Kaggle Competition Writeup

CompSci 671

Due: Dec 9th 2022
Kaggle Competition Link

1 Exploratory Analysis

How did you make sense of the provided dataset and get an idea of what might work? Did you use histograms, scatter plots or some sort of clustering algorithm? Did you do any feature engineering? Describe your thought process in detail for how you approached the problem and if you did any feature engineering in order to get the most out of the data.

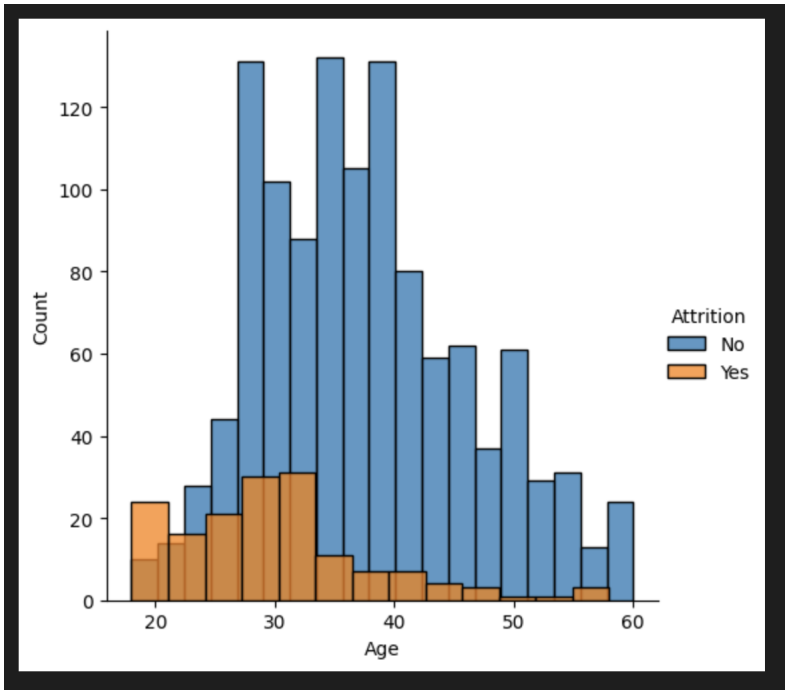
1.1 Initial Exploration:

The first thing I did once I loaded in the data as a pandas DataFrame was look at the general info and description. From the info, I can see the type of data in each feature as well as the number of non-null values. For all 34 features (not including the label column, Attrition), there are no non-null features. This means I won't have to eliminate any rows or columns due to null values! From the description, I was able to take a cursory glance at each feature and gain a coarse understanding from the mean, standard deviation, quartiles, and outliers.

I saw that some features, like EmployeeCount, only had one unique value. As this value was the same for each data point, I dropped this column as it provides no additional information. I was able to gain a sense of what the values were under each feature and what their spread was like.

To gain a deeper understanding of the data, I made a histogram of each feature using seaborn. The distribution of values was overlayed by the distribution of labels, so I could quickly see if certain values in features would be helpful in building a classifier. Below

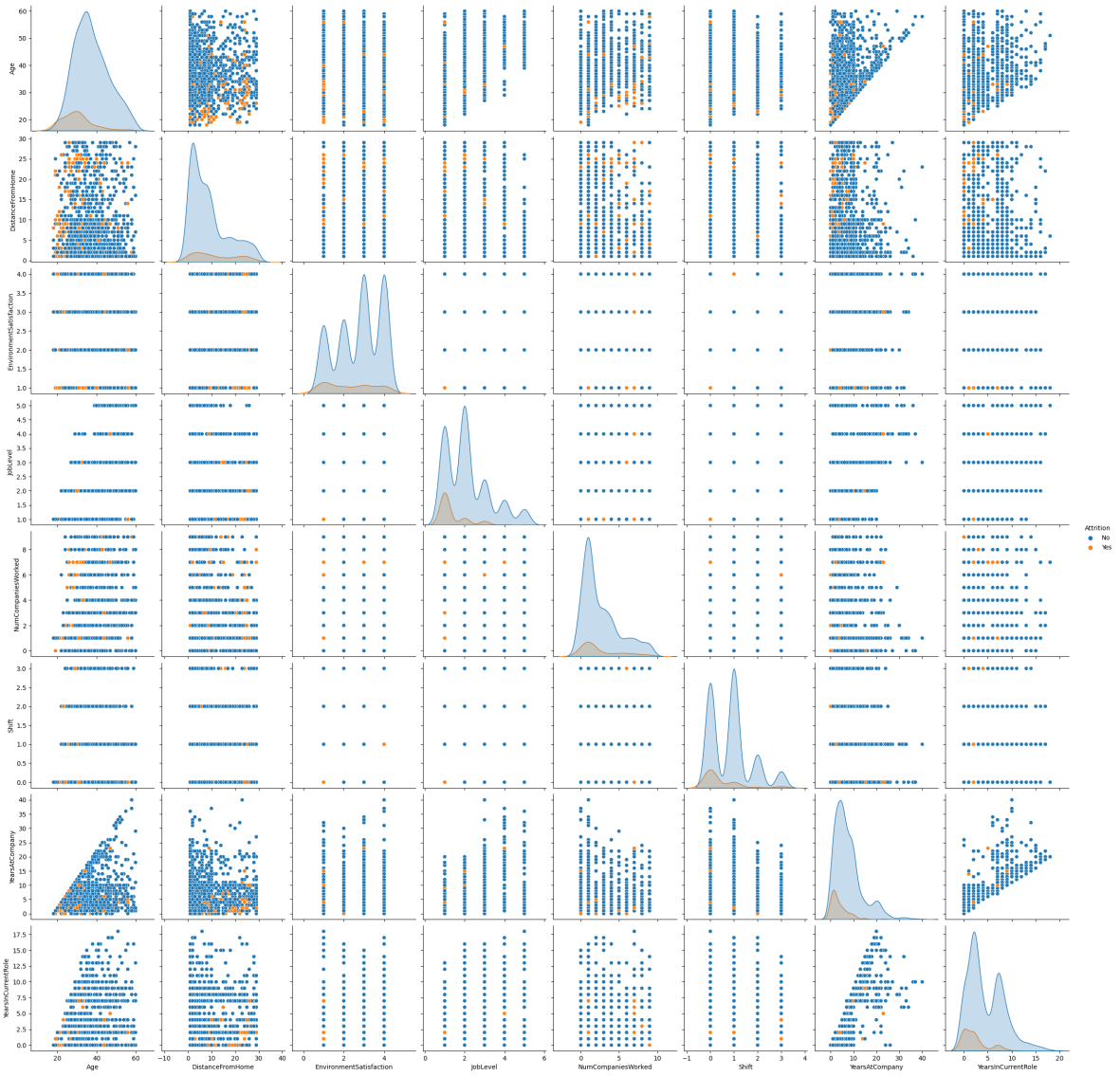
is an example, with all the plots in the code included in the code section of the report. [https://seaborn.pydata.org/]



From this step, I was able to see some features with decent separation or skewness for the labels. These were features such as Age, DistanceFromHome, and JobLevel. I took note of 11 of these promising features and set the columns aside for further analysis.

I also realized that the distribution of Attrition labels was very imbalanced, with there being almost 6x more No's than Yes's. This will be important when choosing a models, as well as for the training, tuning, and evaluation processes.

Next, I generated a pairwise scatter plot for each pair of these promising features to see if I could see a good separation of the labels (as seen below). Sadly, I didn't see anything amazing.



1.2 Feature Engineering

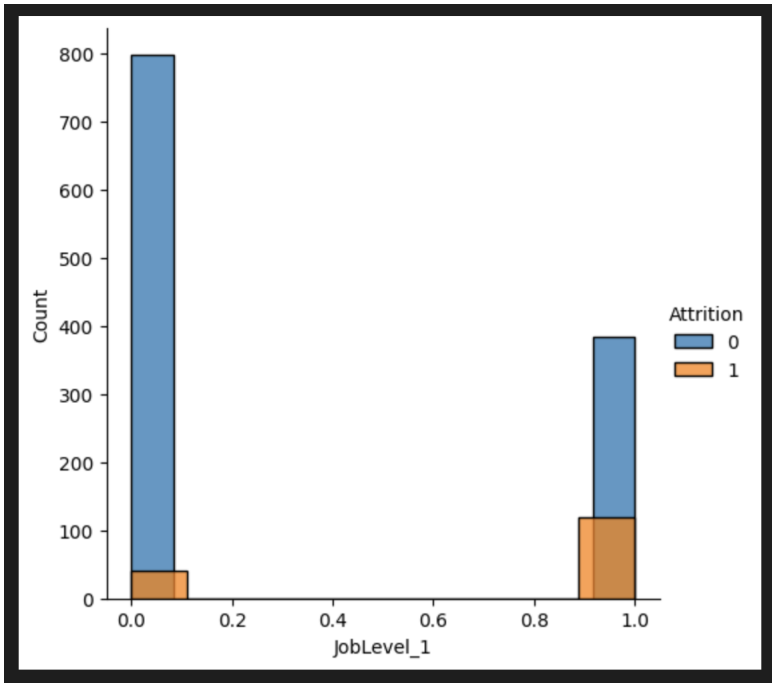
Now that I had found some promising features to work with, it was time to try and see if I could create some new features that could provide my models with features highly correlated to Attrition.

First, I found 13 features with numeric data, such as HourlyRate, which had many unique values over a large range. To simplify the information given from these columns, I created a new columns that binned the data by quartile. Instead of the large range of values, each value was assigned a number 0 through 3.

Next, I made dummy variables from all of the features with categorical data. For exam-

ple, three new columns (NonTravel, TravelRarely, TravelFrequently) were created from the BusinessTravel column, each only containing values of 0 or 1. This could simplify some of the relationships, such as if Attrition was highly correlated with only those who travelled frequently. I also factorized the strings into numbers to make the values easier to work with for the models.

I then made a histogram for each new feature created the same way as done above, where the distribution of values was overlayed by the distribution of labels. I saw some potentially useful features, such as the one shown below.



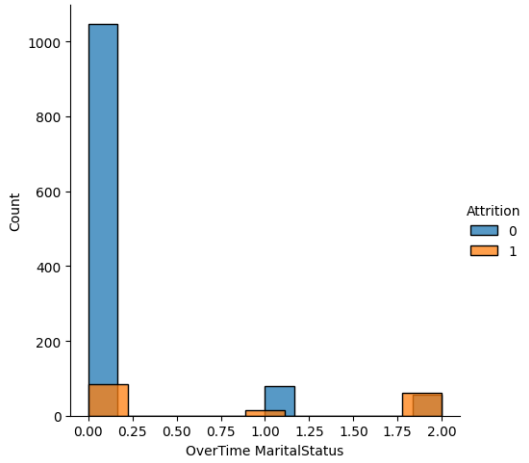
As I now had 100+ features, many with redundant information, I needed a way to select ones most correlated with Attrition, while also not giving repeated information. I used sklearn's cross-validated Recursive Feature Elimination with Logistic Regression to choose the most useful features. RFE trains a model, Logistic Regression in my case, and then drops the features with the lowest assigned weights. After many rounds, it returns the most promising features. [https://scikit-learn.org/stable/index.html]

Now that I had promising features, I needed to remove the features that gave redundant information, features that were highly correlated with one another. For this, I used minimum Redundancy - Maximum Relevance. Essentially, the mRMR selects features based on correlation to the target, Attrition, and for not being correlated to other features in the dataset. [https://github.com/smazzanti/mrmr]

Now, I had a nice set of features chosen by correlation to the labels and for unique

information. I wanted to try and see if I could engineer some other features that could also be helpful. I used sklearn's PolynomialFeatures to generate new features from the interactions with other features. For example, generating a new feature based on Age x OverTime. Even though I didn't see any clear standouts from the pairwise plot made earlier, that doesn't mean that there couldn't be any useful new features generate this way. As this process essentially squares the number of features in the dataset, I would have to repeat the feature selection step using mRMR, as this algorithm also has the advantage of being fast.

To check the usefulness of the features selected from this process, I generated histograms using the same method as before. It appears that some of the features selected were good at identifying the edge cases where Attrition was true, such as in the example below. This is exactly what I was looking for as the labels are imbalanced, with few true Attrition labels - making correctly classifying those Yes's very important for the F1 score. However, a concern is that the features selected are overfit to the training data.



2 Models

You are required to use at least two different algorithms for generating predictions. These do not need to be algorithms we used in class. It would not be acceptable to use the same algorithm but with two different parameters or kernels. In this section, you will explain your reasoning behind the choice of algorithms. Specific motivations for choosing a certain algorithm may include computational efficiency, simple parameterization, ease of use, ease of training, or the availability of high-quality libraries online, among many other possible factors. If external libraries were used, describe them and identify the source or authors of the code (make sure to cite all references and figures that you use if someone else designed them). Try to be adventurous!

When considering which algorithms to use, the main priority was choosing an algorithm that was good at handling imbalanced data. Other important considerations were well-documented and widely used libraries to make troubleshooting much easier, as well as lightweight models that are quick to train so that hyperparameter tuning will be less computationally expensive.

The first algorithm I chose was AdaBoost. AdaBoost is an ensemble method, meaning that it combines multiple weak learners (usually Decision Trees) to create a - hopefully - strong learner. AdaBoost trains the weak learners sequentially, where the weights assigned to training data are updated each step based on how difficult they are to classify. This makes AdaBoost particularly useful for imbalanced classification problems. AdaBoost is also widely used and quick to train, checking off all of my considerations. [https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html]

The second algorithm I chose was LightGBM. LightGBM another ensemble method that usually uses Decision Trees and trains its weak learners sequentially. It uses the gradient of the loss function from the previous learner to help minimize the loss of the next learner. LightGBM is known for running extremely quickly, making it very easy to tune. It is also widely used in data competitions. [https://lightgbm.readthedocs.io/en/latest/index.html]

3 Training

Here, for each of the algorithms used, briefly describe (5-6 sentences) the training algorithm used to optimize the parameter settings for that model. For example, if you used a support vector regression approach, you would probably need to reference the quadratic solver that works under-the-hood to fit the model. You may need to read the documentation for the code libraries you use to determine how the model is fit. This is part of the applied machine learning process! Also, provide estimates of runtime (either wall time or CPU time) required to train your model.

AdaBoost stands for Adaptive Boosting. The key to the algorithm are the weights assigned to each data point. Lets assume that the weak learner being used are decision trees with depth=1. First, weights are initialized to $\frac{1}{N}$. Then, based on how many features are available for consideration and how many data points are available to evaluate over for that iteration, a feature is chosen to split over. The feature is chosen by choosing the feature with the largest information gained, usually using the Gini Index metric.

Then, the total error is calculated for this weak learner, and based on its performance, it is assigned a higher or lower weight on classification importance. This importance score is then used to update the weights on each data point, making misclassfied points more important and correctly classified points less important. This process is done repeatedly, with the next weak learner using the re-weighted points to make a new decision tree. The end model is a weighted sum of classifications over all weak learners, weighted by their importance score. AdaBoost takes about 0.1 seconds to train on my personal computer with default parameters. [https://www.researchgate.net/publication/321583409_AdaBoost_typical_Algorithm_and_its_application_research]

LightGBM is a gradient boosting ensemble method. Lets assume that the weak learner being used are decision trees. The special thing with LightGBM is that it grows leaf wise. It uses a histogram bucketing method where data is binned using the data's distribution. Unlike with AdaBoost which only uses a single point, the bins are used to calculate the information gain for each feature. It does this for the leaf with the largest loss, and then chooses a new split to lower the loss. Other unique features of LightGBM is Gradient-based One-Side Sampling (GOSS). GOSS excludes low gradient data points, meaning that data points with low contribution to calculating information gain can be excluded. This speeds up computation time. LightGBM take about 0.5 seconds to train on my personal computer with default parameters. [https://dl.acm.org/doi/10.5555/3294996.3295074]

4 Hyperparameter Selection

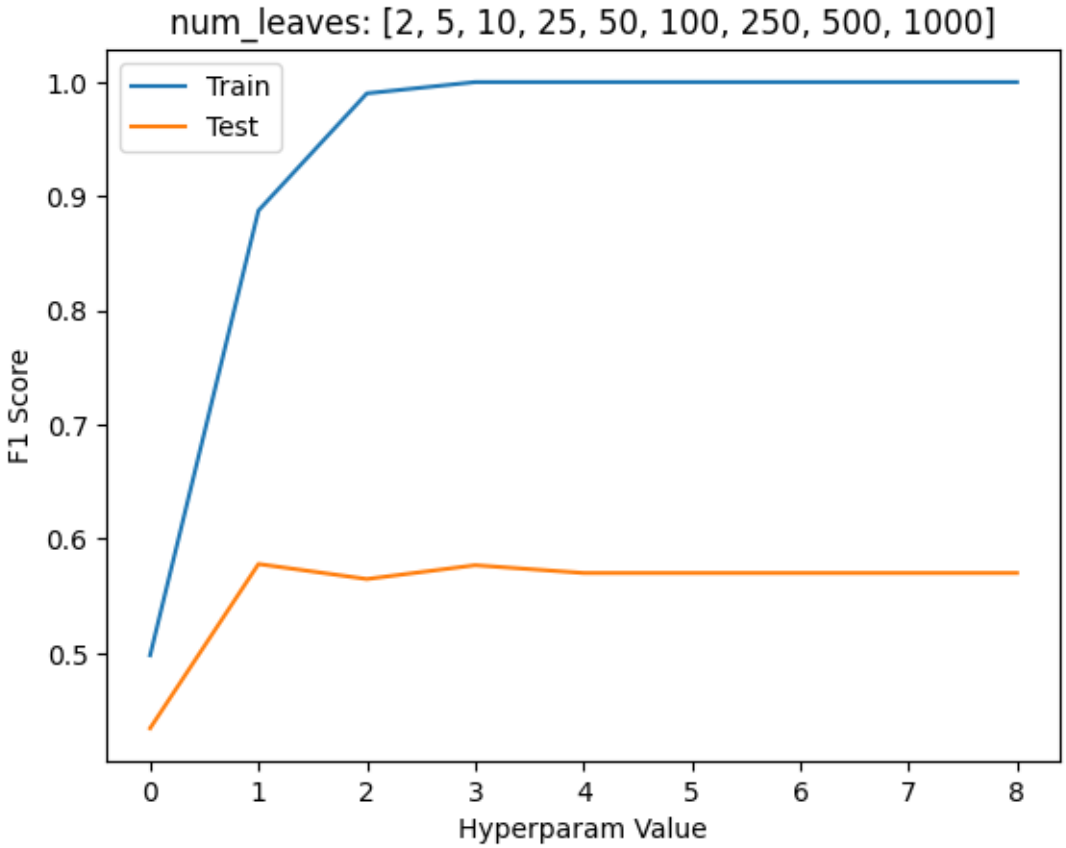
You also need to explain how the model hyperparameters were tuned to achieve some degree of optimality. Examples of what we consider hyperparameters are the number of trees used in a random forest model, the regularization parameter for LASSO or the type of activation / number of neurons in a neural network model. These must be chosen according to some search or heuristic. It would not be acceptable to pick a single setting of your hyperparameters and not tune them further. You also need to make at least one plot showing the functional relation between predictive accuracy on some subset of the training data and a varying hyperparameter.

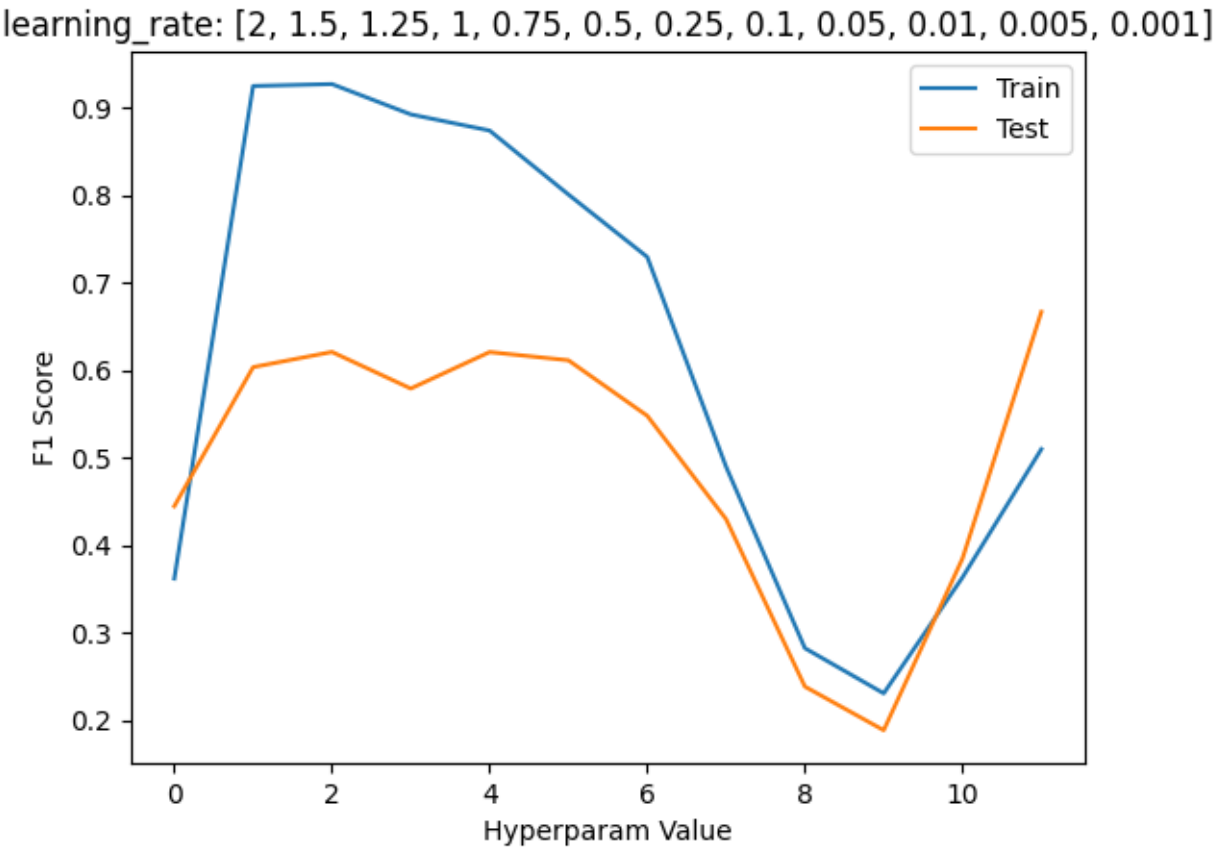
For AdaBoost, the hyperparameters being optimized were the number of estimators, the learning rate, and the maximum depth of the base estimator (decision trees).

For LightGBM, the hyperparamters being optimized were the number of estimators, the learning rate, the number of leaves, the maximum depth of the base estimator (decision

trees), as well as starting data weights based on the labels.

For both algorithms, I tuned the hyperparameters one at a time rather than a grid search. This saved a lot of computation time. For each new hyperparameter being tested, I trained the model with that new value on 0.8 of the training set. I then tested its performance by getting its predictions on 0.2 of the training set, the calculating the F1 score with the true labels. I saved the F1 scores on the training and test sets to choose the optimal values while accounting for overfitting, as well as generate the plot, two of which are seen below. The rest are included in the code section.





5 Data Splits

Finally, we need to know how you split up the training data provided for cross validation. Again, briefly describe your scheme for making sure that you did not overfit to the training data.

First, I used sklearn's test-train-split to split the training data into 0.8 training data, 0.2 test data. I also set stratify to true, which ensures that the new training and test datasets have the same proportion of true labels. This is important for imbalanced datasets and ensures that the data we are training and testing on has a representative distribution so that it can generalize well. The test data was only used for checking scores, not for training. This prevents data leakage.

When tuning hyperparameters, I used sklearn's StratifiedKFold for cross validation. This split the train data into a smaller training set (0.8) and a validation set (0.2). After K folds, I took the mean of the validation F1 scores. This ensured that the parameters being chosen were good for many different subsets of the data. Once again, by reserving the test data for

checking, it protects against overfitting to the dataset.

Additionally, I utilized upsampling the dataset. This means that I resampled positive labelled data points, as these were rarer in the data. This allows the models to perform better on these rarer labels, which is important for F1 performance.

6 Errors and Mistakes

Making missteps is a natural part of the process. If there were any steps or bugs that really slowed your progress, put them here! What was the hardest part of this competition?

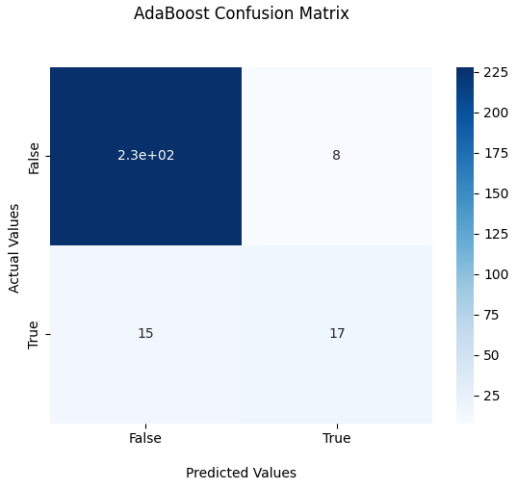
I made a lot of mistakes during the duration of this project! The first was not spending enough time at the data exploration step. I rushed through it, excited to get to the modelling. I ended up paying for this with low performance with my modelling, and having to go back to do the step more throughly. The next was how I was generating and selecting my features. First, I was only selecting too few, meaning that I wasn't feeding my models enough information. When evaluating which features to choose, I first used a linear kernel SVM rather than a simple logistic regression model for RFE. This ran for more than 14 hours without completing, as it was going over the polynomial features as well. I also originally did GridSearch for hyperparameter tuning, which took 10+ hours without completing. This led to my more simplified approach after this combinatorial explosion. At one point, I kept getting 1.0 on train and test according to F1 score, which was because I ran the python notebook out of order and accidentally included the labels in the training data! I also ran into a lot of overfitting problems, especially with LightGBM.

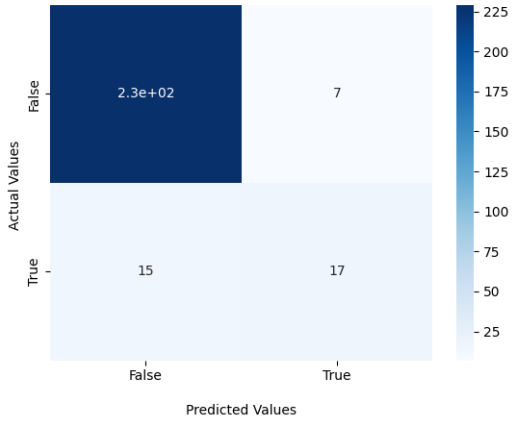
7 Predictive Accuracy

Upload the submissions from your best model to Kaggle and put your Kaggle username in this section so we can verify that you uploaded something. Also, compare the effectiveness of the models that you used via the F1 score that we are using to evaluate you on the Kaggle site. Half (10) of the points from this section will be awarded based on your performance relative to your peers. Scoring in the 90th percentile and above will give 10 points while being in the bottom 10 percent will give one point. Note that it is possible to do very poorly in the competition but still get an A on this assignment if the other sections are filled out satisfactorily. This encourages you to take risks! Use plots or other diagrams to visually

represent the accuracy of your model and the predictions it makes.

The python notebooks used for my AdaBoost and LightGBM submissions are included in the code section below! To visualize the accuracy of my model, I created a confusion matrix heatmap for each model on the test data set.





8 Code

Copy and paste your code into your write-up document. Also, attach all the code needed for your competition. The code should be commented so that the grader can understand what is going on. Points will be taken off if if the code or the comments do not explain what is taking place. Your code should be executable with the installed libraries and only minor modifications.

In [ ]:
```
### Imports
import mrmr
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
### Import data
data = pd.read_csv("train.csv")

# Get general info
print(data.info(), "\n\n\n")
print(data.describe(), "\n\n\n")
```

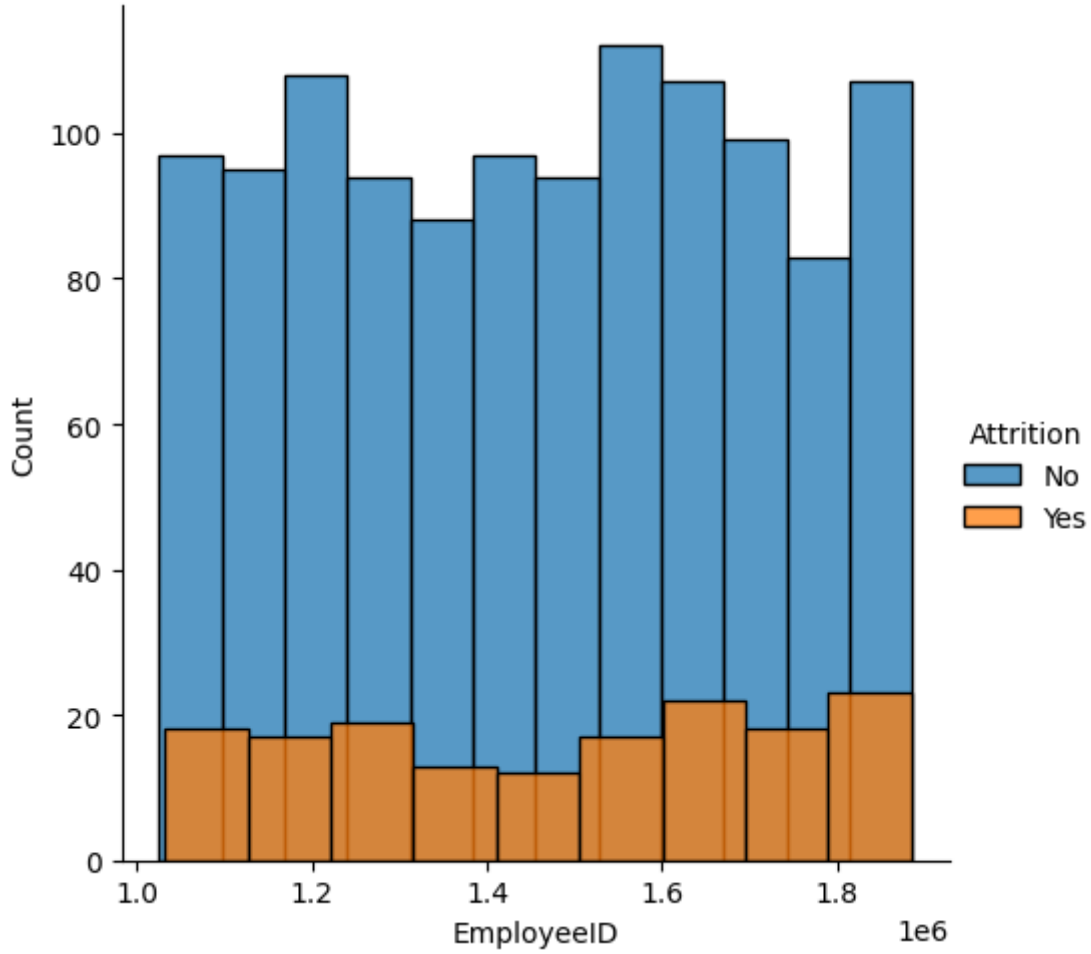
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1340 entries, 0 to 1339
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   EmployeeID                1340 non-null   int64 
 1   Age                       1340 non-null   int64 
 2   Attrition                 1340 non-null   object
 3   BusinessTravel            1340 non-null   object
 4   DailyRate                 1340 non-null   int64 
 5   Department                1340 non-null   object
 6   DistanceFromHome          1340 non-null   int64 
 7   Education                 1340 non-null   int64 
 8   EducationField            1340 non-null   object
 9   EmployeeCount             1340 non-null   int64 
 10  EnvironmentSatisfaction   1340 non-null   int64 
 11  Gender                    1340 non-null   object
 12  HourlyRate                1340 non-null   int64 
 13  JobInvolvement            1340 non-null   int64 
 14  JobLevel                  1340 non-null   int64 
 15  JobRole                   1340 non-null   object
 16  JobSatisfaction           1340 non-null   int64 
 17  MaritalStatus             1340 non-null   object
 18  MonthlyIncome             1340 non-null   int64 
 19  MonthlyRate               1340 non-null   int64 
 20  NumCompaniesWorked        1340 non-null   int64 
 21  Over18                    1340 non-null   object
 22  OverTime                  1340 non-null   object
 23  PercentSalaryHike         1340 non-null   int64 
 24  PerformanceRating         1340 non-null   int64 
 25  RelationshipSatisfaction  1340 non-null   int64 
 26  StandardHours             1340 non-null   int64 
 27  Shift                     1340 non-null   int64 
 28  TotalWorkingYears         1340 non-null   int64 
 29  TrainingTimesLastYear     1340 non-null   int64 
 30  WorkLifeBalance           1340 non-null   int64 
 31  YearsAtCompany            1340 non-null   int64 
 32  YearsInCurrentRole        1340 non-null   int64 
 33  YearsSinceLastPromotion   1340 non-null   int64 
 34  YearsWithCurrManager      1340 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 366.5+ KB
None
```

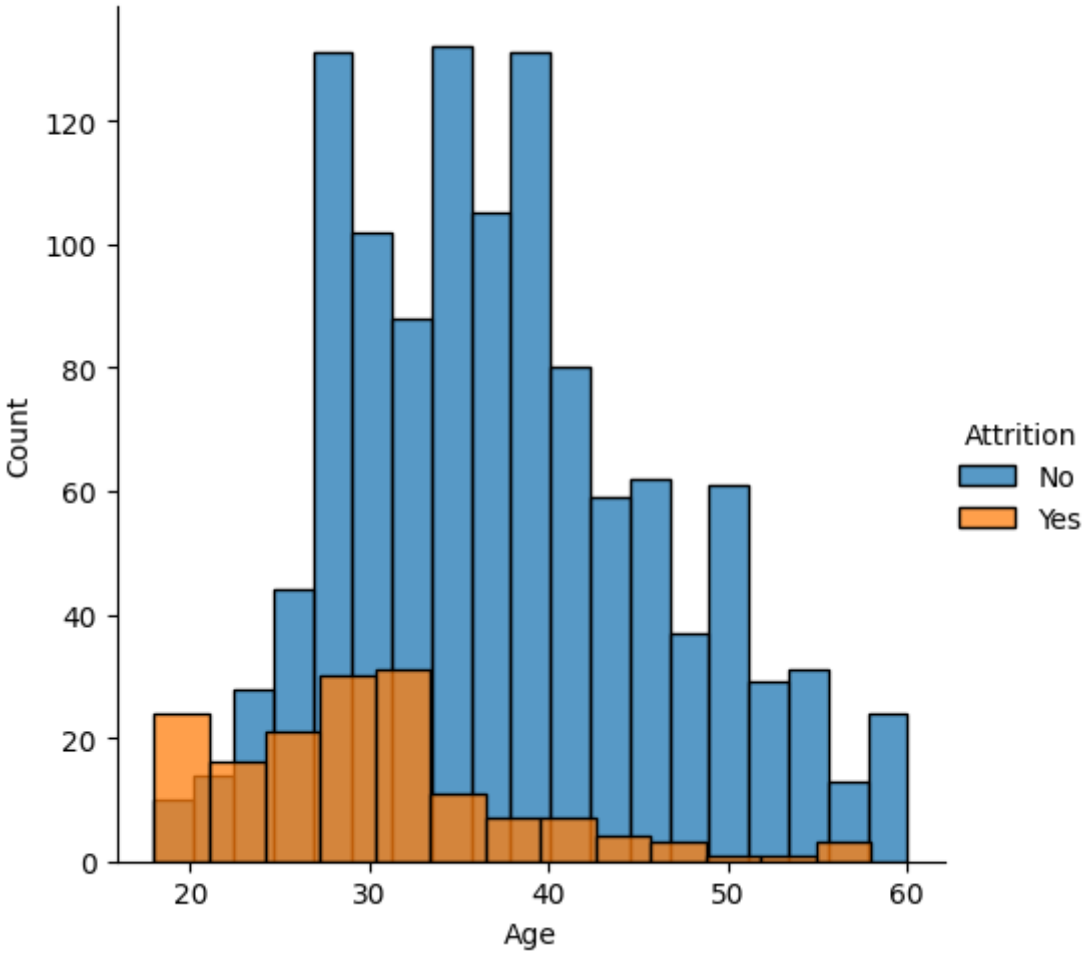
```
         EmployeeID          Age    DailyRate  DistanceFromHome    Educatio
n  \
count  1.340000e+03  1340.000000  1340.000000       1340.000000  1340.00000
0
mean   1.460265e+06    36.580597   799.197761          9.193284     2.92462
7
std    2.494821e+05     9.013072   399.333256          8.141621     1.03608
8
min    1.025177e+06    18.000000   102.000000          1.000000     1.00000
0
```

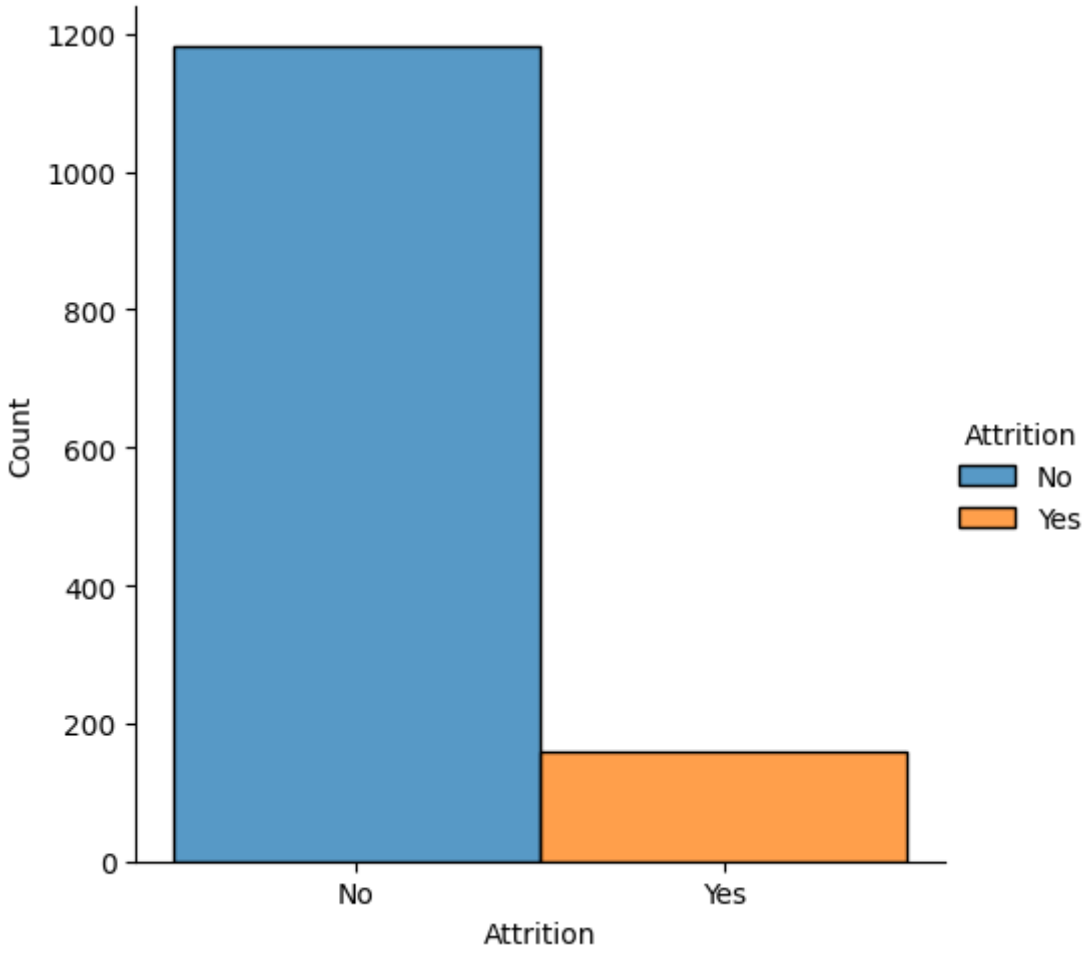
```
25%    1.237599e+06    30.000000   465.000000          2.000000     2.00000
0
50%    1.469862e+06    35.000000   796.000000          7.000000     3.00000
0
75%    1.670131e+06    42.000000  1153.000000         14.000000     4.00000
0
max    1.886378e+06    60.000000  1499.000000         29.000000     5.00000
0

       EmployeeCount  EnvironmentSatisfaction   HourlyRate  JobInvolvement
\
count         1340.0              1340.000000  1340.000000     1340.000000
mean             1.0                 2.709701    65.559701        2.717910
std              0.0                 1.099961    20.335025        0.717523
min              1.0                 1.000000    30.000000        1.000000
25%              1.0                 2.000000    48.000000        2.000000
50%              1.0                 3.000000    65.000000        3.000000
75%              1.0                 4.000000    83.000000        3.000000
max              1.0                 4.000000   100.000000        4.000000

          JobLevel  ...  RelationshipSatisfaction  StandardHours        Shi
ft  \
count  1340.000000  ...               1340.000000         1340.0  1340.0000
00
mean      2.051493  ...                  2.700000           80.0     0.8082
09
std       1.104491  ...                  1.079858            0.0     0.8562
51
min       1.000000  ...                  1.000000           80.0     0.0000
00
25%       1.000000  ...                  2.000000           80.0     0.0000
00
50%       2.000000  ...                  3.000000           80.0     1.0000
00
75%       3.000000  ...                  4.000000           80.0     1.0000
00
max       5.000000  ...                  4.000000           80.0     3.0000
00

       TotalWorkingYears  TrainingTimesLastYear  WorkLifeBalance  \
count        1340.000000            1340.000000      1340.000000
mean           11.222388               2.785821         2.771642
std             7.696043               1.263473         0.700007
min             0.000000               0.000000         1.000000
25%             6.000000               2.000000         2.000000
50%            10.000000               3.000000         3.000000
75%            15.000000               3.000000         3.000000
max            40.000000               6.000000         4.000000

