Employee Attriton Prediction

Employee attrition prediction is a data-driven approach to identify factors that lead to employees leaving a company and forecasting which employees are at risk of attrition. By analyzing historical employee data, organizations can develop predictive models to improve retention strategies, reduce turnover costs, and enhance workplace satisfaction.

Import the necessary libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaB
from xgboost import XGBClassifier
from sklearn.metrics import classification_report,ConfusionMatrixDisplay,accuracy_sco
from sklearn.model_selection import train_test_split,RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
import pickle
import warnings
```

```
warnings.filterwarnings('ignore')
```

Read the dataset

```
df=pd.read_csv('/content/drive/MyDrive/train_main.csv')
df
```

	Employee ID	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Perfor F
0	8410	31	Male	19	Education	5390	Excellent	Medium	A
1	64756	59	Female	4	Media	5534	Poor	High	
2	30257	24	Female	10	Healthcare	8159	Good	High	
3	65791	36	Female	7	Education	3989	Good	High	
4	65026	56	Male	41	Education	4821	Fair	Very High	A
...	...	...	...	...	...	...	...	...	
59593	37195	50	Female	12	Education	4414	Fair	High	A
59594	6266	18	Male	4	Healthcare	8040	Fair	High	
59595	54887	22	Female	14	Technology	7944	Fair	High	
59596	861	23	Male	8	Education	2931	Fair	Very High	A
59597	15796	56	Male	19	Technology	6660	Good	High	A

59598 rows × 24 columns

```
df.columns
```

```
Index(['Employee ID', 'Age', 'Gender', 'Years at Company', 'Job Role',
       'Monthly Income', 'Work-Life Balance', 'Job Satisfaction',
       'Performance Rating', 'Number of Promotions', 'Overtime',
       'Distance from Home', 'Education Level', 'Marital Status',
       'Number of Dependents', 'Job Level', 'Company Size', 'Company Tenure',
       'Remote Work', 'Leadership Opportunities', 'Innovation Opportunities',
       'Company Reputation', 'Employee Recognition', 'Attrition'],
      dtype='object')
```

Check for the missing values

