1. Understanding the Dataset

Import Required Libraries

In [22]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style='whitegrid')
```

Load the Dataset

In [23]:
```
# Load the dataset
df = pd.read_csv(r"C:\Users\SRAVANI\Documents\datasets\WA_Fn-UseC_-HR-Employee-Attrition.csv")

df.head()
```

Out[23]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNur
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	

5 rows × 35 columns

Basic Dataset Info

In [24]:

```
# Check shape
print("Dataset Shape:", df.shape)
print()
# Data types and non-null counts
df.info()
print()
# Check for missing values
print("\nMissing values:\n", df.isnull().sum())
print()
```

```
Dataset Shape: (1470, 35)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EmployeeNumber            1470 non-null   int64
 10  EnvironmentSatisfaction   1470 non-null   int64
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64
 13  JobInvolvement            1470 non-null   int64
 14  JobLevel                  1470 non-null   int64
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64
 19  MonthlyRate               1470 non-null   int64
 20  NumCompaniesWorked        1470 non-null   int64
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64
 24  PerformanceRating         1470 non-null   int64
 25  RelationshipSatisfaction  1470 non-null   int64
 26  StandardHours             1470 non-null   int64
 27  StockOptionLevel          1470 non-null   int64
 28  TotalWorkingYears         1470 non-null   int64
 29  TrainingTimesLastYear     1470 non-null   int64
 30  WorkLifeBalance           1470 non-null   int64
 31  YearsAtCompany            1470 non-null   int64
 32  YearsInCurrentRole        1470 non-null   int64
 33  YearsSinceLastPromotion   1470 non-null   int64
```

```
 34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB


Missing values:
 Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
Over18                      0
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StandardHours               0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
```

```
dtype: int64
```

In [25]:
```
# Basic statistics
df.describe()
```

Out[25]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	HourlyRat
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.000000	1470.000000	1470.00000
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.865306	2.721769	65.89115
std	9.135373	403.509100	8.106864	1.024165	0.0	602.024335	1.093082	20.32942
min	18.000000	102.000000	1.000000	1.000000	1.0	1.000000	1.000000	30.00000
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.250000	2.000000	48.00000
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.500000	3.000000	66.00000
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.750000	4.000000	83.75000
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.000000	4.000000	100.00000

8 rows × 26 columns

Understand Target Variable (Attrition)

In [26]:
```
df['Attrition'].value_counts(normalize=True) * 100
```

Out[26]:
```
Attrition
No     83.877551
Yes    16.122449
Name: proportion, dtype: float64
```

Understand the Features

In [27]:
```
# grouping of columns by type for easier analysis:
```

```
categorical_cols = df.select_dtypes(include='object').columns.tolist()
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()

print("Categorical columns:\n", categorical_cols)

print()

print("Numerical columns:\n", numerical_cols)
```

```
Categorical columns:
 ['Attrition', 'BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus', 'Over18', 'OverTime']

Numerical columns:
 ['Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EmployeeCount', 'EmployeeNumber', 'EnvironmentSatisfaction', 'HourlyRat
e', 'JobInvolvement', 'JobLevel', 'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'PercentSalaryHike',
'PerformanceRating', 'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYe
ar', 'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager']
```

Check for Duplicate Records

In [28]:
```
print("Duplicate Rows:", df.duplicated().sum())
```