       YearsAtCompany  YearsInCurrentRole  YearsSinceLastPromotion  \
count     1340.000000         1340.000000              1340.000000
mean         7.070149            4.272388                 2.175373
std          6.039663            3.677798                 3.222376
min          0.000000            0.000000                 0.000000
25%          3.000000            2.000000                 0.000000
50%          5.000000            3.000000                 1.000000
```

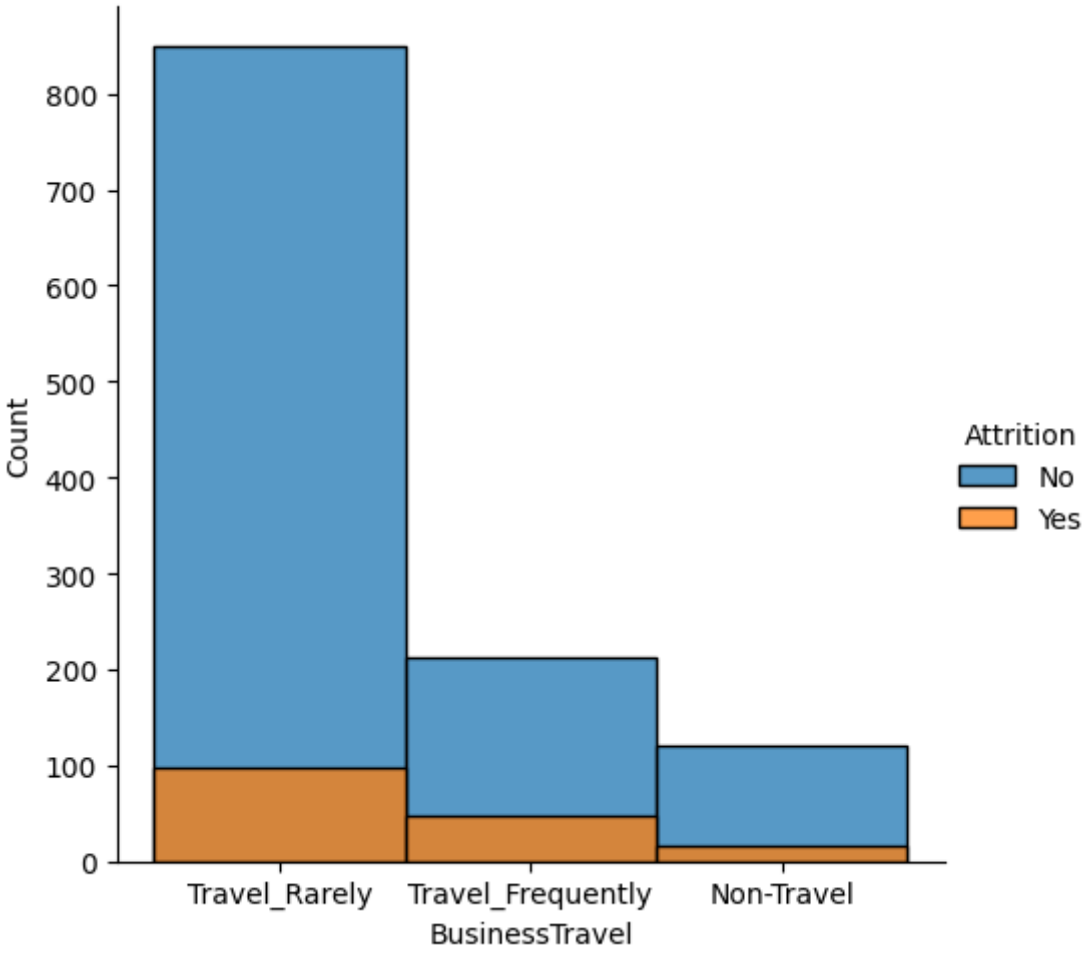
```
75%          10.000000             7.000000                3.000000
max          40.000000            18.000000               15.000000

       YearsWithCurrManager
count           1340.000000
mean               4.167164
std                3.581605
min                0.000000
25%                2.000000
50%                3.000000
75%                7.000000
max               17.000000