```
df.isna().sum()
```

	0
Employee ID	0
Age	0
Gender	0
Years at Company	0
Job Role	0
Monthly Income	0
Work-Life Balance	0
Job Satisfaction	0
Performance Rating	0
Number of Promotions	0
Overtime	0
Distance from Home	0
Education Level	0
Marital Status	0
Number of Dependents	0
Job Level	0
Company Size	0
Company Tenure	0
Remote Work	0
Leadership Opportunities	0
Innovation Opportunities	0
Company Reputation	0
Employee Recognition	0
Attrition	0

dtype: int64

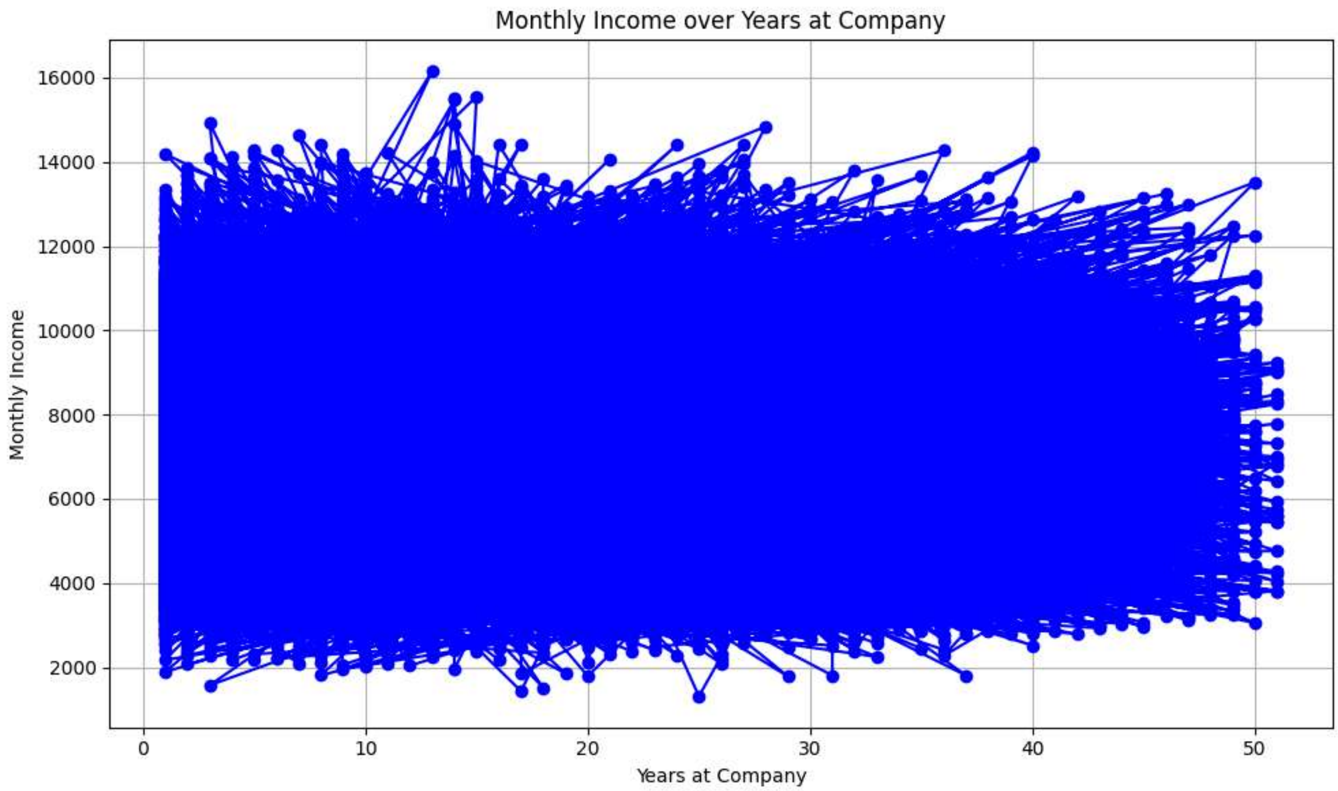
```
df.dtypes
```

	0
Employee ID	int64
Age	int64
Gender	object
Years at Company	int64
Job Role	object
Monthly Income	int64
Work-Life Balance	object
Job Satisfaction	object
Performance Rating	object
Number of Promotions	int64
Overtime	object
Distance from Home	int64
Education Level	object
Marital Status	object
Number of Dependents	int64
Job Level	object
Company Size	object
Company Tenure	int64
Remote Work	object
Leadership Opportunities	object
Innovation Opportunities	object
Company Reputation	object
Employee Recognition	object
Attrition	object

dtype: object

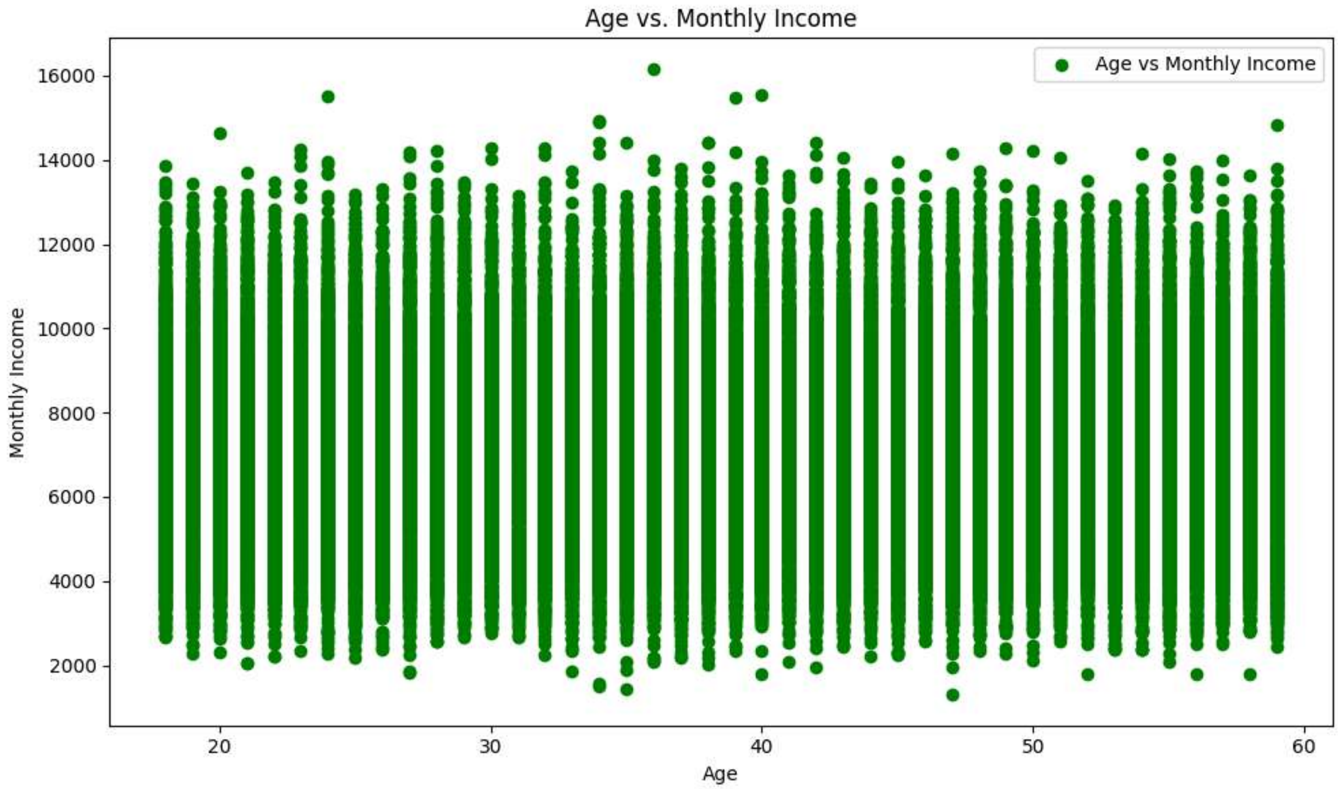
```
# 1. Line Plot - Example for Monthly Income over Years at Company
plt.figure(figsize=(10,6))
plt.plot(df['Years at Company'], df['Monthly Income'], marker='o', linestyle='-', color='blu
plt.title('Monthly Income over Years at Company')
plt.xlabel('Years at Company')
plt.ylabel('Monthly Income')
```

