (4.1.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from lightgbm) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from lightgbm) (1.11.4)
```

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.11.4)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Assuming 'Termd' as the target variable (indicating whether an employee has quit)
target_variable = 'Termd'

# Choose relevant features, adjust as needed
features = ['PerformanceScore', 'EmpSatisfaction', 'ManagerID', 'EngagementSurvey', 'PositionID','SpecialProjectsCount

# Prepare the data
X = df[features]
y = df[target_variable]

# Encode categorical variables
X_encoded = pd.get_dummies(X)

# Split the data into training and testing sets
X_train, X_testx, y_train, y_testx = train_test_split(X_encoded, y, test_size=0.3, random_state=42)

# Build a RandomForestClassifier with hyperparameter tuning
param_grid = {
    'n_estimators': [100, 500, 1000],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print(f'Best Hyperparameters: {best_params}')

# Use the best model
best_model = grid_search.best_estimator_
y_predx = best_model.predict(X_testx)

# Predict the probability of each class
y_probx = best_model.predict_proba(X_testx)

# Print the predicted probabilities
print('\nPredicted Probabilities:')
print(y_probx)

# Choose the probability for the positive class (quitting)
quit_probability = y_probx[:, 1]

# Example: Print the probability for the first few instances
for i in range(5):
    print(f'Employee {i + 1} - Quit Probability: {quit_probability[i]:.4f}')


# Evaluate the model
accuracyx = accuracy_score(y_testx, y_predx)
print(f'Accuracy: {accuracyx:.2f}')

# Additional evaluation metrics
print('\nClassification Report:')
print(classification_report(y_testx, y_predx))
```

```
 [0.69807723 0.30192277]
 [0.80850209 0.19149791]
 [0.73871108 0.26128892]
 [0.65956448 0.34043552]
 [0.76152985 0.23847015]
 [0.67720044 0.32279956]
 [0.73711018 0.26288982]
 [0.67598777 0.32401223]
 [0.69200665 0.30799335]
 [0.4317626  0.5682374 ]
 [0.48481973 0.51518027]
 [0.42566888 0.57433112]
 [0.47902427 0.52097573]
 [0.68142377 0.31857623]
 [0.74776809 0.25223191]
 [0.5545539  0.4454461 ]
 [0.78477794 0.21522206]
 [0.59760911 0.40239089]
 [0.63030374 0.36969626]
 [0.74974821 0.25025179]
 [0.67462781 0.32537219]
 [0.63887632 0.36112368]
 [0.52196098 0.47803902]
 [0.79499319 0.20500681]
 [0.73878528 0.26121472]
 [0.71246034 0.28753966]
 [0.44927375 0.55072625]
 [0.65377237 0.34622763]
 [0.46019867 0.53980133]
 [0.70136478 0.29863522]
 [0.53531973 0.46468027]
 [0.43351552 0.56648448]]
Employee 1 - Quit Probability: 0.4309
Employee 2 - Quit Probability: 0.1524
Employee 3 - Quit Probability: 0.2626
Employee 4 - Quit Probability: 0.4875
Employee 5 - Quit Probability: 0.1545
Accuracy: 0.68

Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.86      0.79        66
           1       0.44      0.25      0.32        28

    accuracy                           0.68        94
   macro avg       0.58      0.56      0.55        94
weighted avg       0.64      0.68      0.65        94
```

```
#better to use above for classification models.
```

```
'''%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    '%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
```