[8 rows x 26 columns]
```

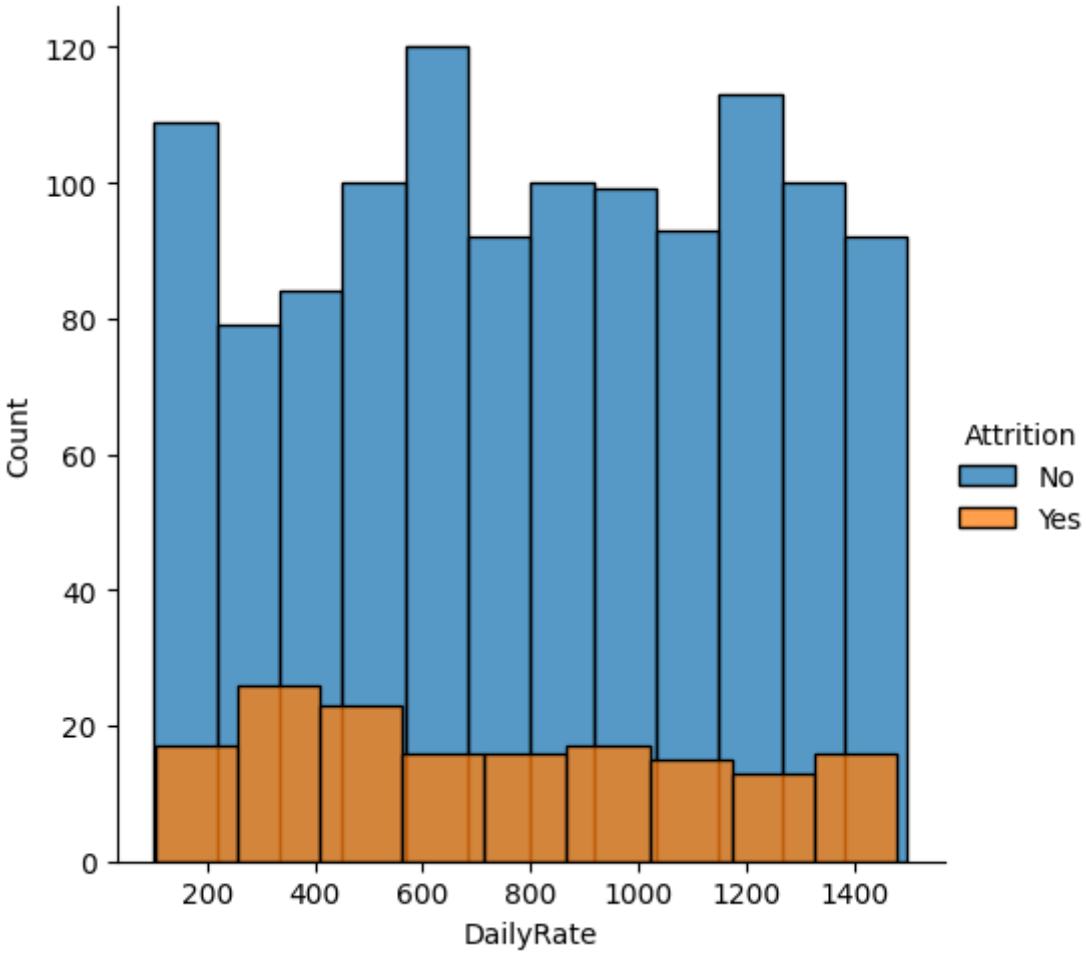
In [ ]:
```
### Initial dataset visualization with histograms
for c in data.columns:
    sns.FacetGrid(data,
                  hue="Attrition",
                  height= 5).map(sns.histplot,c).add_legend()
```

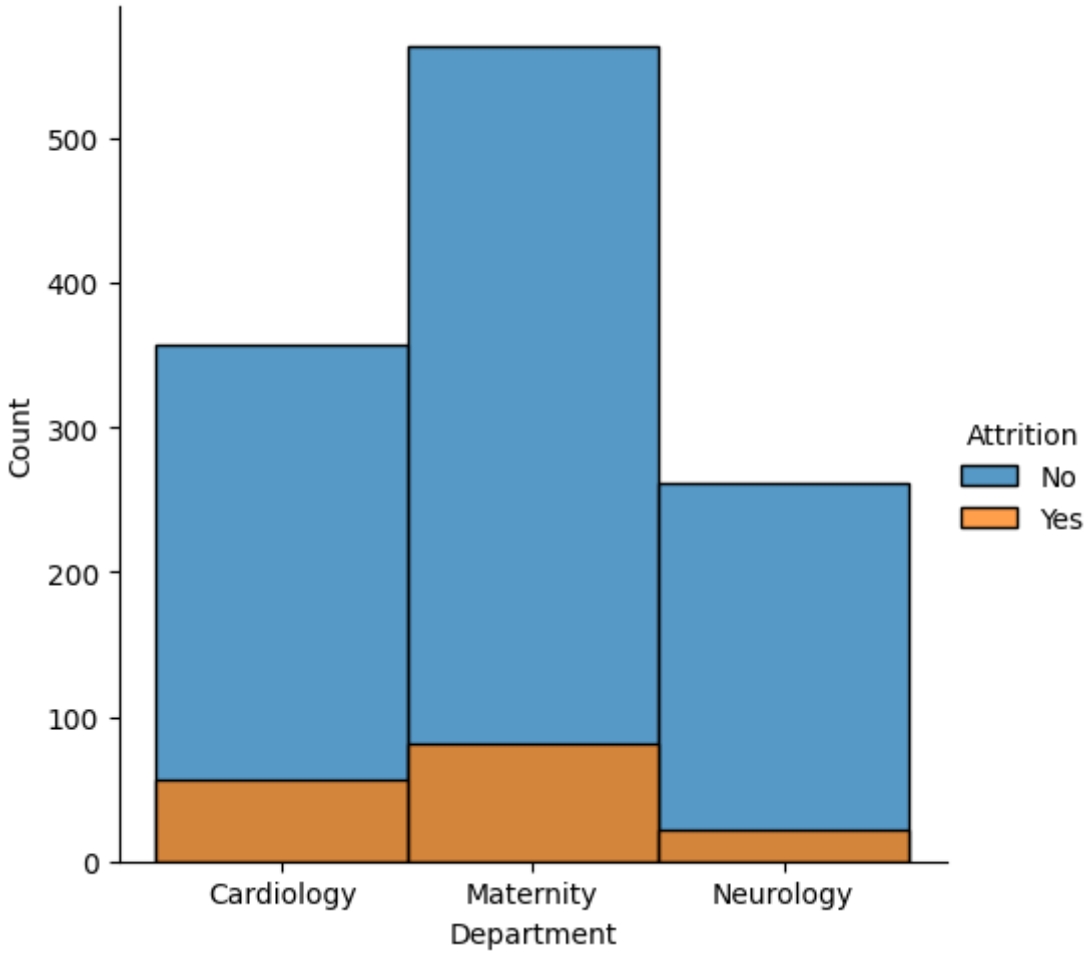


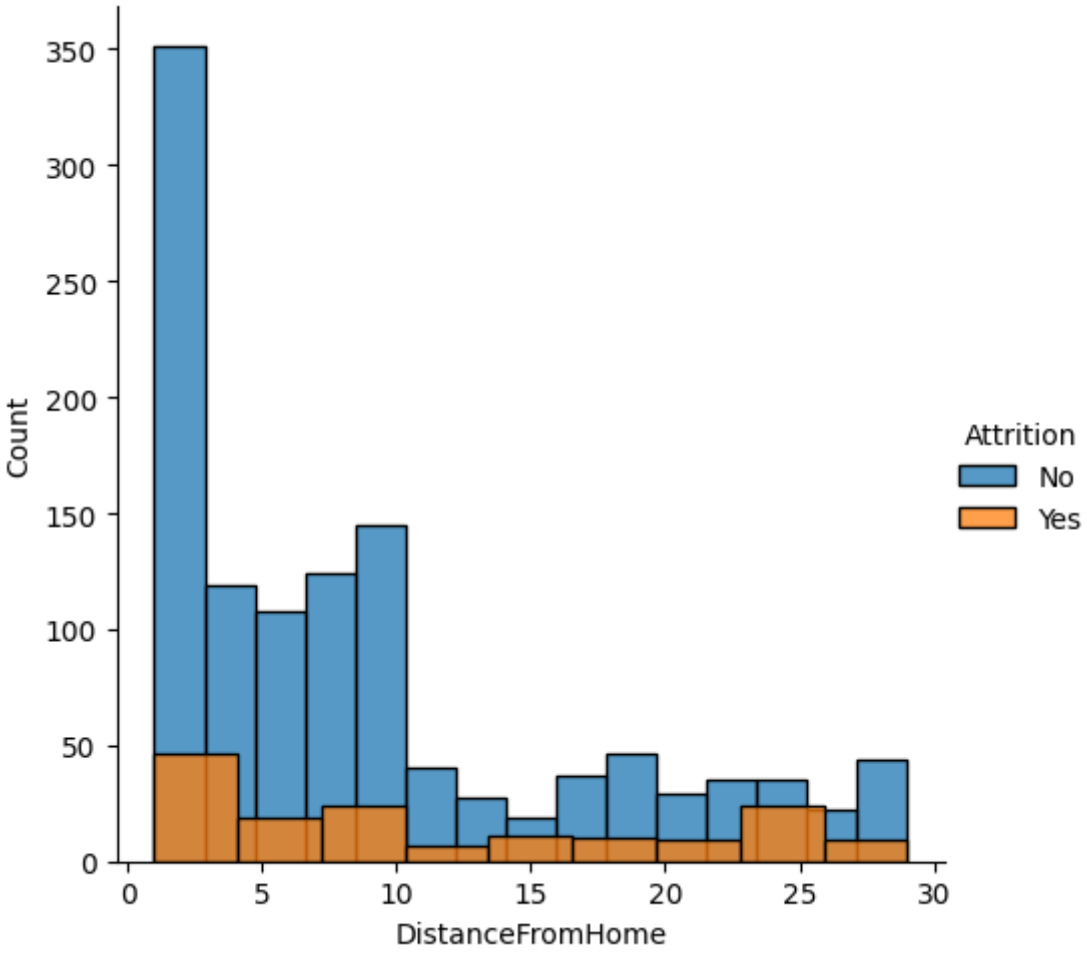

Count
120
100
80
60
40
20
0
20
30
40
50
60
Age
Attrition
No
Yes

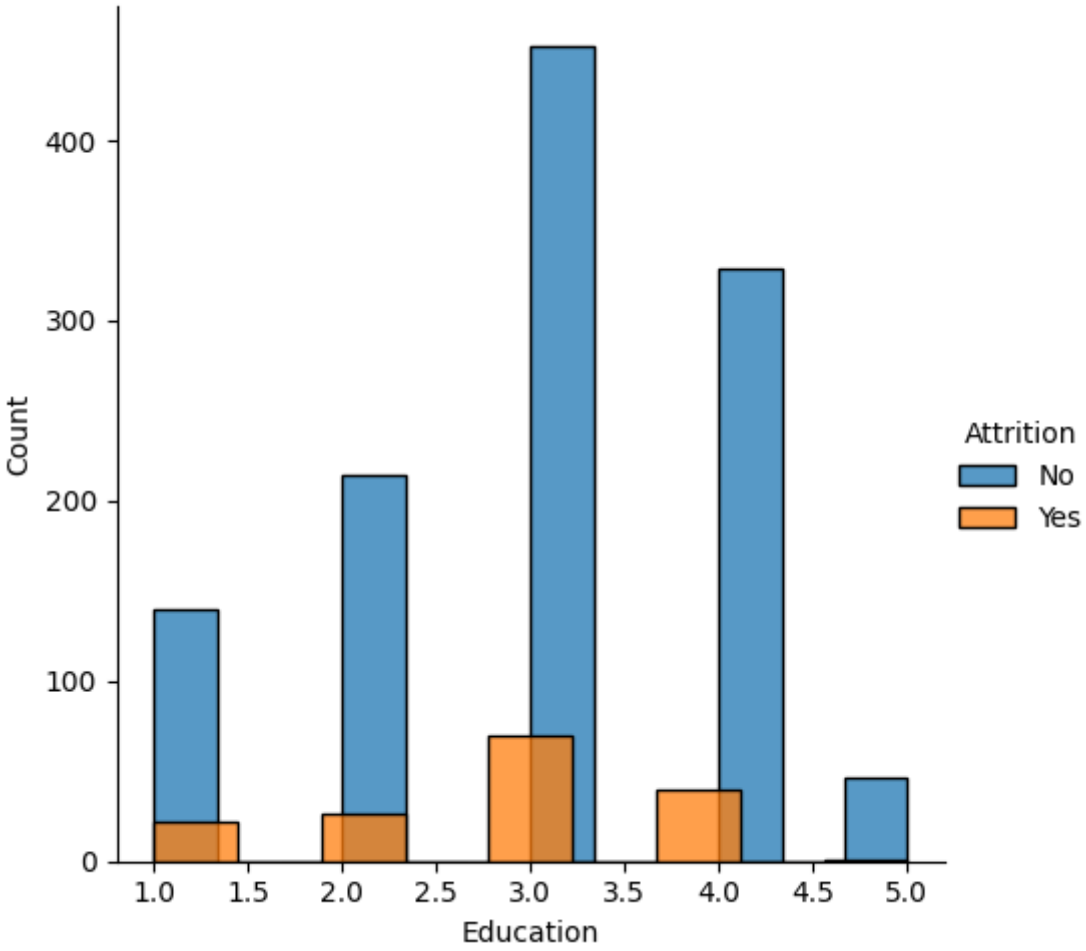

1200
1000
800
600
400
200
0
Count
No
Yes
Attrition
Attrition
No
Yes

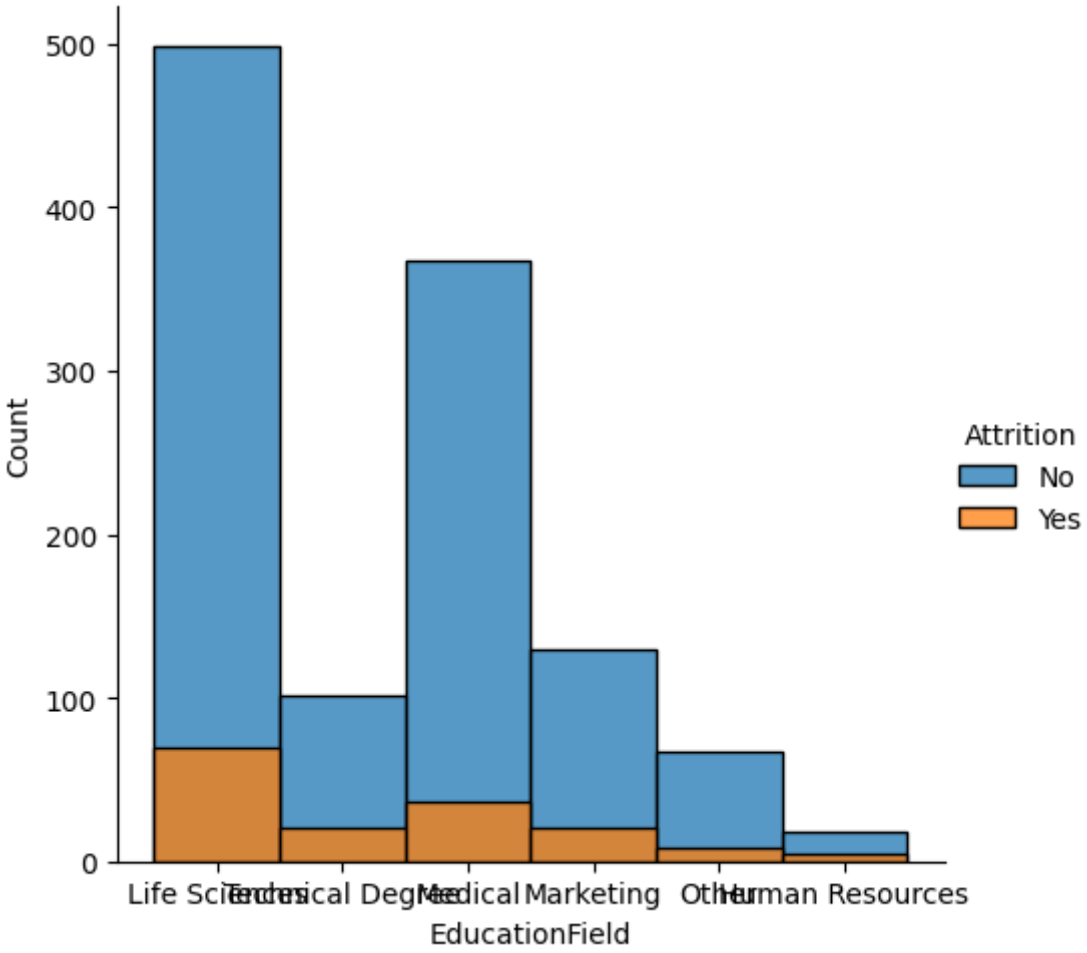

800
700
600
500
400
300
200
100
0
Count
Attrition
No
Yes
Travel_Rarely
Travel_Frequently
Non-Travel
BusinessTravel

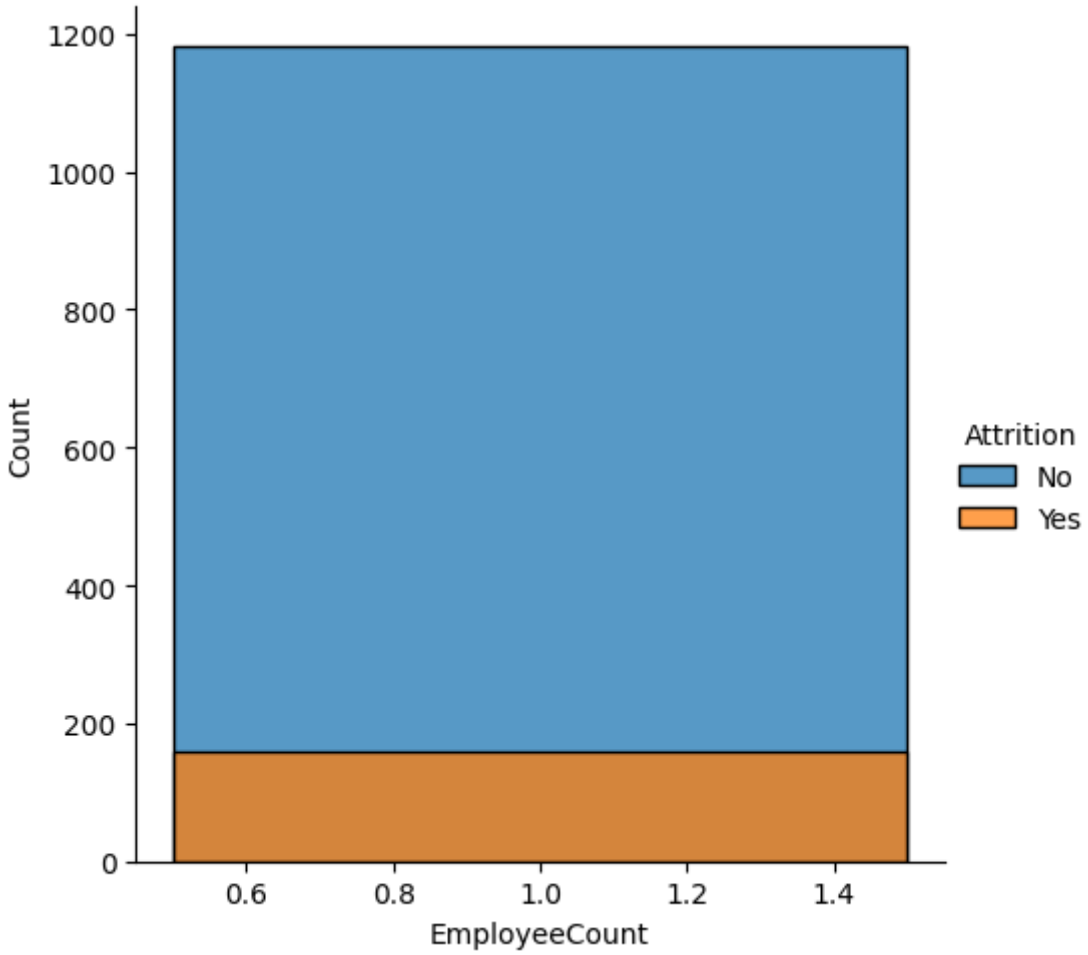

120
100
80
60
40
20
0
Count
200
400
600
800
1000
1200
1400
DailyRate
Attrition
No
Yes

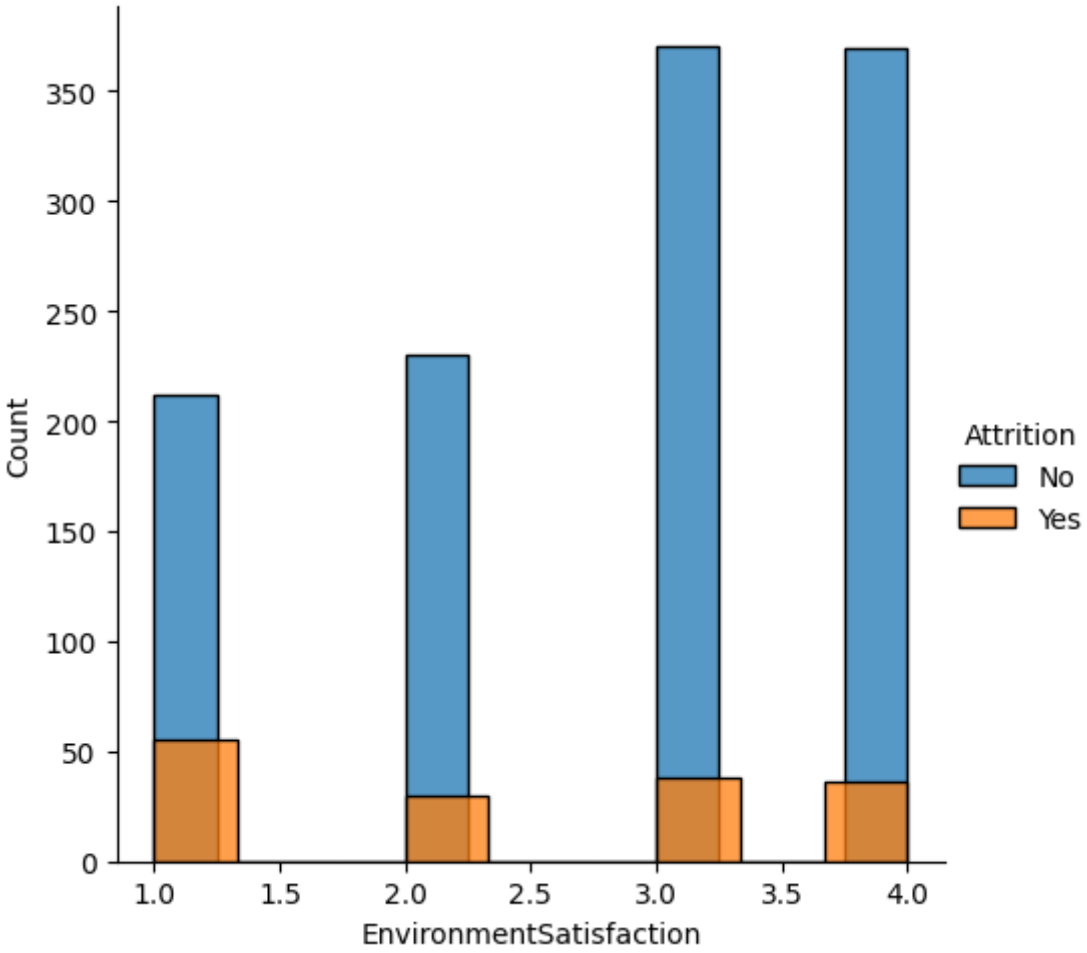

500
400
300
200
100
0
Count
Cardiology
Maternity
Neurology
Department
Attrition
No
Yes

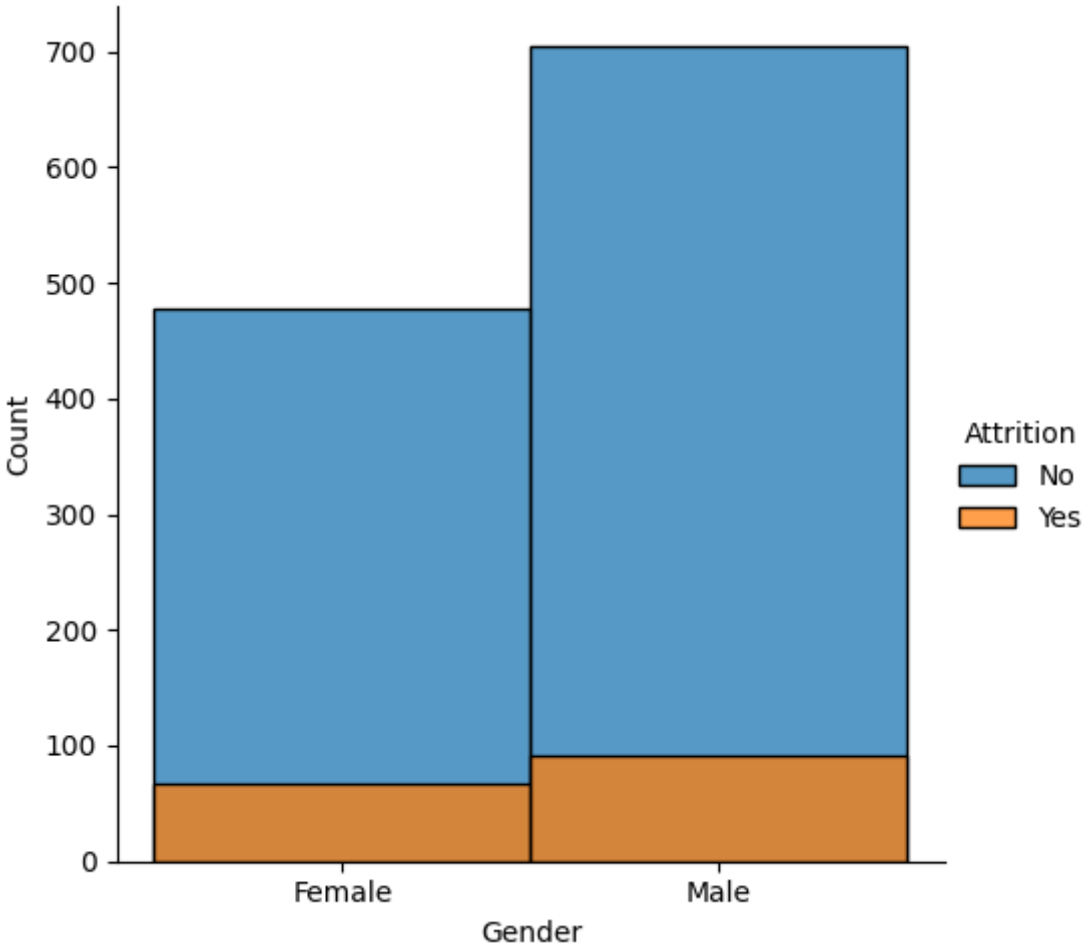

350
300
250
200
150
100
50
0
Count
0
5
10
15
20
25
30
DistanceFromHome
Attrition
No
Yes

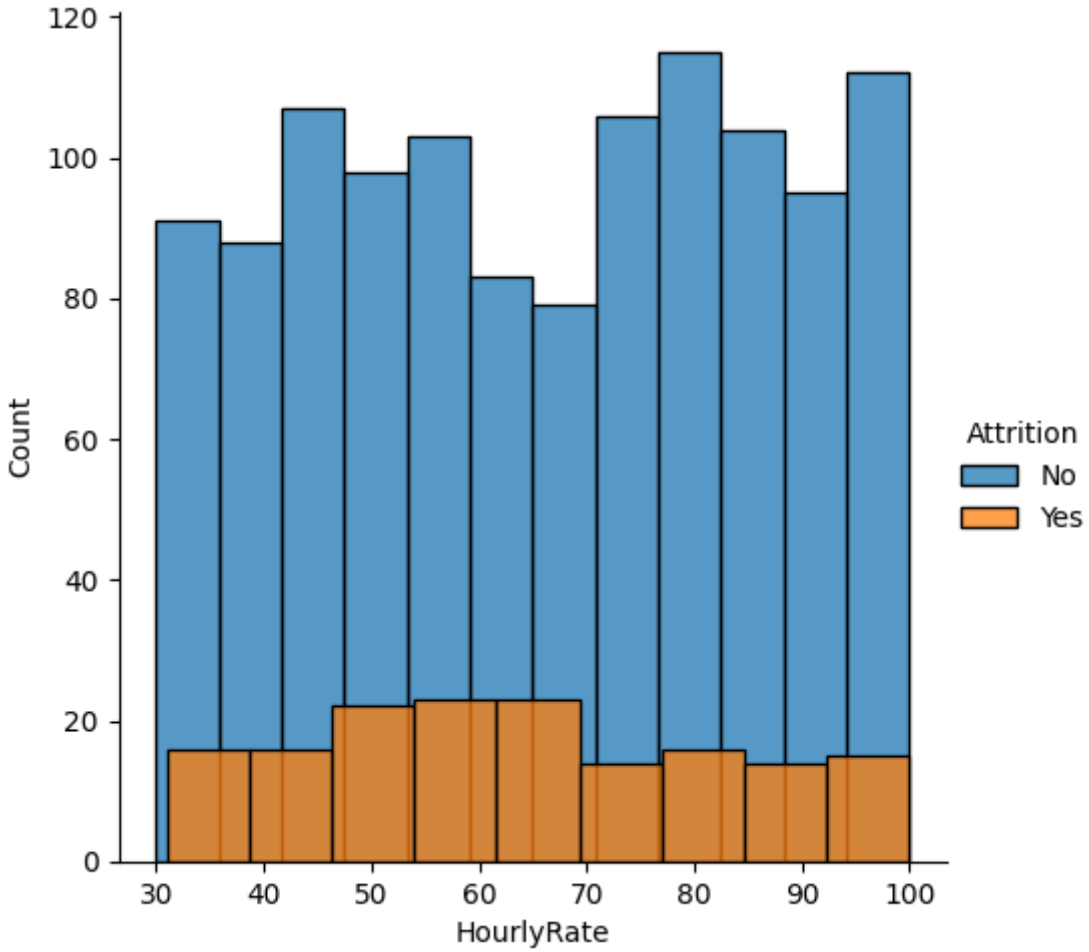

400
300
200
100
0
Count
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Education
Attrition
No
Yes

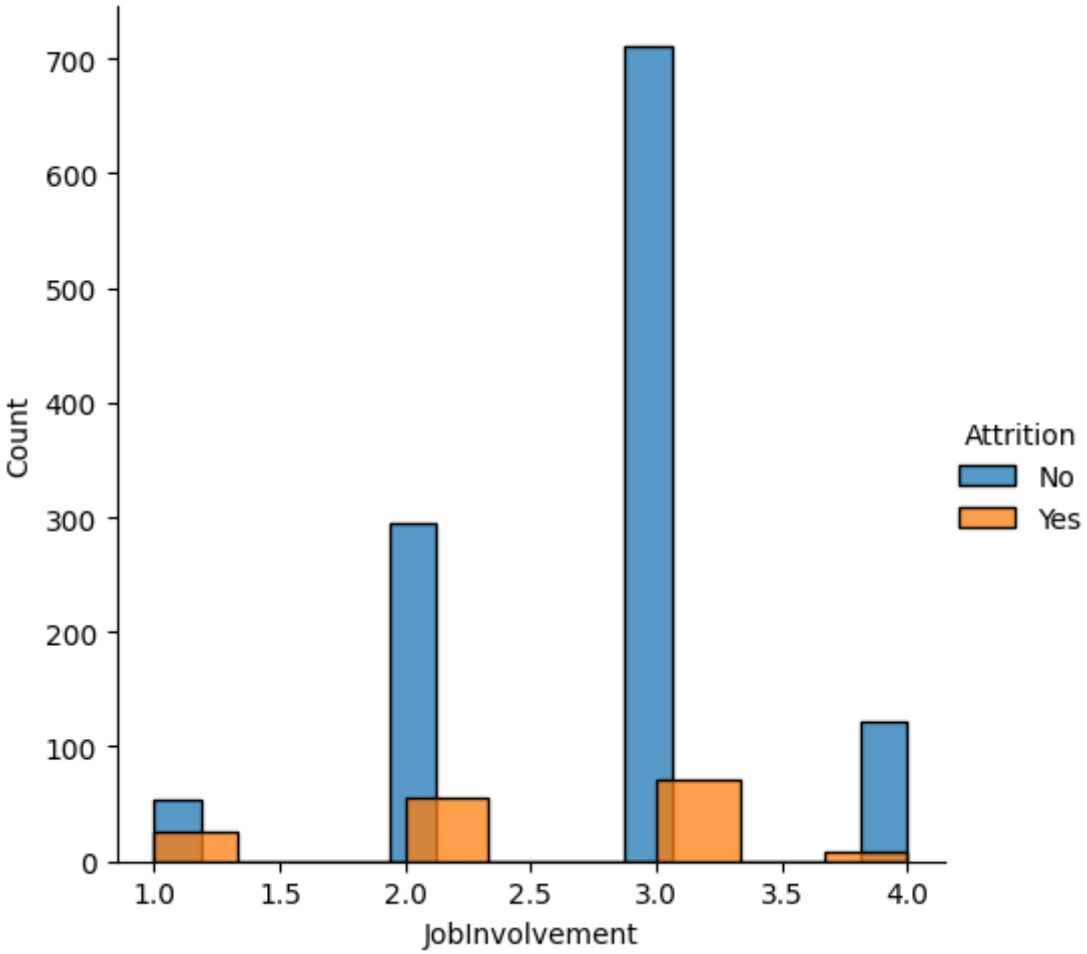

500
400
300
200
100
0
Count
Attrition
No
Yes
Life Sciences
Technical Degree
Medical
Marketing
Other
Human Resources
EducationField

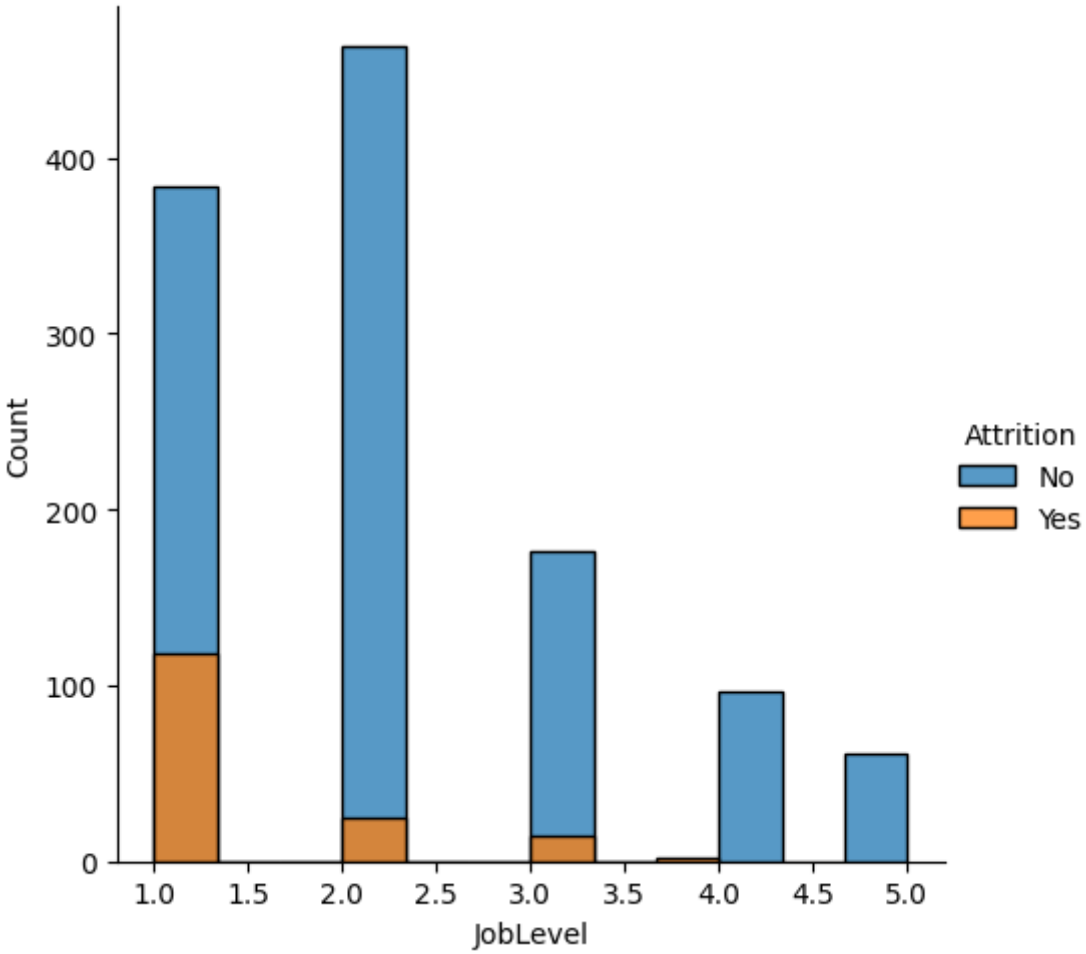

1200
1000
800
600
400
200
0
Count
0.6
0.8
1.0
1.2
1.4
EmployeeCount
Attrition
No
Yes

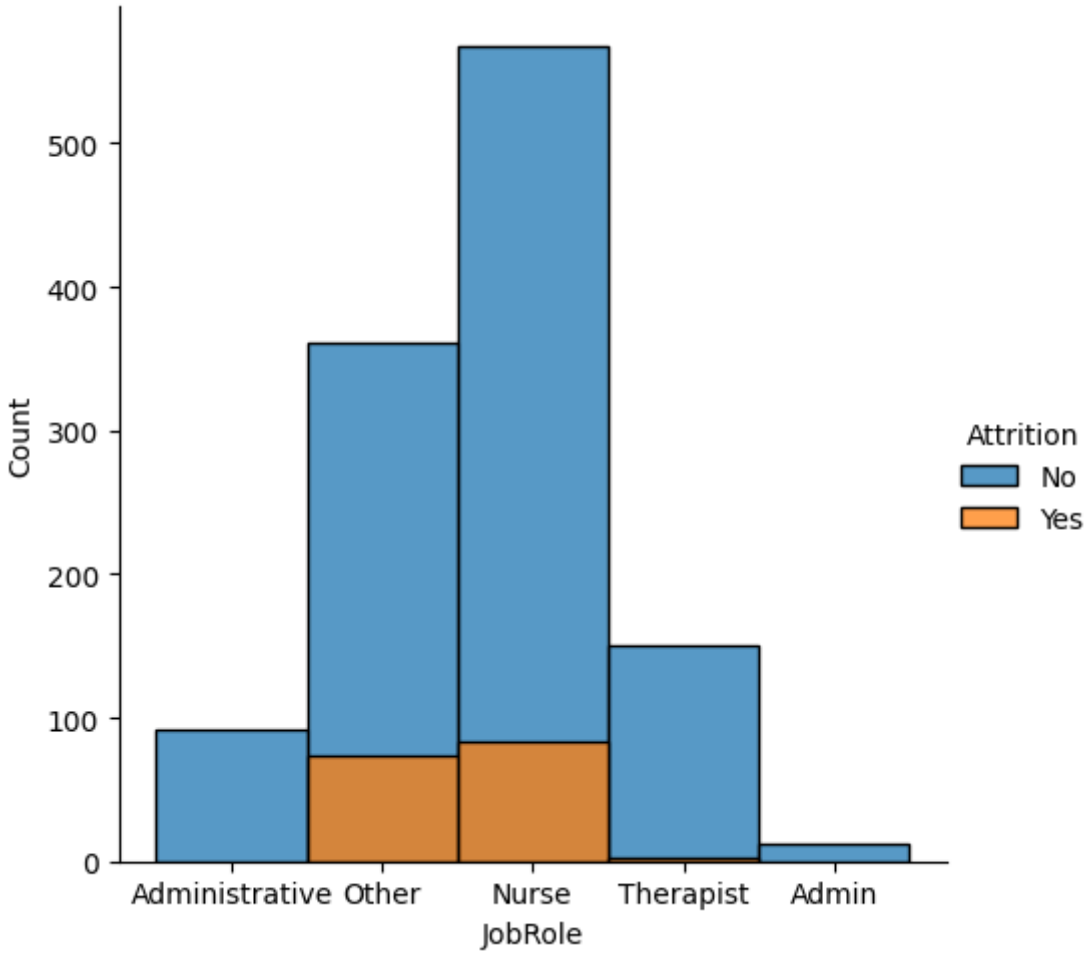

Count
350
300
250
200
150
100
50
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
EnvironmentSatisfaction
Attrition
No
Yes

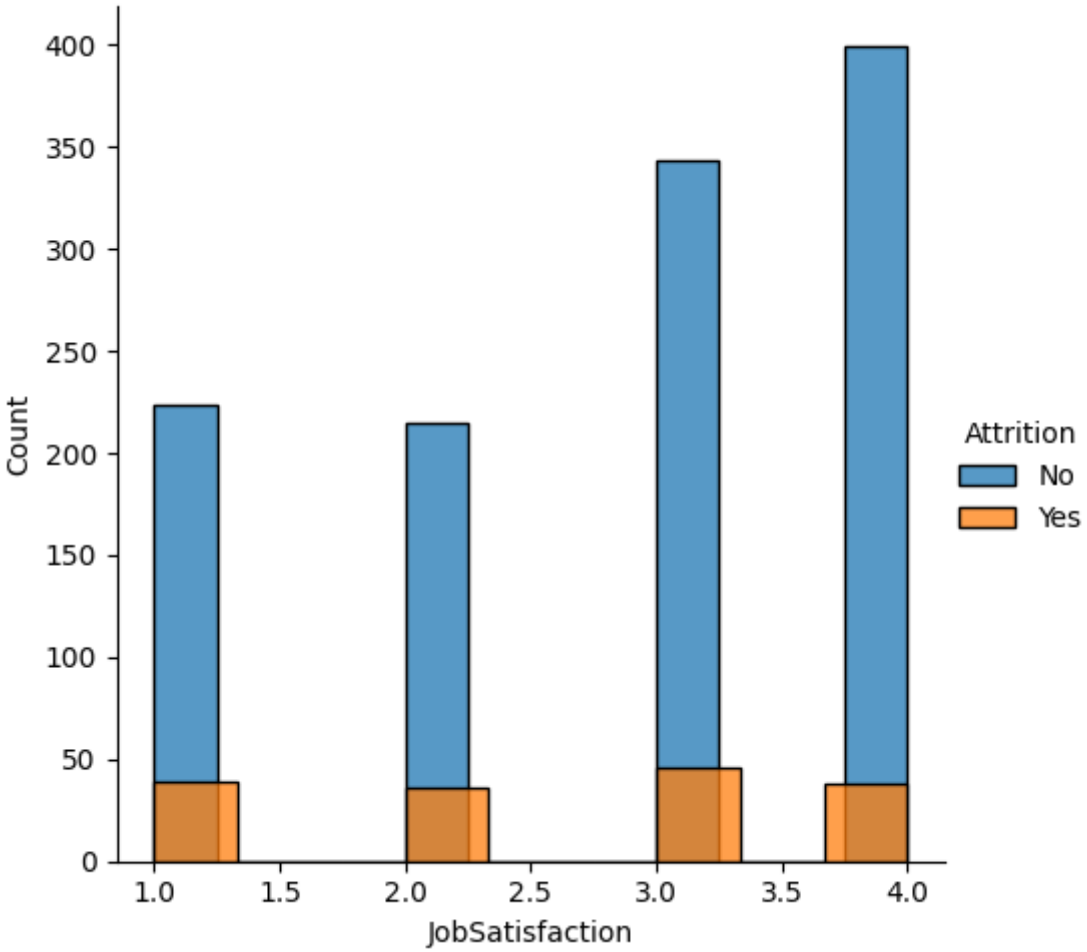

700
600
500
400
300
200
100
0
Count
Female
Male
Gender
Attrition
No
Yes

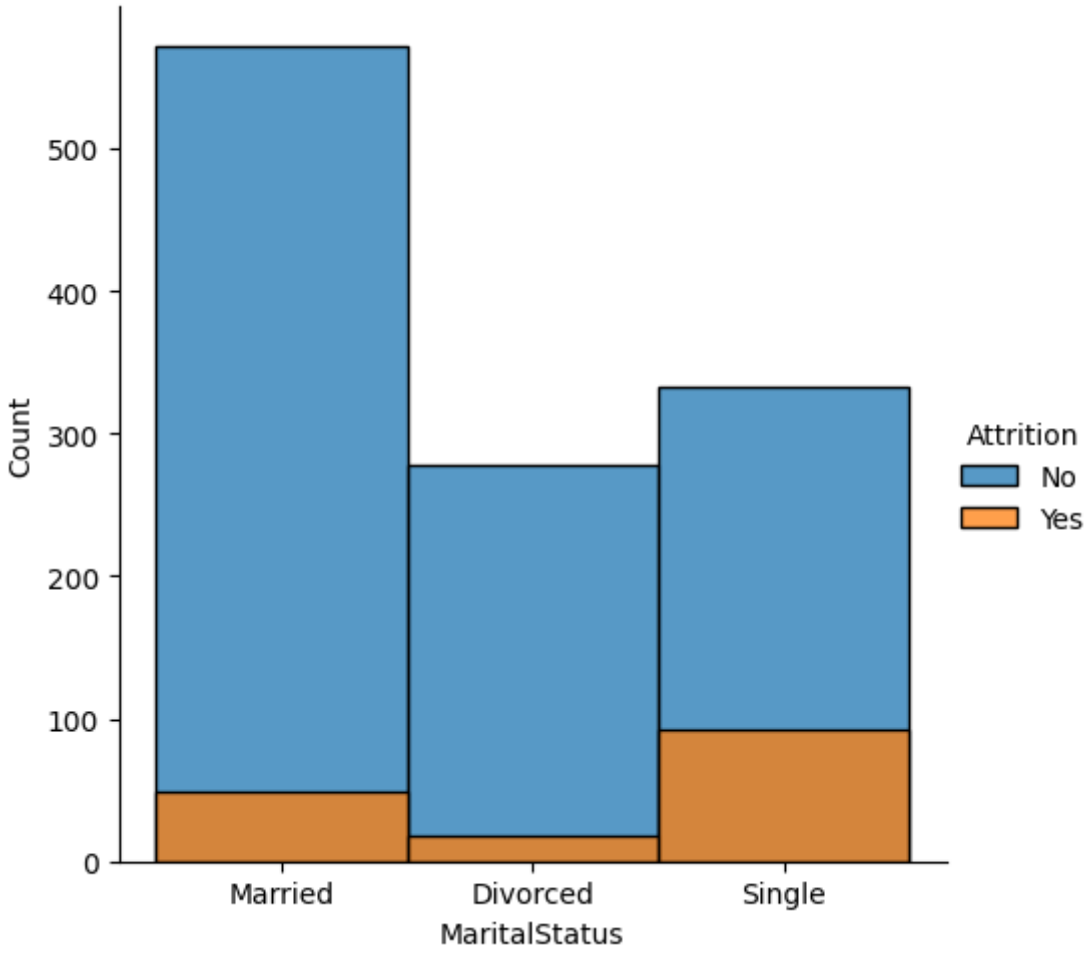

120
100
80
60
40
20
0
Count
30
40
50
60
70
80
90
100
HourlyRate
Attrition
No
Yes

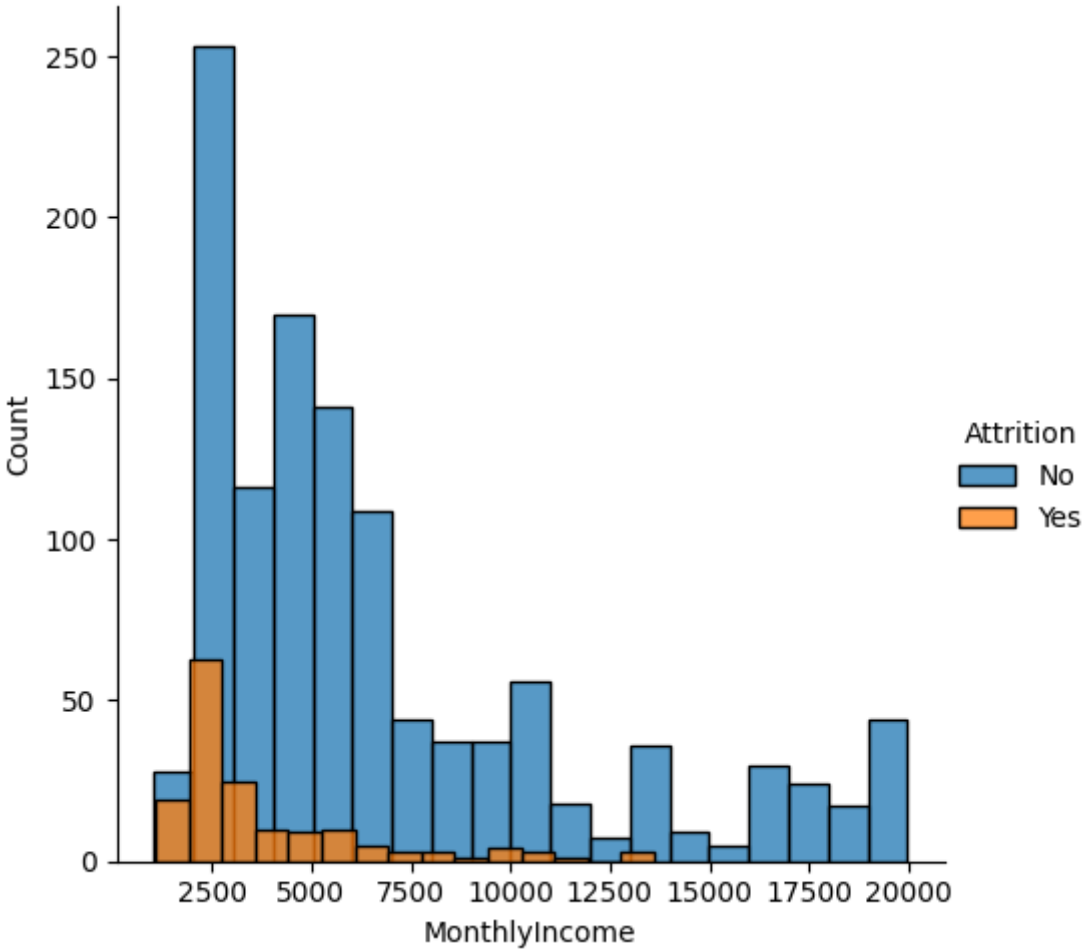

700
600
500
400
300
200
100
0
Count
1.0
1.5
2.0
2.5
3.0
3.5
4.0
JobInvolvement
Attrition
No
Yes

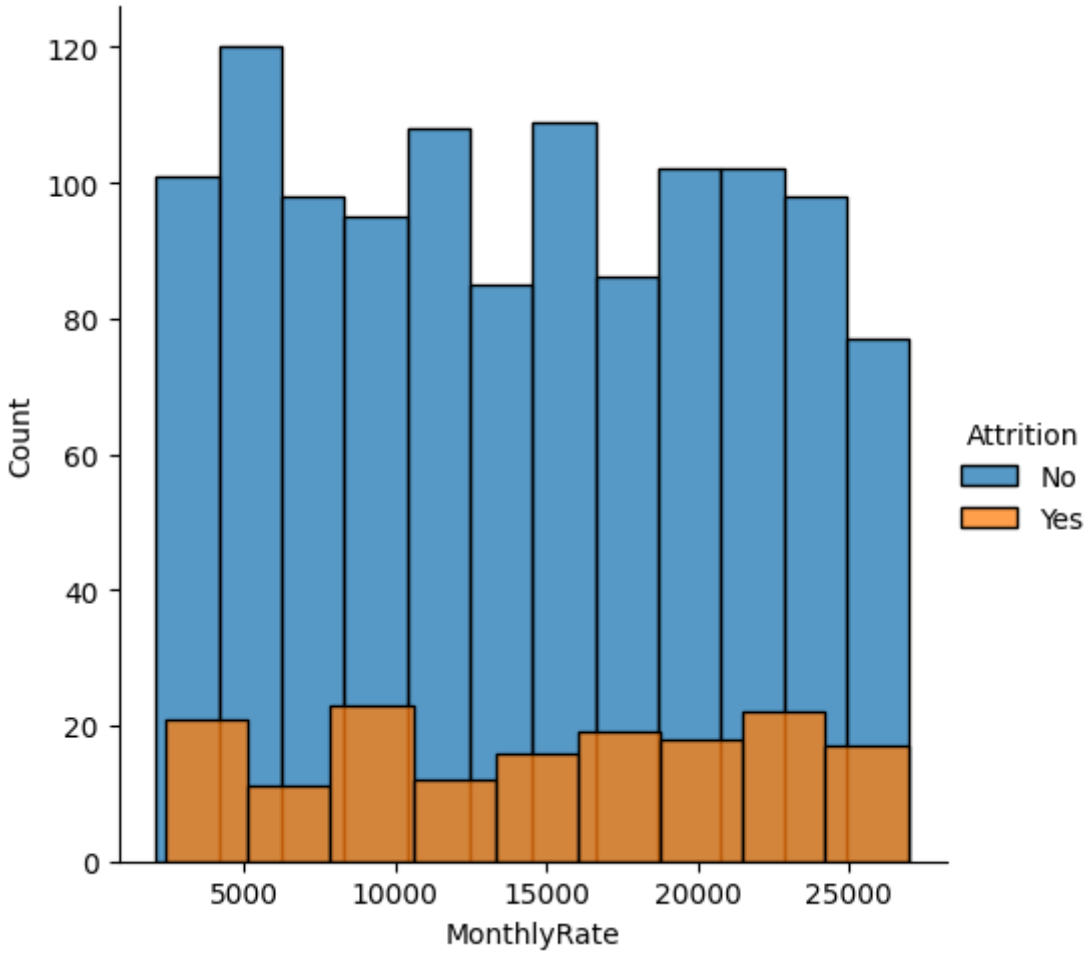

Count
400
300
200
100
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
JobLevel
Attrition
No
Yes

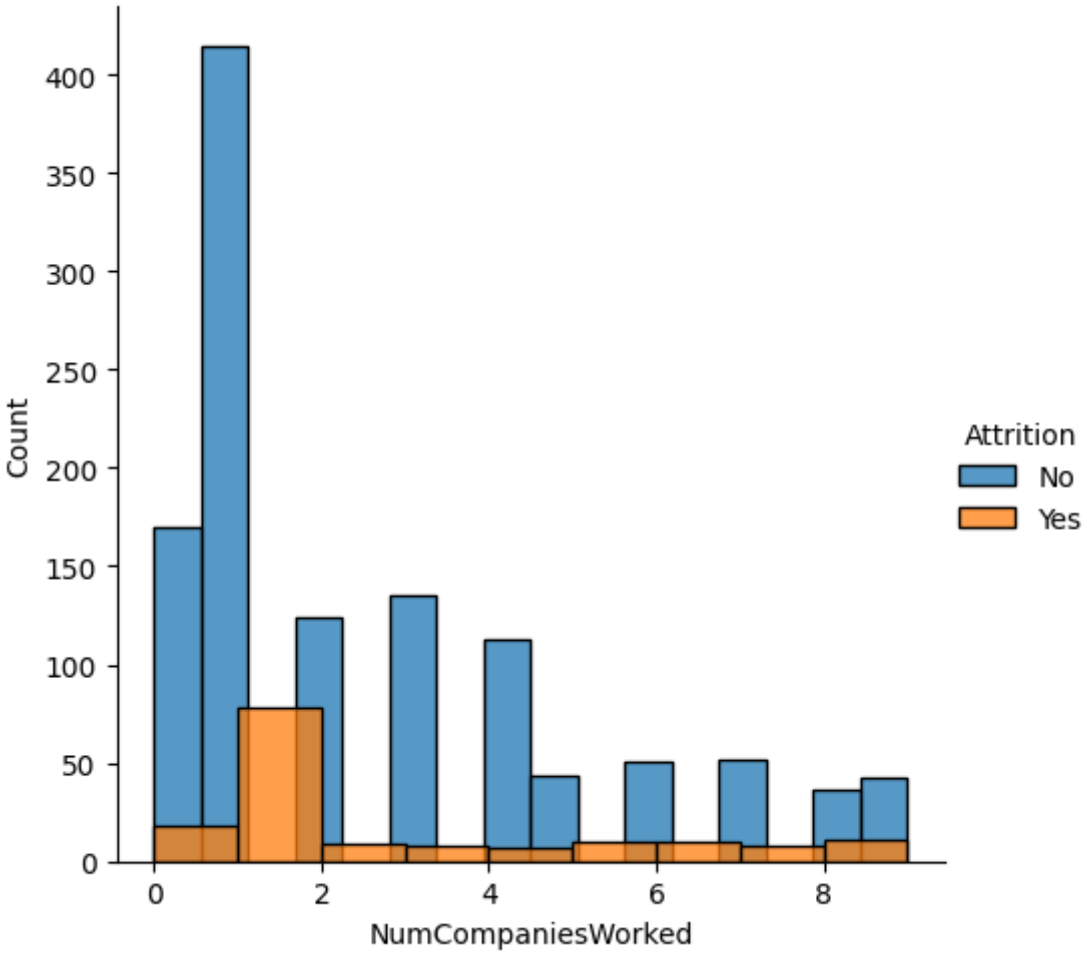

500
400
300
200
100
0
Count
Administrative
Other
Nurse
Therapist
Admin
JobRole
Attrition
No
Yes

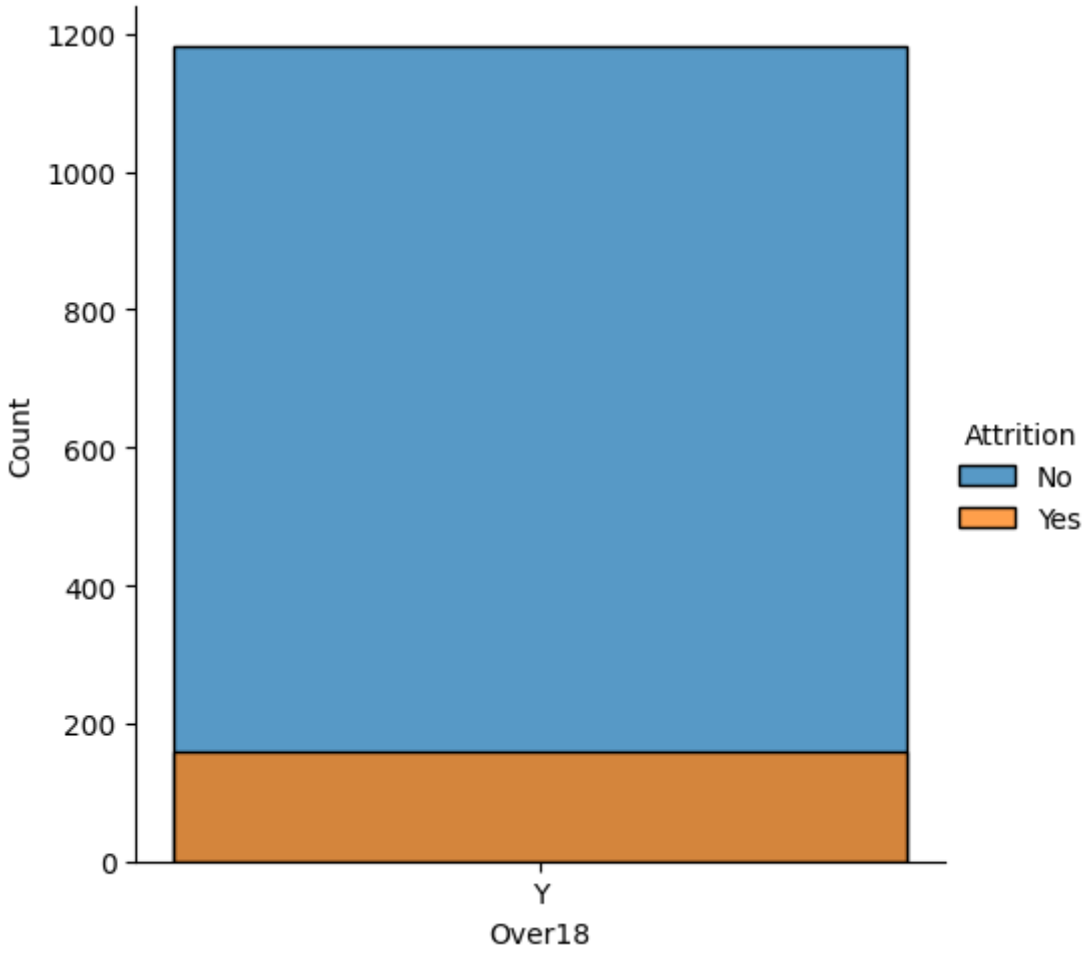

400
350
300
250
200
150
100
50
0
Count
1.0
1.5
2.0
2.5
3.0
3.5
4.0
JobSatisfaction
Attrition
No
Yes

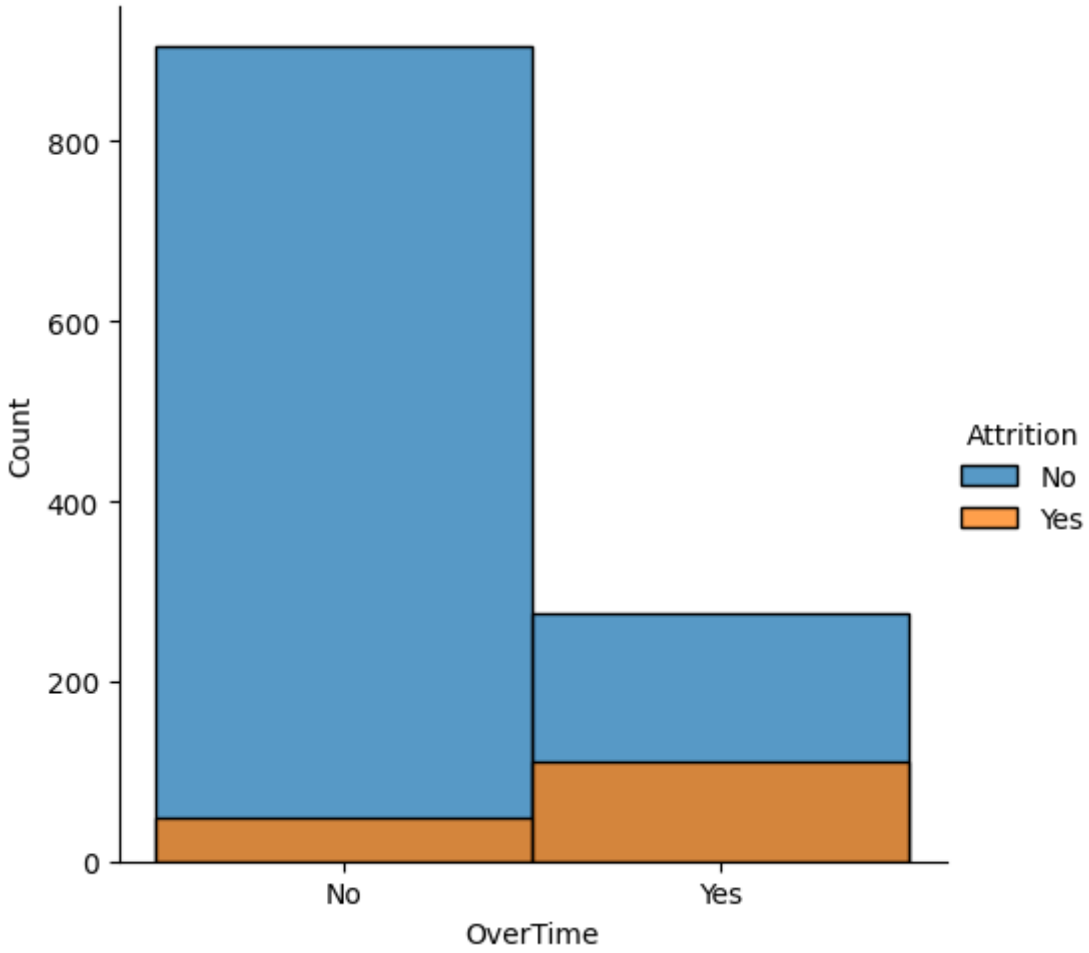

500
400
300
200
100
0
Count
Married
Divorced
Single
MaritalStatus
Attrition
No
Yes

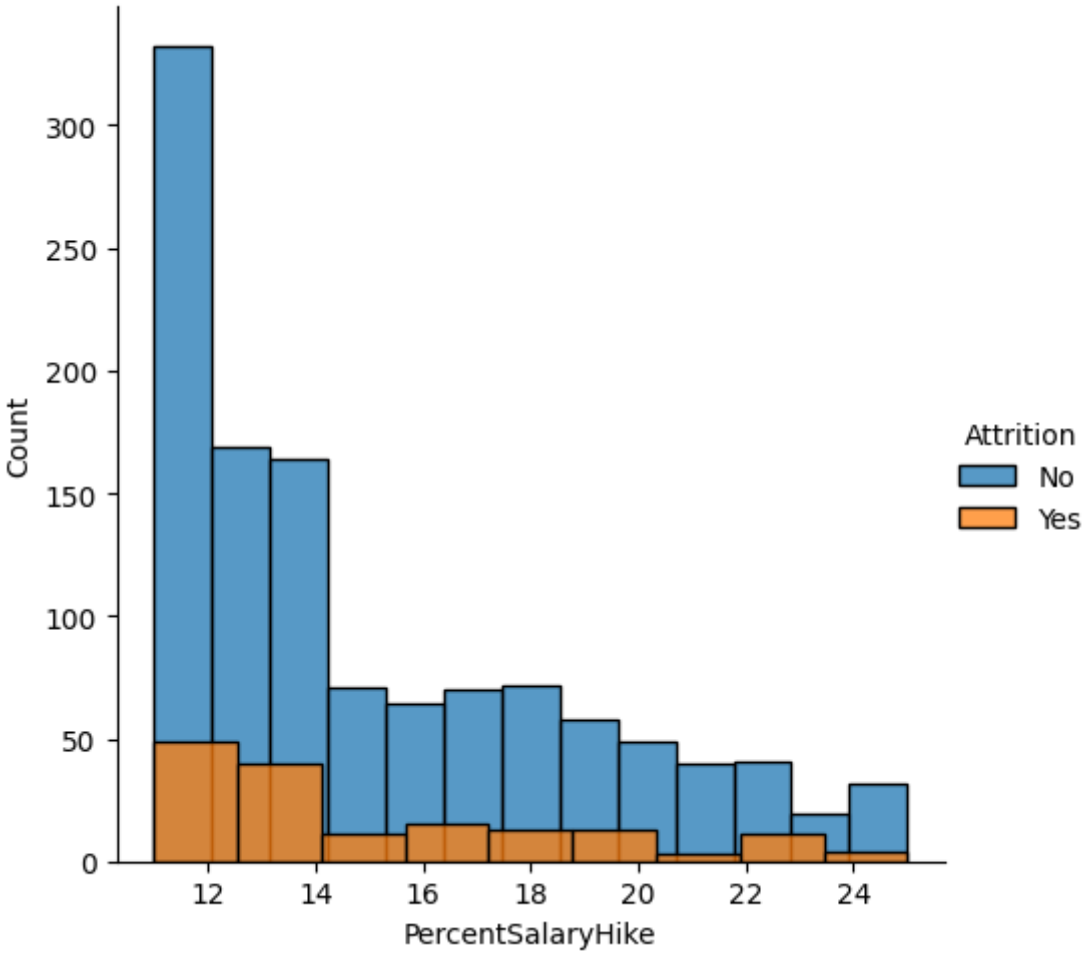

250
200
150
100
50
0
Count
2500
5000
7500
10000
12500
15000
17500
20000
MonthlyIncome
Attrition
No
Yes

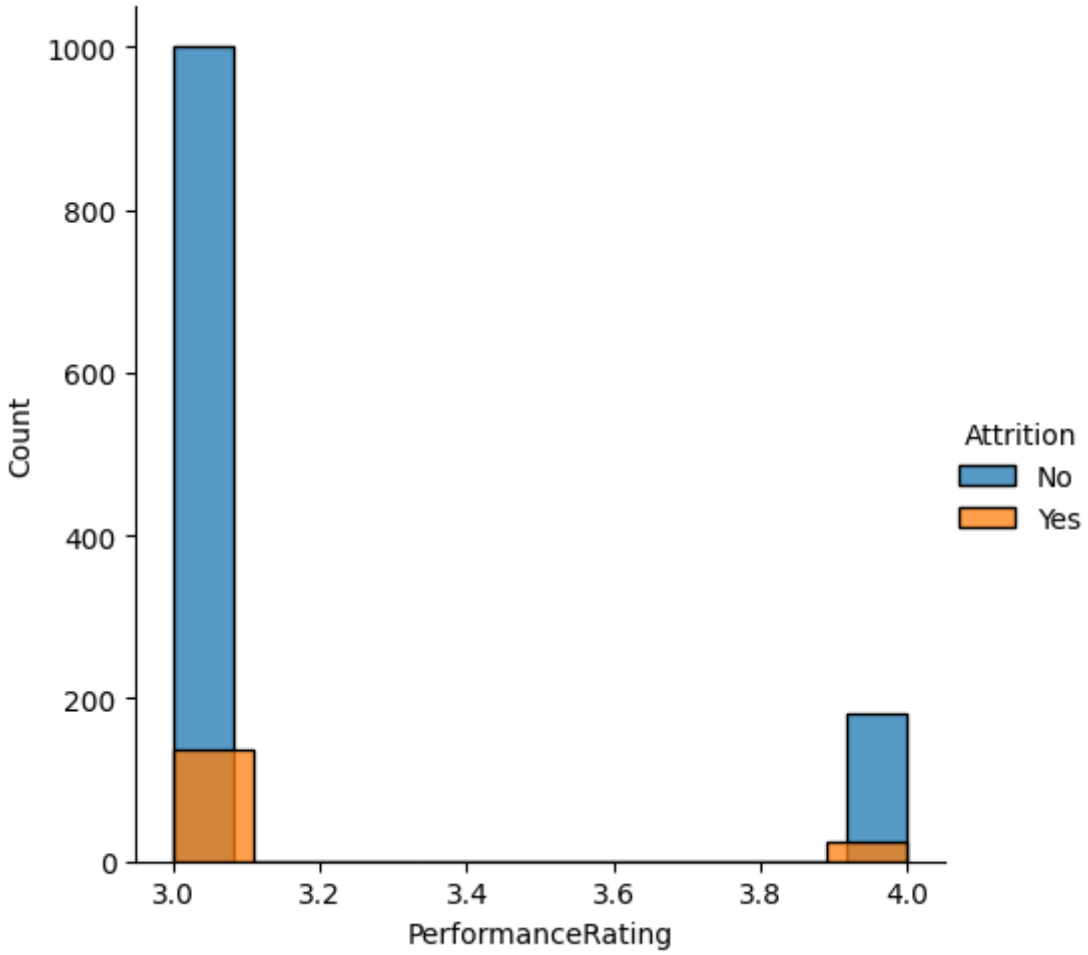

120
100
80
60
40
20
0
Count
5000
10000
15000
20000
25000
MonthlyRate
Attrition
No
Yes

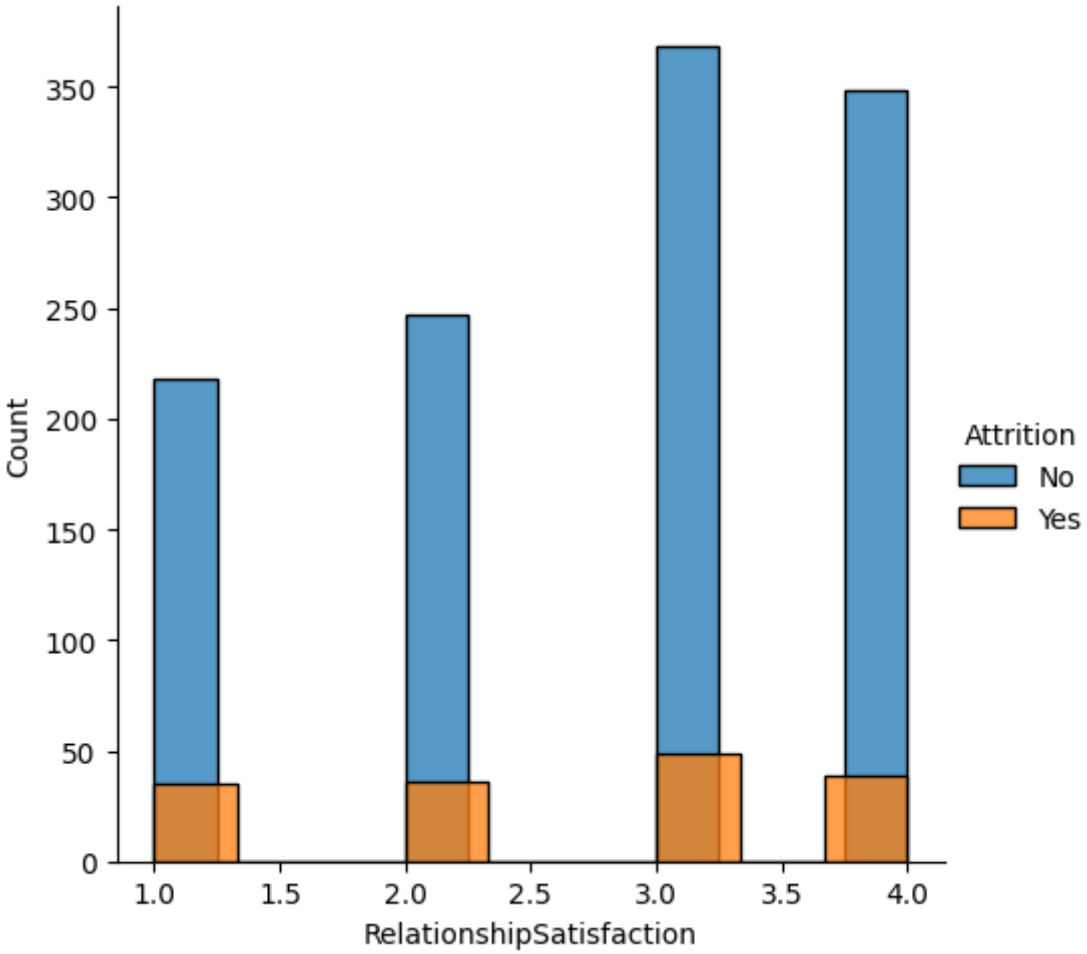

Count
400
350
300
250
200
150
100
50
0
0
2
4
6
8
NumCompaniesWorked
Attrition
No
Yes

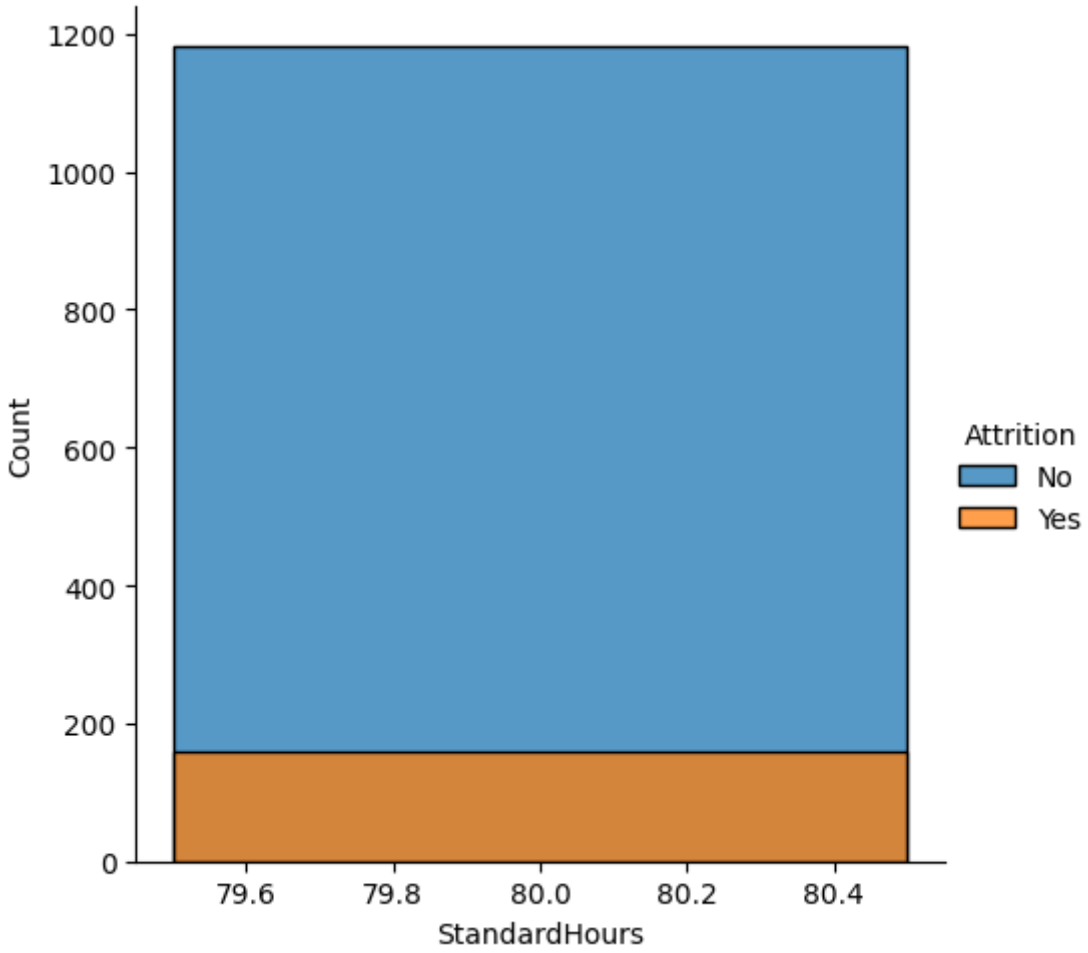

1200
1000
800
600
400
200
0
Count
Y
Over18
Attrition
No
Yes

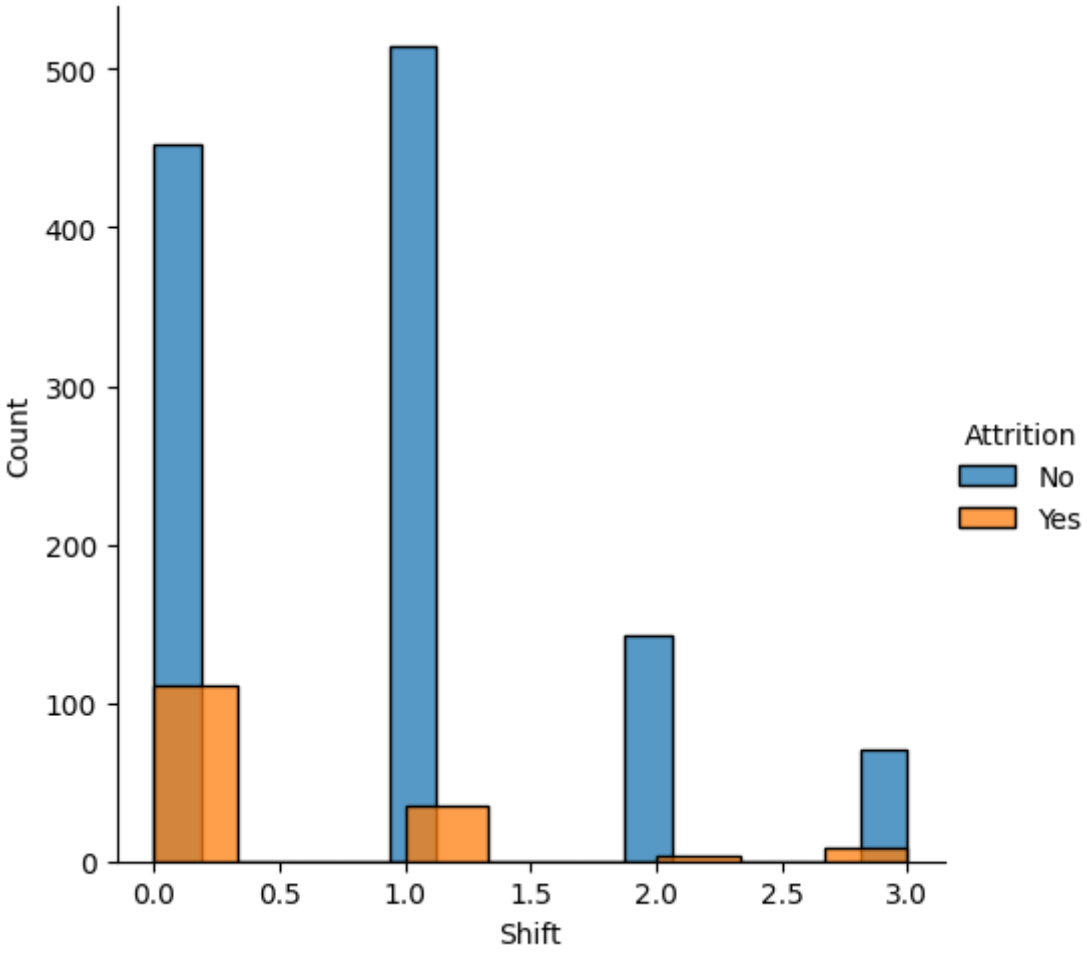

800
600
400
200
0
Count
No
Yes
OverTime
Attrition
No
Yes

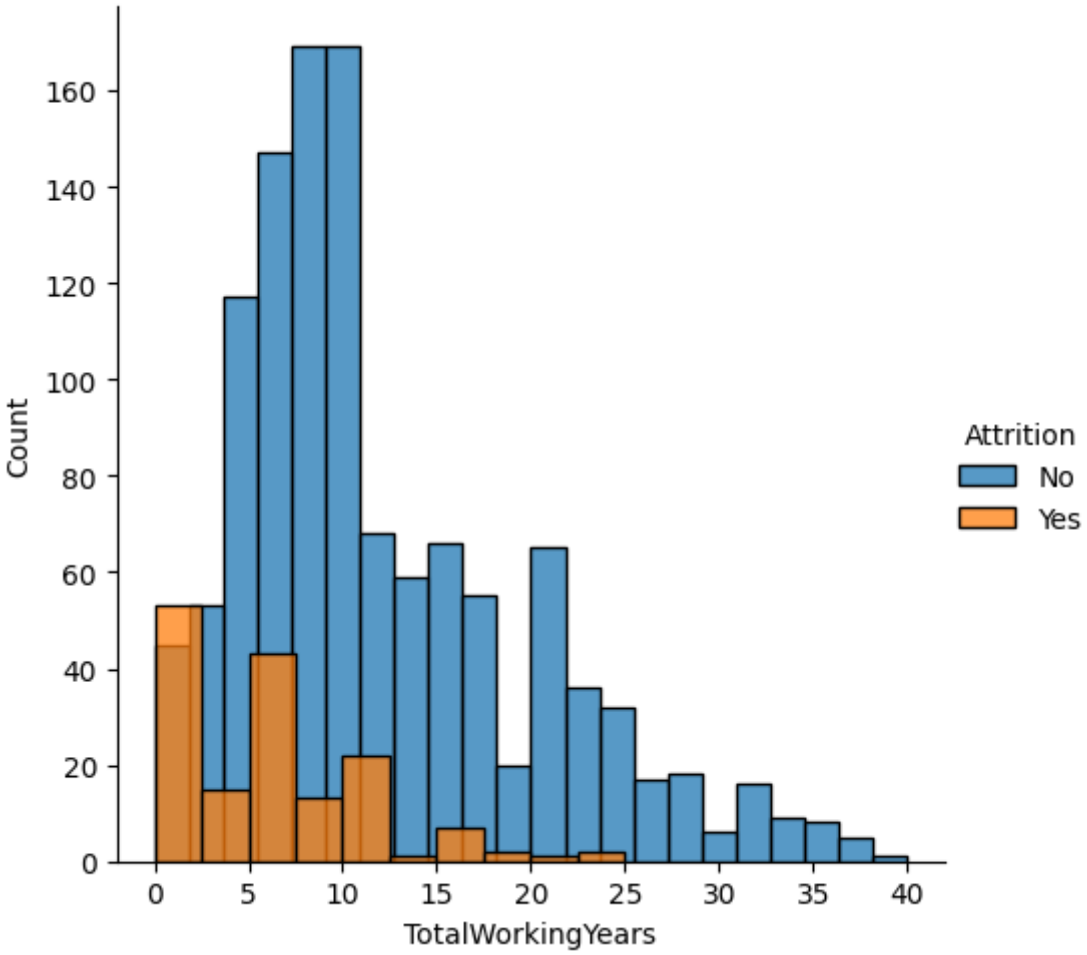

300
250
200
150
100
50
0
Count
12
14
16
18
20
22
24
PercentSalaryHike
Attrition
No
Yes

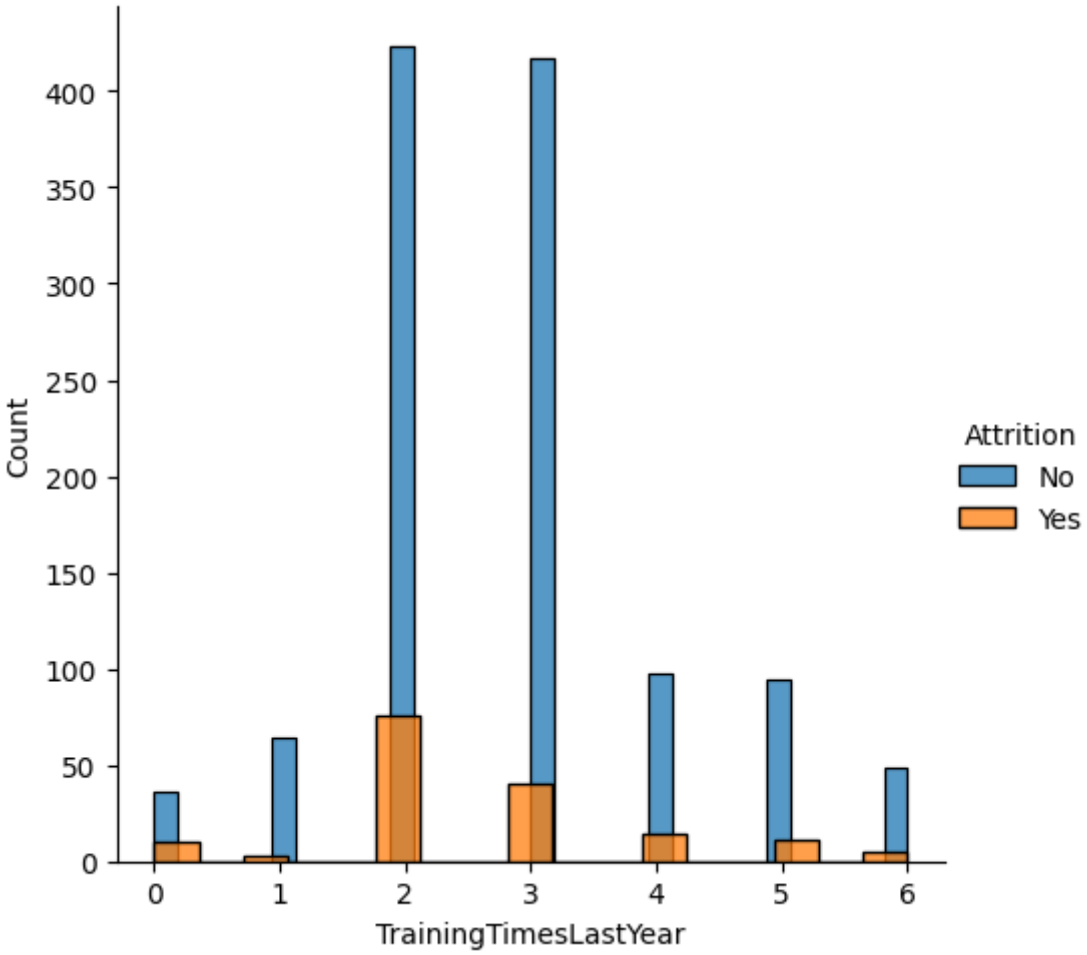

1000
800
600
400
200
0
Count
3.0
3.2
3.4
3.6
3.8
4.0
PerformanceRating
Attrition
No
Yes

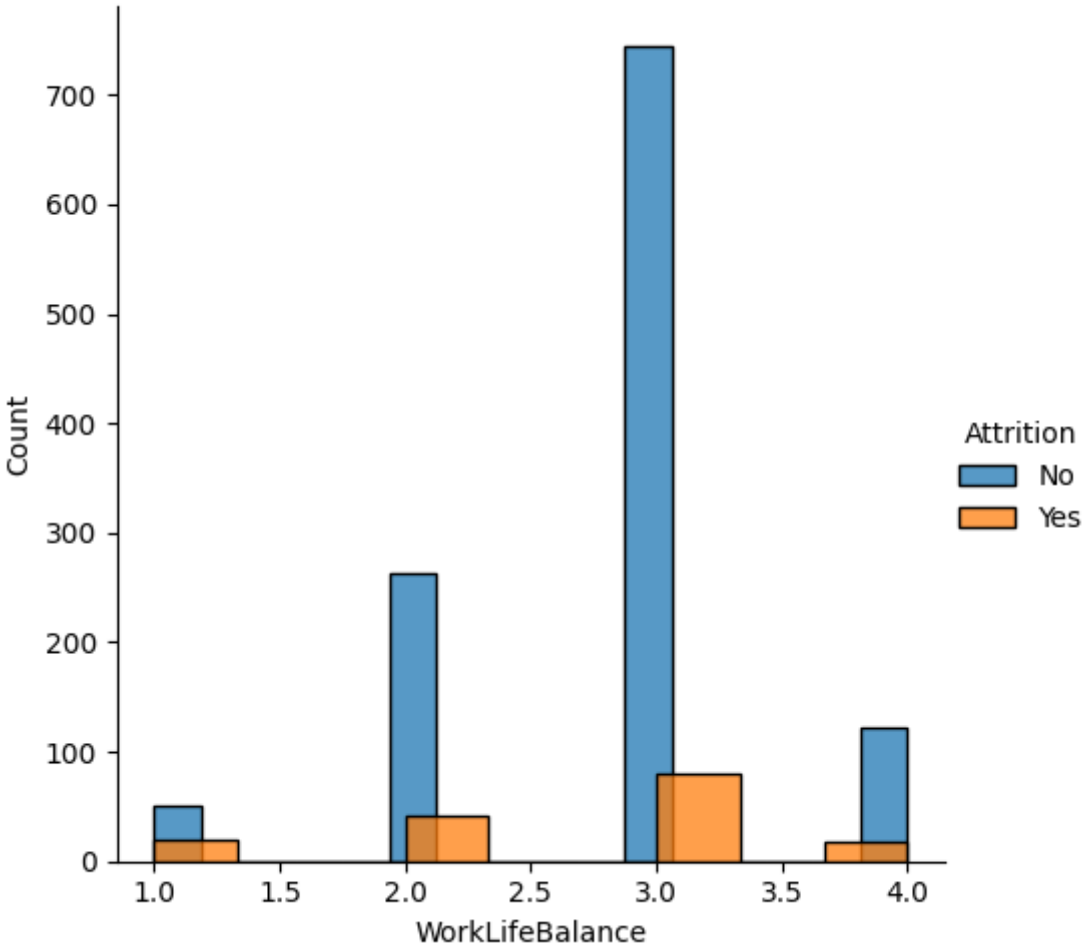

350
300
250
200
150
100
50
0
Count
1.0
1.5
2.0
2.5
3.0
3.5
4.0
RelationshipSatisfaction
Attrition
No
Yes

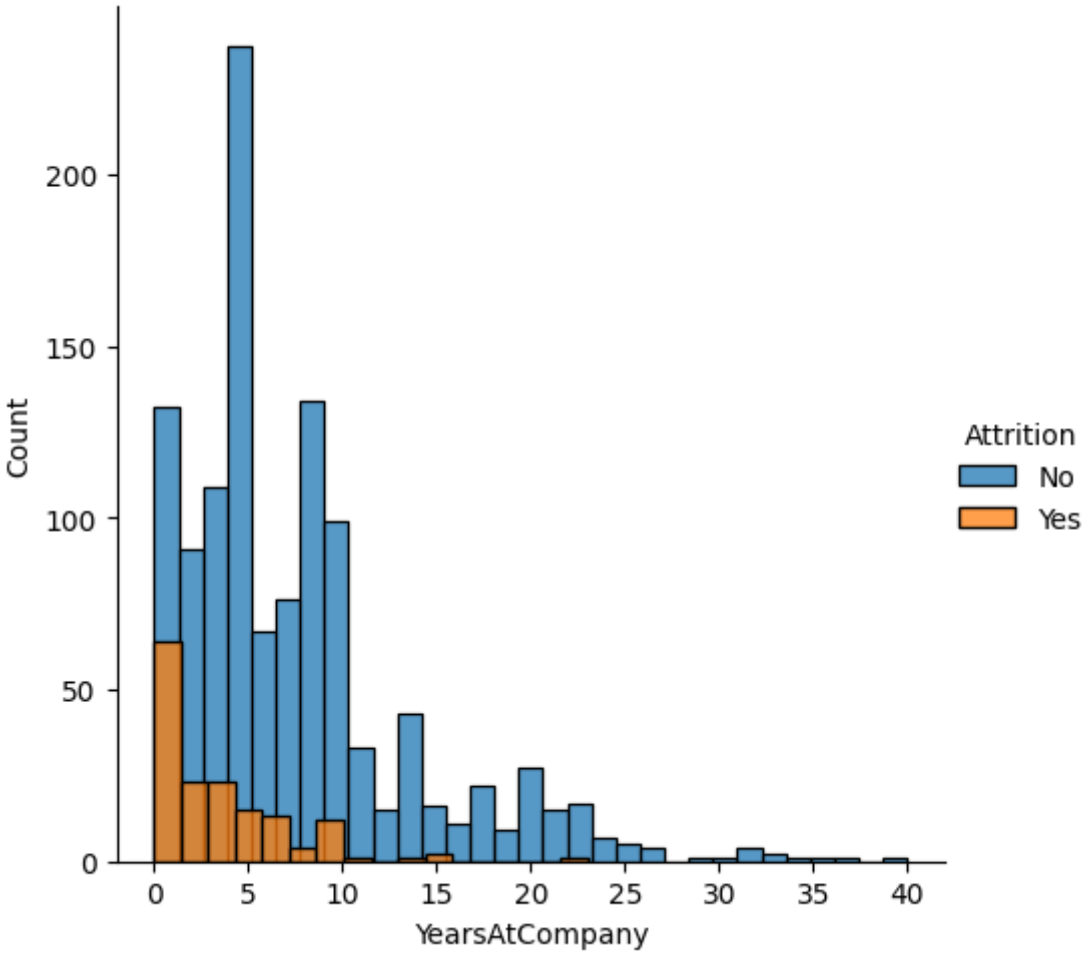

1200
1000
800
600
400
200
0
Count
79.6
79.8
80.0
80.2
80.4
StandardHours
Attrition
No
Yes

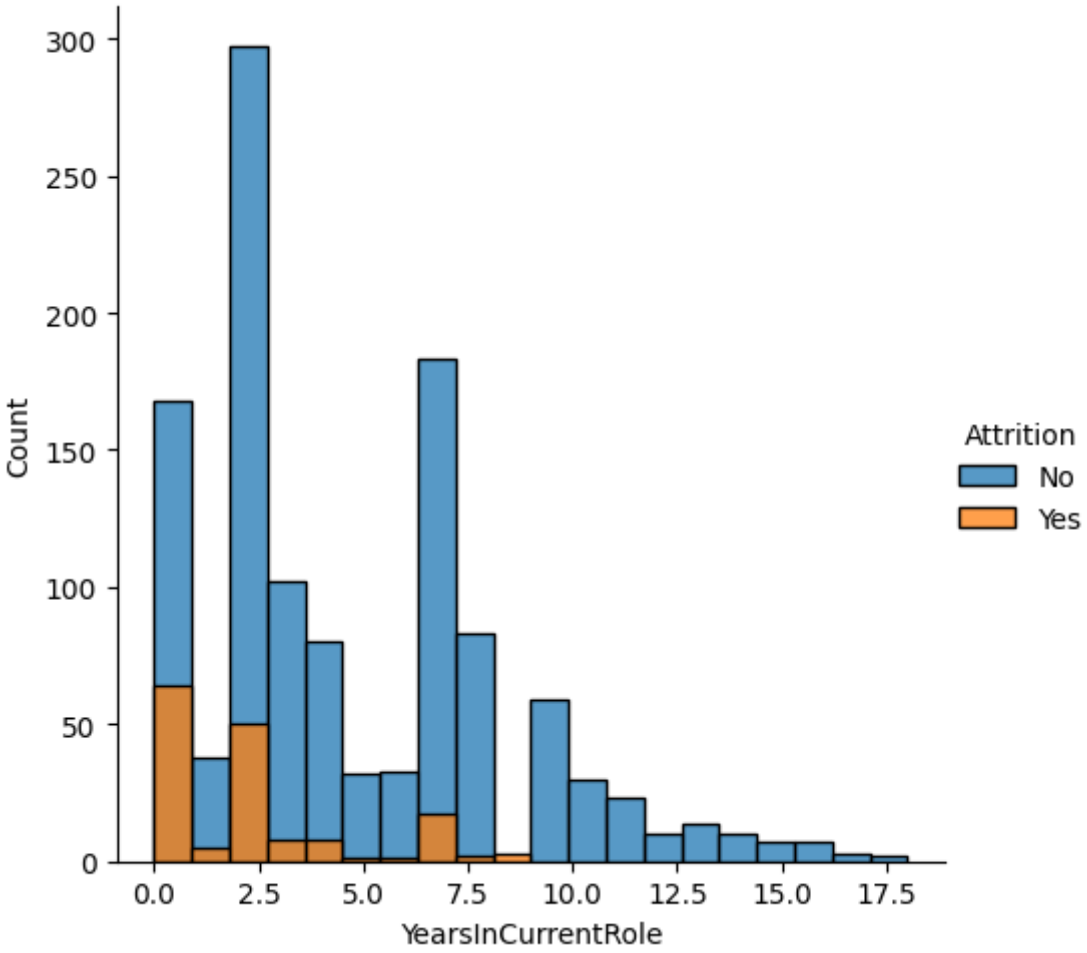

500
400
300
200
100
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Shift
Attrition
No
Yes

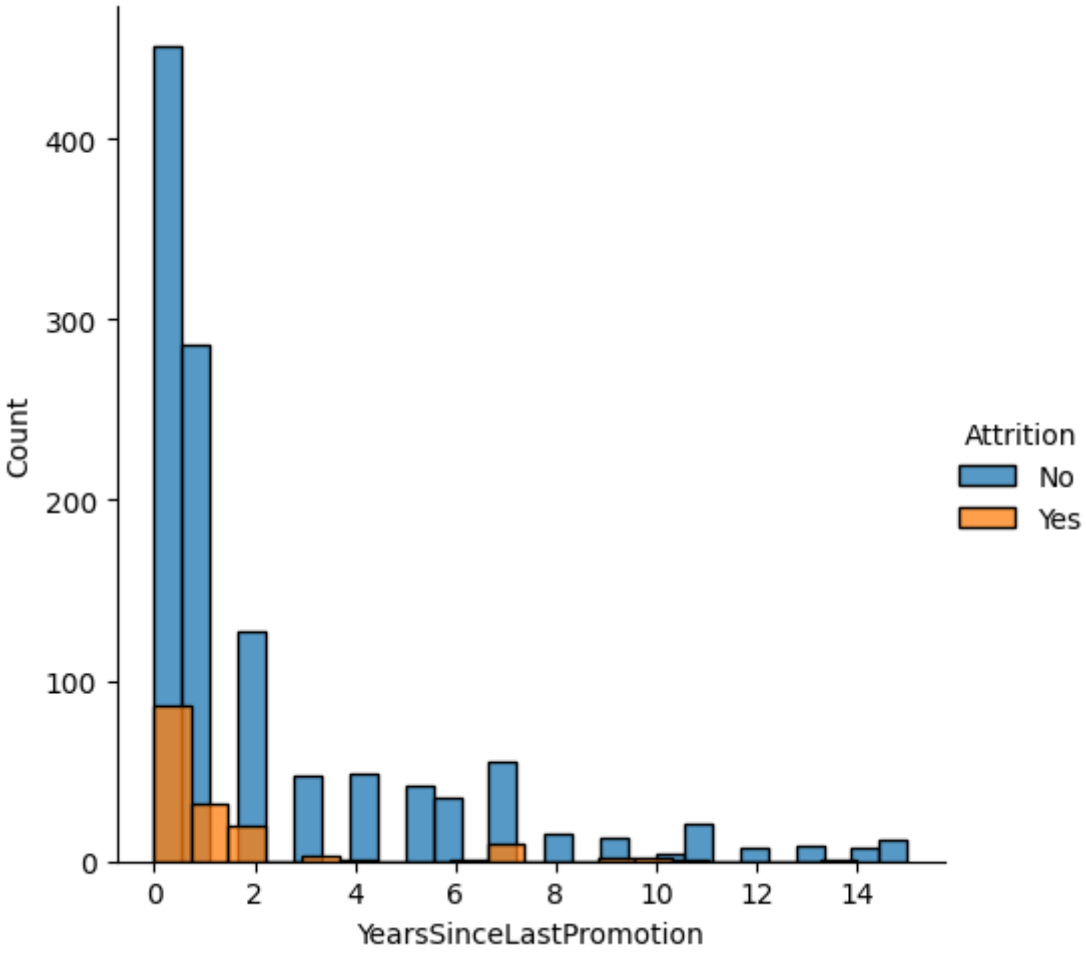

Count
160
140
120
100
80
60
40
20
0
0
5
10
15
20
25
30
35
40
TotalWorkingYears
Attrition
No
Yes

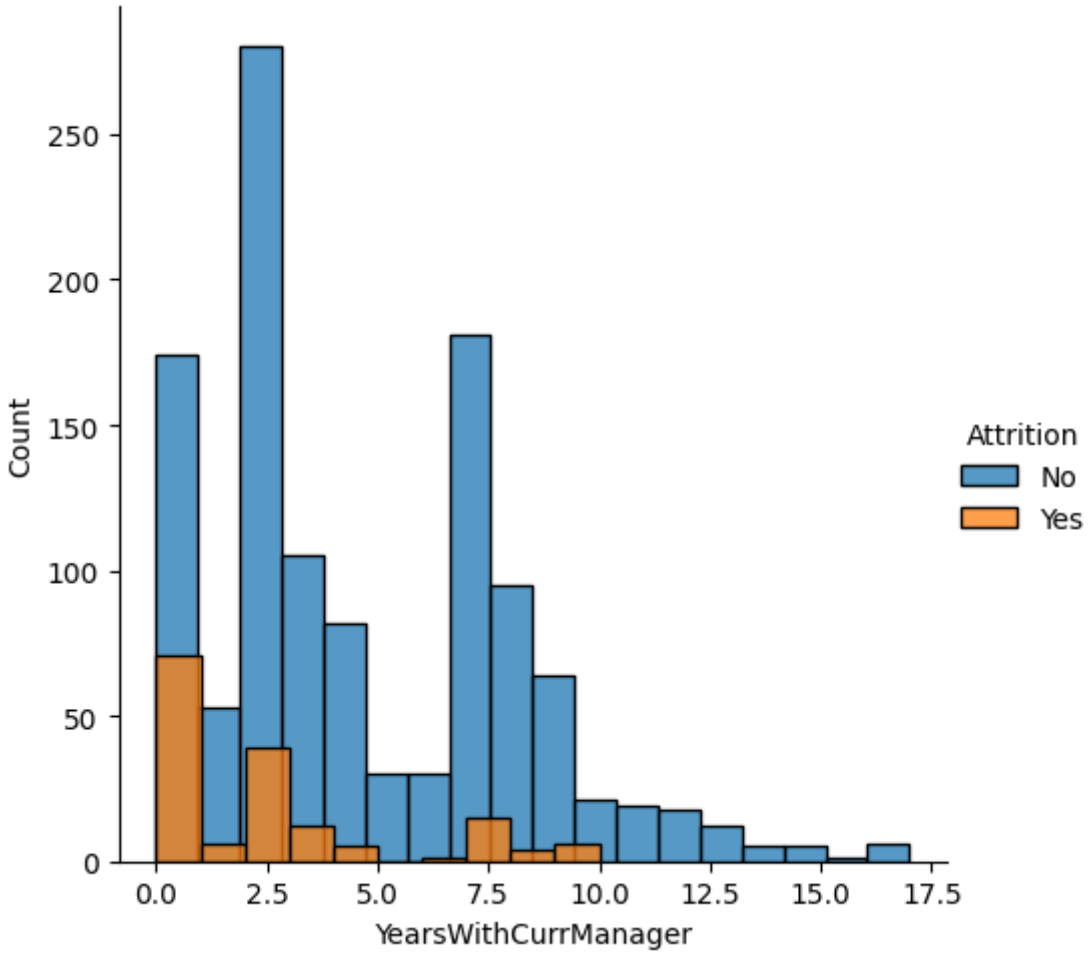

Count
400
350
300
250
200
150
100
50
0
0
1
2
3
4
5
6
TrainingTimesLastYear
Attrition
No
Yes


700
600
500
400
300
200
100
0
Count
1.0
1.5
2.0
2.5
3.0
3.5
4.0
WorkLifeBalance
Attrition
No
Yes


Count
200
150
100
50
0
0
5
10
15
20
25
30
35
40
YearsAtCompany
Attrition
No
Yes

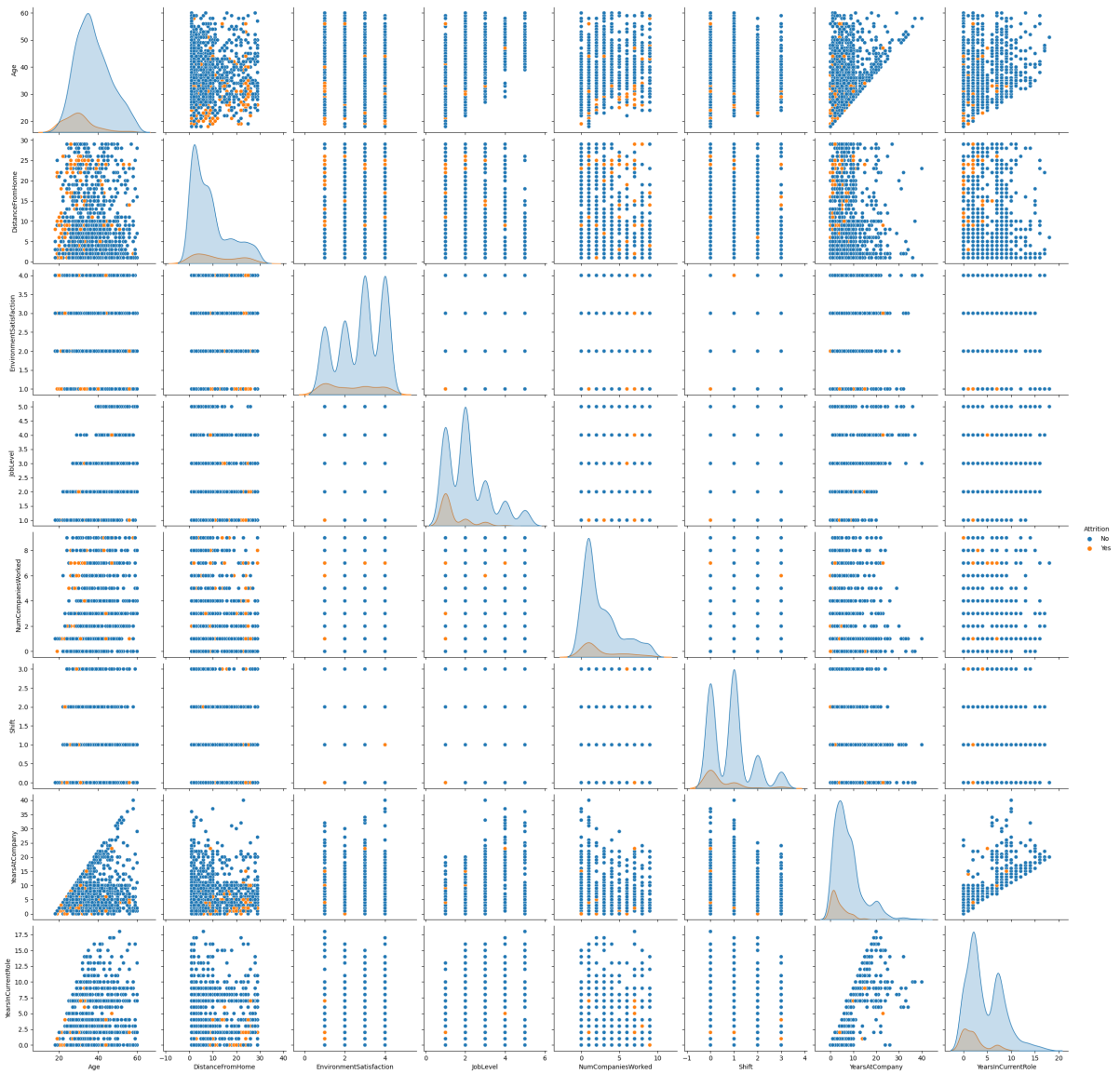

300
250
200
150
100
50
0
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
YearsInCurrentRole
Attrition
No
Yes


Count
400
300
200
100
0
0
2
4
6
8
10
12
14
YearsSinceLastPromotion
Attrition
No
Yes



In [ ]:
```
# Make pairwise scatter plots promising features seen in above analysis
promising_features = ["Age", "DistanceFromHome",
                      "EnvironmentSatisfaction",
                      "JobLevel", "JobRole",
                      "MaritalStatus",
                      "NumCompaniesWorked", "OverTime",
                      "Shift", "YearsAtCompany",
                      "YearsInCurrentRole",
                      "Attrition"]