```
Duplicate Rows: 0
```

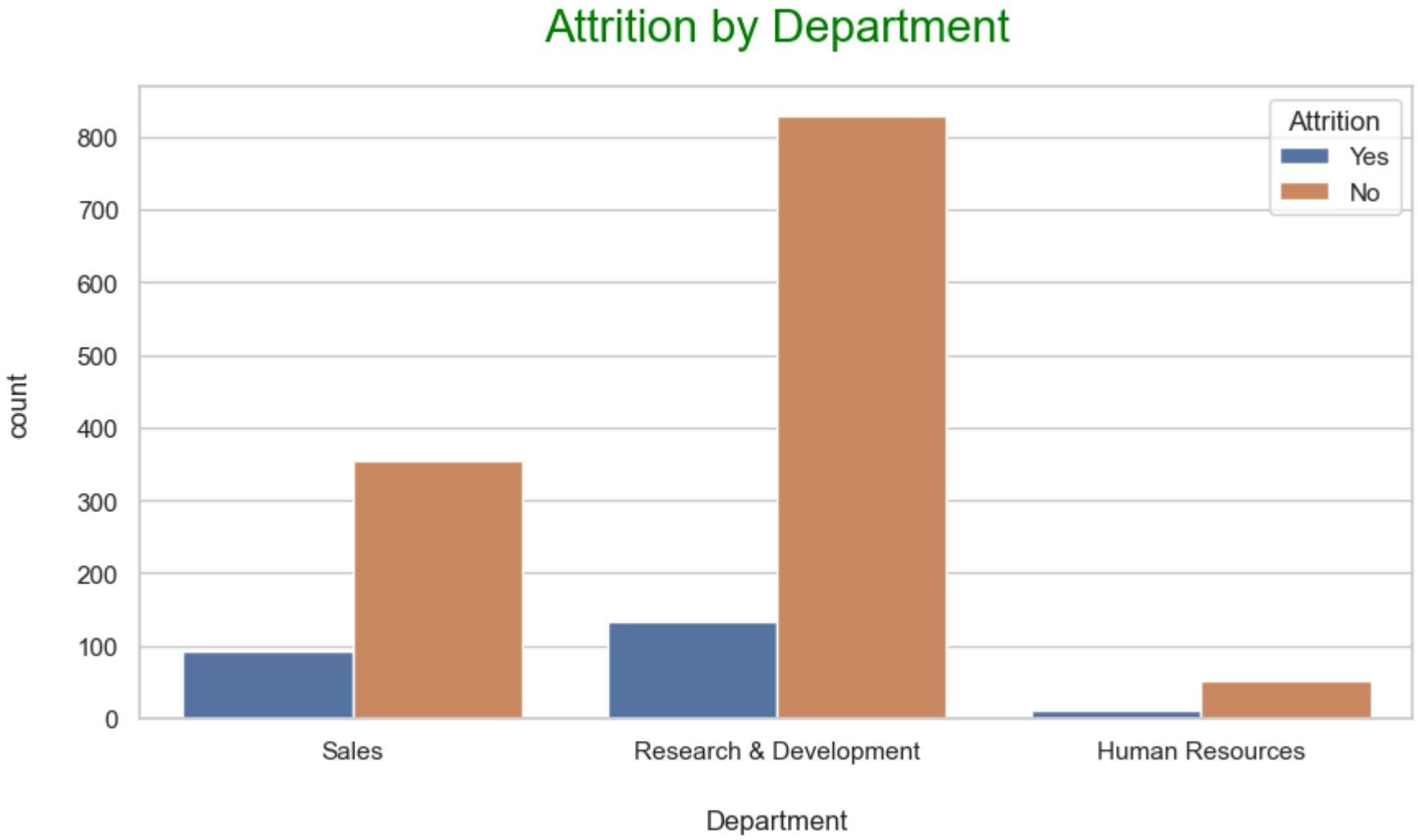
In [29]:
```
#If duplicates exist:

df.drop_duplicates(inplace=True)
```

2. Perform Exploratory Data Analysis (EDA)

Univariate Analysis (One Feature at a Time)

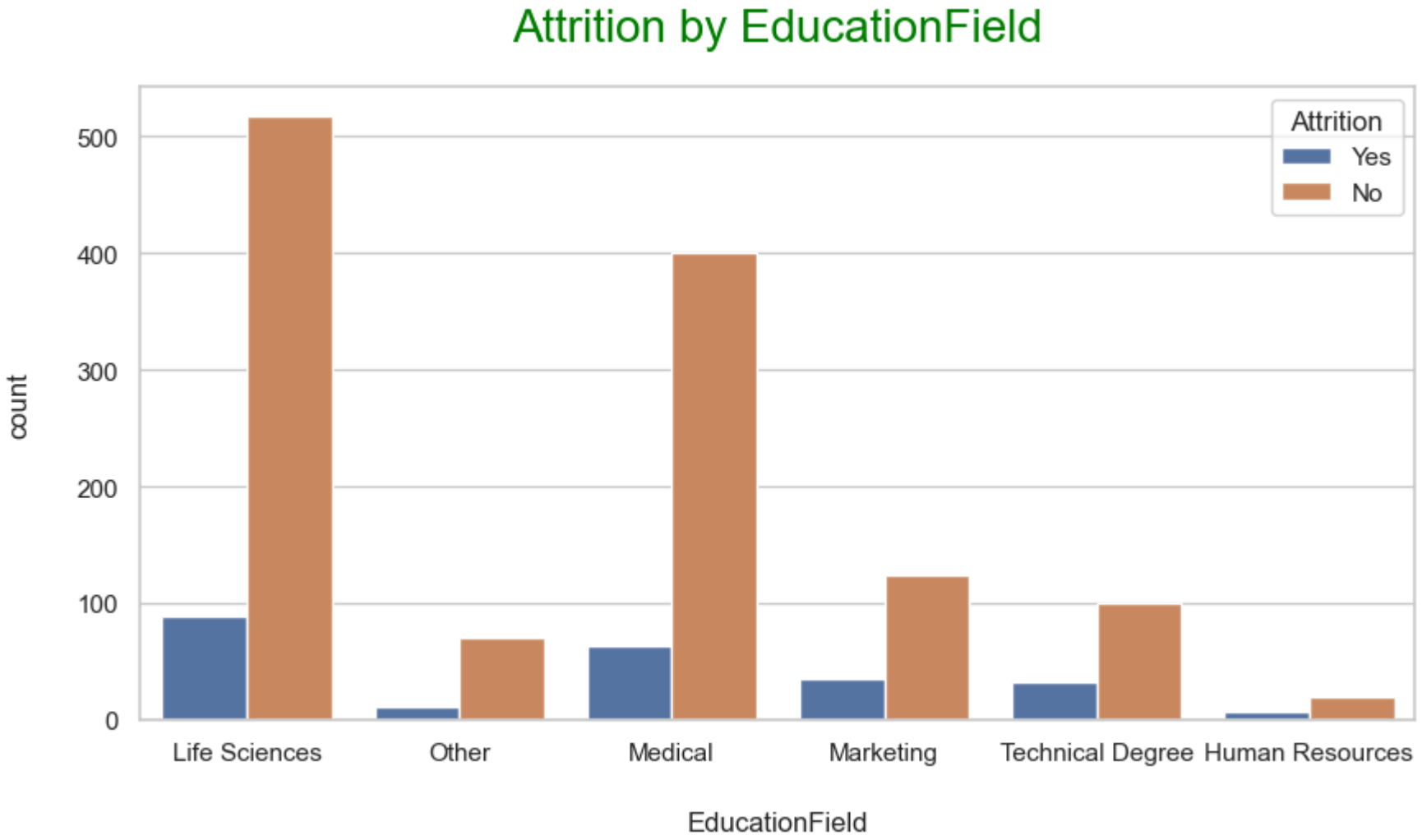
In [30]:
```
# Count Plot for Categorical Variables
plt.figure(figsize=(10,5))
sns.countplot(data=df, x='Department', hue='Attrition')
plt.title('Attrition by Department',pad=20,color="green",size=20)
plt.xlabel("Department",labelpad=20)
plt.ylabel("count",labelpad=20)
```