```
plt.grid(True)
plt.tight_layout()
plt.show()
```

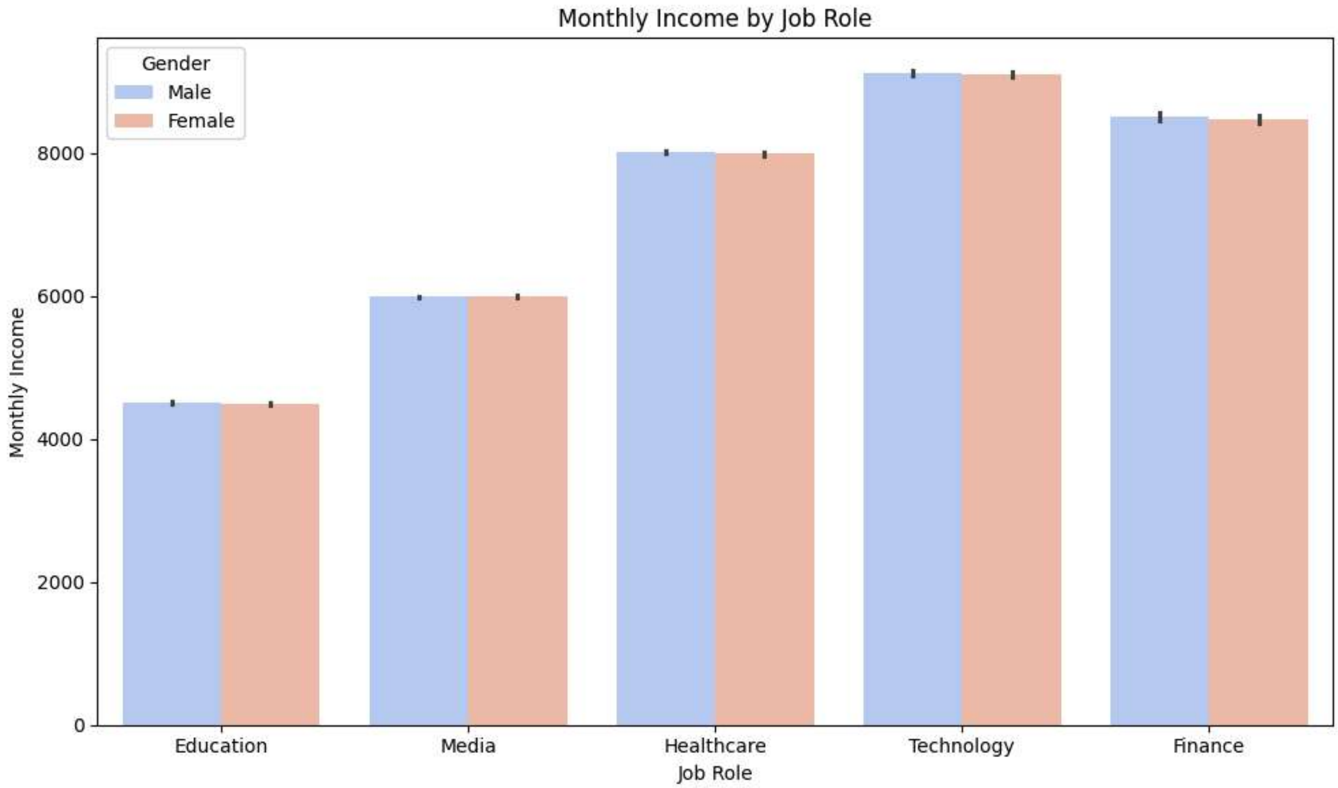


Employees with more years at the company tend to have higher salaries, but growth is not always consistent.Some employees with fewer years may earn more, possibly due to job roles, promotions, or performance