promising_data = data[promising_features]

sns.pairplot(promising_data,
             hue="Attrition", height= 3)
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x2abee75b0>
```



In [ ]:
```
# Import other data
labels = data.pop("Attrition")
submission_data = pd.read_csv("test.csv")
print(data.shape)
print(submission_data.shape)
```

```
(1340, 34)
(336, 34)
```

In [ ]:
```
# drop columns with only 1 unique value
for c in data.columns:
    print(c, " num unique values: ", len(data[c].unique()), "\n")
    if len(data[c].unique()) == 1:
        data.drop([c], axis=1, inplace=True)
        submission_data.drop([c], axis=1, inplace=True)
print(data.shape)
print(submission_data.shape)
```

```
EmployeeID  num unique values:  1340
Age  num unique values:  43
BusinessTravel  num unique values:  3
DailyRate  num unique values:  793
Department  num unique values:  3
DistanceFromHome  num unique values:  29
Education  num unique values:  5
EducationField  num unique values:  6
EmployeeCount  num unique values:  1
EnvironmentSatisfaction  num unique values:  4
Gender  num unique values:  2
HourlyRate  num unique values:  71
JobInvolvement  num unique values:  4
JobLevel  num unique values:  5
JobRole  num unique values:  5
JobSatisfaction  num unique values:  4
MaritalStatus  num unique values:  3
MonthlyIncome  num unique values:  1134
MonthlyRate  num unique values:  1191
NumCompaniesWorked  num unique values:  10
Over18  num unique values:  1
OverTime  num unique values:  2
PercentSalaryHike  num unique values:  15
PerformanceRating  num unique values:  2
RelationshipSatisfaction  num unique values:  4
StandardHours  num unique values:  1
Shift  num unique values:  4
TotalWorkingYears  num unique values:  40
```

```
TrainingTimesLastYear  num unique values:  7

WorkLifeBalance  num unique values:  4

YearsAtCompany  num unique values:  37

YearsInCurrentRole  num unique values:  19

YearsSinceLastPromotion  num unique values:  16

YearsWithCurrManager  num unique values:  18

(1340, 31)
(336, 31)
```

See that there are 9 categorical columns which need to be converted to numerical.

See that there are many numerical columns which need to be binned by quantile.

Dropped columns with only 1 unique value -> can't get any information from those.

Can see that there are no null values, so we do not need to clean out rows or columns containing nulls.

In [ ]:
```
# Binning
int_cols = []
for c in data.columns:
    if data[c].dtype == "int64":
        print(c, len(data[c].unique()))
        print(data[c].describe(), "\n\n\n")
```

```
EmployeeID 1340
count    1.340000e+03
mean     1.460265e+06
std      2.494821e+05
min      1.025177e+06
25%      1.237599e+06
50%      1.469862e+06
75%      1.670131e+06
max      1.886378e+06
Name: EmployeeID, dtype: float64
```

```
Age 43
count    1340.000000
mean       36.580597
std         9.013072
min        18.000000
25%        30.000000
50%        35.000000
75%        42.000000
max        60.000000
Name: Age, dtype: float64
```

```
DailyRate 793
count    1340.000000
mean      799.197761
std       399.333256
min       102.000000
25%       465.000000
50%       796.000000
75%      1153.000000
max      1499.000000
Name: DailyRate, dtype: float64
```

```
DistanceFromHome 29
count    1340.000000
mean        9.193284
std         8.141621
min         1.000000
25%         2.000000
50%         7.000000
75%        14.000000
max        29.000000
Name: DistanceFromHome, dtype: float64
```

```
Education 5
count    1340.000000
mean        2.924627
std         1.036088
```

```
min         1.000000
25%         2.000000
50%         3.000000
75%         4.000000
max         5.000000
Name: Education, dtype: float64
```

```
EnvironmentSatisfaction 4
count    1340.000000
mean        2.709701
std         1.099961
min         1.000000
25%         2.000000
50%         3.000000
75%         4.000000
max         4.000000
Name: EnvironmentSatisfaction, dtype: float64
```

```
HourlyRate 71
count    1340.000000
mean       65.559701
std        20.335025
min        30.000000
25%        48.000000
50%        65.000000
75%        83.000000
max       100.000000
Name: HourlyRate, dtype: float64
```

```
JobInvolvement 4
count    1340.000000
mean        2.717910
std         0.717523
min         1.000000
25%         2.000000
50%         3.000000
75%         3.000000
max         4.000000
Name: JobInvolvement, dtype: float64
```

```
JobLevel 5
count    1340.000000
mean        2.051493
std         1.104491
min         1.000000
25%         1.000000
50%         2.000000
75%         3.000000
```

```
max         5.000000
Name: JobLevel, dtype: float64
```

```
JobSatisfaction 4
count    1340.000000
mean        2.746269
std         1.111328
min         1.000000
25%         2.000000
50%         3.000000
75%         4.000000
max         4.000000
Name: JobSatisfaction, dtype: float64
```

```
MonthlyIncome 1134
count     1340.000000
mean      6433.381343
std       4687.058380
min       1051.000000
25%       2870.000000
50%       4876.500000
75%       8038.750000
max      19973.000000
Name: MonthlyIncome, dtype: float64
```

```
MonthlyRate 1191
count     1340.000000
mean     14290.377612
std       7166.995911
min       2094.000000
25%       7967.250000
50%      14288.500000
75%      20472.500000
max      26997.000000
Name: MonthlyRate, dtype: float64
```

```
NumCompaniesWorked 10
count    1340.000000
mean        2.600000
std         2.472794
min         0.000000
25%         1.000000
50%         1.000000
75%         4.000000
max         9.000000
Name: NumCompaniesWorked, dtype: float64
```

```
PercentSalaryHike 15
count    1340.000000
mean       15.168657
std         3.661956
min        11.000000
25%        12.000000
50%        14.000000
75%        18.000000
max        25.000000
Name: PercentSalaryHike, dtype: float64
```

```
PerformanceRating 2
count    1340.000000
mean        3.152239
std         0.359386
min         3.000000
25%         3.000000
50%         3.000000
75%         3.000000
max         4.000000
Name: PerformanceRating, dtype: float64
```

```
RelationshipSatisfaction 4
count    1340.000000
mean        2.700000
std         1.079858
min         1.000000
25%         2.000000
50%         3.000000
75%         4.000000
max         4.000000
Name: RelationshipSatisfaction, dtype: float64
```

```
Shift 4
count    1340.000000
mean        0.808209
std         0.856251
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         3.000000
Name: Shift, dtype: float64
```

```
TotalWorkingYears 40
count    1340.000000
mean       11.222388
```

```
std         7.696043
min         0.000000
25%         6.000000
50%        10.000000
75%        15.000000
max        40.000000
Name: TotalWorkingYears, dtype: float64
```

```
TrainingTimesLastYear 7
count    1340.000000
mean        2.785821
std         1.263473
min         0.000000
25%         2.000000
50%         3.000000
75%         3.000000
max         6.000000
Name: TrainingTimesLastYear, dtype: float64
```

```
WorkLifeBalance 4
count    1340.000000
mean        2.771642
std         0.700007
min         1.000000
25%         2.000000
50%         3.000000
75%         3.000000
max         4.000000
Name: WorkLifeBalance, dtype: float64
```

```
YearsAtCompany 37
count    1340.000000
mean        7.070149
std         6.039663
min         0.000000
25%         3.000000
50%         5.000000
75%        10.000000
max        40.000000
Name: YearsAtCompany, dtype: float64
```

```
YearsInCurrentRole 19
count    1340.000000
mean        4.272388
std         3.677798
min         0.000000
25%         2.000000
50%         3.000000
```

```
75%         7.000000
max        18.000000
Name: YearsInCurrentRole, dtype: float64
```

```
YearsSinceLastPromotion 16
count    1340.000000
mean        2.175373
std         3.222376
min         0.000000
25%         0.000000
50%         1.000000
75%         3.000000
max        15.000000
Name: YearsSinceLastPromotion, dtype: float64
```

```
YearsWithCurrManager 18
count    1340.000000
mean        4.167164
std         3.581605
min         0.000000
25%         2.000000
50%         3.000000
75%         7.000000
max        17.000000
Name: YearsWithCurrManager, dtype: float64
```

In [ ]:
```
# Binning
# found manually
cols_to_bin = ["Age", "DailyRate", "DistanceFromHome",
               "HourlyRate", "MonthlyIncome", "MonthlyRate",
               "PercentSalaryHike", "TotalWorkingYears",
               "YearsAtCompany","YearsInCurrentRole",
               "YearsWithCurrManager", "NumCompaniesWorked",
               "YearsSinceLastPromotion",]

print(data.shape)
print(submission_data.shape)

# uneven 4 groups
for c in cols_to_bin:
    try:
        data[c+"_Even"] = pd.cut(data[c], 4, labels=False)
        submission_data[c+"_Even"] = pd.cut(submission_data[c], 4, labels=Fa
    except:
        print("failed")
        pass
```

```
print(data.shape)
print(submission_data.shape)
```

```
(1340, 31)
(336, 31)
(1340, 44)
(336, 44)
```

In [ ]:
```
# get dummies from int categorical data

# found manually
already_categorical = ["Education", "EnvironmentSatisfaction",
                       "JobInvolvement", "JobLevel", "JobSatisfaction",
                       "PerformanceRating", "RelationshipSatisfaction",
                       "Shift", "TrainingTimesLastYear",
                       "WorkLifeBalance"]

for c in already_categorical:
    temp_dummy = pd.get_dummies(data[c], prefix=c)
    data = pd.concat([data, temp_dummy], axis=1)

    sub_temp_dummy = pd.get_dummies(submission_data[c], prefix=c)
    submission_data = pd.concat([submission_data, sub_temp_dummy], axis=1)

print(data.shape)
print(submission_data.shape)
```

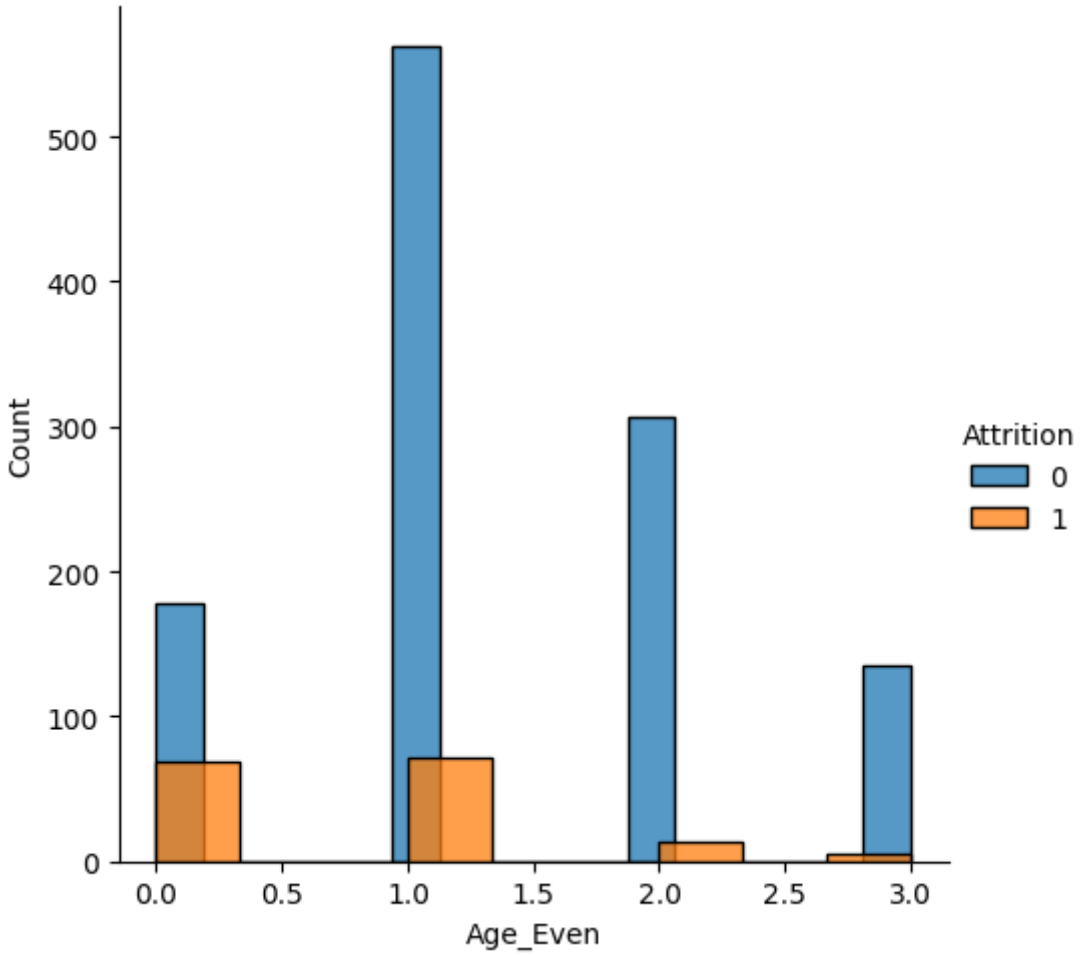
```
(1340, 87)
(336, 87)
```

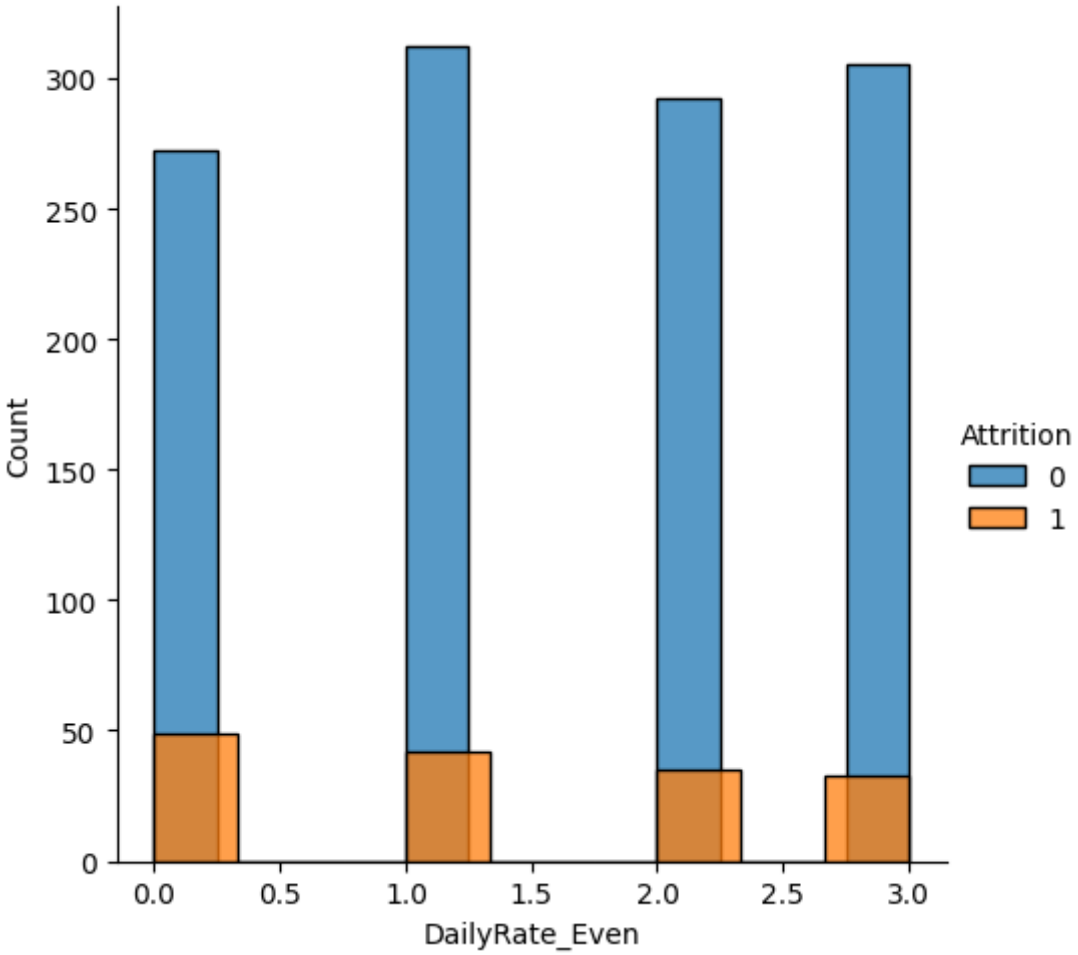
In [ ]:
```
# get dummies from obj categorical data
for c in data.columns:
    if data[c].dtype == "object":
        if len(data[c].unique()) == 2:
            data[c] = pd.factorize(data[c])[0]

            submission_data[c] = pd.factorize(submission_data[c])[0]
        else:
            temp_dummy = pd.get_dummies(data[c], prefix=c)
            data = pd.concat([data, temp_dummy], axis=1)
            data[c] = pd.factorize(data[c])[0]

            sub_temp_dummy = pd.get_dummies(submission_data[c], prefix=c)
            submission_data = pd.concat([submission_data, sub_temp_dummy], a
            submission_data[c] = pd.factorize(submission_data[c])[0]

print(data.shape)
print(submission_data.shape)