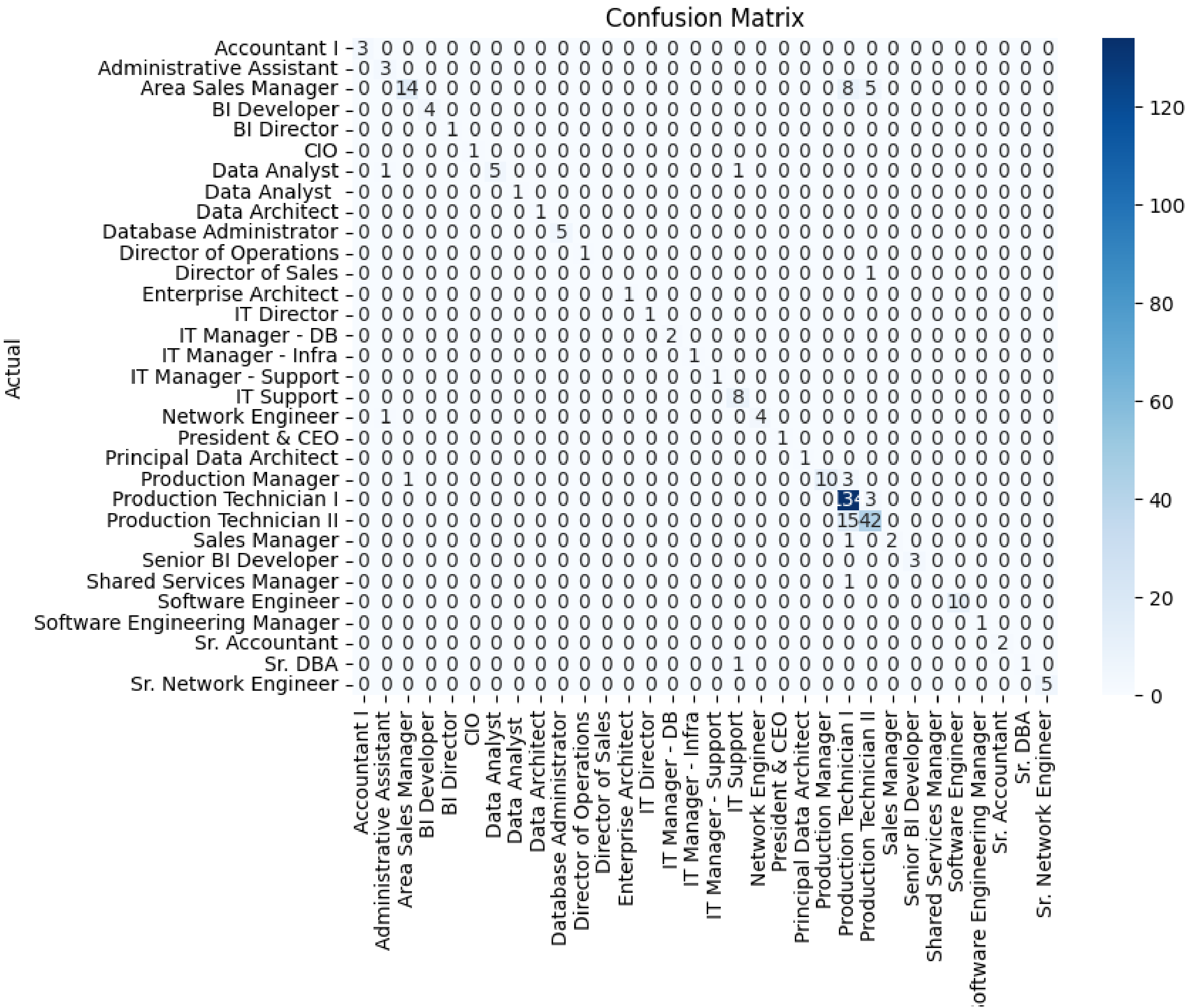
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

# Assuming 'Position' as the target variable for job recommendation
target_variable = 'Position'

# Assume relevant features, adjust as needed
features = ['PerfScoreID', 'EmpSatisfaction', 'EngagementSurvey', 'SpecialProjectsCount']

# Prepare the data
X = df[features]

# Label encode the target variable
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df[target_variable])

# Build a RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X, y)

# Assume a new employee's features (replace these values with actual employee data)
new_employee_data = pd.DataFrame({
    'PerfScoreID': [4],
    'EmpSatisfaction': [3],
    'EngagementSurvey': [4.5],
    'SpecialProjectsCount': [2]
})

# Make a job recommendation for the new employee
recommended_job_label = model.predict(new_employee_data)[0]

# Convert the predicted label back to the original category
recommended_job = label_encoder.inverse_transform([recommended_job_label])[0]

print(f'Recommended Job for the New Employee: {recommended_job}')
```

```
Recommended Job for the New Employee: Production Technician II
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Predict on the training set
y_pred_train = model.predict(X)

# Calculate Mean Squared Error (MSE)
mse_train = mean_squared_error(y, y_pred_train)

# Calculate Mean Absolute Error (MAE)
mae_train = mean_absolute_error(y, y_pred_train)

# Calculate Root Mean Squared Error (RMSE)
rmse_train = mean_squared_error(y, y_pred_train, squared=False)

#print(f'Mean Squared Error (MSE): {mse_train}')
#print(f'Mean Absolute Error (MAE): {mae_train}')
print(f'Root Mean Squared Error (RMSE): {rmse_train}')
```

```
Root Mean Squared Error (RMSE): 4.5909879566392275
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Make predictions on the entire dataset
predictions = model.predict(X)

# Create a confusion matrix
cm = confusion_matrix(y, predictions)

# Plot the confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.class
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
# Extract feature importances from the model
feature_importances = model.feature_importances_

# Create a bar plot for feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importances, y=features)
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Random Forest Feature Importance')
plt.show()
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from itertools import cycle

# Binarize the labels
y_bin = label_binarize(y, classes=range(len(label_encoder.classes_)))

# Get predicted probabilities for each class
y_prob = model.predict_proba(X)

# Compute ROC curve and area under the curve (AUC) for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(label_encoder.classes_)):
    fpr[i], tpr[i], _ = roc_curve(y_bin[:, i], y_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curves for each class
plt.figure(figsize=(10, 8))
colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(len(label_encoder.classes_)), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'ROC curve (class {label_encoder.classes_[i]}) (AUC = {roc_auc[

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Multiclass')
plt.legend(loc='lower right')
plt.show()
```


Confusion Matrix
Actual
Accountant I
Administrative Assistant
Area Sales Manager
BI Developer
BI Director
CIO
Data Analyst
Data Analyst
Data Architect
Database Administrator
Director of Operations
Director of Sales
Enterprise Architect
IT Director
IT Manager - DB
IT Manager - Infra
IT Manager - Support
IT Support
Network Engineer
President & CEO
Principal Data Architect
Production Manager
Production Technician I
Production Technician II
Sales Manager
Senior BI Developer
Shared Services Manager
Software Engineer
Software Engineering Manager
Sr. Accountant
Sr. DBA
Sr. Network Engineer
120
100
80
60
40
20
0