```
# 2. Scatter Plot - Age vs. Monthly Income
plt.figure(figsize=(10,6))
plt.scatter(df['Age'], df['Monthly Income'], c='green', label='Age vs Monthly Income')
plt.title('Age vs. Monthly Income')
plt.xlabel('Age')
plt.ylabel('Monthly Income')
plt.legend()
plt.tight_layout()
plt.show()
```

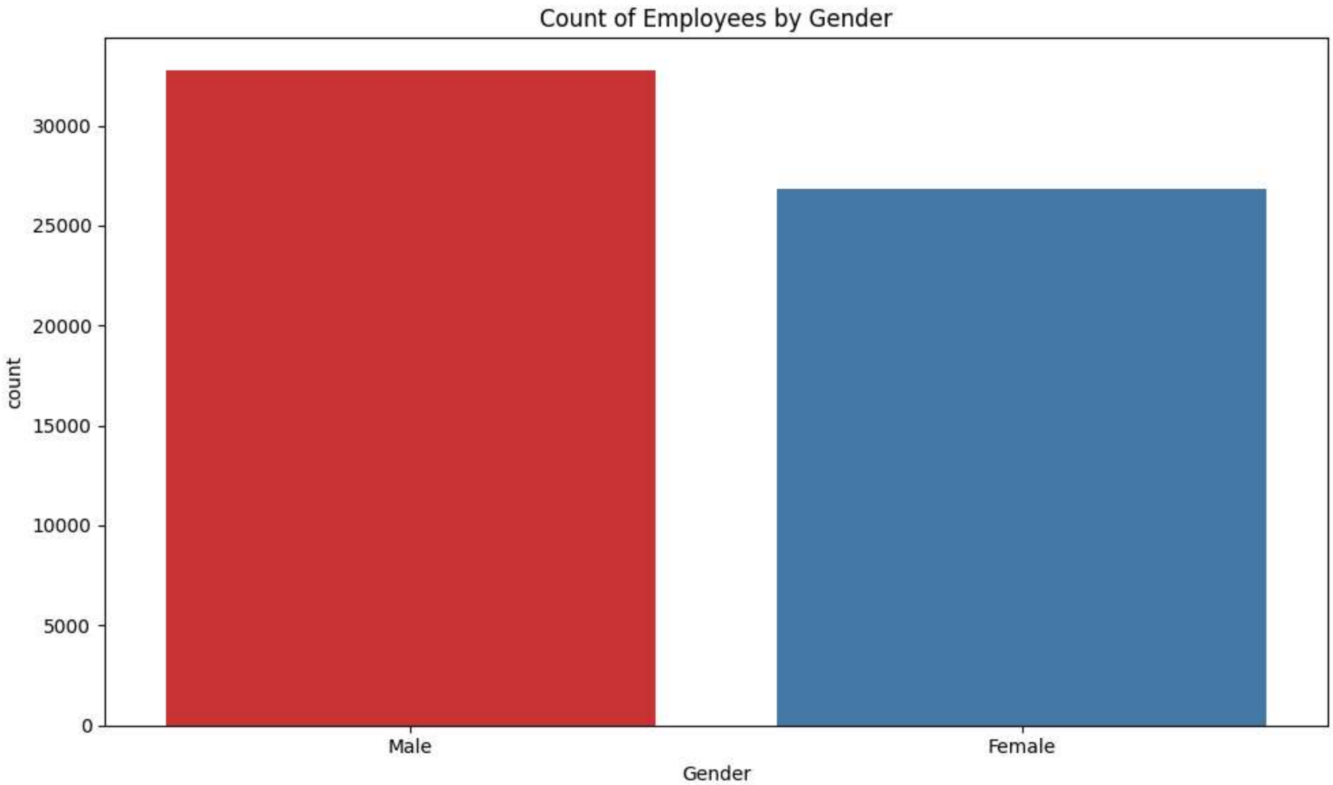