# turn label column to binary
labels = pd.DataFrame(pd.factorize(labels)[0], columns=["Attrition"])
```

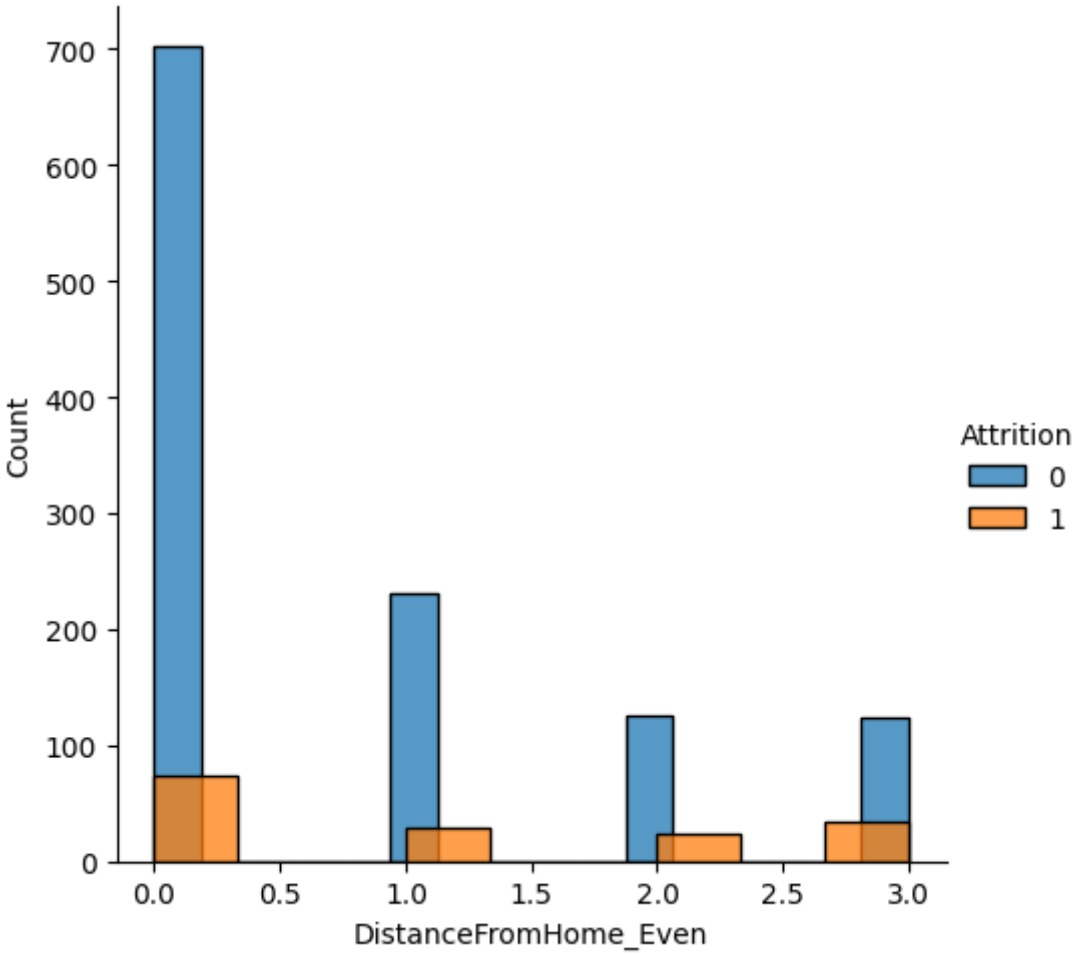
```
(1340, 107)
(336, 107)
```

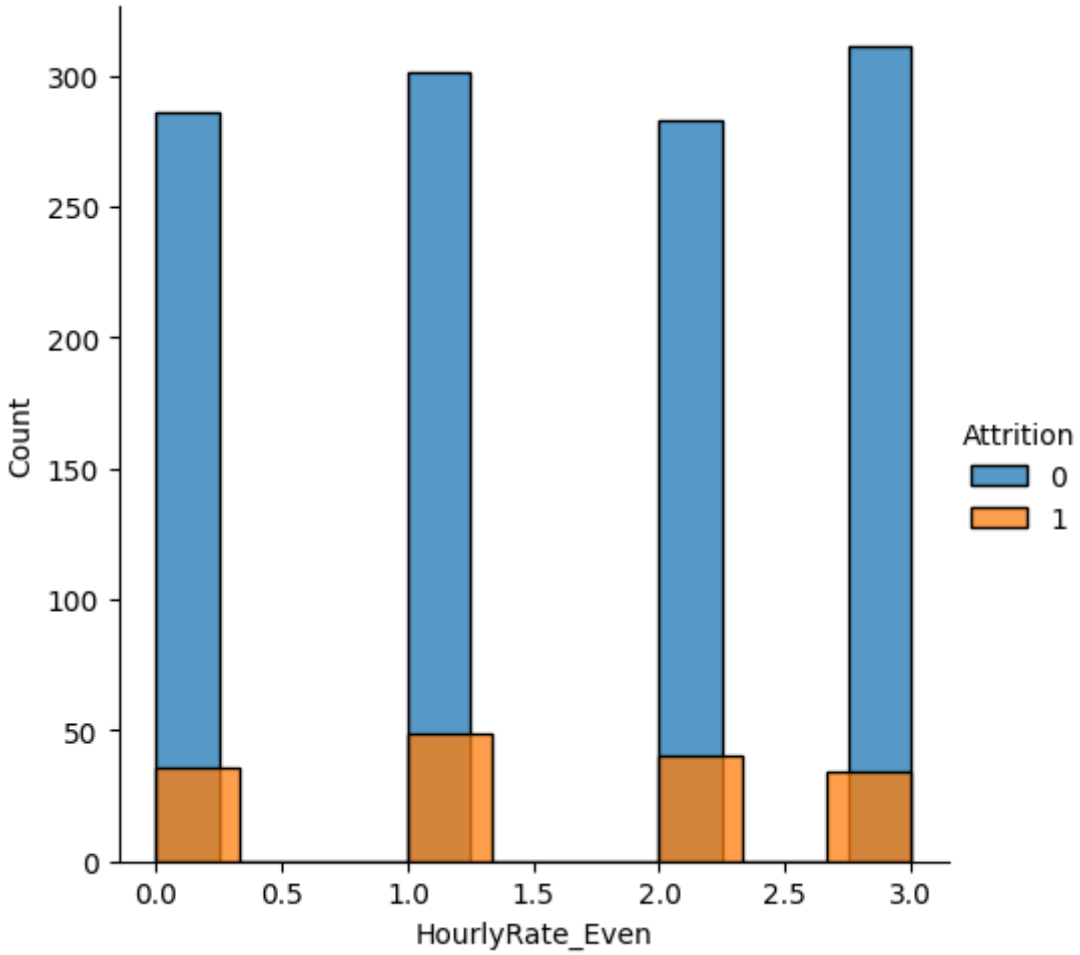
In [ ]:
```
# Histograms for binned data and dummmy data created
promising_data2 = data[data.columns[31:]]
```

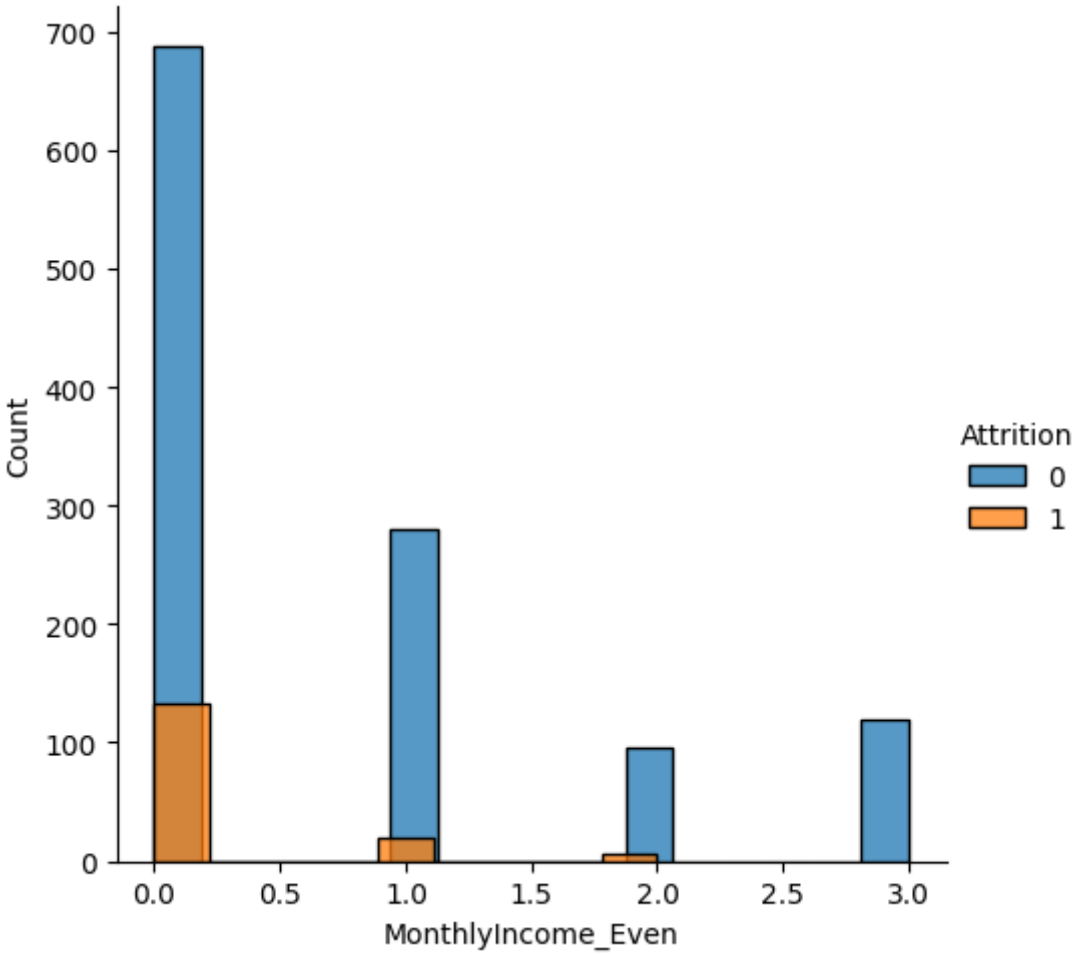
```
promising_data2 = pd.concat([promising_data2, labels], axis=1)