```
plt.xticks(rotation=0)
plt.show()
```



In [31]:
```
# Count Plot for Categorical Variables
plt.figure(figsize=(10,5))
sns.countplot(data=df, x='EducationField', hue='Attrition')
plt.title('Attrition by EducationField',pad=20,color="green",size=20)
plt.xlabel("EducationField",labelpad=20)
plt.ylabel("count",labelpad=20)
```

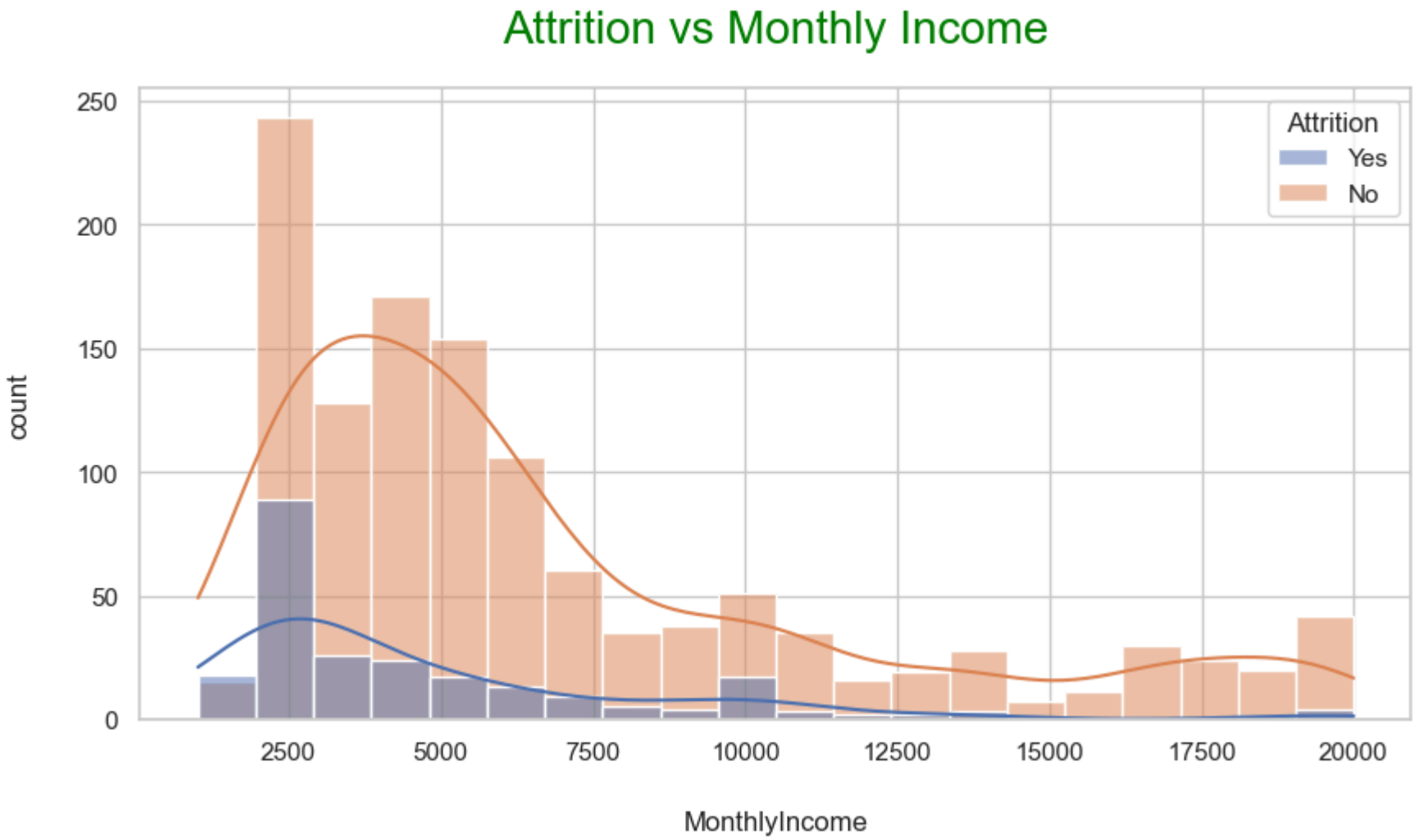
```
plt.xticks(rotation=0)
plt.show()
```



Histogram for Numeric Columns

In [32]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10, 5))
```