```
# 3. Bar Plot (Seaborn) - Job Role vs. Monthly Income
plt.figure(figsize=(10,6))
sns.barplot(x='Job Role', y='Monthly Income', data=df, palette='coolwarm',hue='Gender')
plt.title('Monthly Income by Job Role')
plt.tight_layout()
plt.show()
```



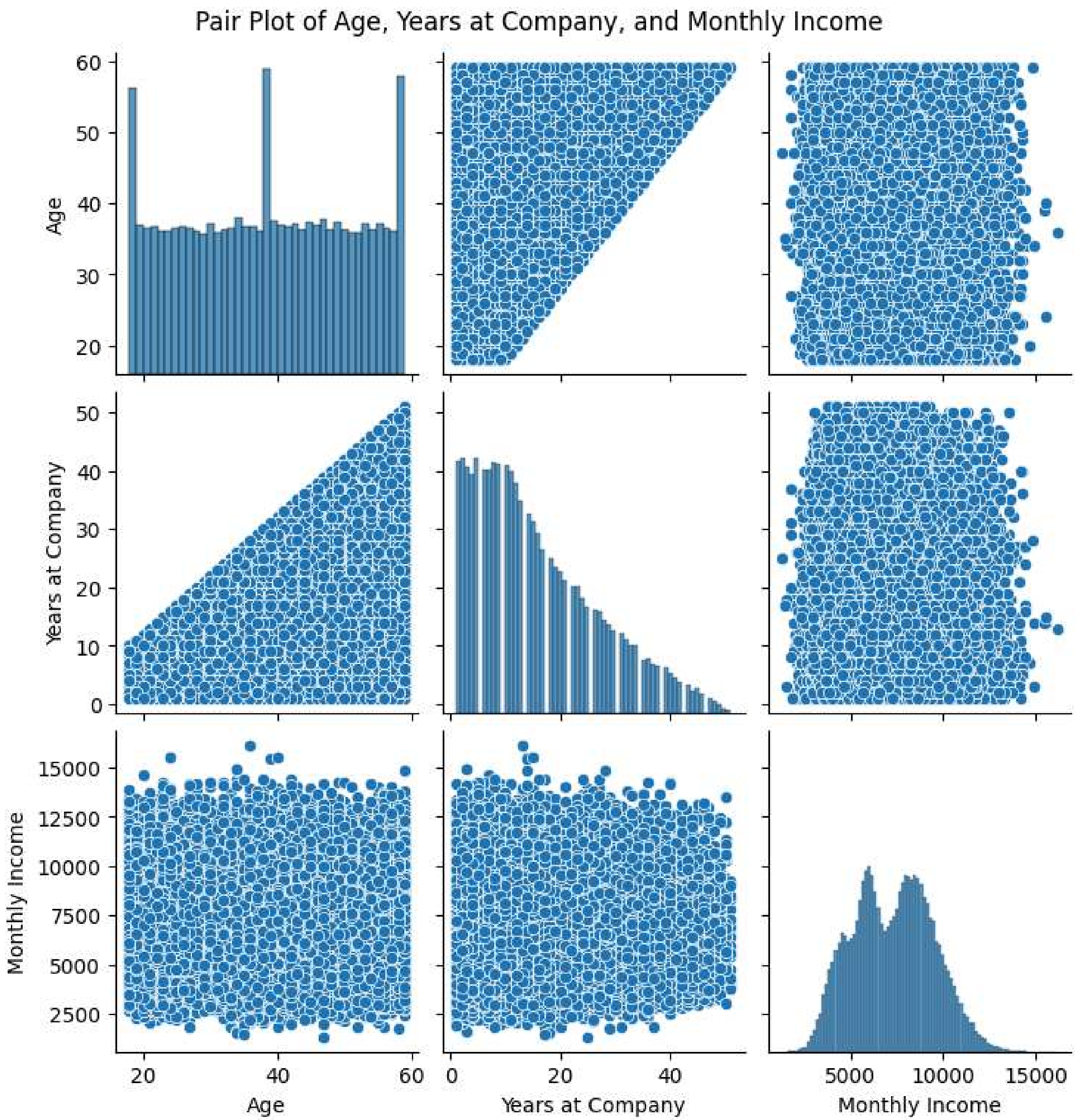
Certain job roles (e.g., Technology, Healthcare) likely have higher average salaries than others (e.g., Education, Media)