for c in promising_data2.columns:
    sns.FacetGrid(promising_data2,
                  hue="Attrition",
                  height= 5).map(sns.histplot,c).add_legend()
```

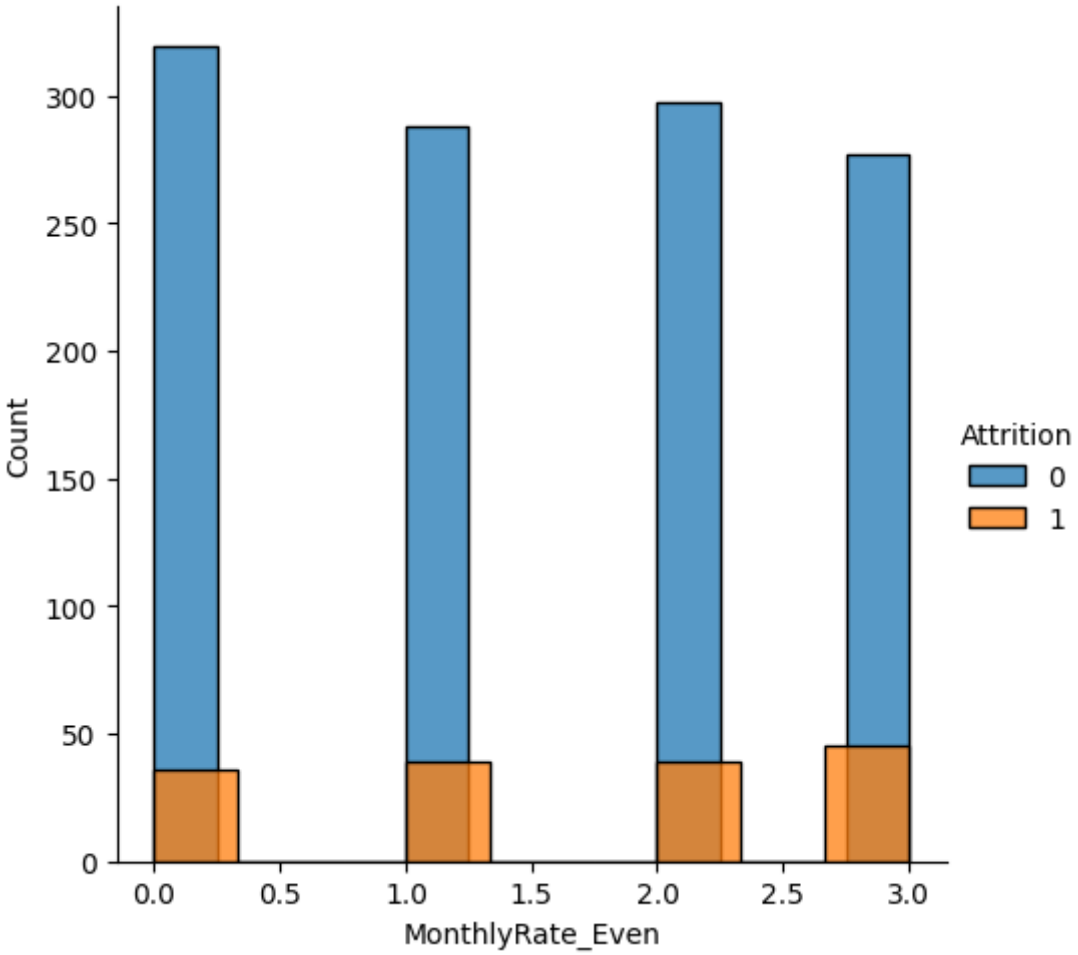


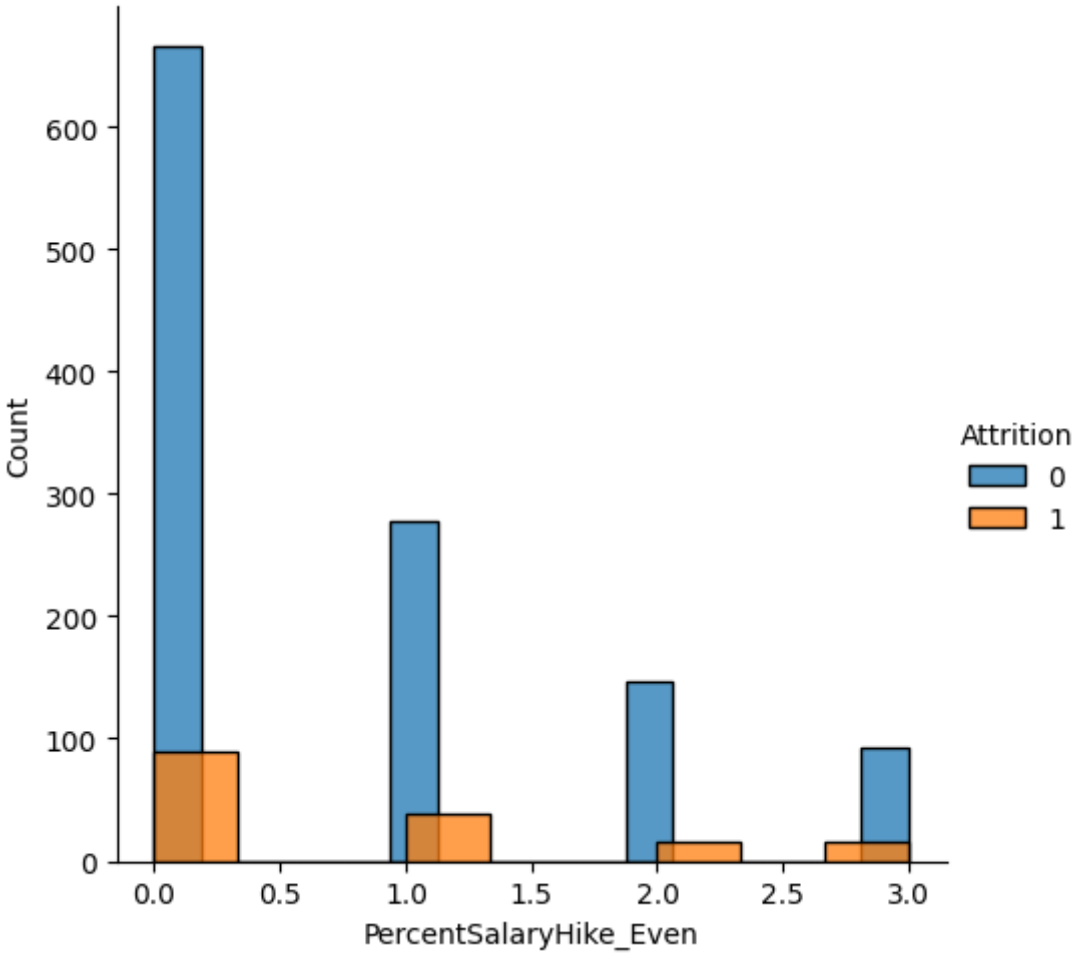

300
250
200
150
100
50
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
DailyRate_Even
Attrition
0
1

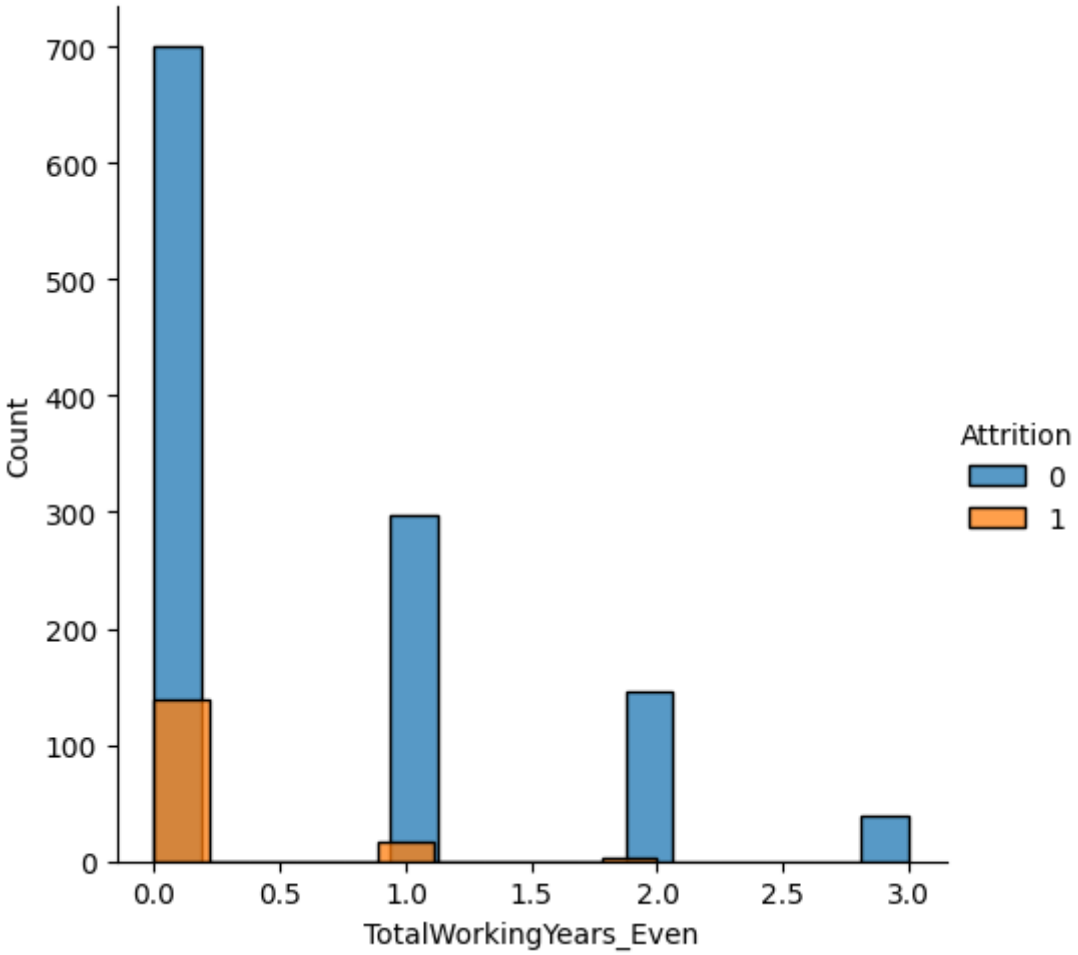

700
600
500
400
300
200
100
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
DistanceFromHome_Even
Attrition
0
1

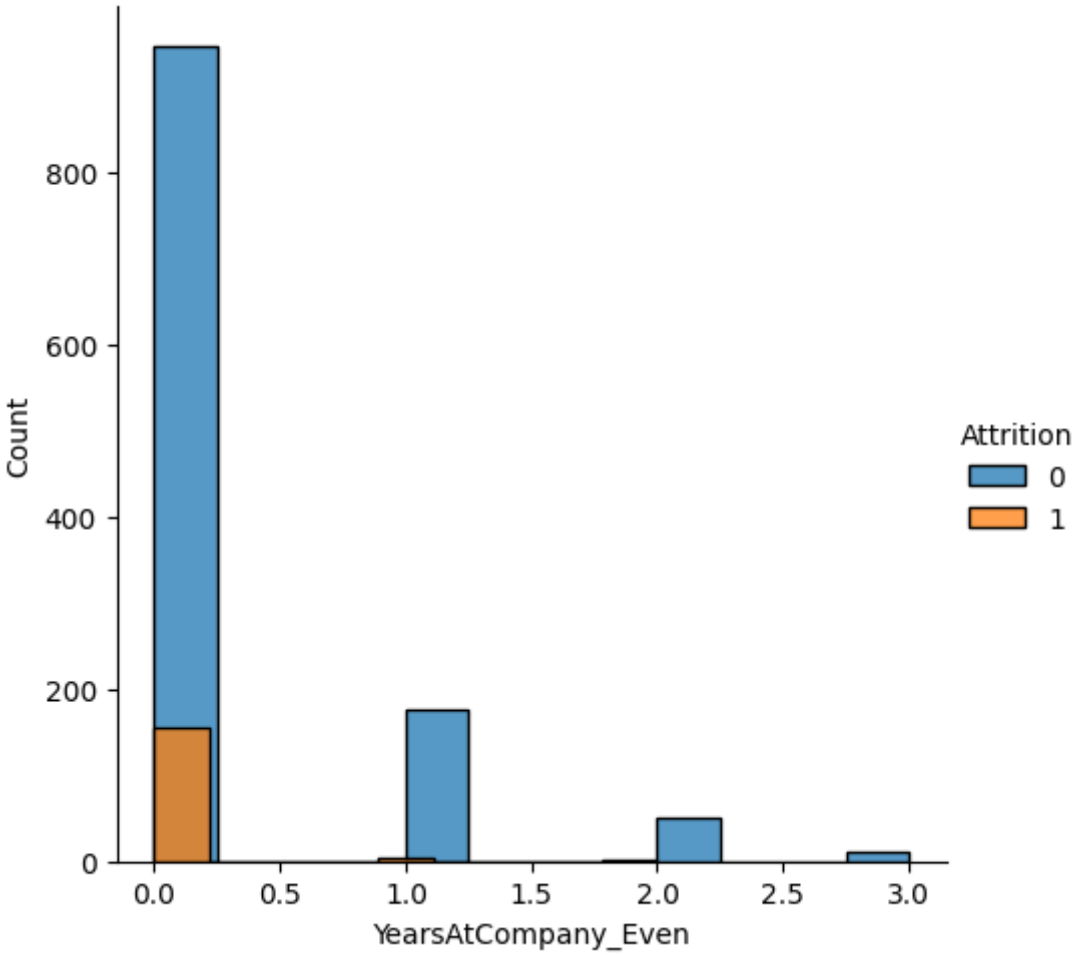

300
250
200
150
100
50
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
HourlyRate_Even
Attrition
0
1

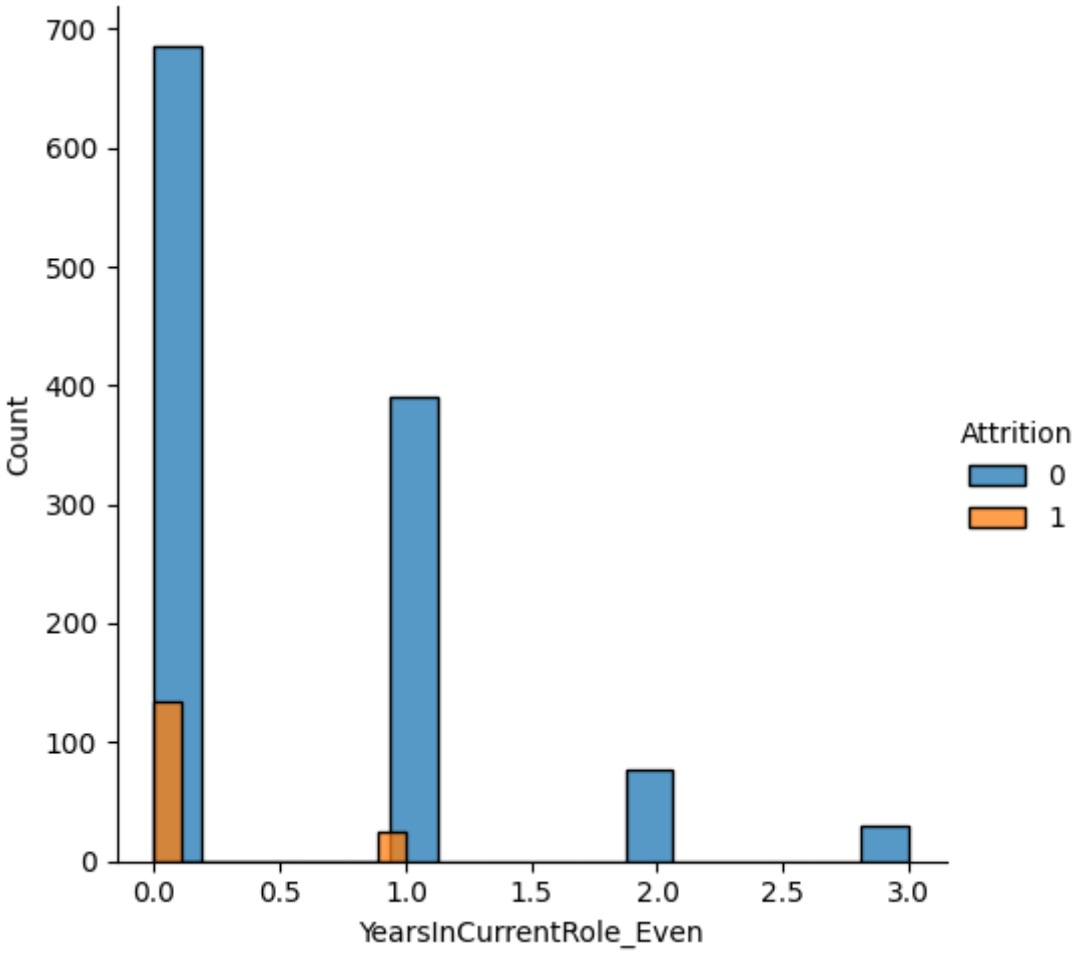

700
600
500
400
300
200
100
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
MonthlyIncome_Even
Attrition
0
1

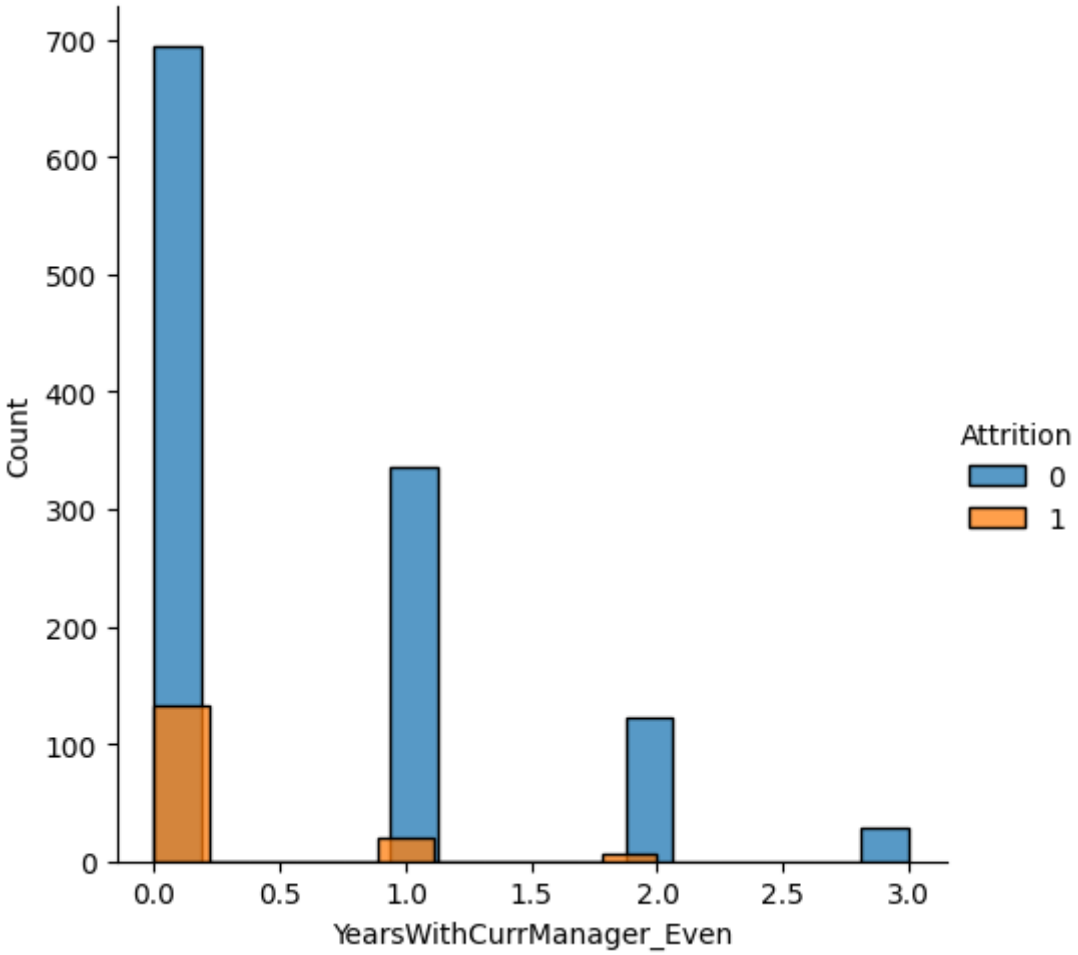

300
250
200
150
100
50
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
MonthlyRate_Even
Attrition
0
1

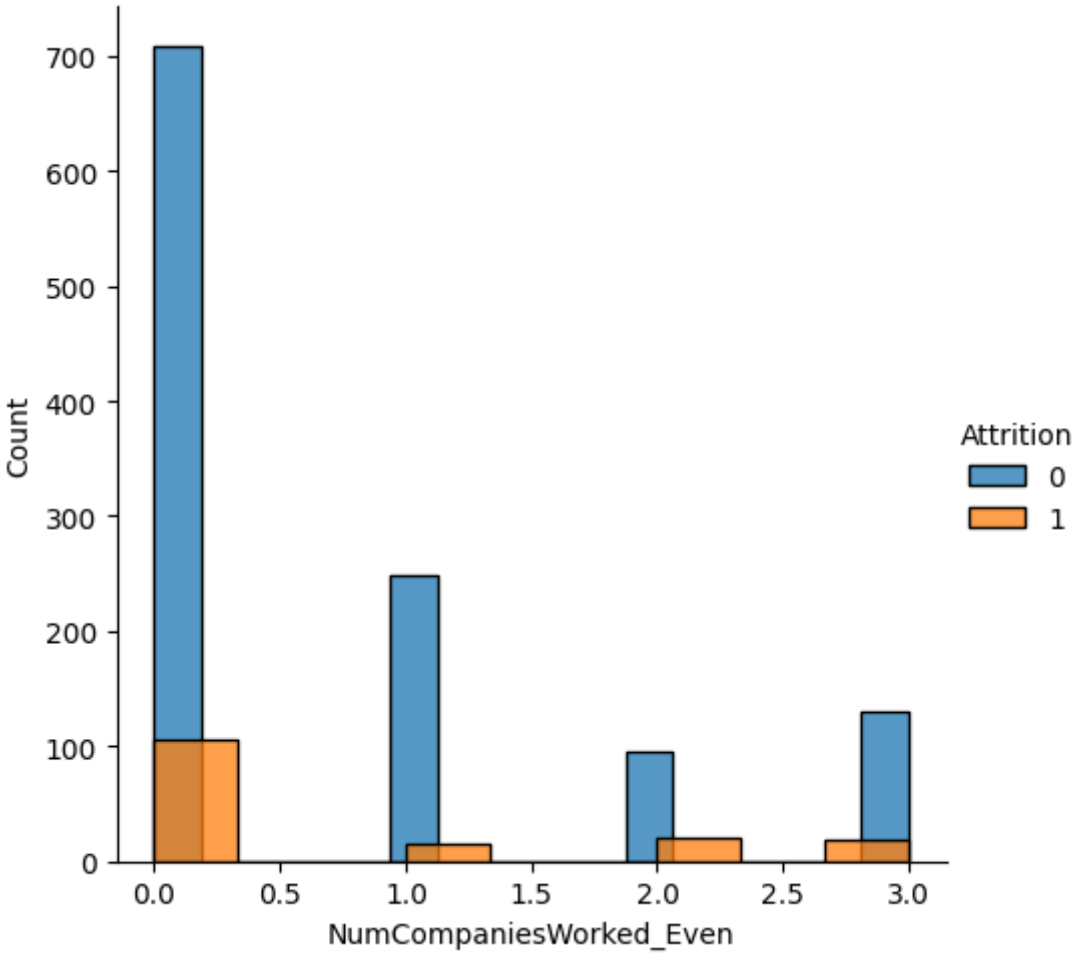

600
500
400
300
200
100
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
PercentSalaryHike_Even
Attrition
0
1

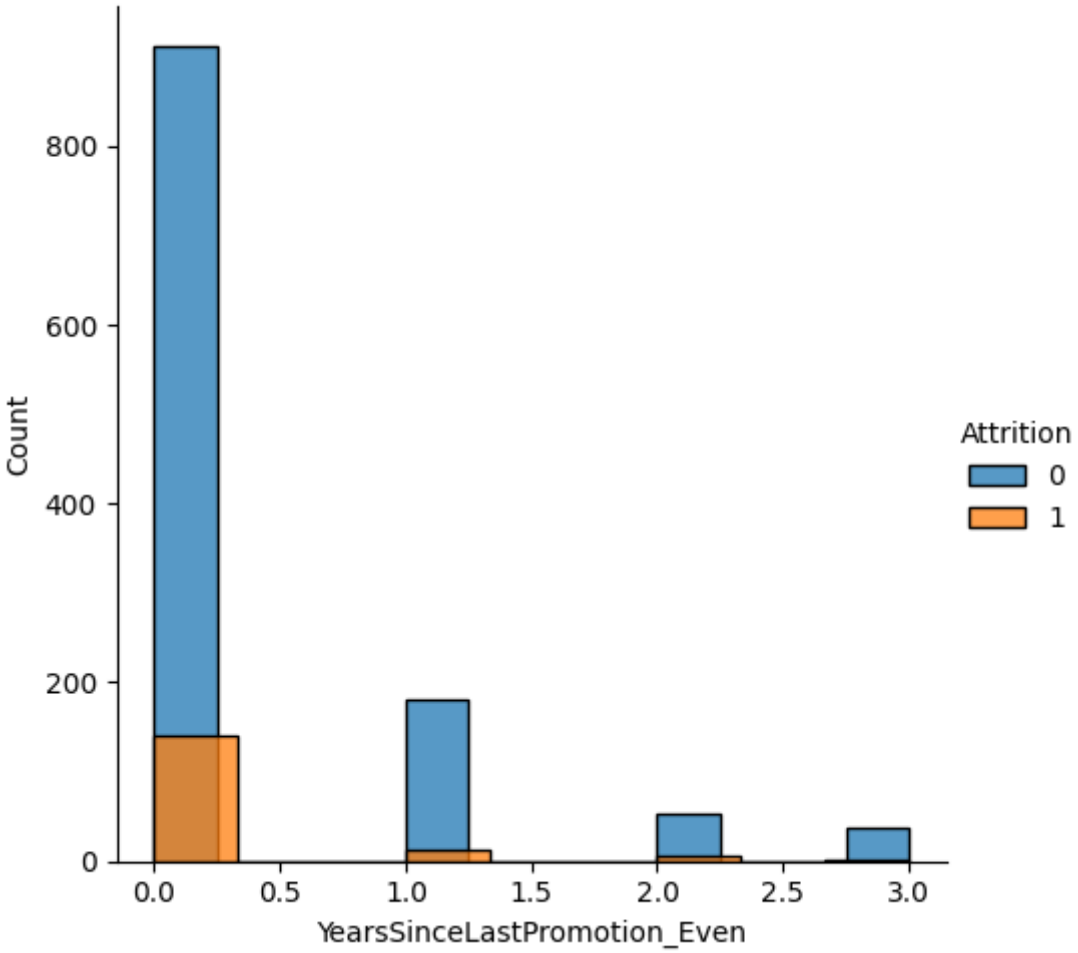

700
600
500
400
300
200
100
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
TotalWorkingYears_Even
Attrition
0
1

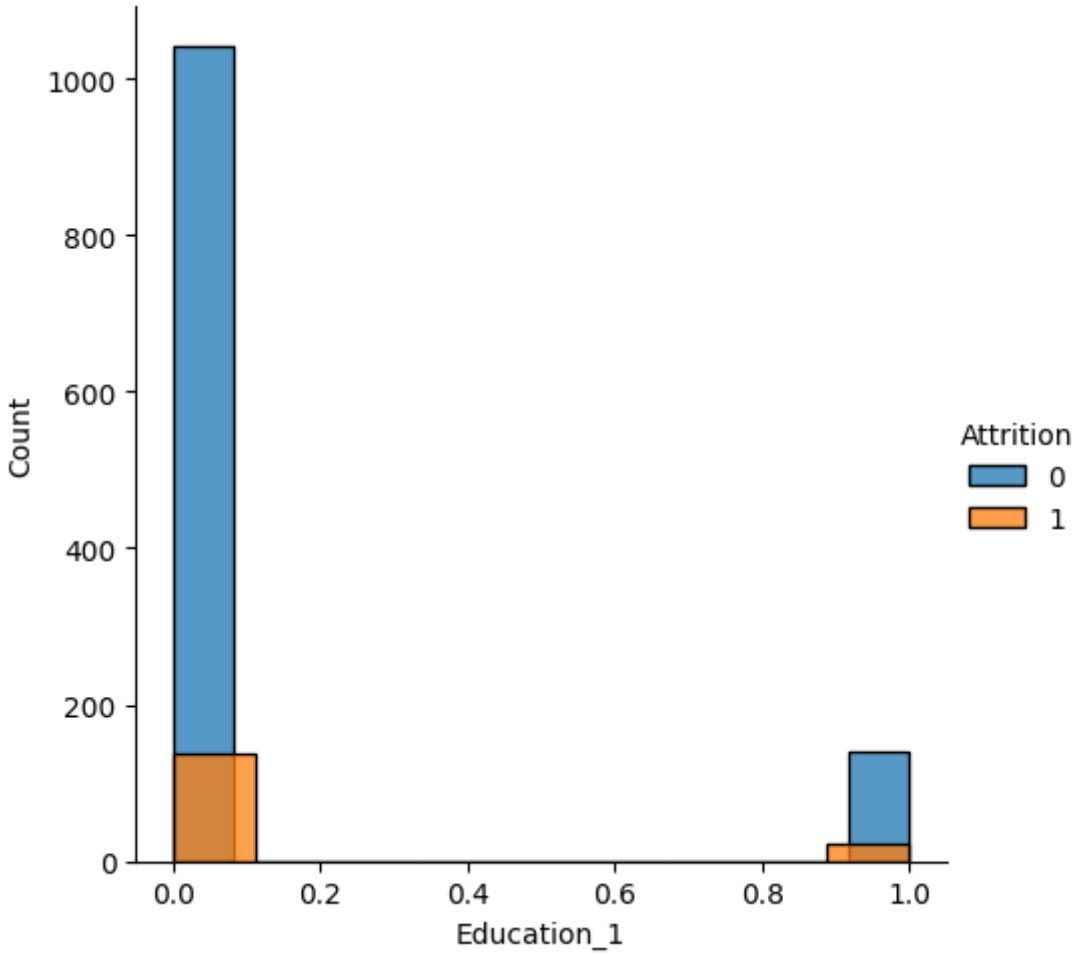

800
600
400
200
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
YearsAtCompany_Even
Attrition
0
1

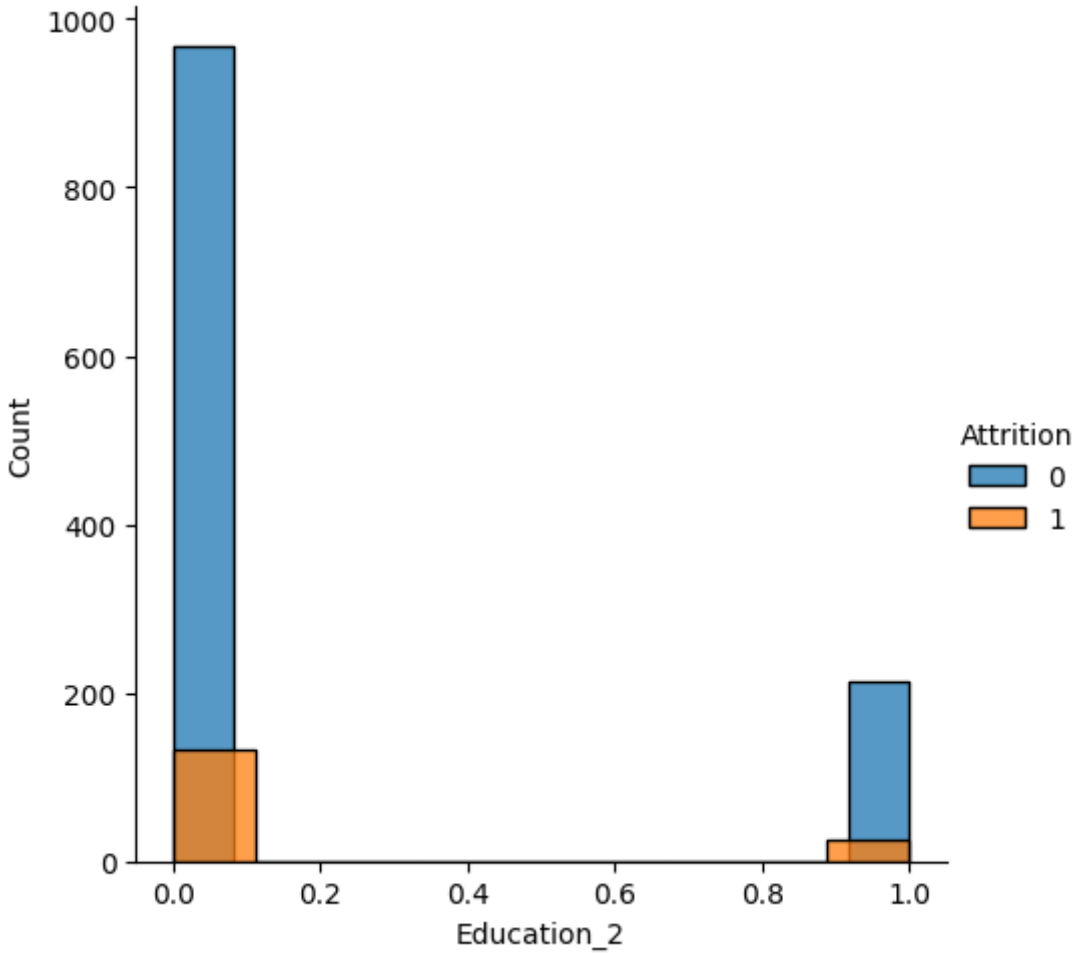

700
600
500
400
300
200
100
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
YearsInCurrentRole_Even
Attrition
0
1

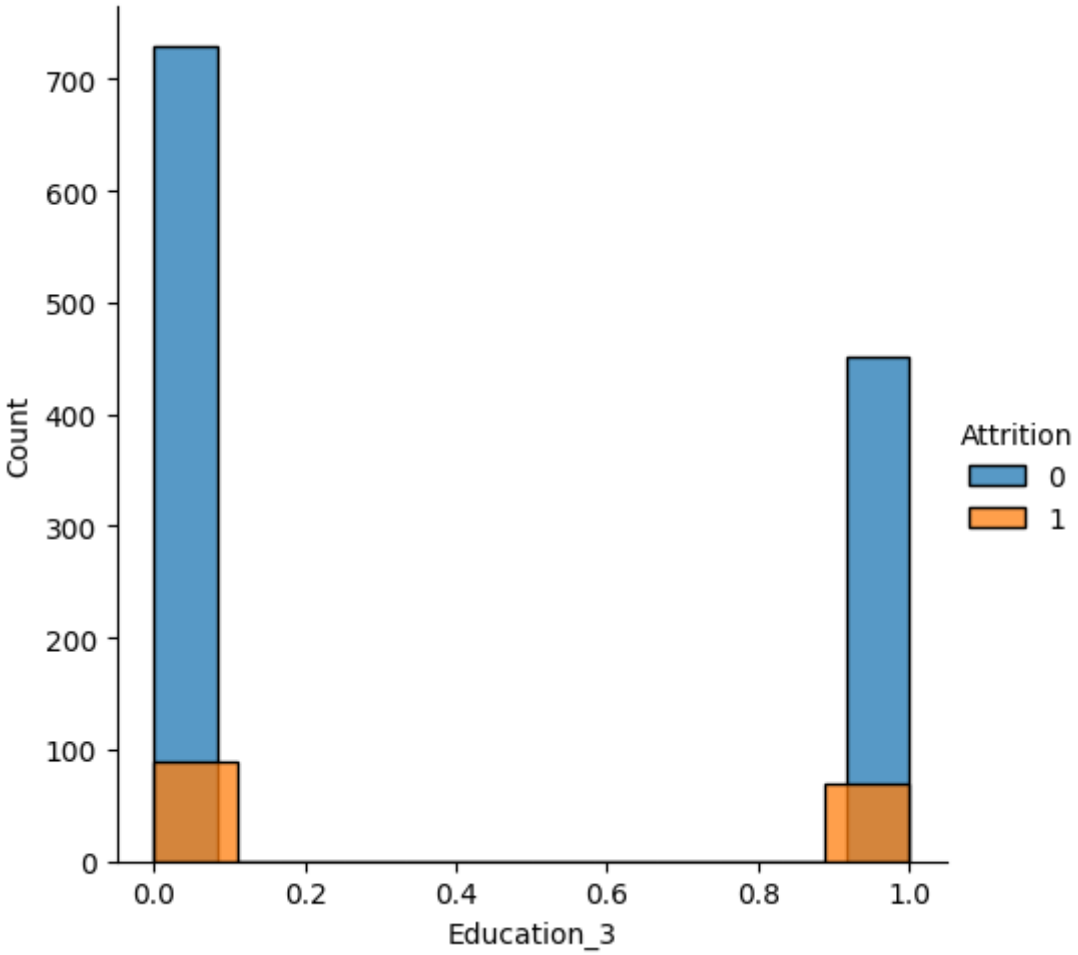

700
600
500
400
300
200
100
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
YearsWithCurrManager_Even
Attrition
0
1

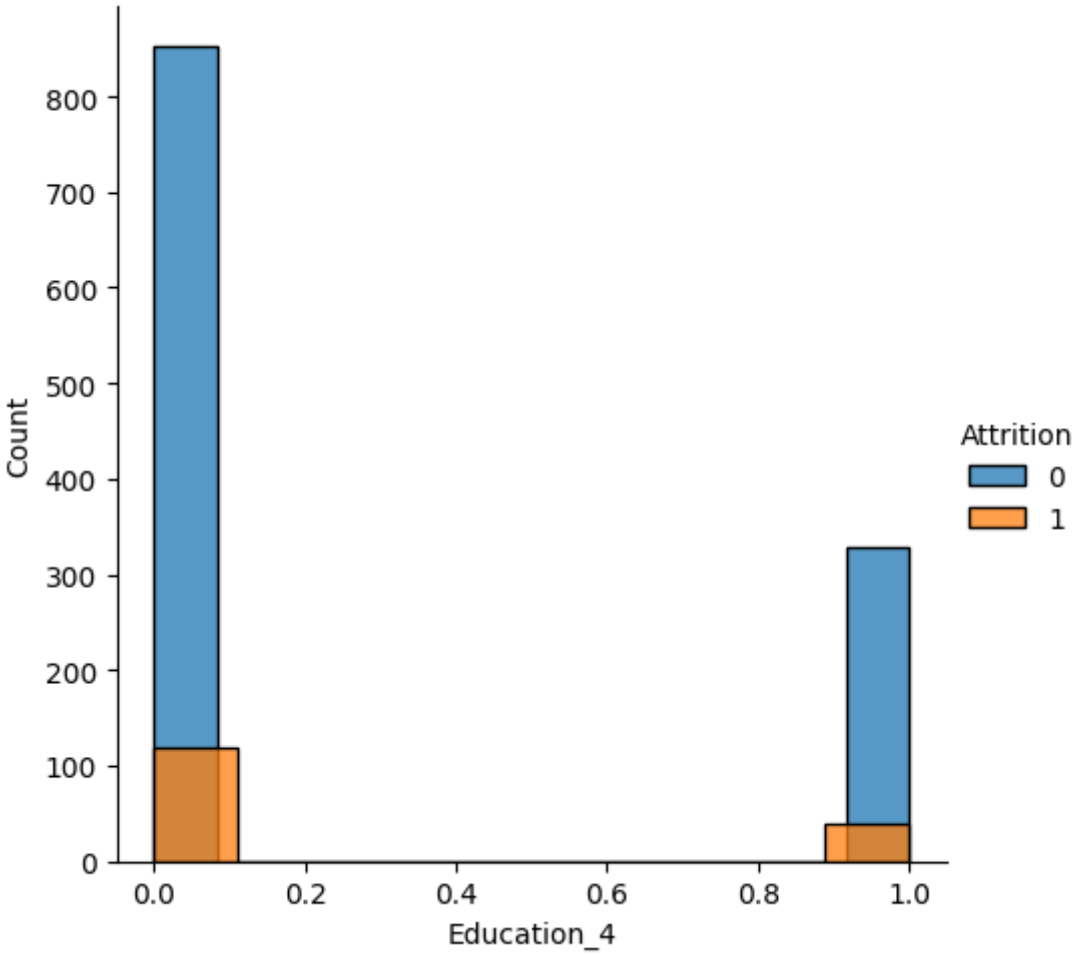

700
600
500
400
300
200
100
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
NumCompaniesWorked_Even
Attrition
0
1

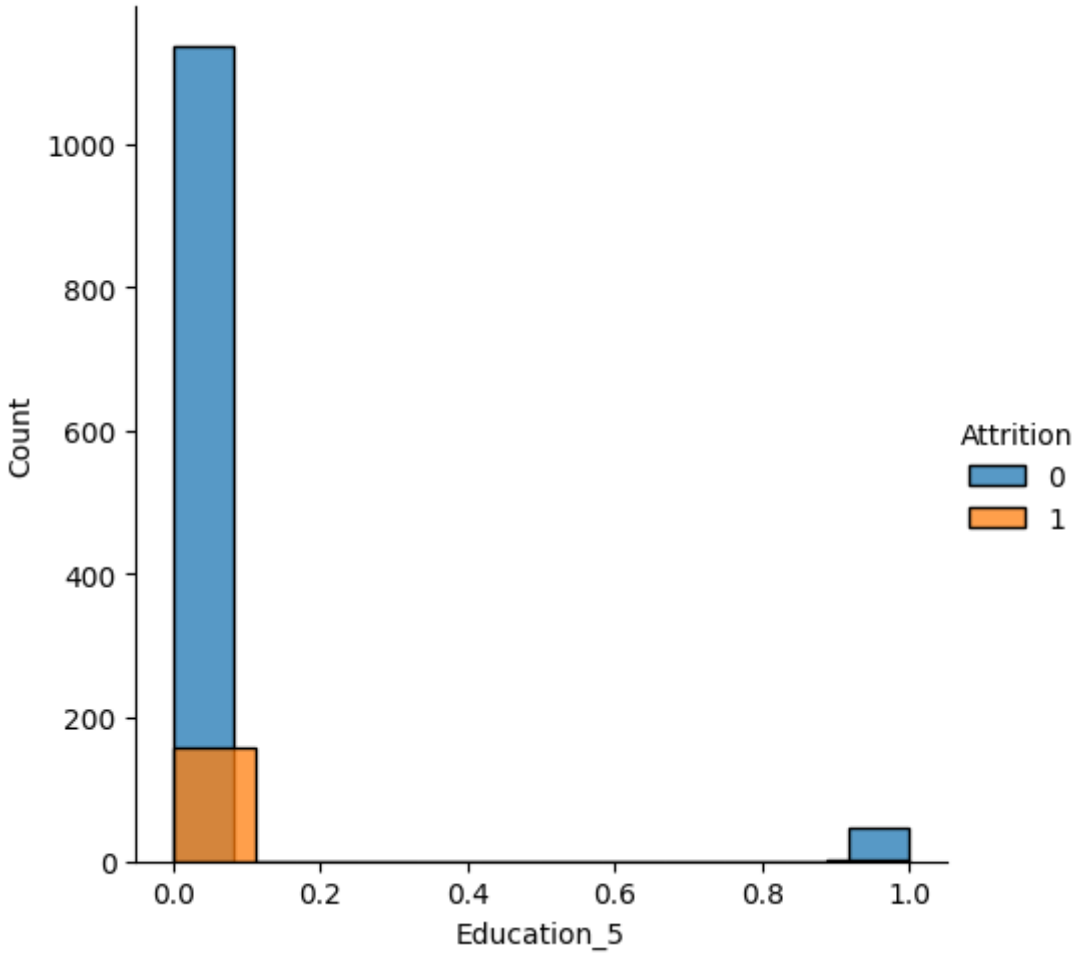

800
600
400
200
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
YearsSinceLastPromotion_Even
Attrition
0
1

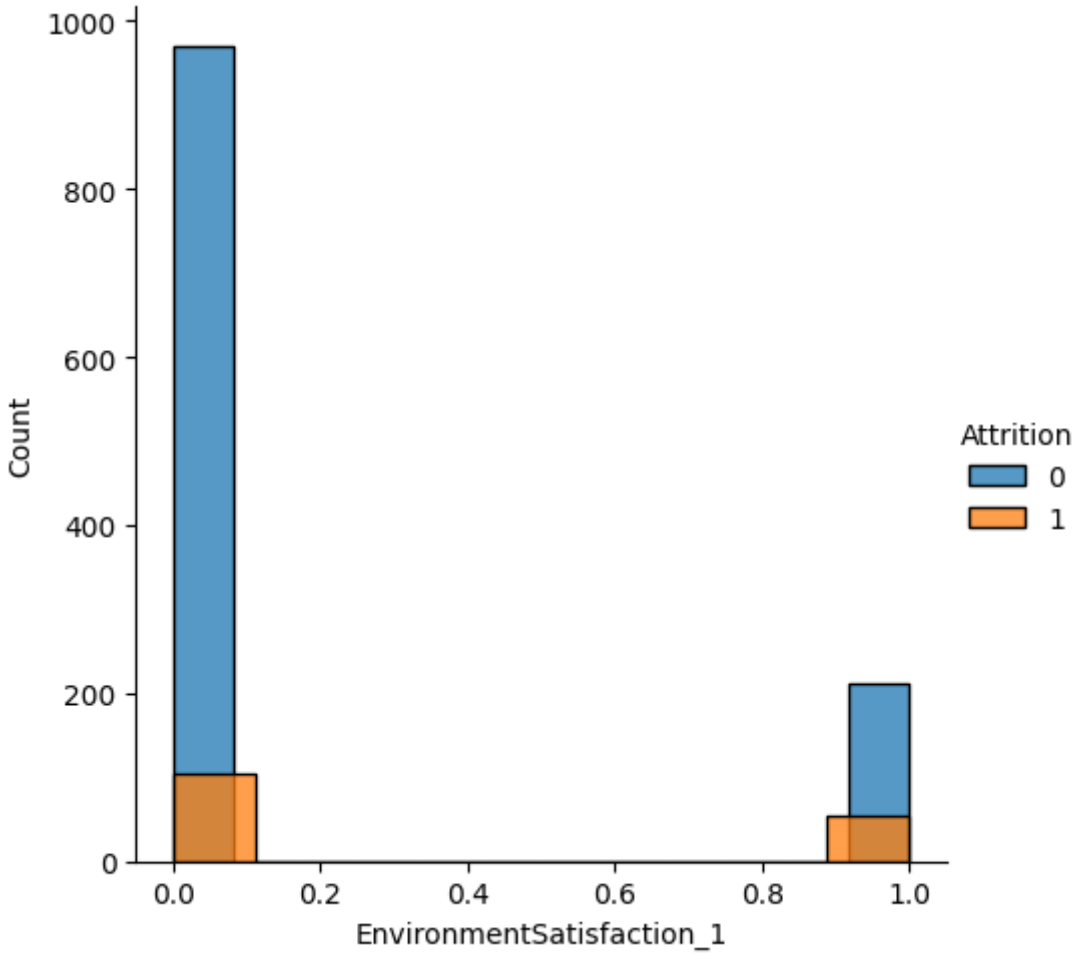

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Education_1
Attrition
0
1

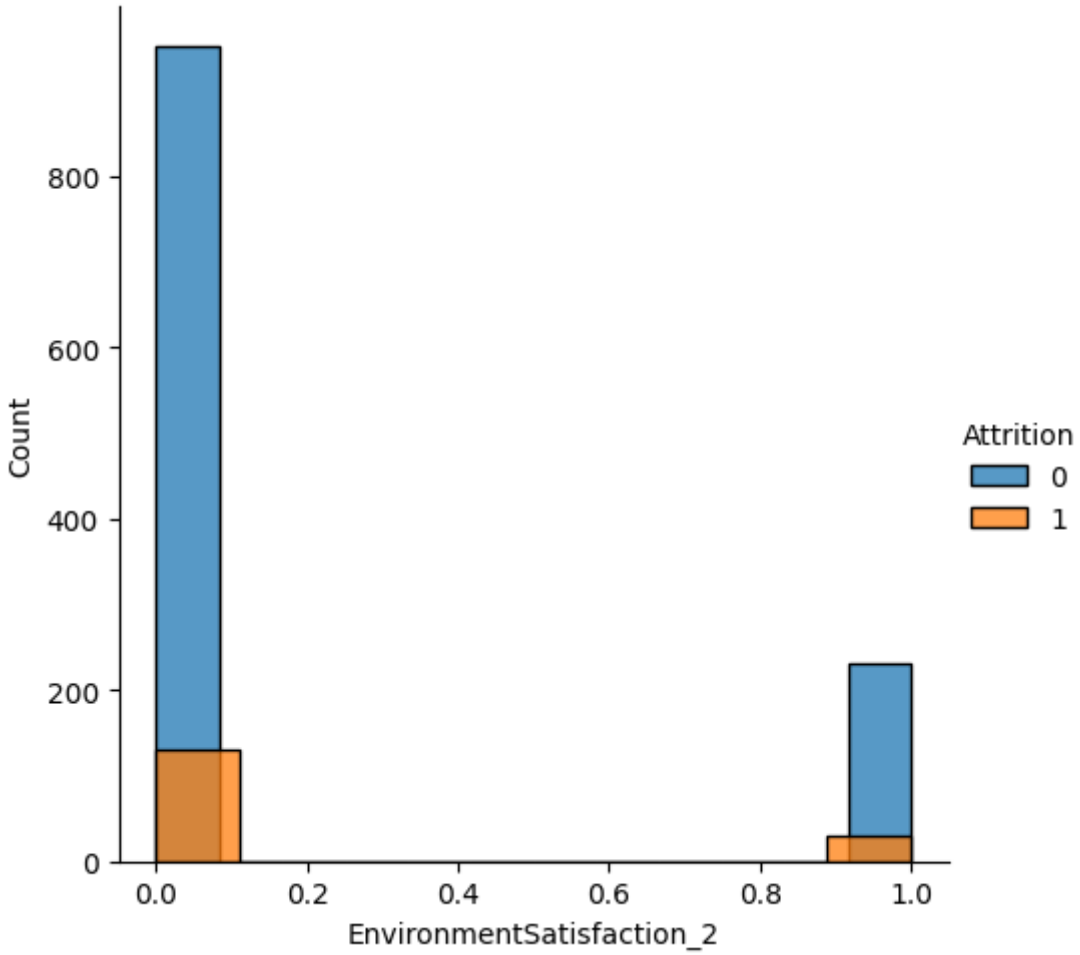