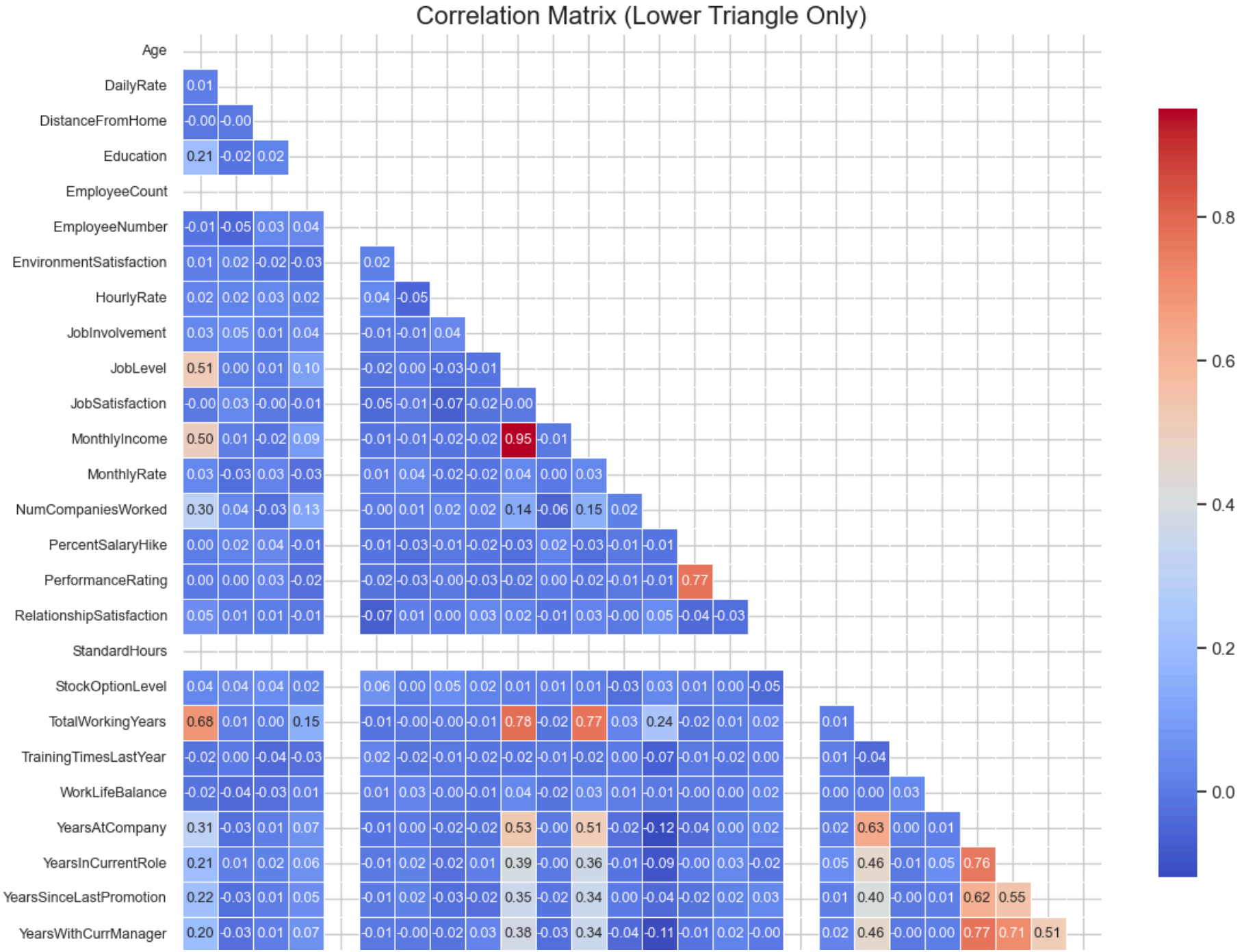
```
sns.histplot(data=df, x='MonthlyIncome', hue='Attrition', kde=True)
plt.title("Attrition vs Monthly Income",color="green",size=20,pad=20)
plt.ylabel("count",labelpad=20)
plt.xlabel("MonthlyIncome",labelpad=20)
plt.show()
```



Bivariate Analysis (Feature vs Feature)

In [33]:

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

# 1. Select only numeric columns
numeric_df = df.select_dtypes(include=['int64', 'float64'])
corr = numeric_df.corr()

# 2. Set up the figure
plt.figure(figsize=(12, 10))  # Adjust size as needed

# 3. Create a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# 4. Customize the heatmap
sns.heatmap(
    corr,
    mask=mask,
    cmap='coolwarm',
    annot=True,
    fmt=".2f",
    annot_kws={"size": 9},  # Smaller annotation font
    linewidths=0.5,         # Lines between cells
    cbar_kws={"shrink": 0.8},  # Colorbar shrink
    square=True
)

# 5. Improve titles and labels
plt.title("Correlation Matrix (Lower Triangle Only)", fontsize=16)
plt.xticks(rotation=45, ha='right', fontsize=9)
plt.yticks(rotation=0, fontsize=9)

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


Correlation Matrix (Lower Triangle Only)
Age
DailyRate
DistanceFromHome
Education
EmployeeCount
EmployeeNumber
EnvironmentSatisfaction
HourlyRate
JobInvolvement
JobLevel
JobSatisfaction
MonthlyIncome
MonthlyRate
NumCompaniesWorked
PercentSalaryHike
PerformanceRating
RelationshipSatisfaction
StandardHours
StockOptionLevel
TotalWorkingYears
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager
0.8
0.6
0.4
0.2
0.0

3.Data Preprocessing

1. Drop Unnecessary Columns

In [34]:
```
df.drop(['EmployeeCount', 'StandardHours', 'Over18', 'EmployeeNumber'], axis=1, inplace=True)
```

2. Encode the Target Column: Attrition

In [35]:
```
df['Attrition'] = df['Attrition'].map({'Yes': 1, 'No': 0})
```

3. Encode Categorical Columns

In [36]:
```
df = pd.get_dummies(df, drop_first=True)
```

4. Split Data into Features and Target

In [37]:
```
X = df.drop('Attrition', axis=1)
y = df['Attrition']
```

5. Train-Test Split

In [38]:
```
# Split into training and testing data (80%-20%).
from sklearn.model_selection import train_test_split

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

4.Build Classification Models

Logistic Regression

In [39]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# 1. Split your data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 2. Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 3. Initialize and train the model
lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train_scaled, y_train)

# 4. Predict
y_pred_lr = lr_model.predict(X_test_scaled)

# 5. Evaluate
print(" 🔹 Logistic Regression")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_lr))
print("Classification Report:\n", classification_report(y_test, y_pred_lr))
```

```
 🔹 Logistic Regression
Accuracy: 0.8809523809523809
Confusion Matrix:
 [[241  14]
 [ 21  18]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.95      0.93       255
           1       0.56      0.46      0.51        39

    accuracy                           0.88       294
   macro avg       0.74      0.70      0.72       294
weighted avg       0.87      0.88      0.88       294
```

Train Decision Tree and Random Forest Models

Train a Decision Tree Classifier

In [40]:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_scaled, y_train)

y_pred_dt = dt_model.predict(X_test_scaled)

print(" 🔹 Decision Tree Classifier")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
print("Classification Report:\n", classification_report(y_test, y_pred_dt))
```

```
 🔹 Decision Tree Classifier
Accuracy: 0.7551020408163265
Confusion Matrix:
 [[215  40]
 [ 32   7]]
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.84      0.86       255
           1       0.15      0.18      0.16        39

    accuracy                           0.76       294
   macro avg       0.51      0.51      0.51       294
weighted avg       0.77      0.76      0.76       294
```

Train a Random Forest Classifier

In [41]:
```python
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

y_pred_rf = rf_model.predict(X_test_scaled)

print(" 🔹 Random Forest Classifier")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
```

```
🔹 Random Forest Classifier
Accuracy: 0.8775510204081632
Confusion Matrix:
 [[254   1]
 [ 35   4]]
Classification Report:
              precision    recall  f1-score   support

           0       0.88      1.00      0.93       255
           1       0.80      0.10      0.18        39

    accuracy                           0.88       294
   macro avg       0.84      0.55      0.56       294
weighted avg       0.87      0.88      0.83       294
```

In [42]:
```
df.to_excel("cleaned_hr_data.xlsx", index=False)
```