```
# 4. Count Plot (Seaborn) - Count of Employees by Gender
plt.figure(figsize=(10,6))
sns.countplot(x='Gender', data=df, palette='Set1',hue='Gender',legend=False)
plt.title('Count of Employees by Gender')
plt.tight_layout()
plt.show()
```

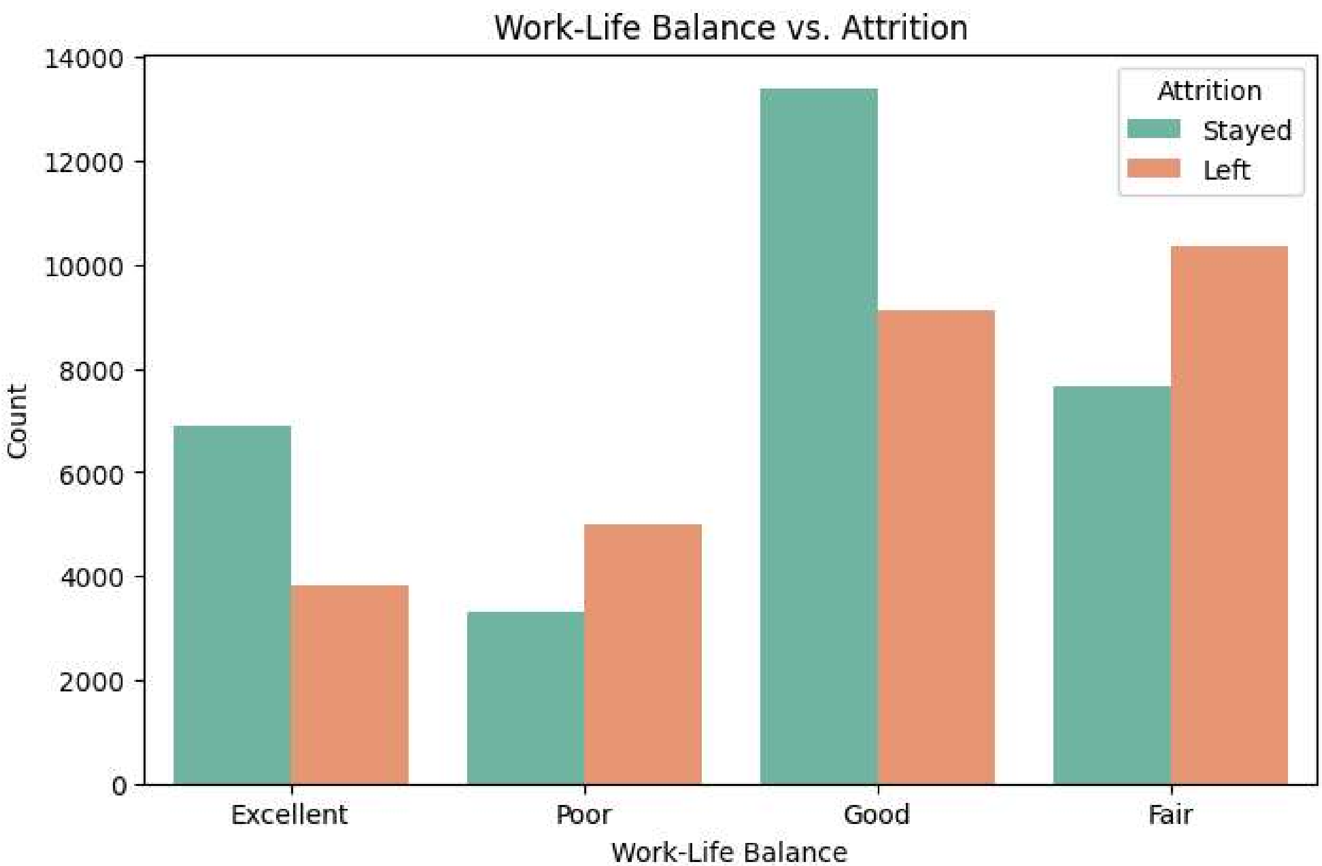


Shows the number of males and females present

```
# 5. Pair Plot - Relationship between multiple variables
sns.pairplot(df[['Age', 'Years at Company', 'Monthly Income']])
plt.suptitle('Pair Plot of Age, Years at Company, and Monthly Income', y=1.02)
plt.show()
```




```
#6.Shows if poor worklife balance leads to higher attrition
plt.figure(figsize=(8,5))
sns.countplot(x="Work-Life Balance", hue="Attrition", data=df, palette="Set2")
plt.title("Work-Life Balance vs. Attrition")
plt.xlabel("Work-Life Balance")
plt.ylabel("Count")
plt.show()
```



Drop Employee ID