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Education_2
Attrition
0
1

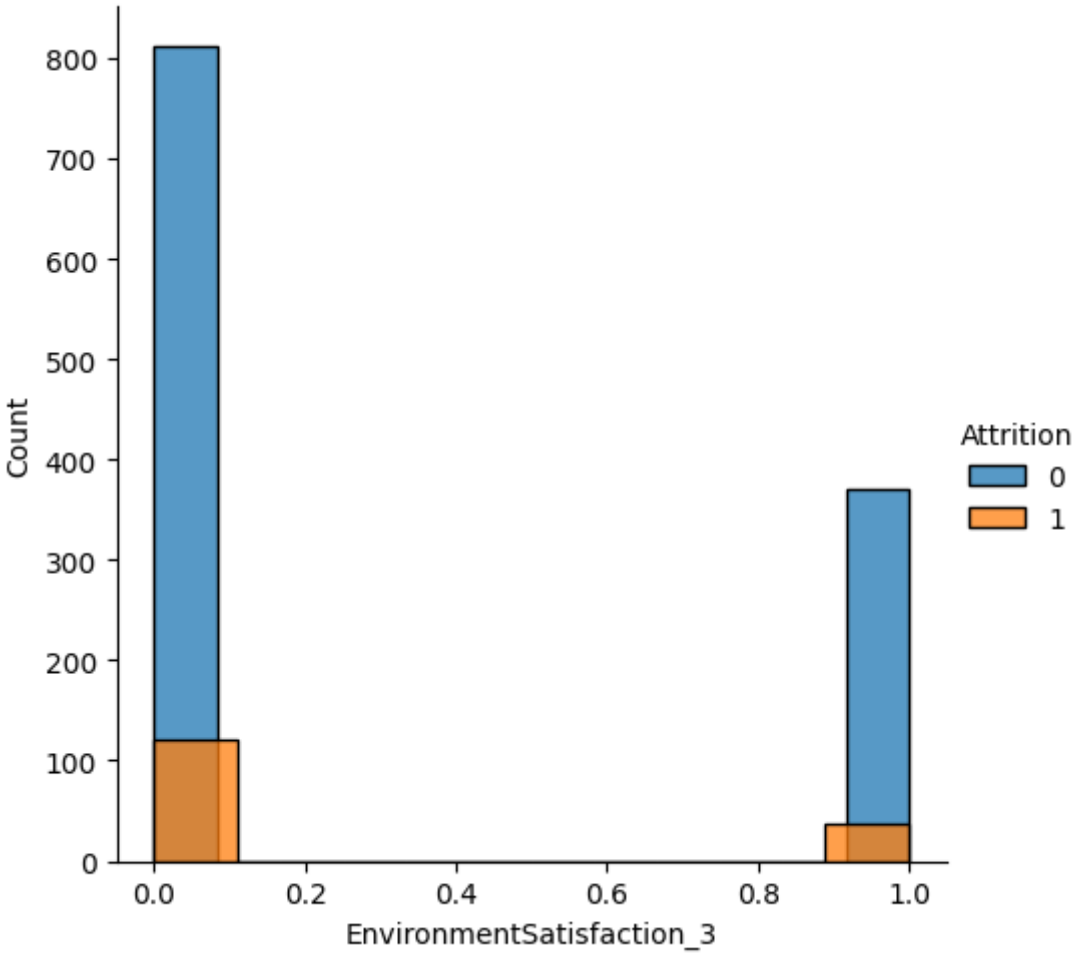

700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Education_3
Attrition
0
1

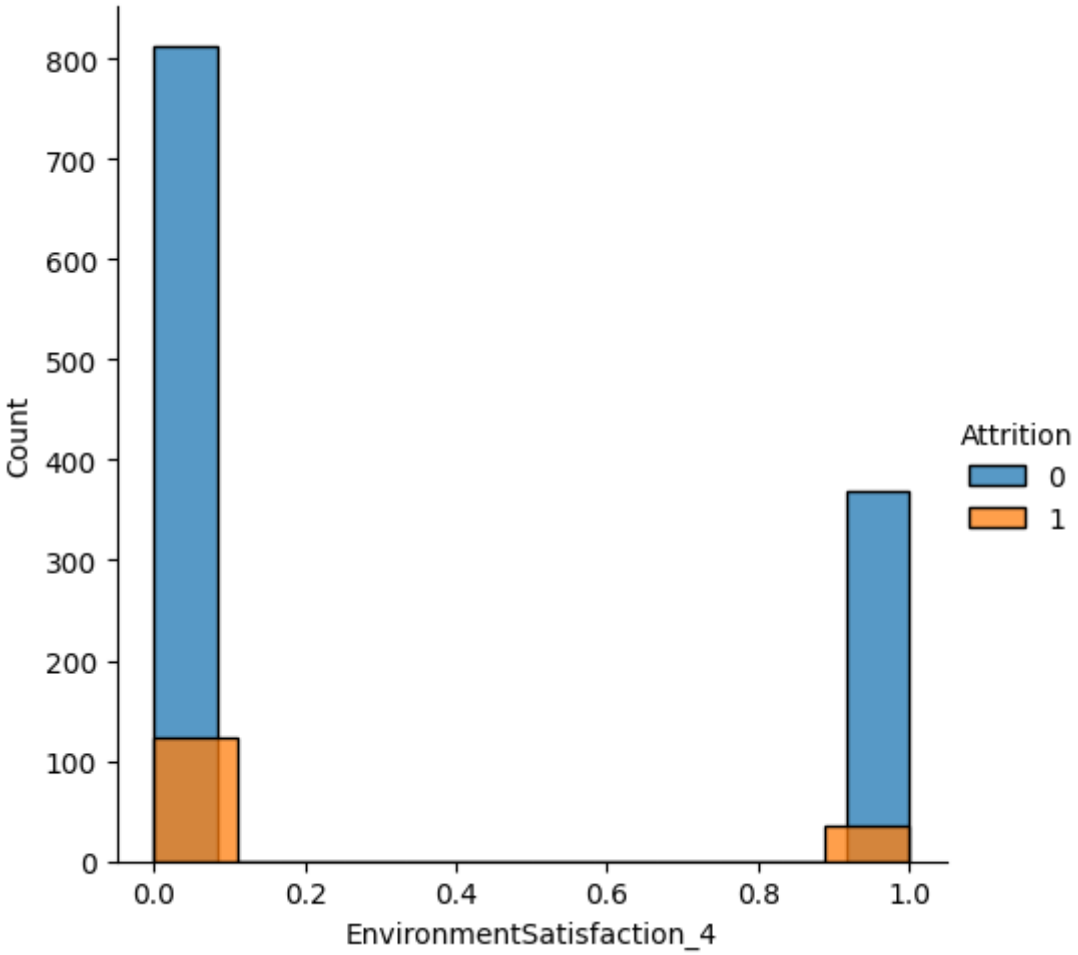

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Education_4
Attrition
0
1

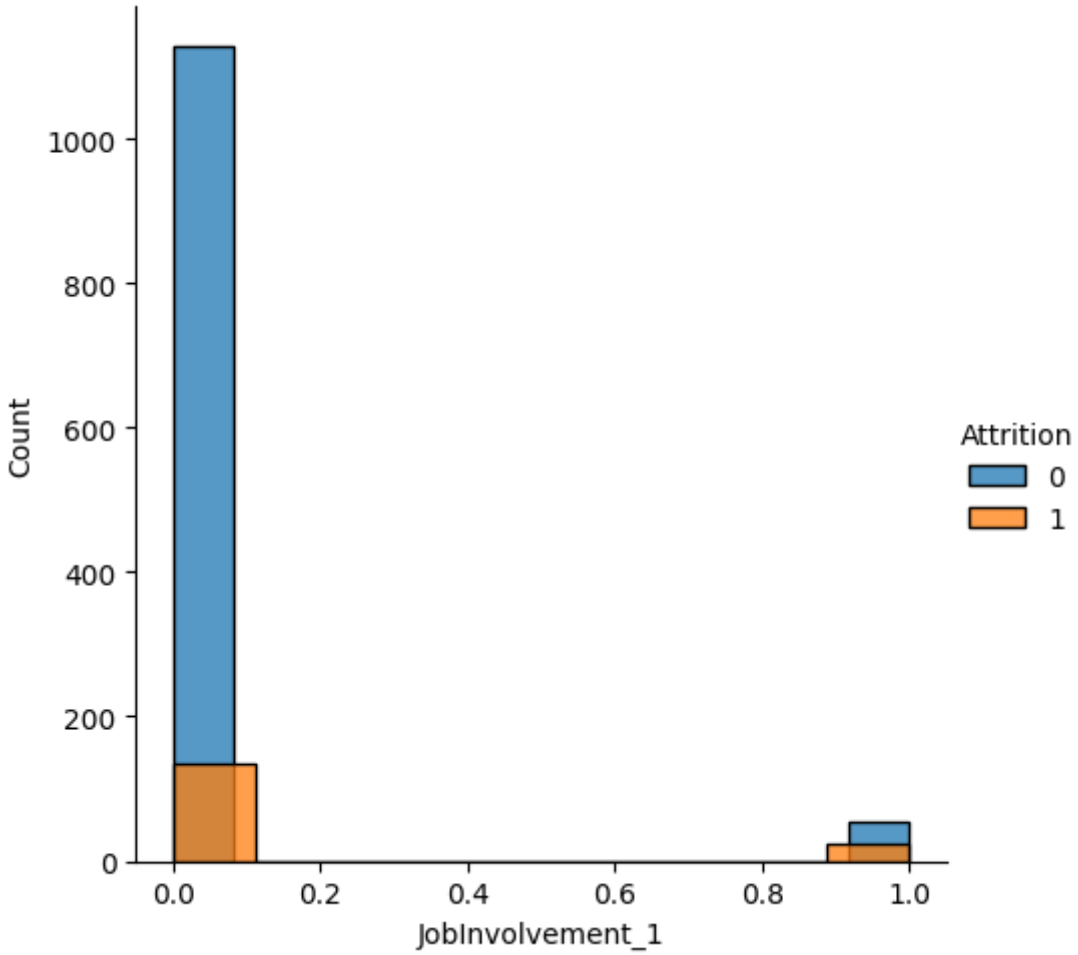

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Education_5
Attrition
0
1

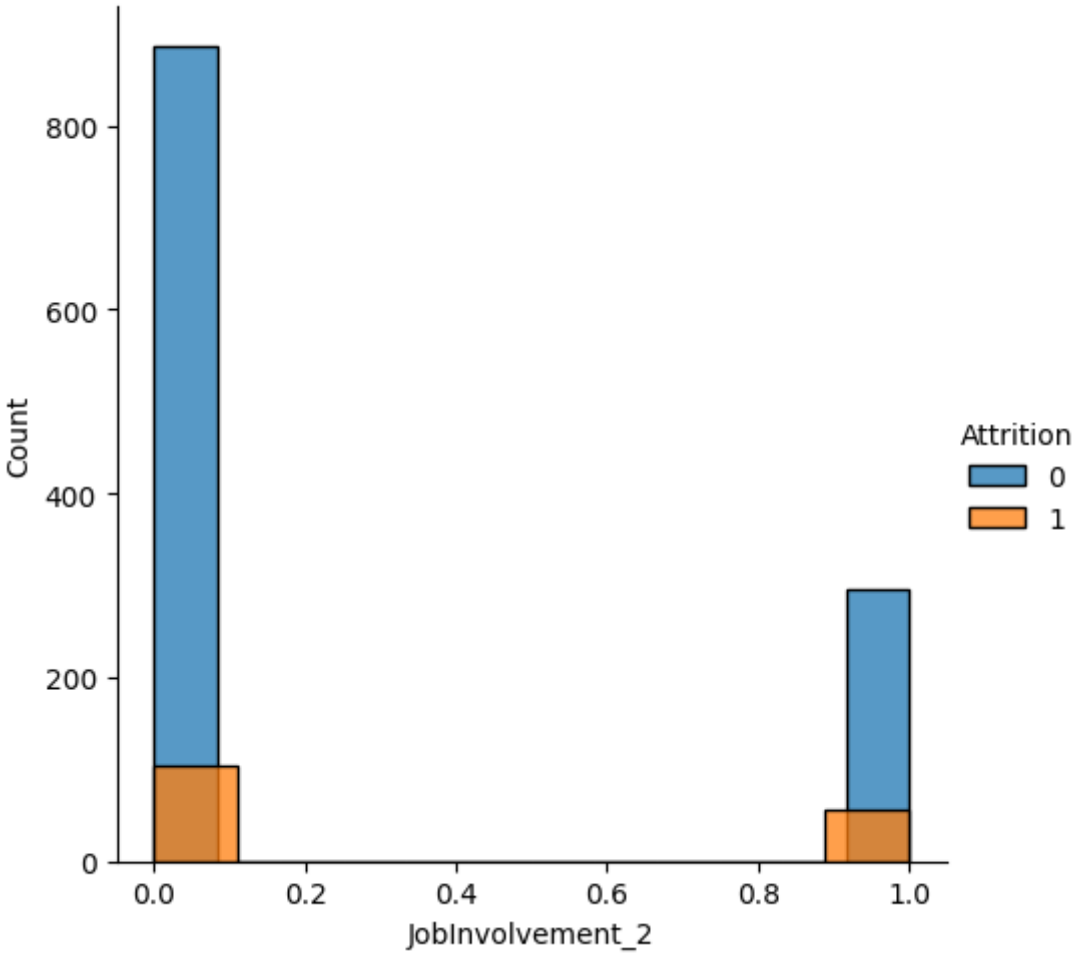

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
EnvironmentSatisfaction_1
Attrition
0
1

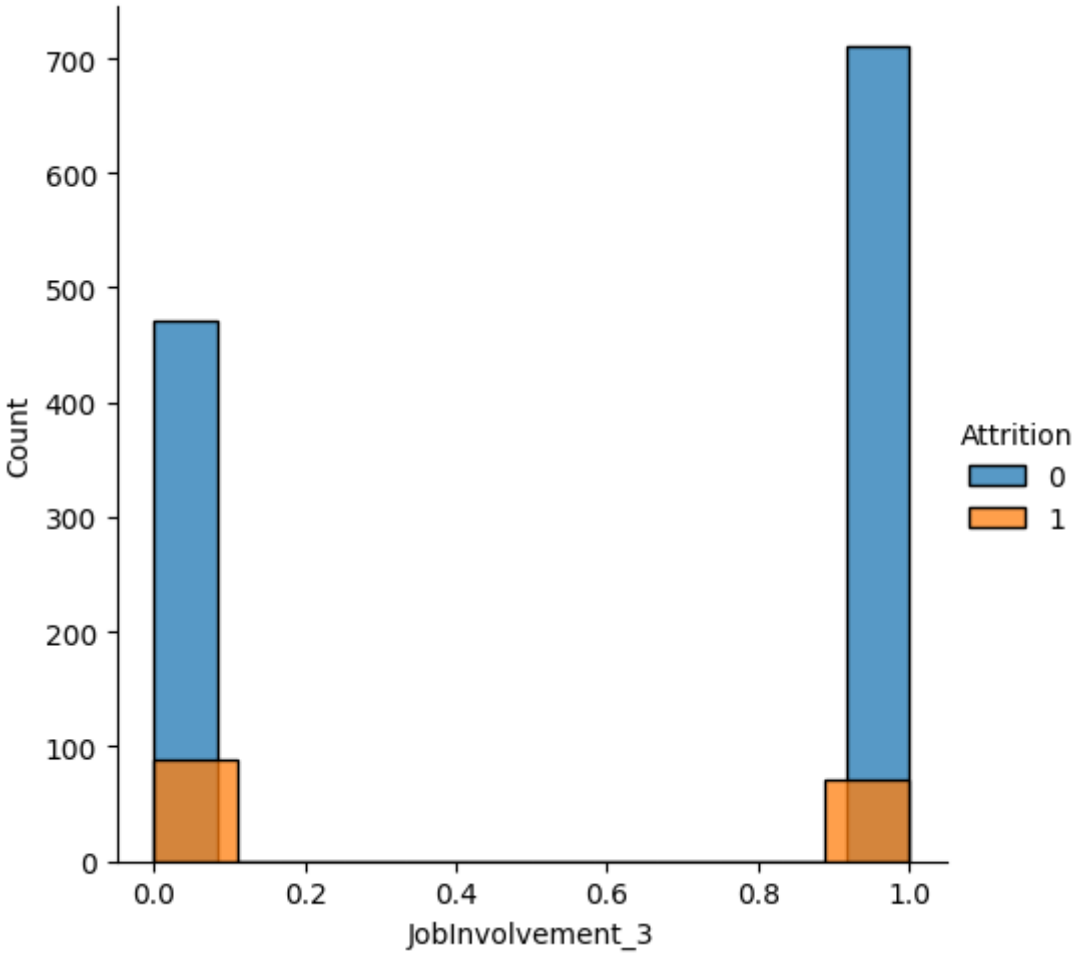

800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
EnvironmentSatisfaction_2
Attrition
0
1

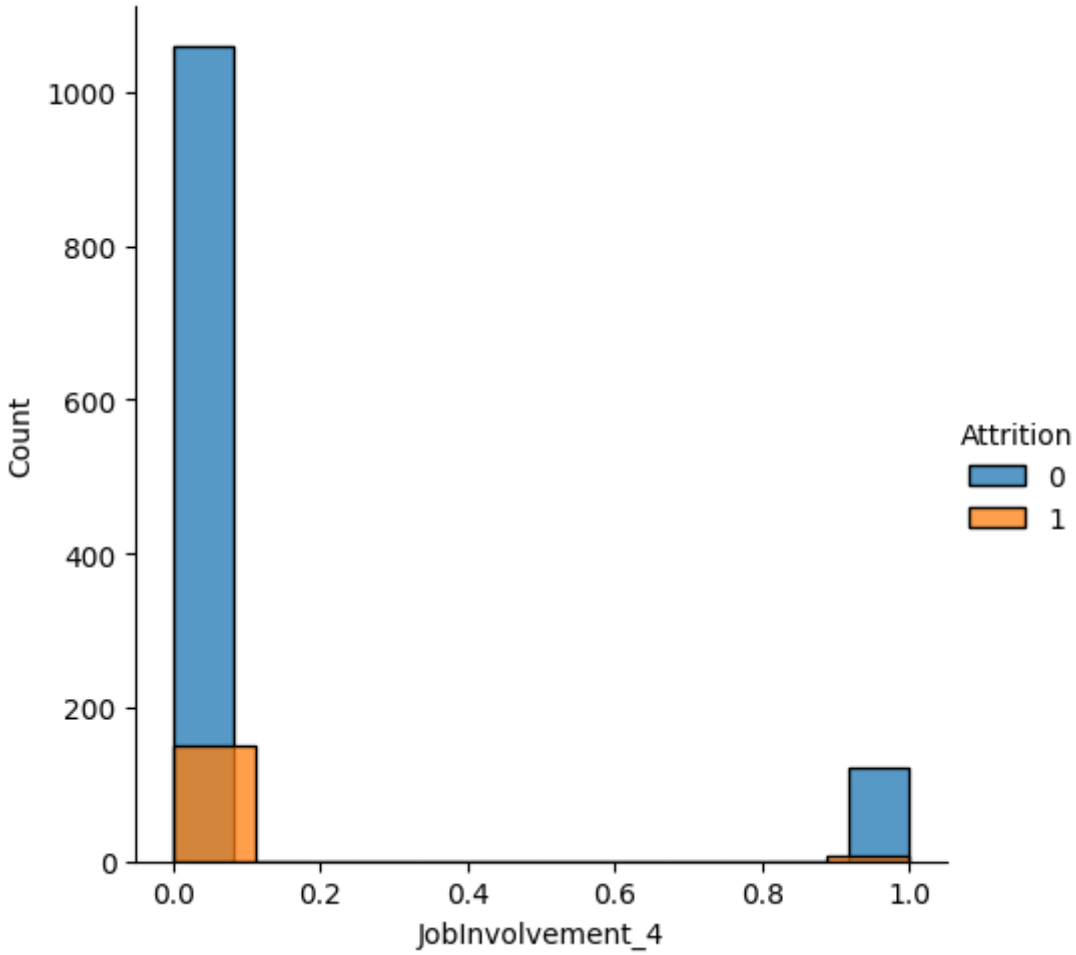

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
EnvironmentSatisfaction_3
Attrition
0
1

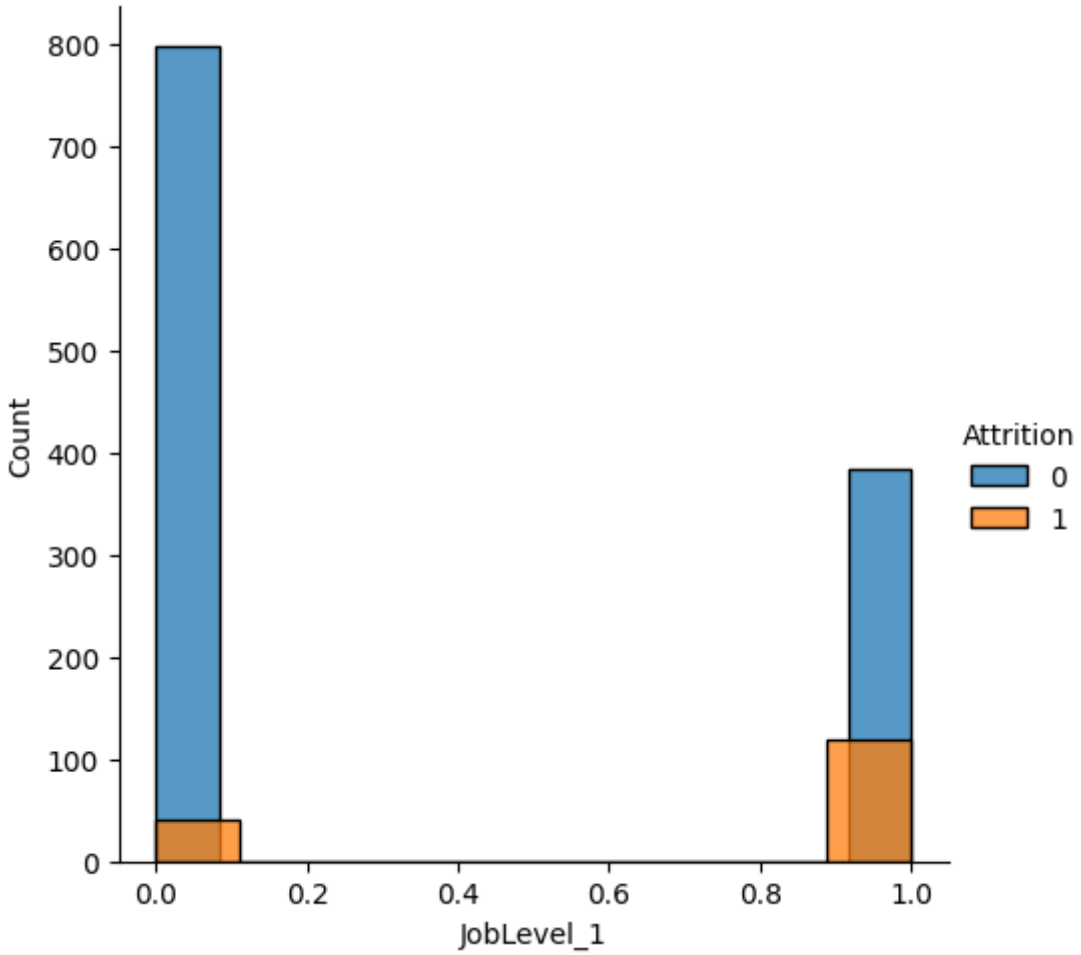

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
EnvironmentSatisfaction_4
Attrition
0
1

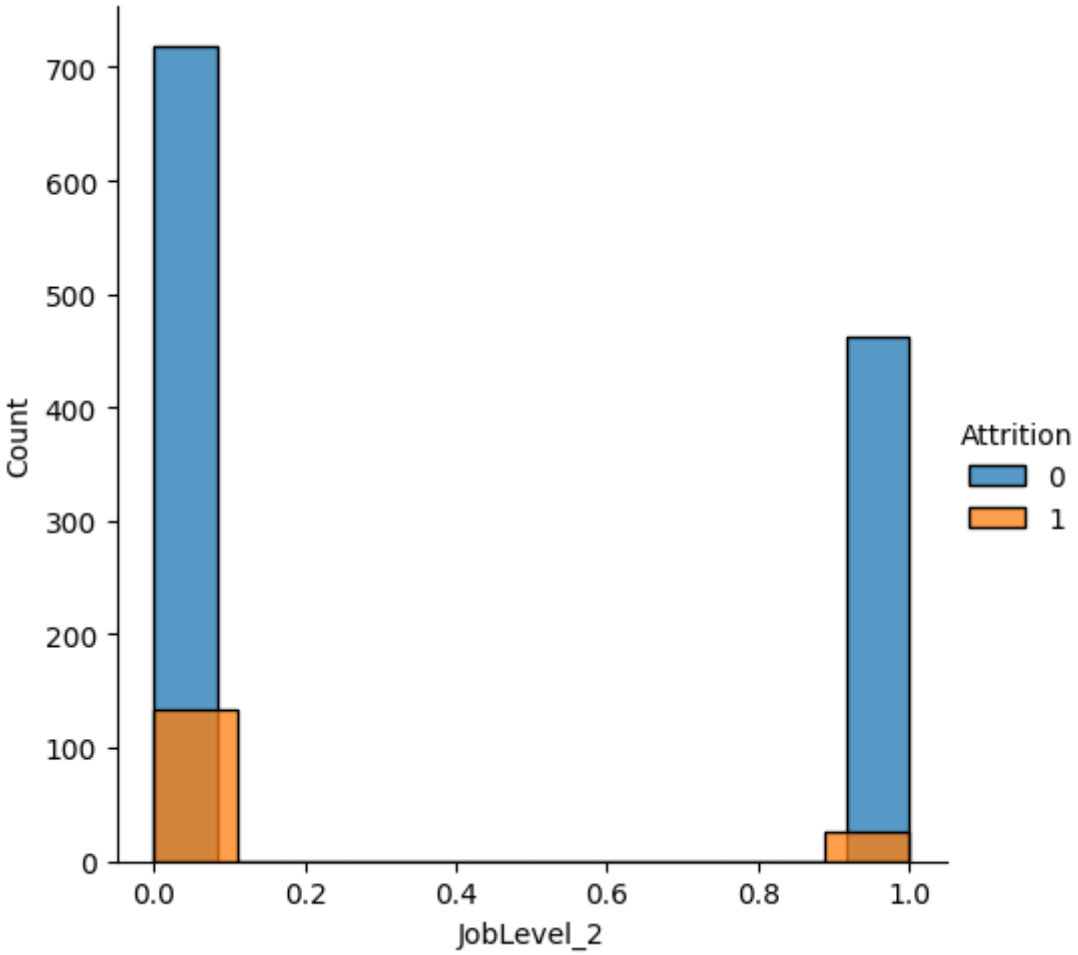

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobInvolvement_1
Attrition
0
1

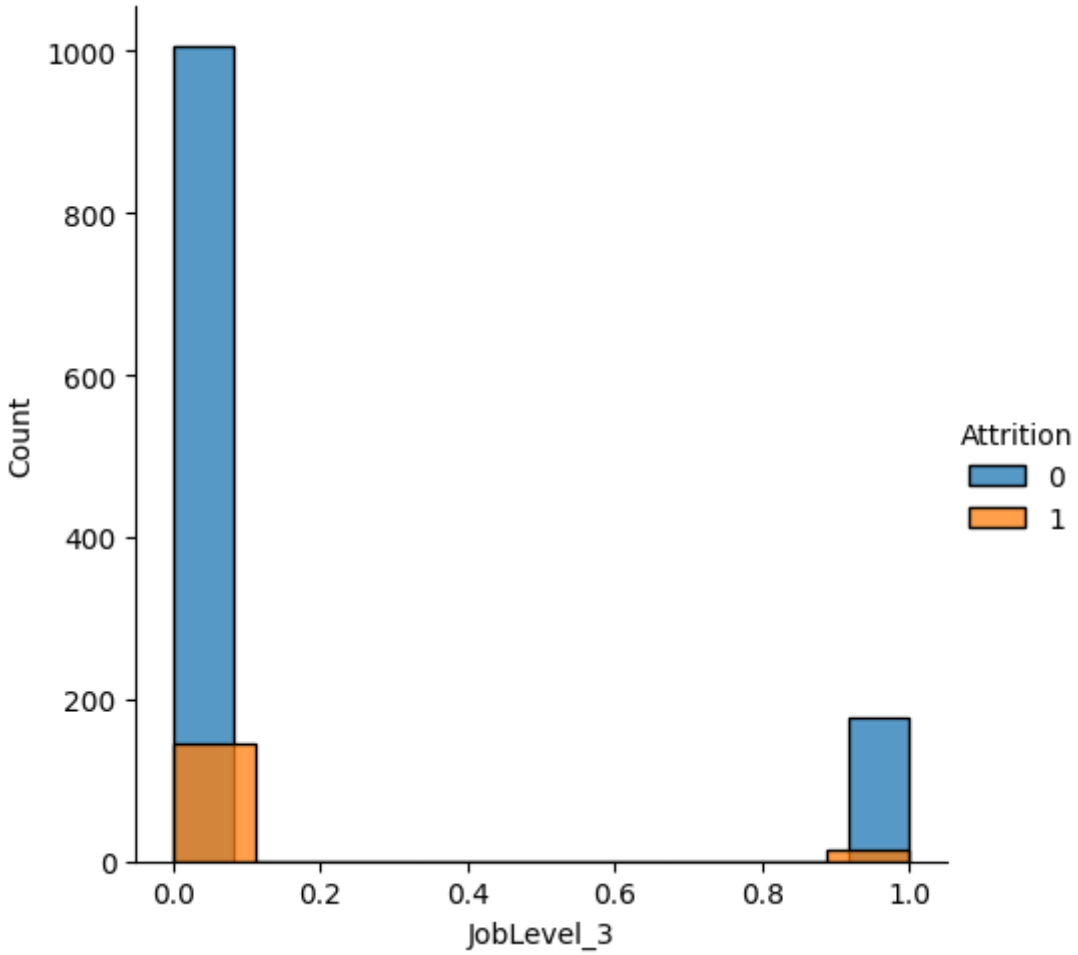

800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobInvolvement_2
Attrition
0
1

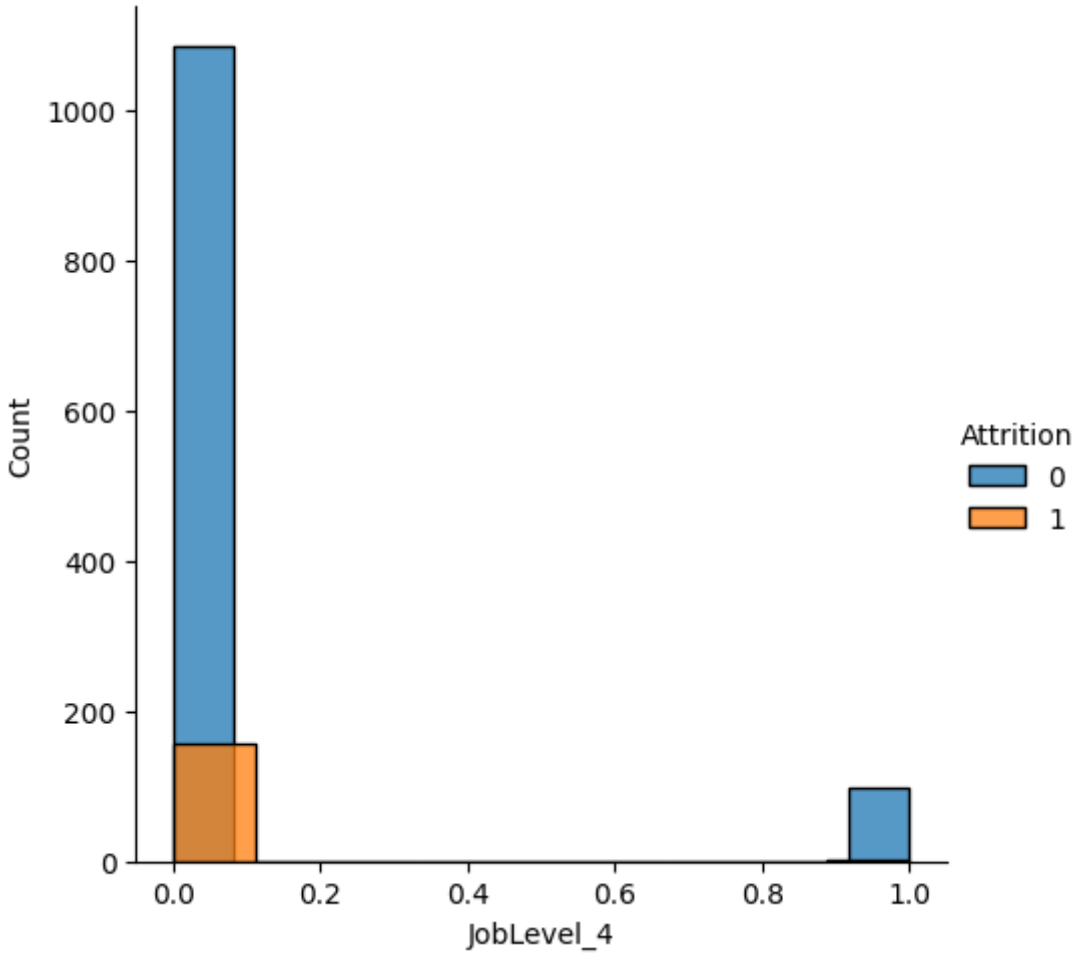

700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobInvolvement_3
Attrition
0
1

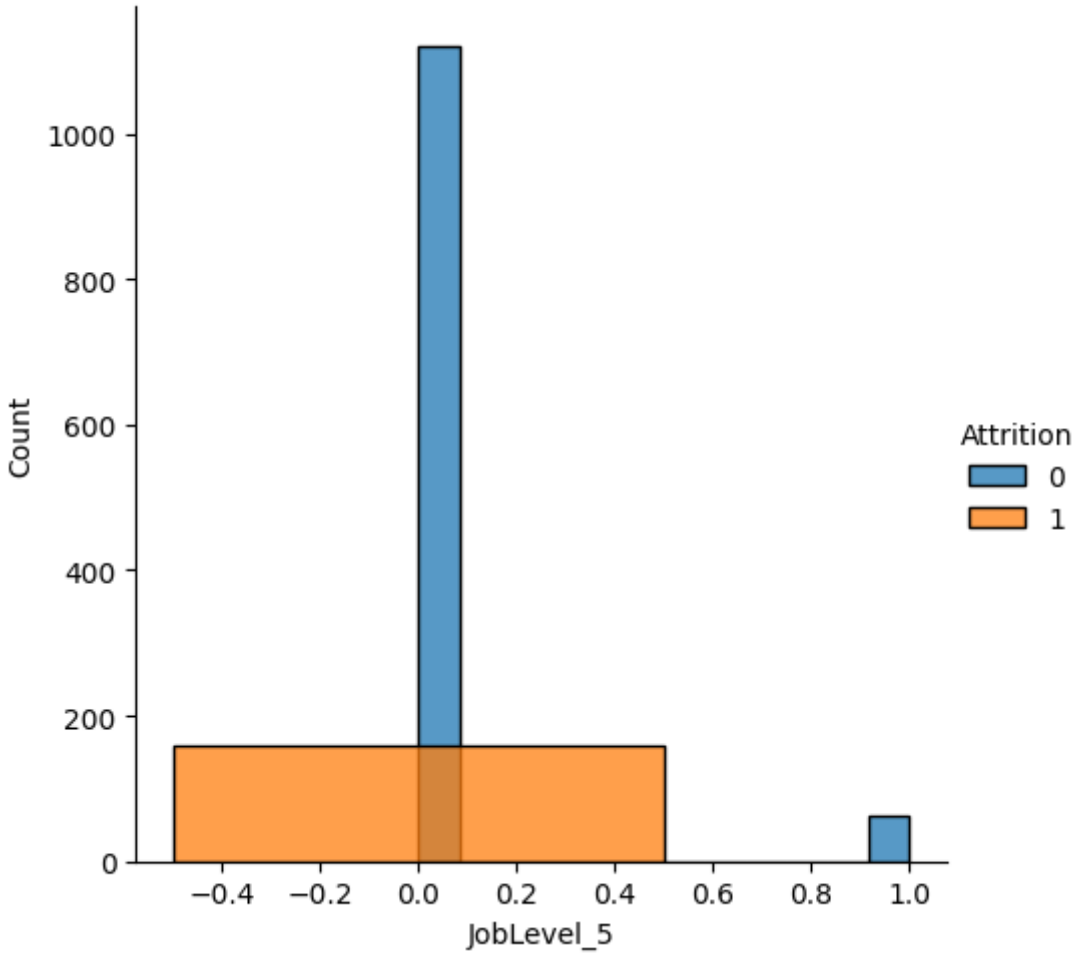

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobInvolvement_4
Attrition
0
1

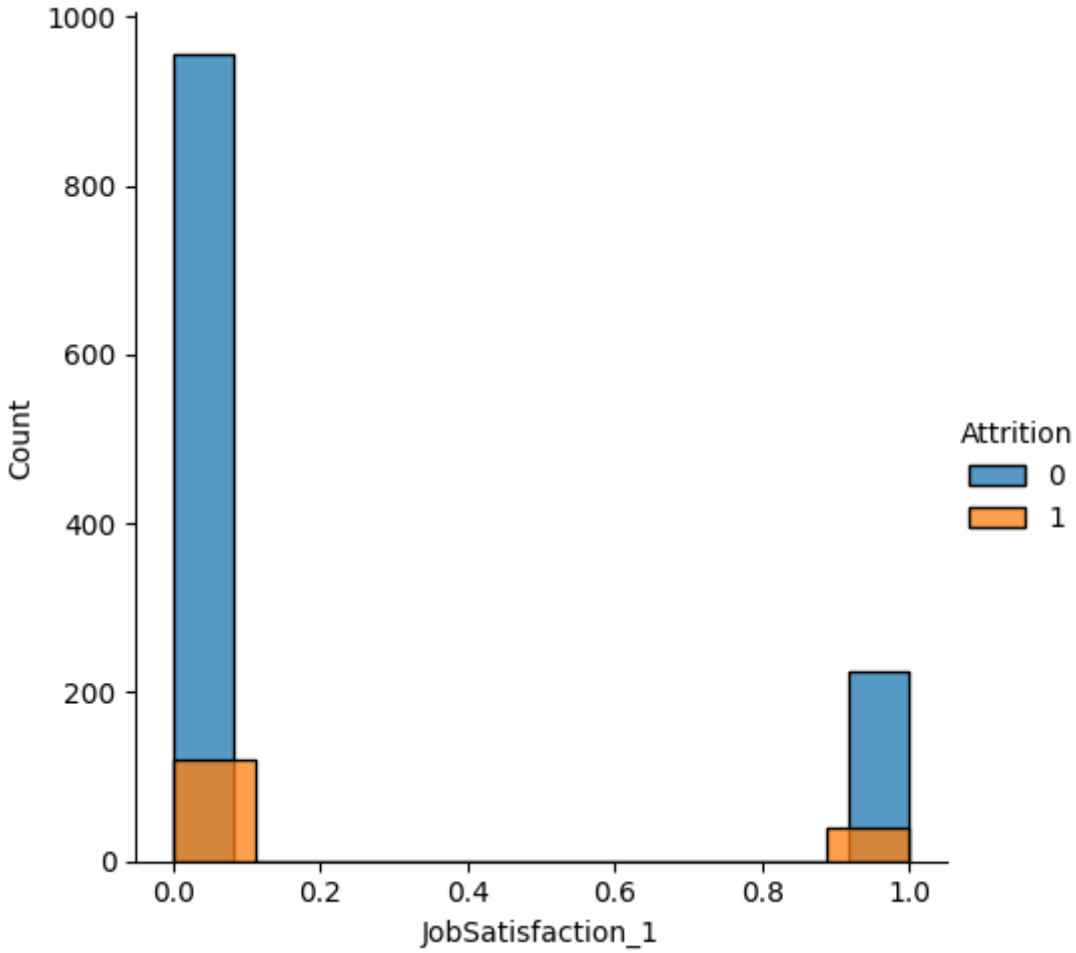

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobLevel_1
Attrition
0
1

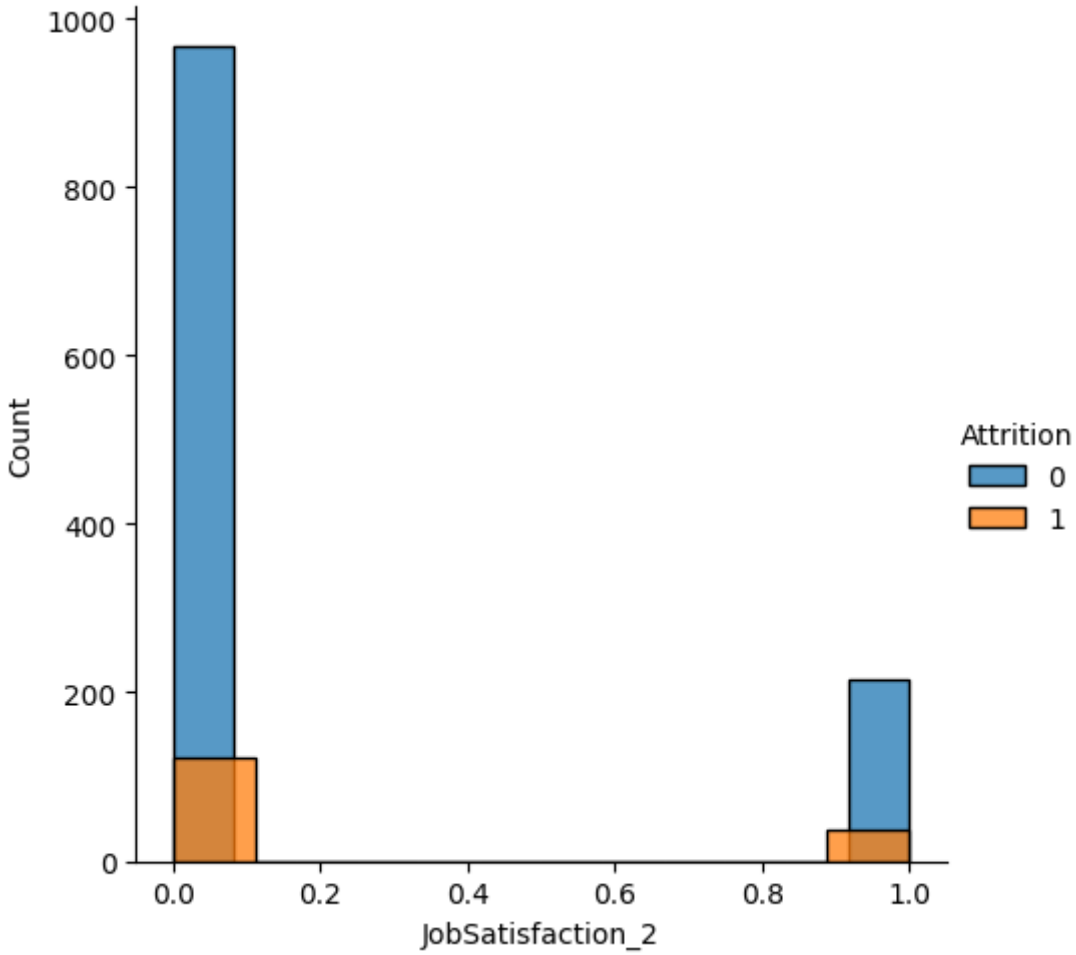

700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobLevel_2
Attrition
0
1

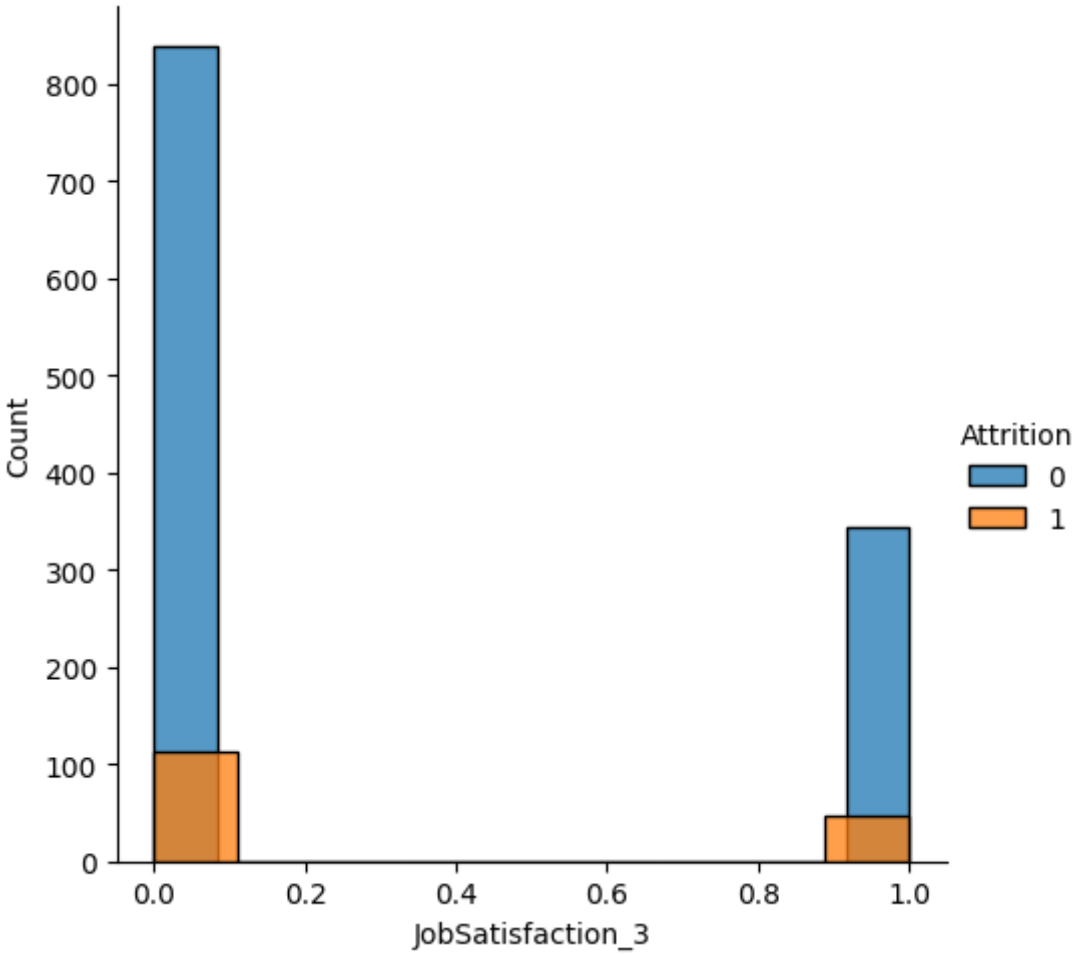

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobLevel_3
Attrition
0
1

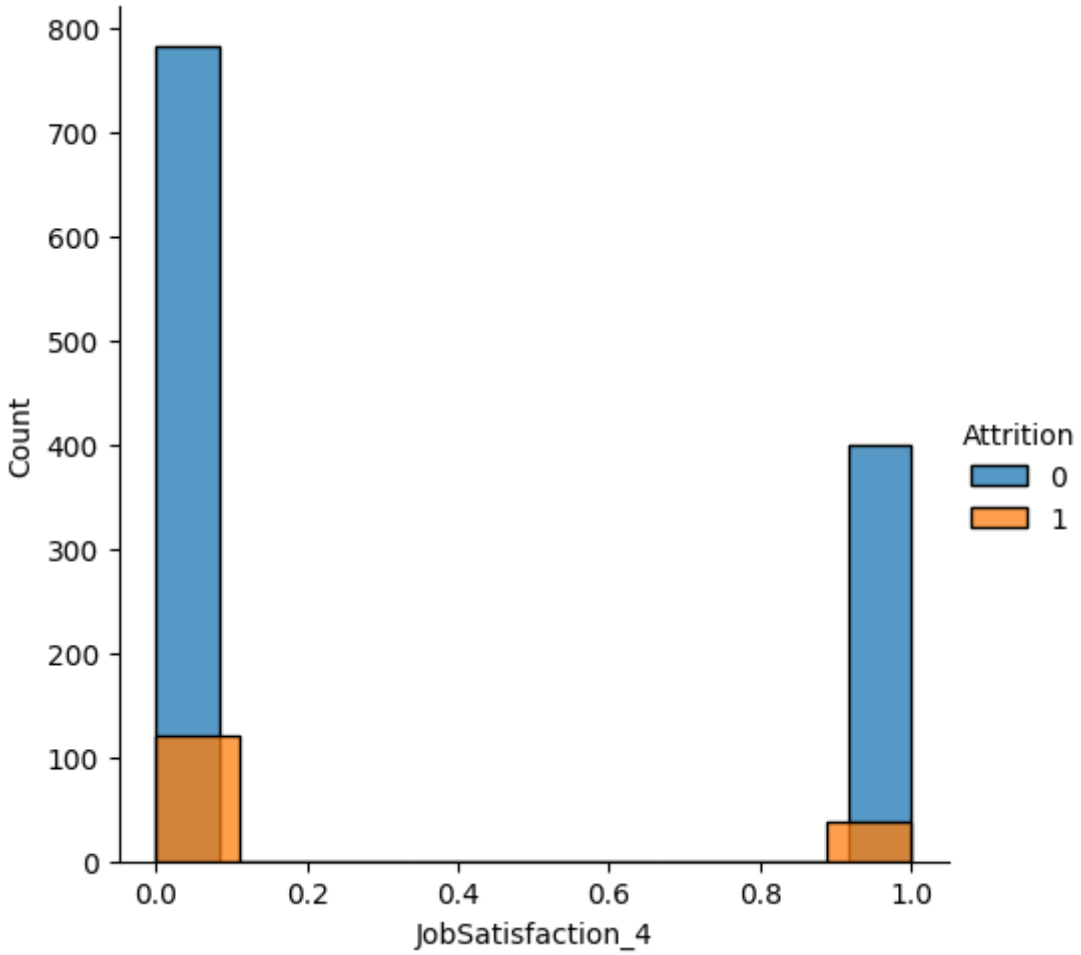

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobLevel_4
Attrition
0
1

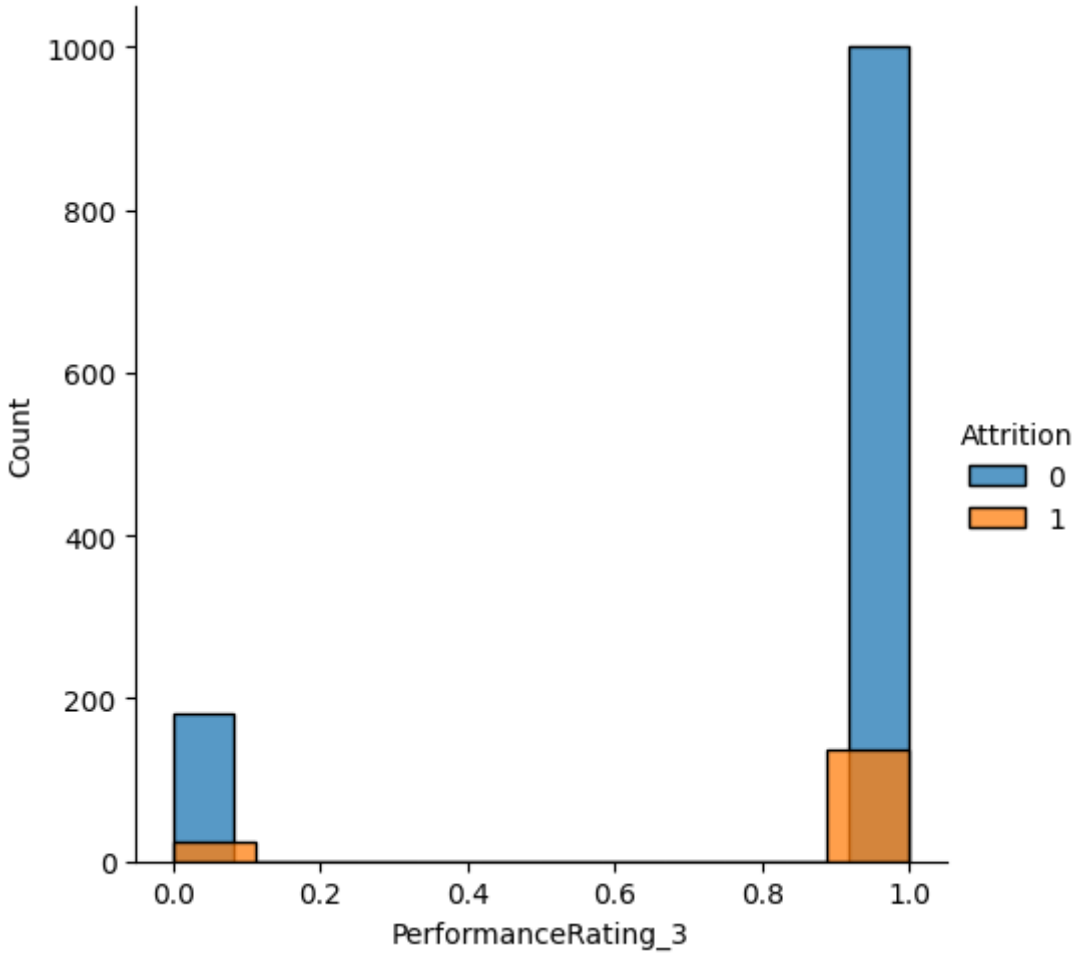

1000
800
600
400
200
0
Count
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
JobLevel_5
Attrition
0
1

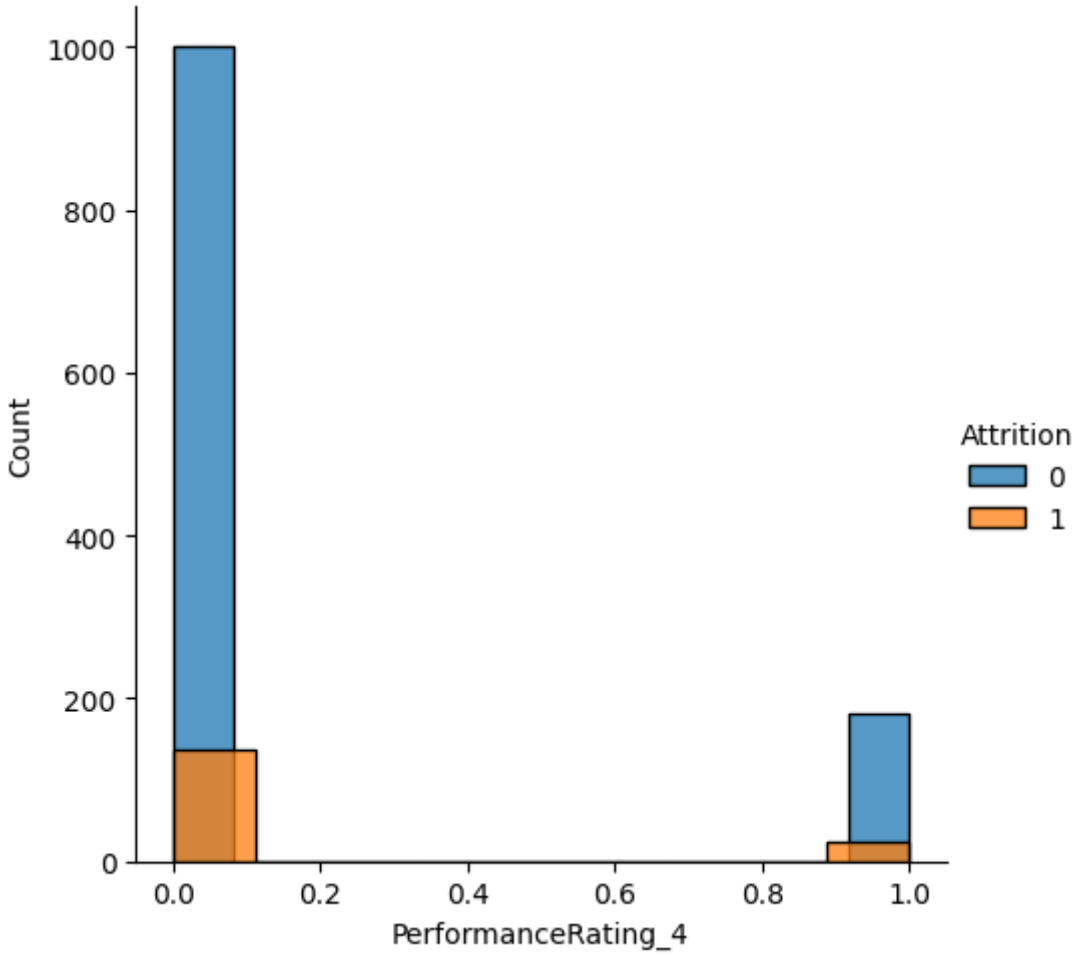

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobSatisfaction_1