```
df.drop(['Employee ID'],axis=1,inplace=True)
```

Label Encoding

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

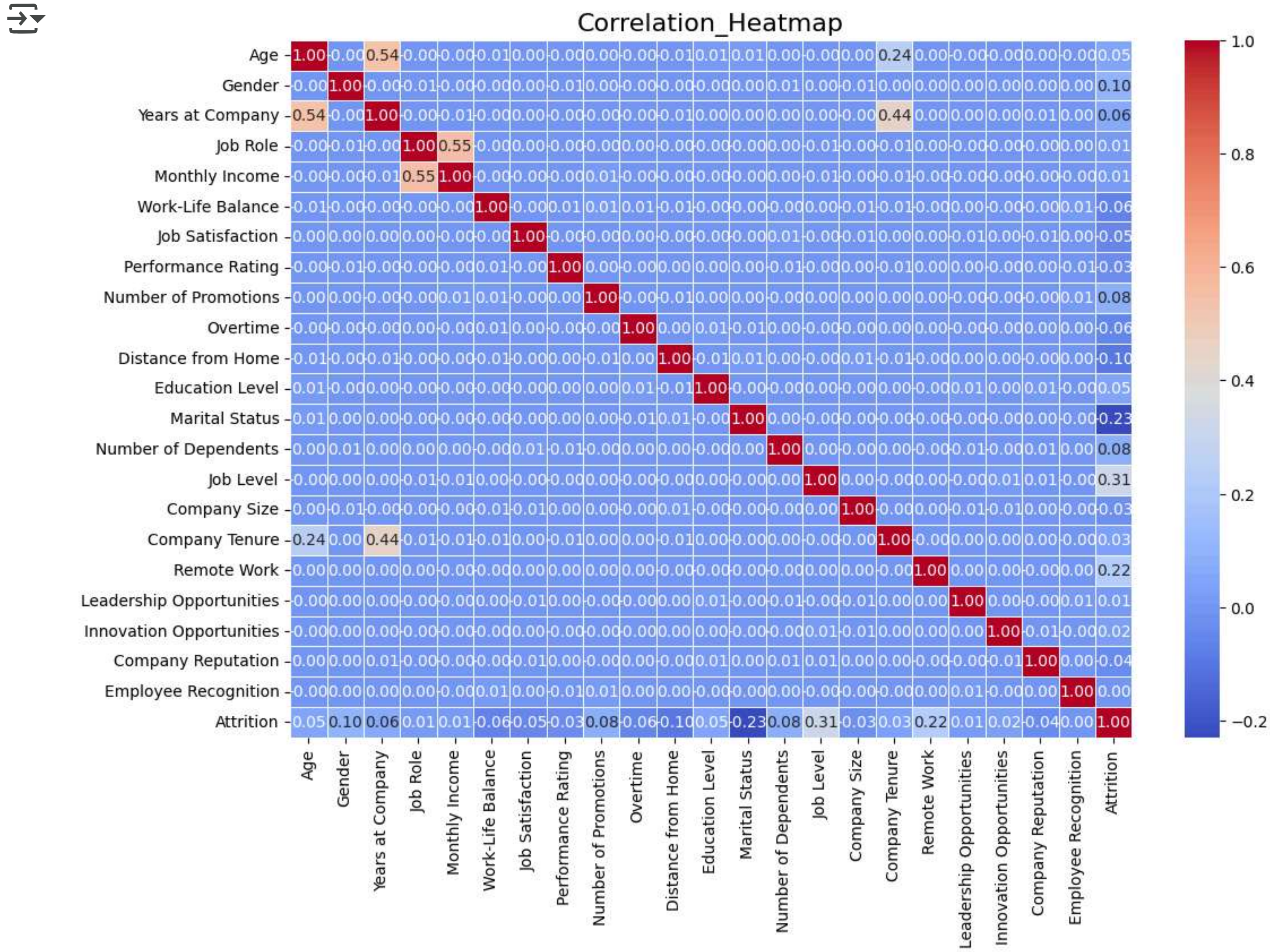
Show hidden output

```
encoder=LabelEncoder()
for col in features:
  df[col]=encoder.fit_transform(df[col])
df
```

	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotio
0	31	1	19	0	5390	0	2	0	
1	59	0	4	3	5534	3	0	3	
2	24	0	10	2	8159	2	0	3	
3	36	0	7	0	3989	2	0	2	
4	56	1	41	0	4821	1	3	0	
...	...	...	...	...	...	...	...	...	
59593	50	0	12	0	4414	1	0	0	
59594	18	1	4	2	8040	1	0	2	
59595	22	0	14	4	7944	1	0	2	
59596	23	1	8	0	2931	1	3	0	
59597	56	1	19	4	6660	2	0	0	

59598 rows × 23 columns

```
#   Compute the correlation matrix
correlation_matrix=df.corr()
 #  Create  a   heatmap to  visualize   the correlation matrix
plt.figure(figsize=(12, 8))     #   Adjust  figure  size
sns.heatmap(correlation_matrix,annot=True,  cmap='coolwarm',    fmt='.2f',  linewidths=0.5)
 #  Customize   the plot
plt.title('Correlation_Heatmap',    fontsize=16)
plt.show()
```



```
corr_matrix_unstacked = correlation_matrix.unstack()  # Convert matrix to a Series
corr_matrix_unstacked = corr_matrix_unstacked[corr_matrix_unstacked != 1]  # Remove self-cor

highest_corr = corr_matrix_unstacked.idxmax()
lowest_corr = corr_matrix_unstacked.idxmin()

print(f"Highest correlation: {highest_corr} -> {corr_matrix_unstacked.max():.4f}")
print(f"Lowest correlation: {lowest_corr} -> {corr_matrix_unstacked.min():.4f}")
```

```
Highest correlation: ('Job Role', 'Monthly Income') -> 0.5484
Lowest correlation: ('Marital Status', 'Attrition') -> -0.2296
```

```
X=df.iloc[:,:-1]           #Input
X
```

	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotio
0	31	1	19	0	5390	0	2	0	
1	59	0	4	3	5534	3	0	3	
2	24	0	10	2	8159	2	0	3	
3	36	0	7	0	3989	2	0	2	
4	56	1	41	0	4821	1	3	0	
...	...	...	...	...	...	...	...	...	
59593	50	0	12	0	4414	1	0	0	
59594	18	1	4	2	8040	1	0	2	
59595	22	0	14	4	7944	1	0	2	
59596	23	1	8	0	2931	1	3	0	
59597	56	1	19	4	6660	2	0	0	

59598 rows × 22 columns

```
y=df.iloc[:,-1]
y
```

	Attrition
0	1
1	1
2	1
3	1
4	1
...	...
59593	0
59594	0
59595	1
59596	0
59597	1

59598 rows × 1 columns

dtype: int64

Scaling

```
scaler=MinMaxScaler()
X_scaled=scaler.fit_transform(X)
```

Training and testing

```
X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=.2,random_state=42)
```

Model Training

```
knn=KNeighborsClassifier()
svc=SVC()
gnb=GaussianNB()
dt=DecisionTreeClassifier()
rf=RandomForestClassifier()
gb=GradientBoostingClassifier()
ab=AdaBoostClassifier()
xg=XGBClassifier()
models=[knn,svc,gnb,dt,rf,gb,ab,xg]
for model in models:
  print('************',model,'************')
```

```
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
print(classification_report(y_test,y_pred,digits=4))
```

```
************ KNeighborsClassifier() ************
              precision    recall  f1-score   support

           0     0.6578    0.6524    0.6551      5667
           1     0.6873    0.6925    0.6899      6253

    accuracy                         0.6734     11920
   macro avg     0.6726    0.6724    0.6725     11920
weighted avg     0.6733    0.6734    0.6733     11920

************ SVC() ************
              precision    recall  f1-score   support

           0     0.7176    0.7081    0.7129      5667
           1     0.7386    0.7475    0.7430      6253

    accuracy                         0.7288     11920
   macro avg     0.7281    0.7278    0.7279     11920
weighted avg     0.7286    0.7288    0.7287     11920

************ GaussianNB() ************
              precision    recall  f1-score   support

           0     0.6642    0.7410    0.7005      5667
           1     0.7378    0.6605    0.6970      6253

    accuracy                         0.6987     11920
   macro avg     0.7010    0.7007    0.6987     11920
weighted avg     0.7028    0.6987    0.6986     11920

************ DecisionTreeClassifier() ************
              precision    recall  f1-score   support

           0     0.6373    0.6391    0.6382      5667
           1     0.6721    0.6704    0.6713      6253

    accuracy                         0.6555     11920
   macro avg     0.6547    0.6548    0.6547     11920
weighted avg     0.6556    0.6555    0.6556     11920

************ RandomForestClassifier() ************
              precision    recall  f1-score   support

           0     0.7243    0.7277    0.7260      5667
           1     0.7522    0.7489    0.7505      6253

    accuracy                         0.7388     11920
   macro avg     0.7382    0.7383    0.7383     11920
weighted avg     0.7389    0.7388    0.7389     11920

************ GradientBoostingClassifier() ************
              precision    recall  f1-score   support
```

```
              0     0.7402    0.7392    0.7397      5667
              1     0.7639    0.7649    0.7644      6253

       accuracy                         0.7527     11920
      macro avg     0.7521    0.7521    0.7521     11920
```

```
selector = SelectKBest(score_func=f_classif, k=15)  # Select Top 10 Features using ANOVA F-t
X_selected = selector.fit_transform(X,y)
selected_features = X.columns[selector.get_support()]
selected_features
```

```
Index(['Age', 'Gender', 'Years at Company', 'Work-Life Balance',
       'Job Satisfaction', 'Performance Rating', 'Number of Promotions',
       'Overtime', 'Distance from Home', 'Education Level', 'Marital Status',
       'Number of Dependents', 'Job Level', 'Remote Work',
       'Company Reputation'],
      dtype='object')
```

```
X_new=X[selected_features]
scaler=MinMaxScaler()
X_new_scaled=scaler.fit_transform(X_new)
```

```
y.value_counts()
```

Attrition	count
1	31260
0	28338

dtype: int64

Sampling

```
from imblearn.over_sampling import SMOTE
os=SMOTE()
X_os,y_os=os.fit_resample(X_new_scaled,y)
```

```
y_os.value_counts()
```

count

Attrition

```
X_scaled=scaler.fit_transform(X_os)
```