Attrition
0
1

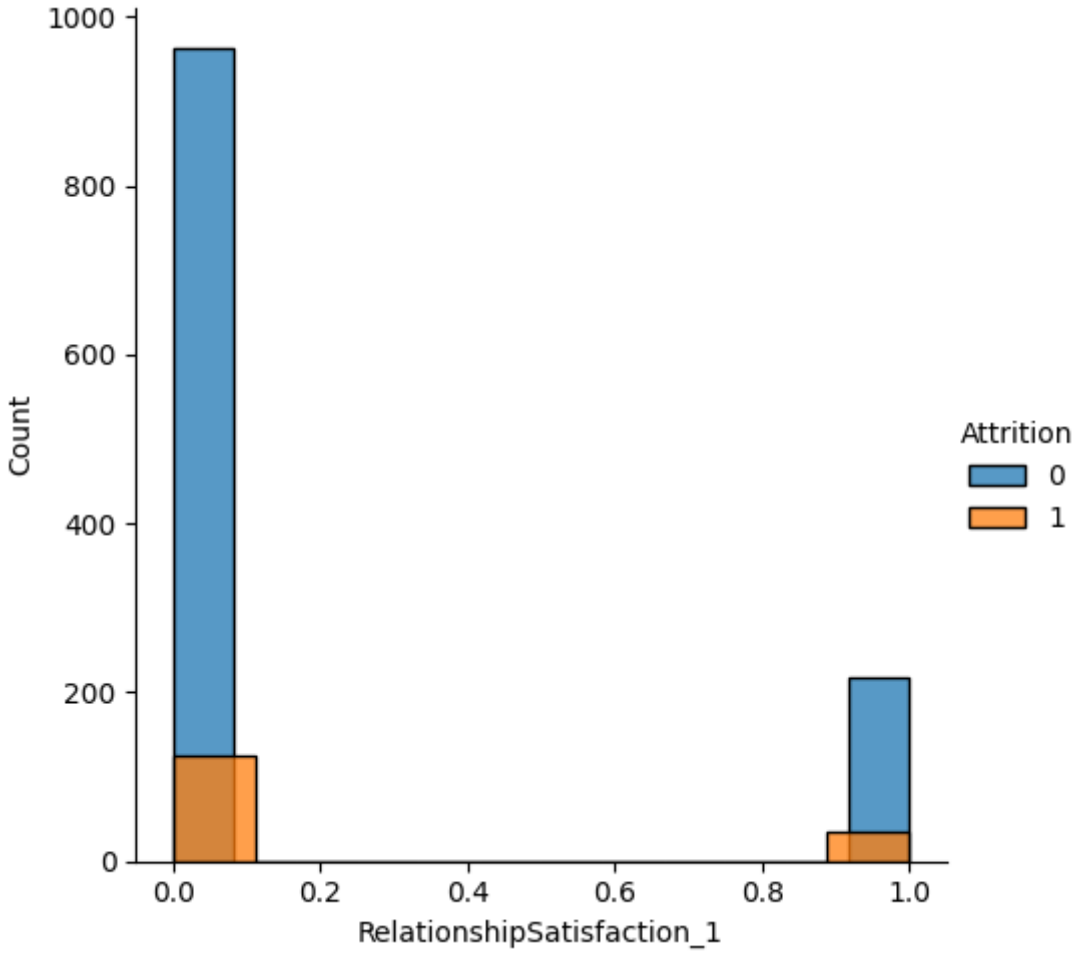

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobSatisfaction_2
Attrition
0
1

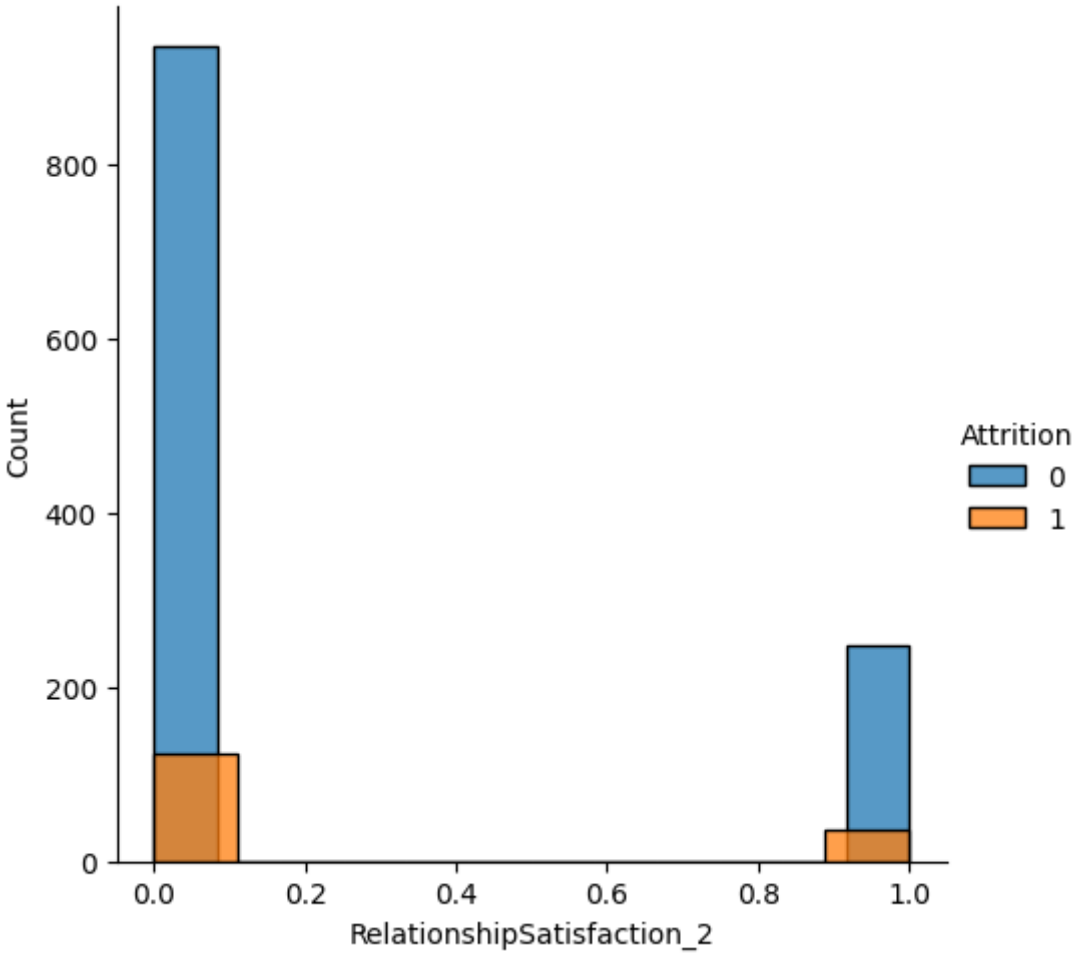

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobSatisfaction_3
Attrition
0
1

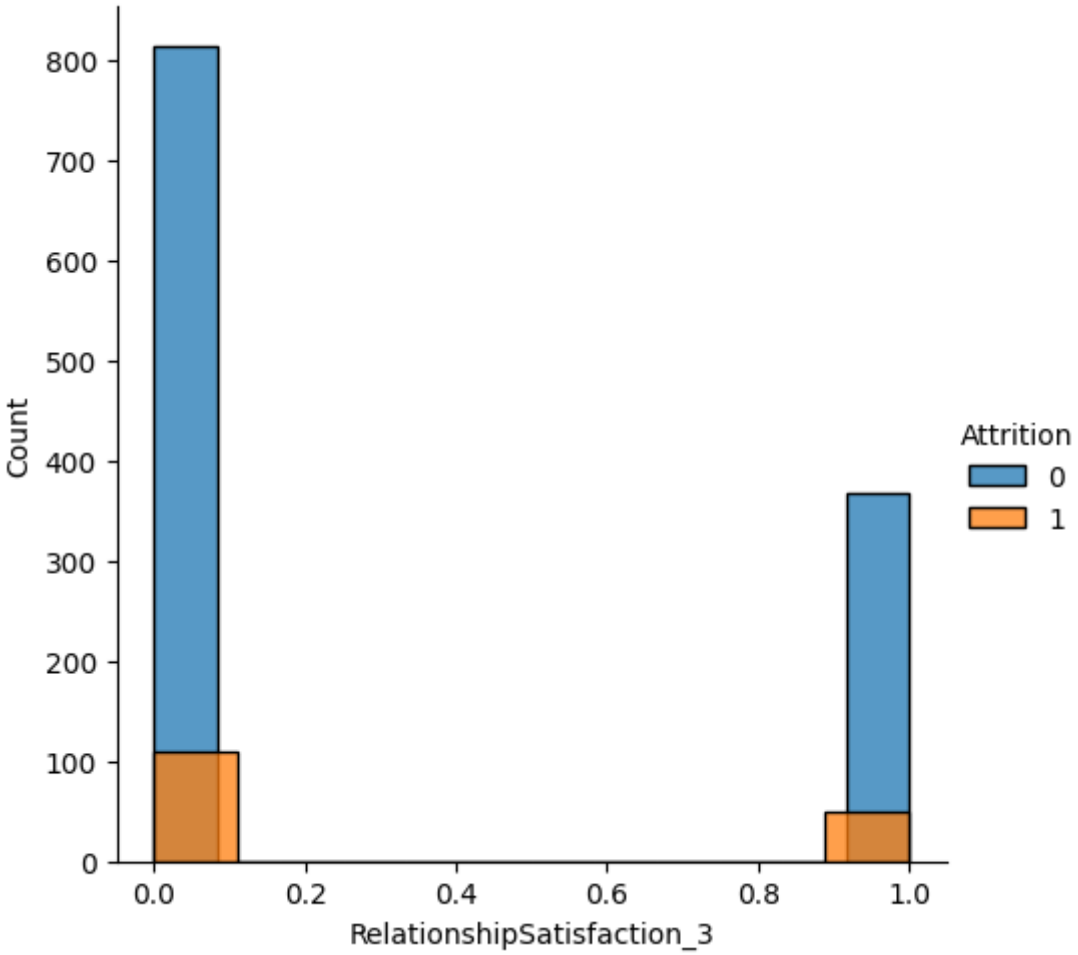

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobSatisfaction_4
Attrition
0
1

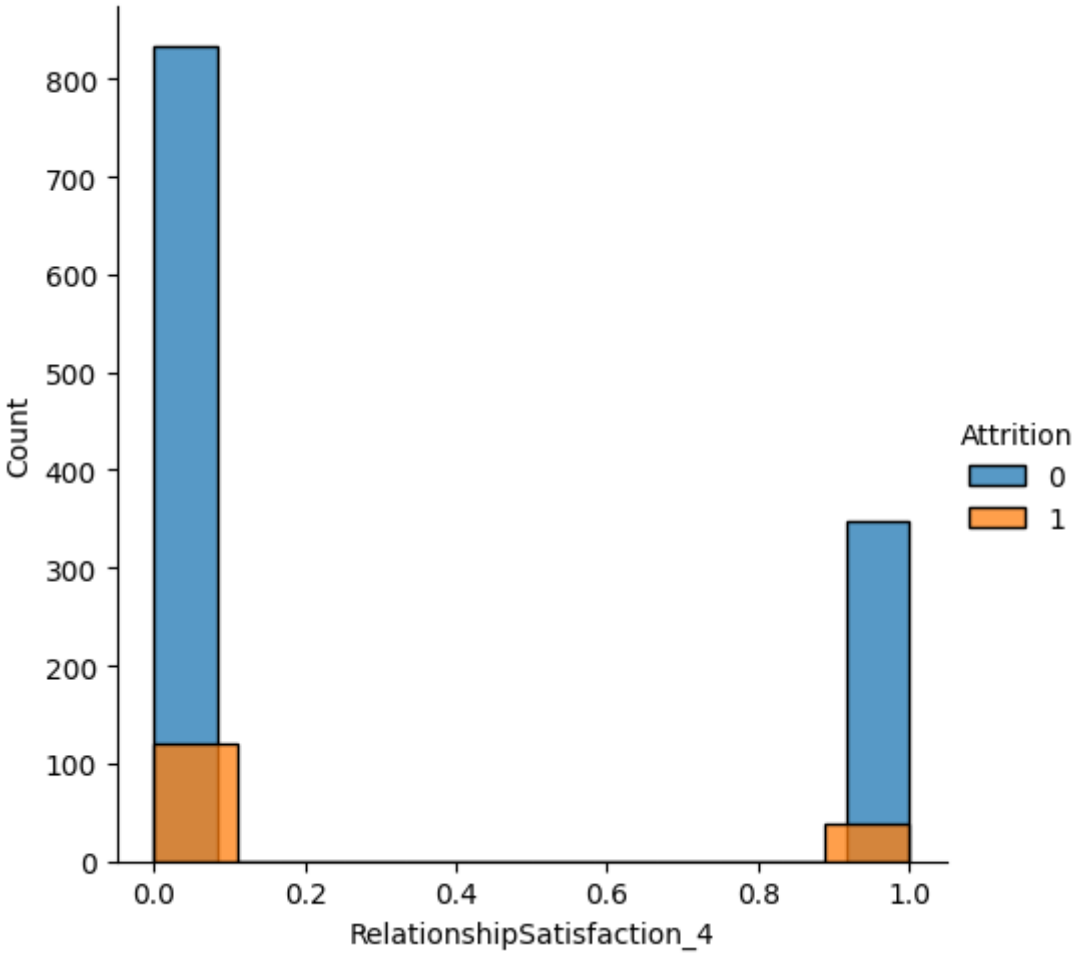

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
PerformanceRating_3
Attrition
0
1

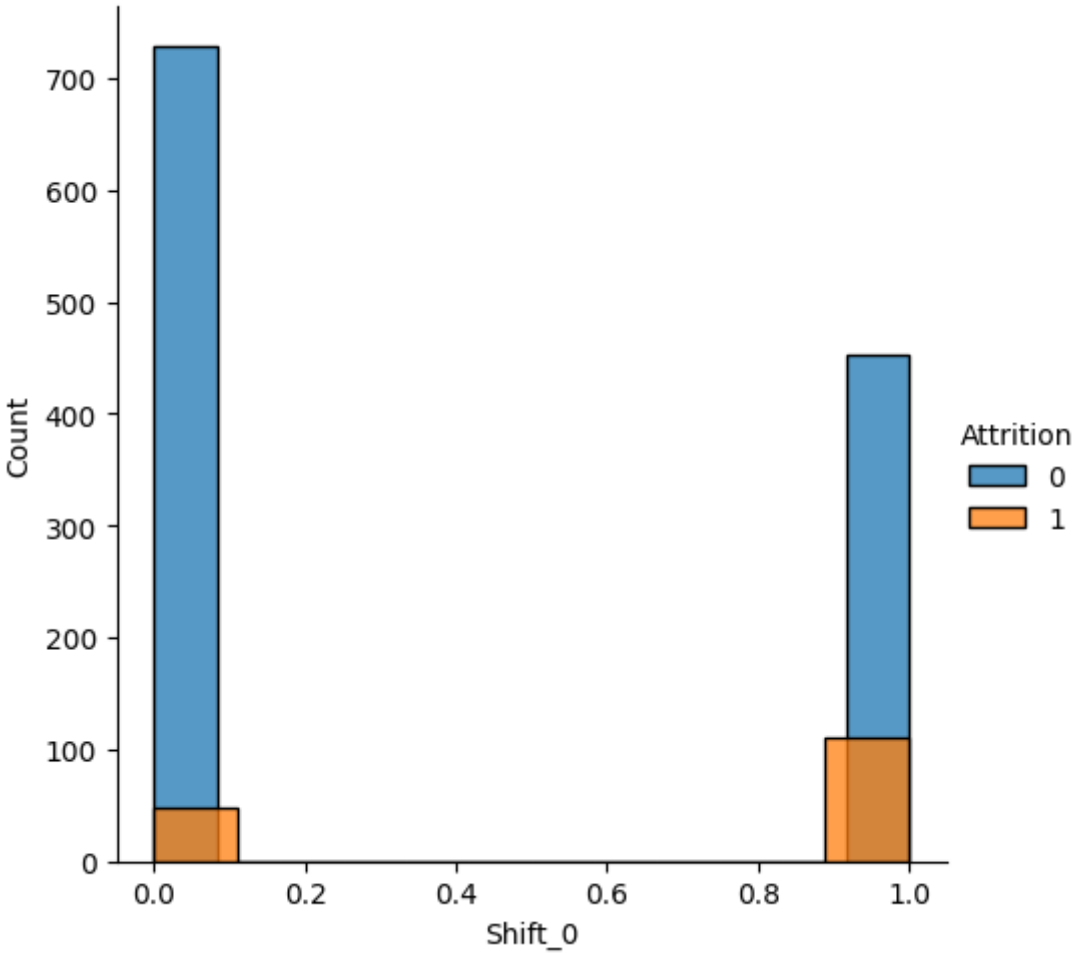

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
PerformanceRating_4
Attrition
0
1

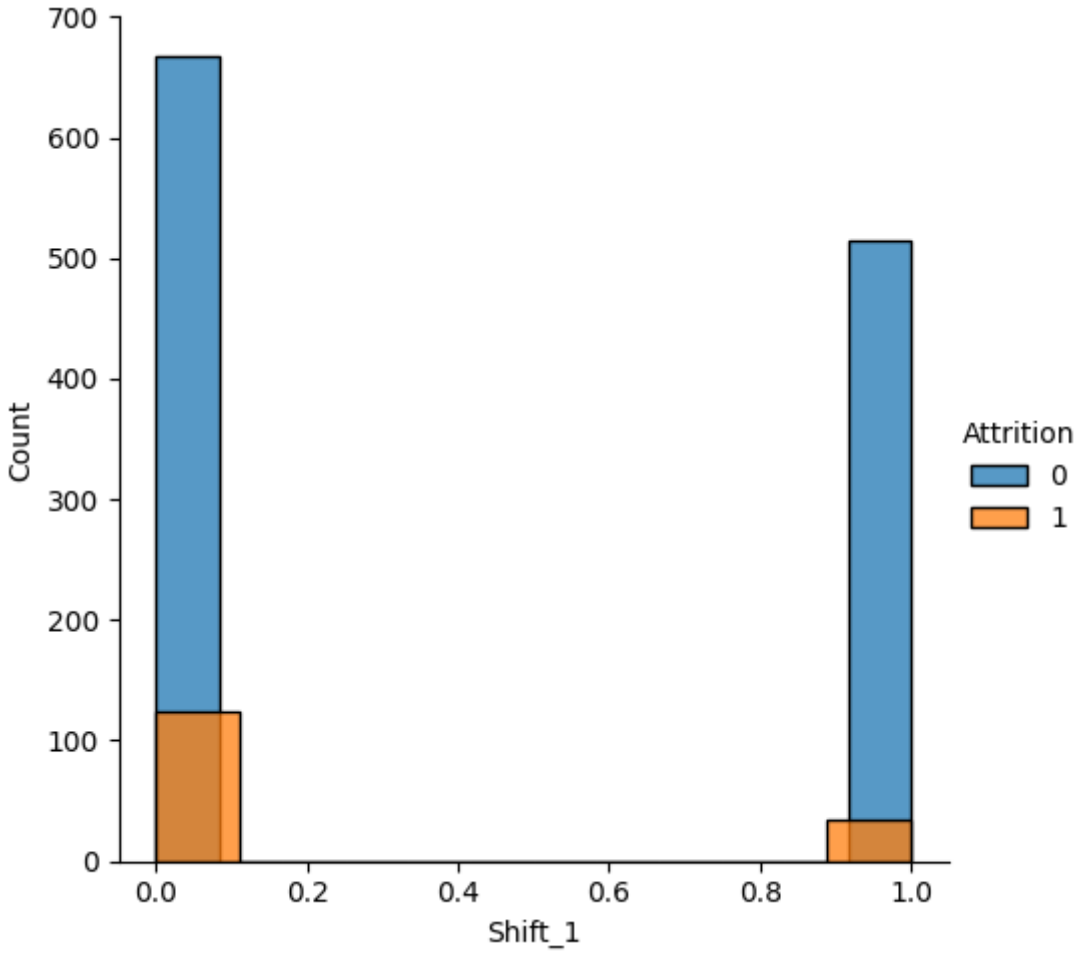

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
RelationshipSatisfaction_1
Attrition
0
1

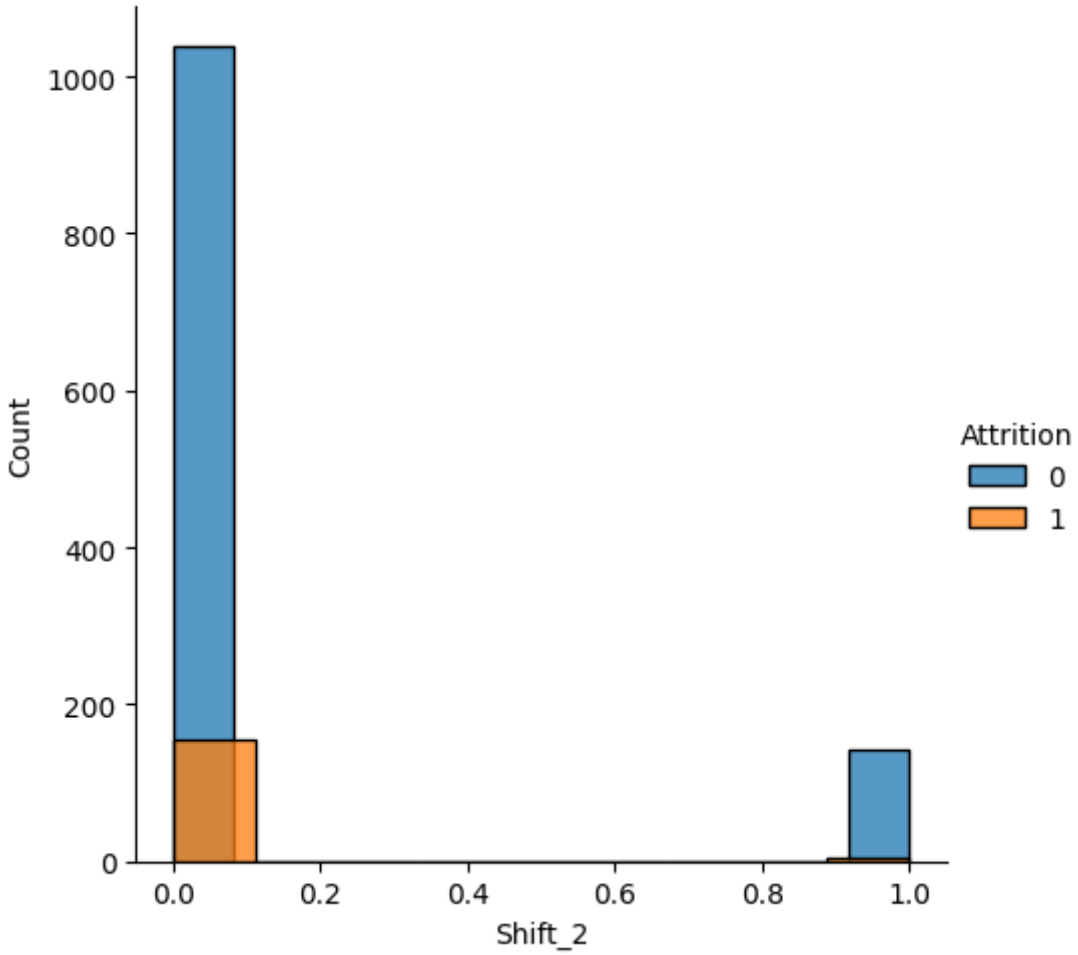

800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
RelationshipSatisfaction_2
Attrition
0
1

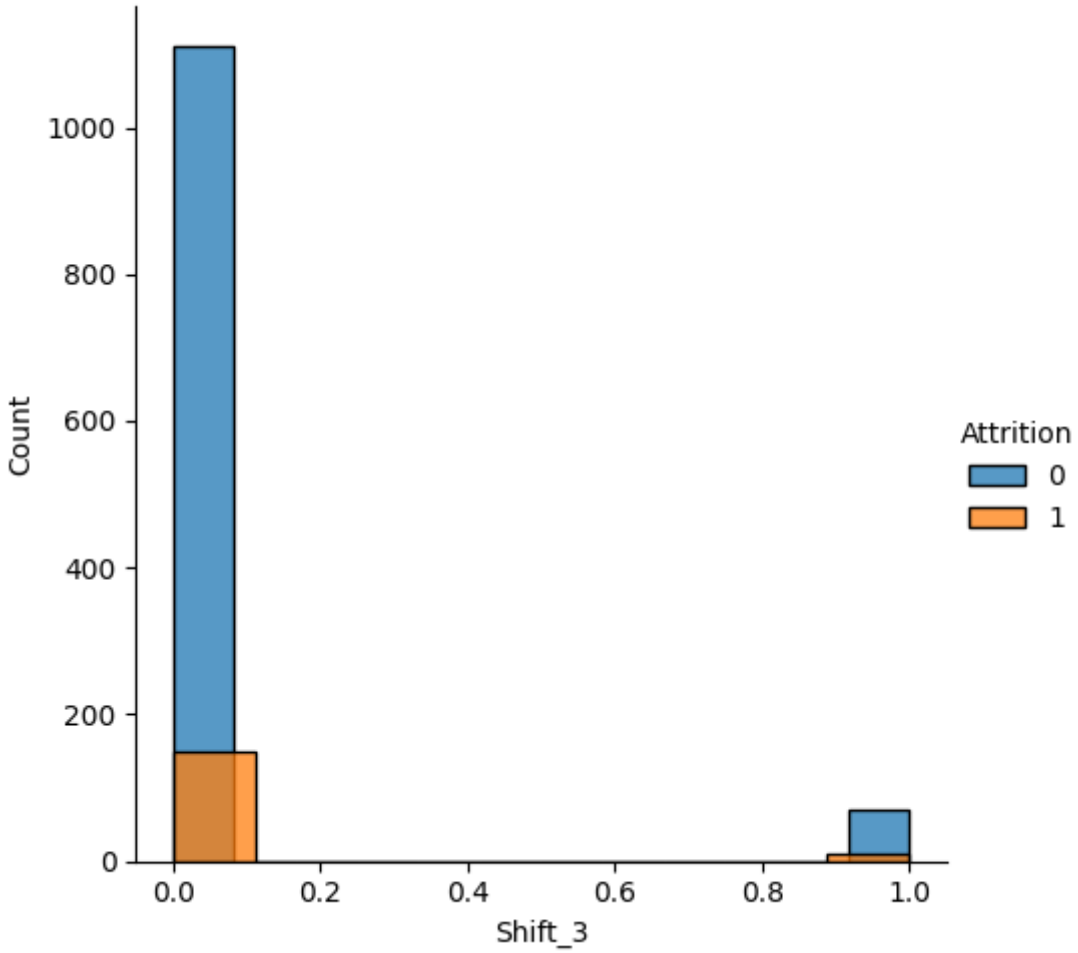

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
RelationshipSatisfaction_3
Attrition
0
1

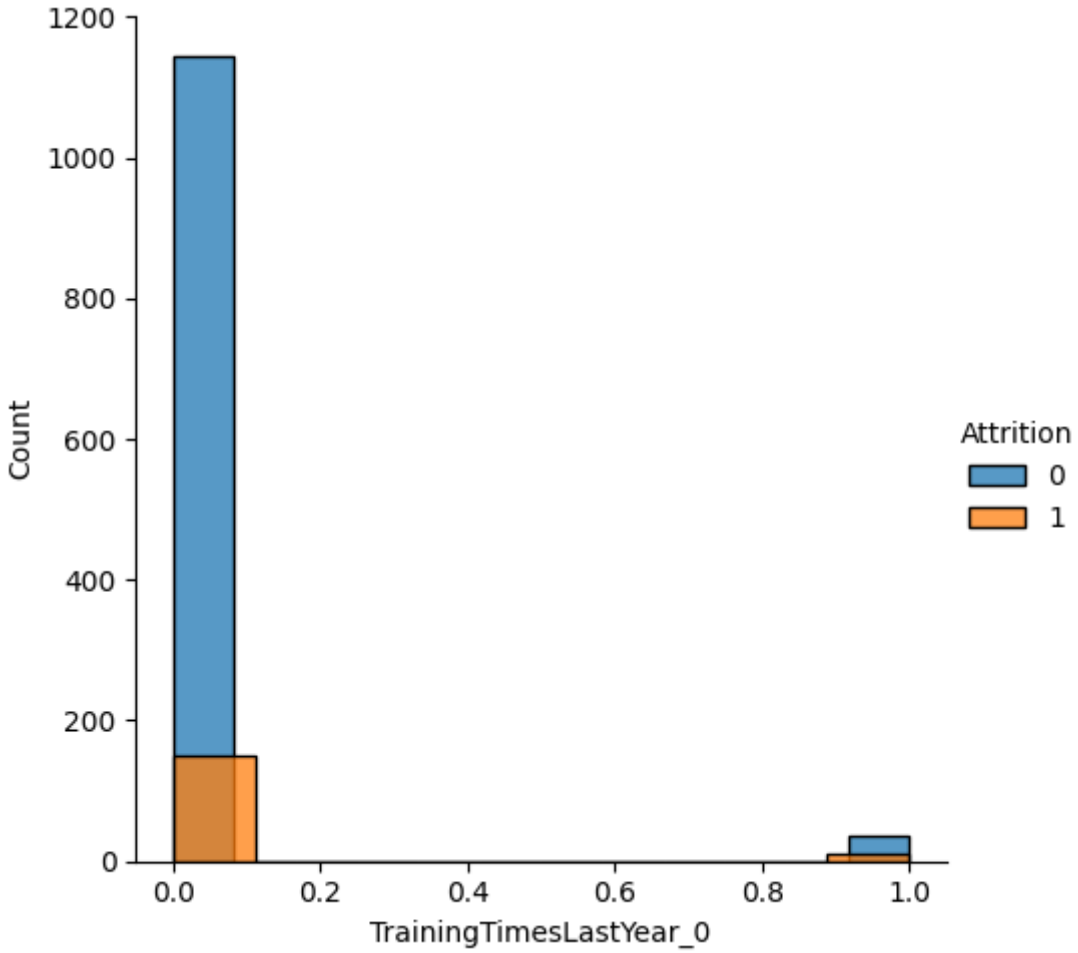

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
RelationshipSatisfaction_4
Attrition
0
1

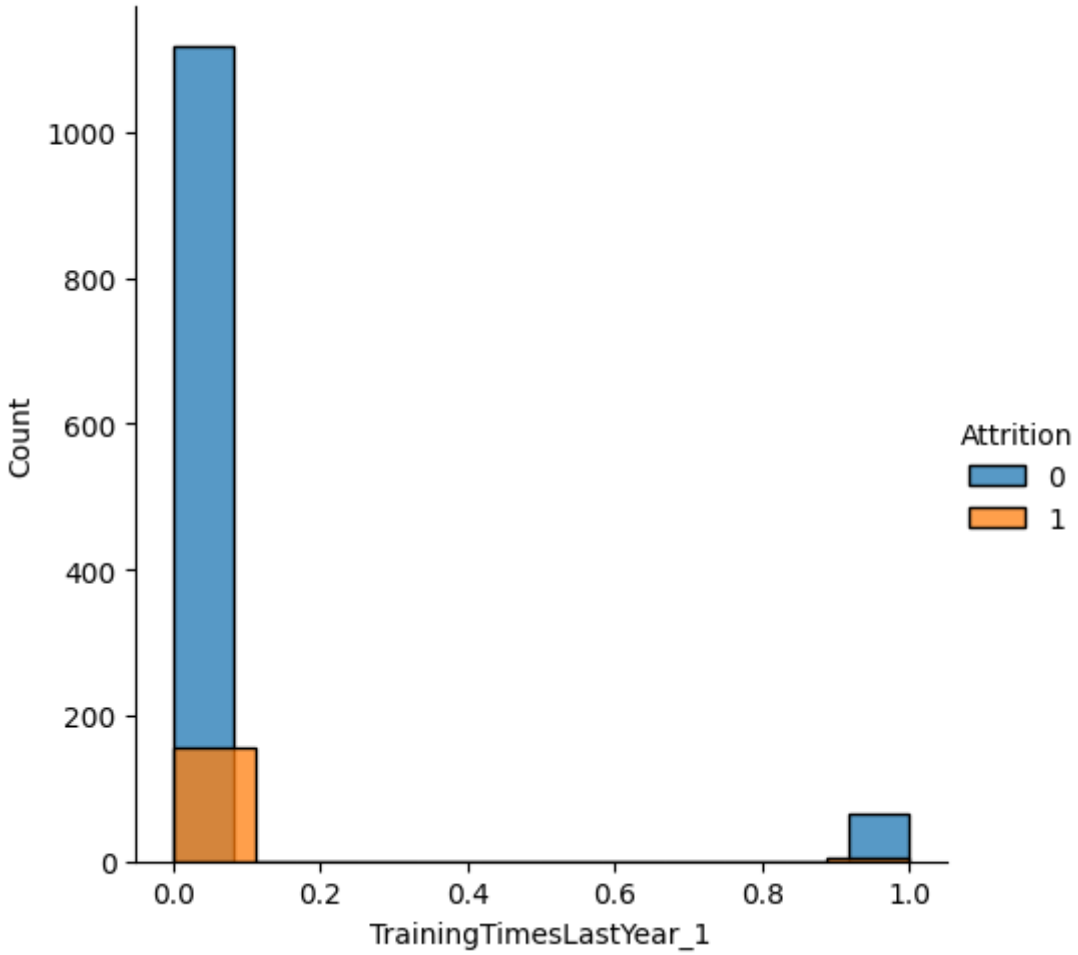

700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Shift_0
Attrition
0
1

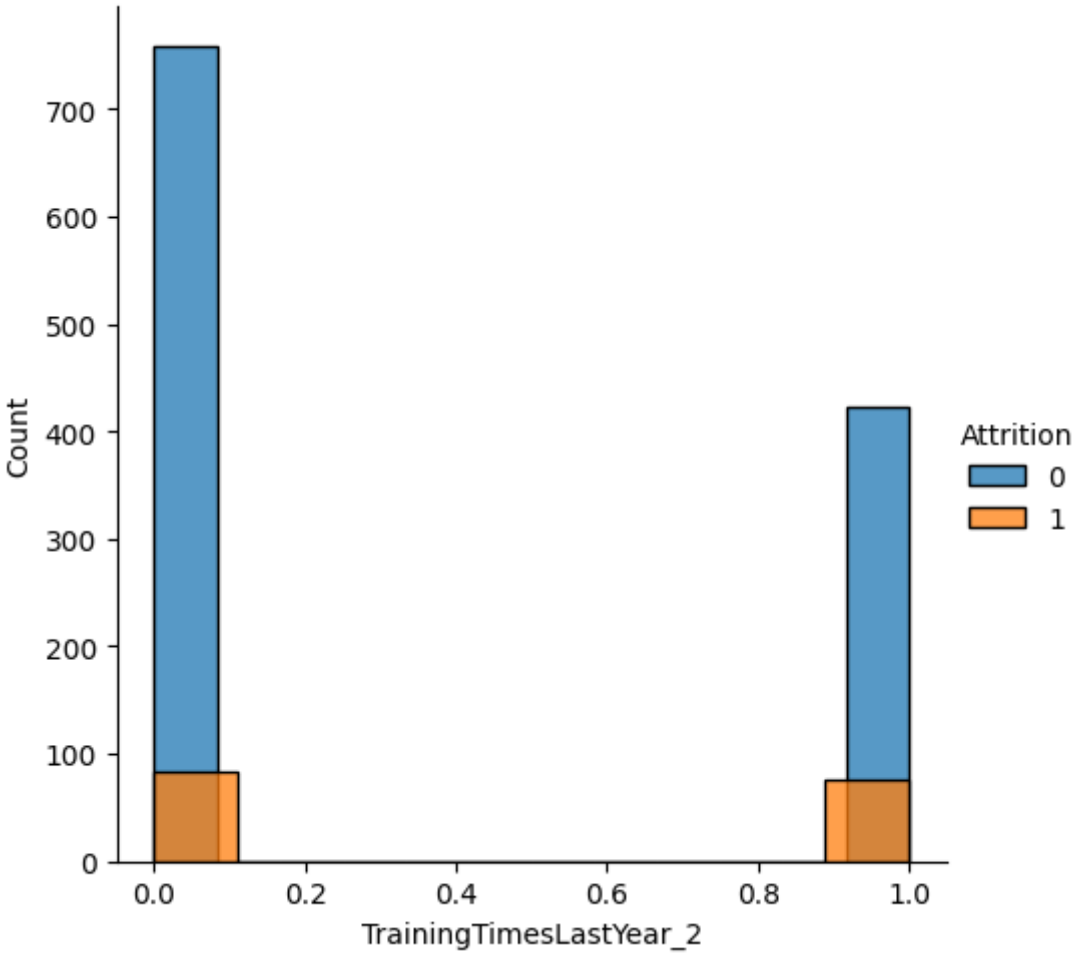

700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Shift_1
Attrition
0
1

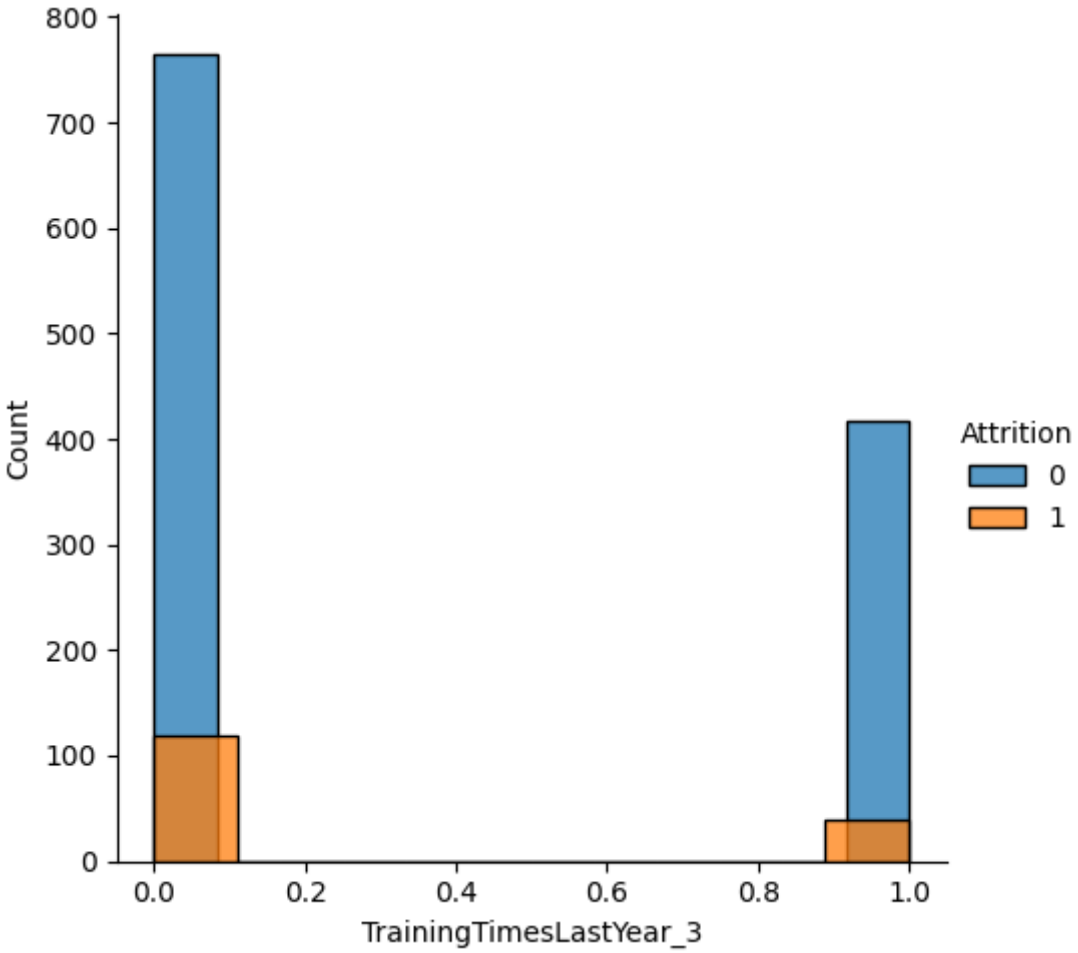

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Shift_2
Attrition
0
1

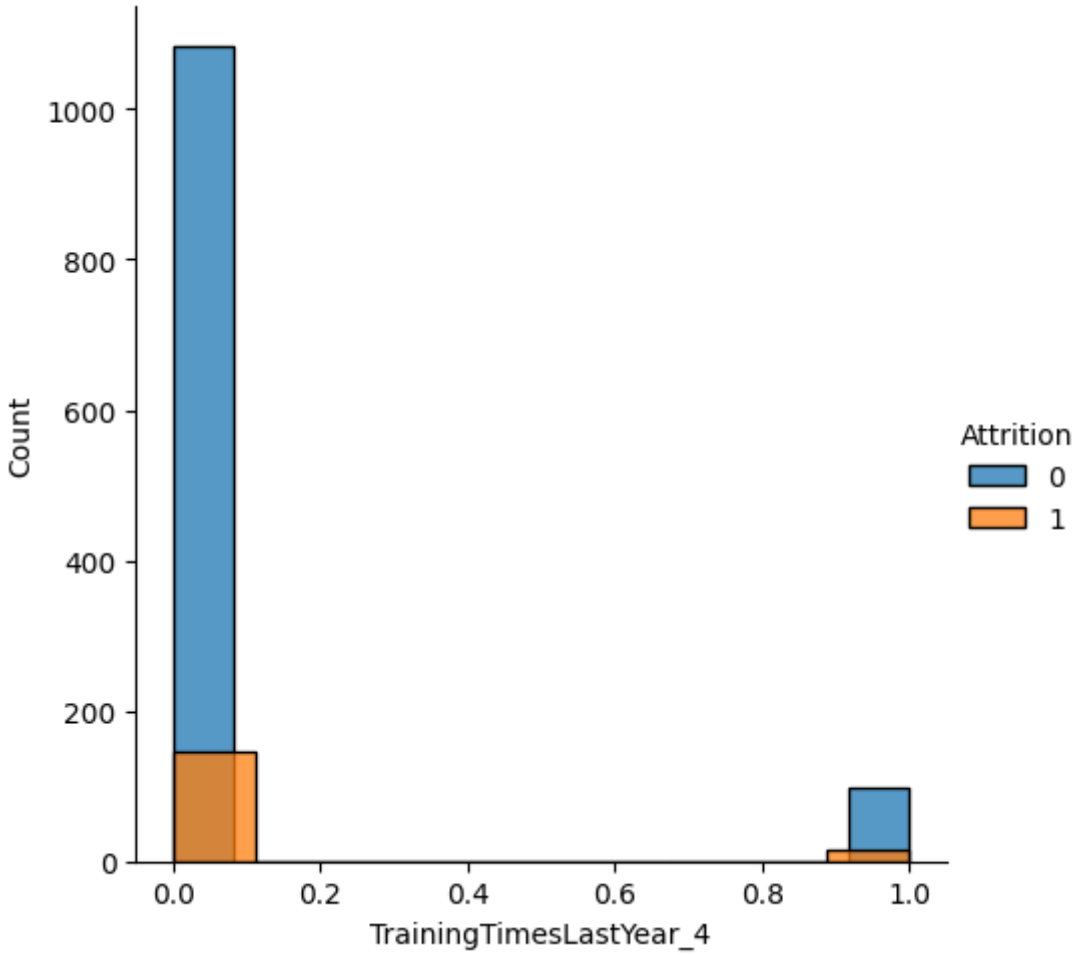

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Shift_3
Attrition
0
1

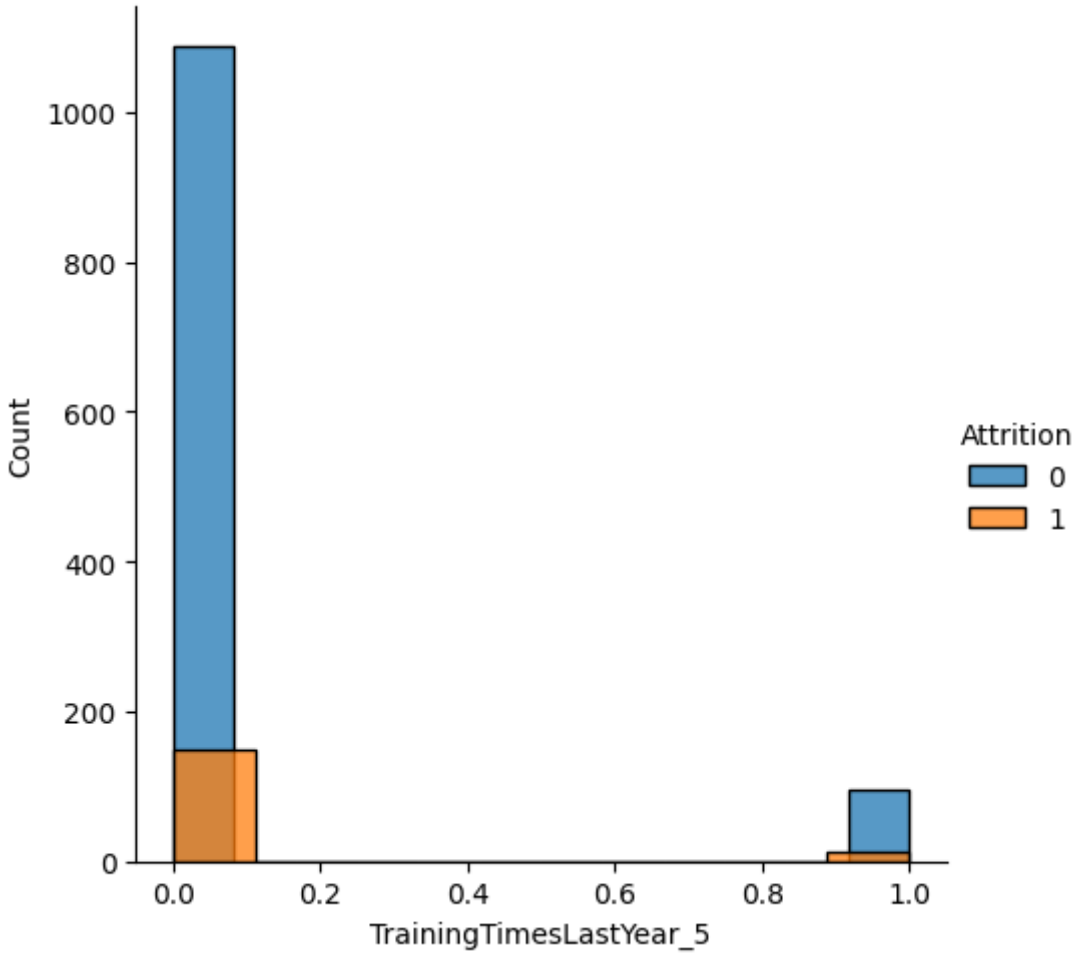

1200
1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
TrainingTimesLastYear_0
Attrition
0
1

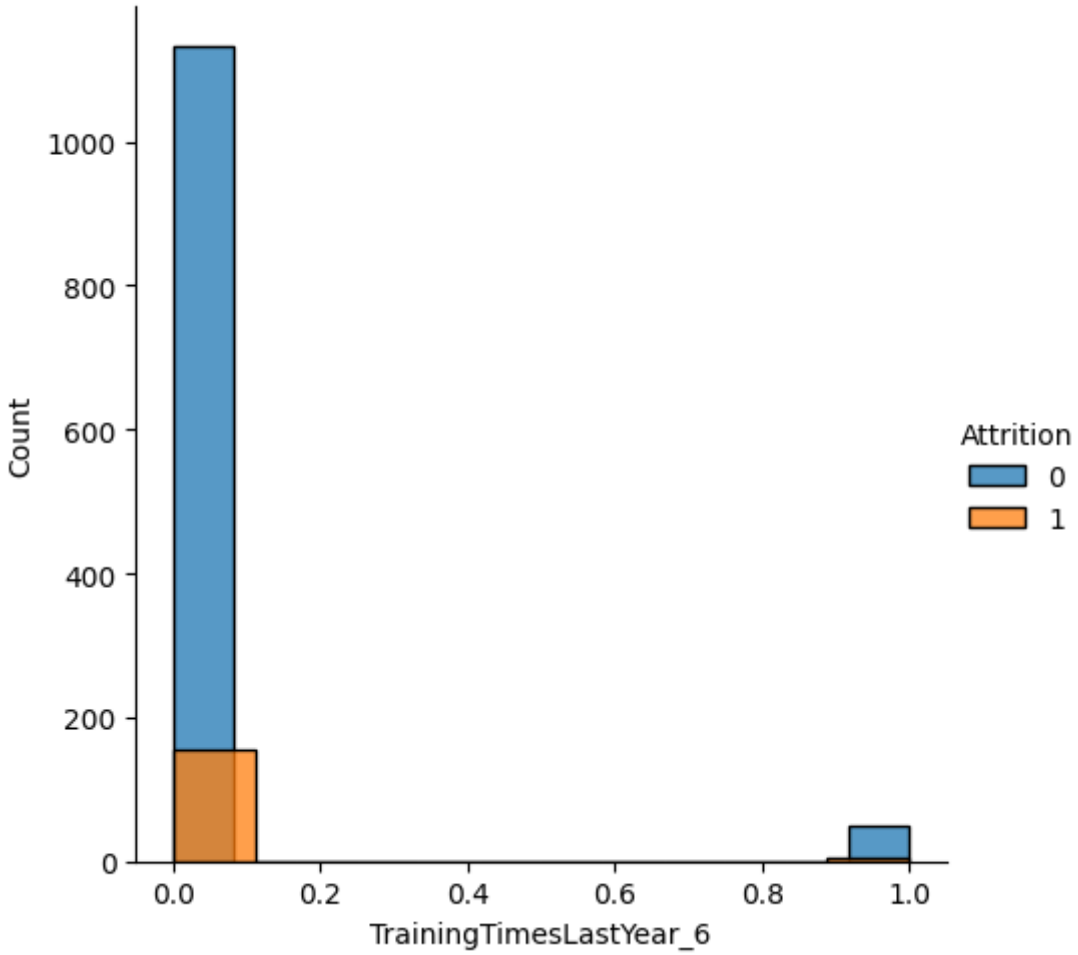

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
TrainingTimesLastYear_1
Attrition
0
1

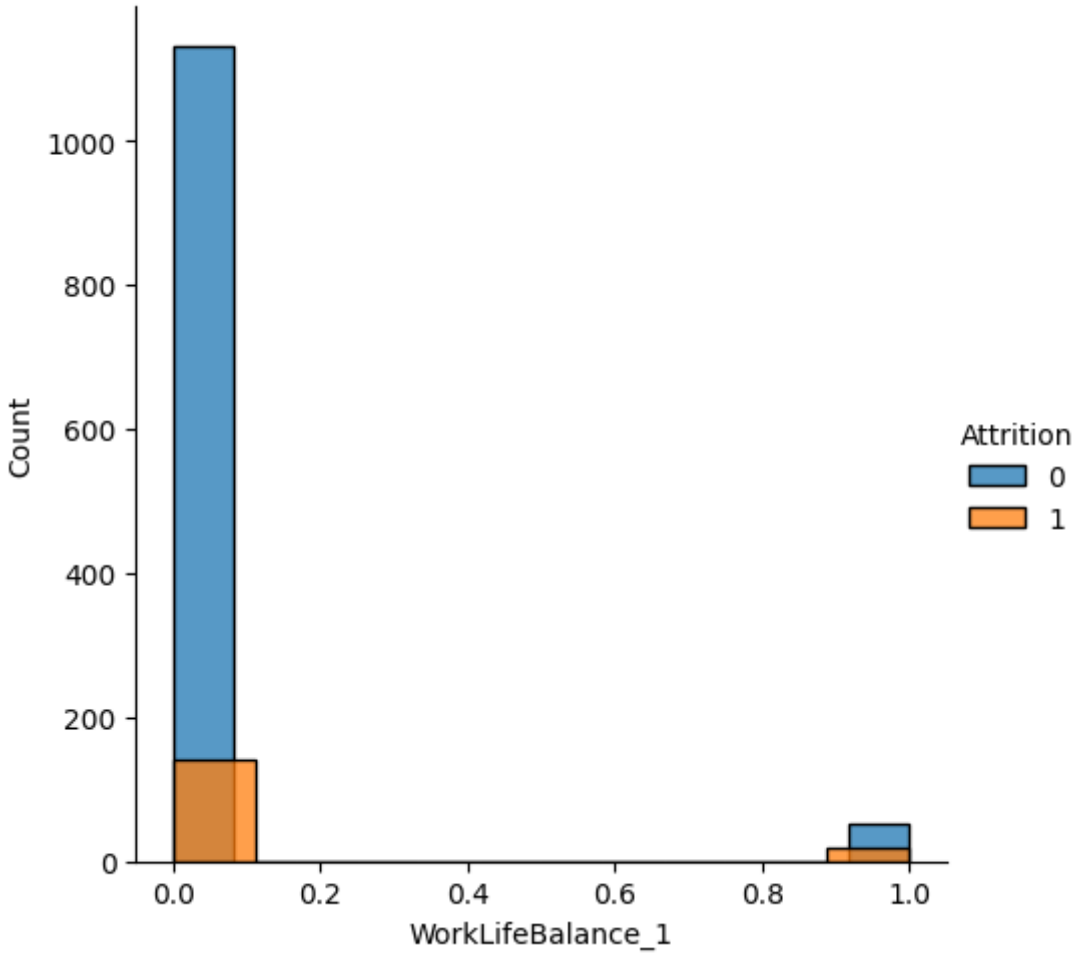

700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
TrainingTimesLastYear_2
Attrition
0
1

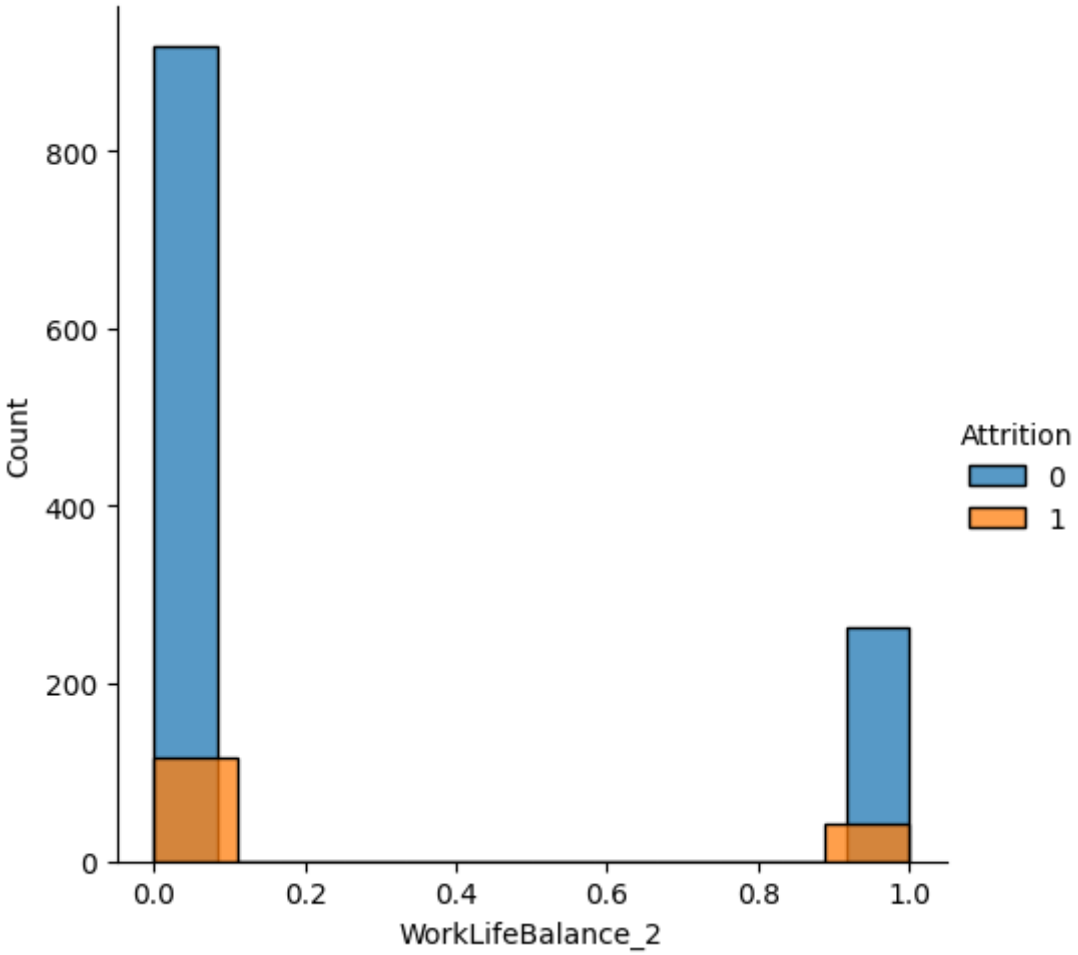

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
TrainingTimesLastYear_3
Attrition
0
1

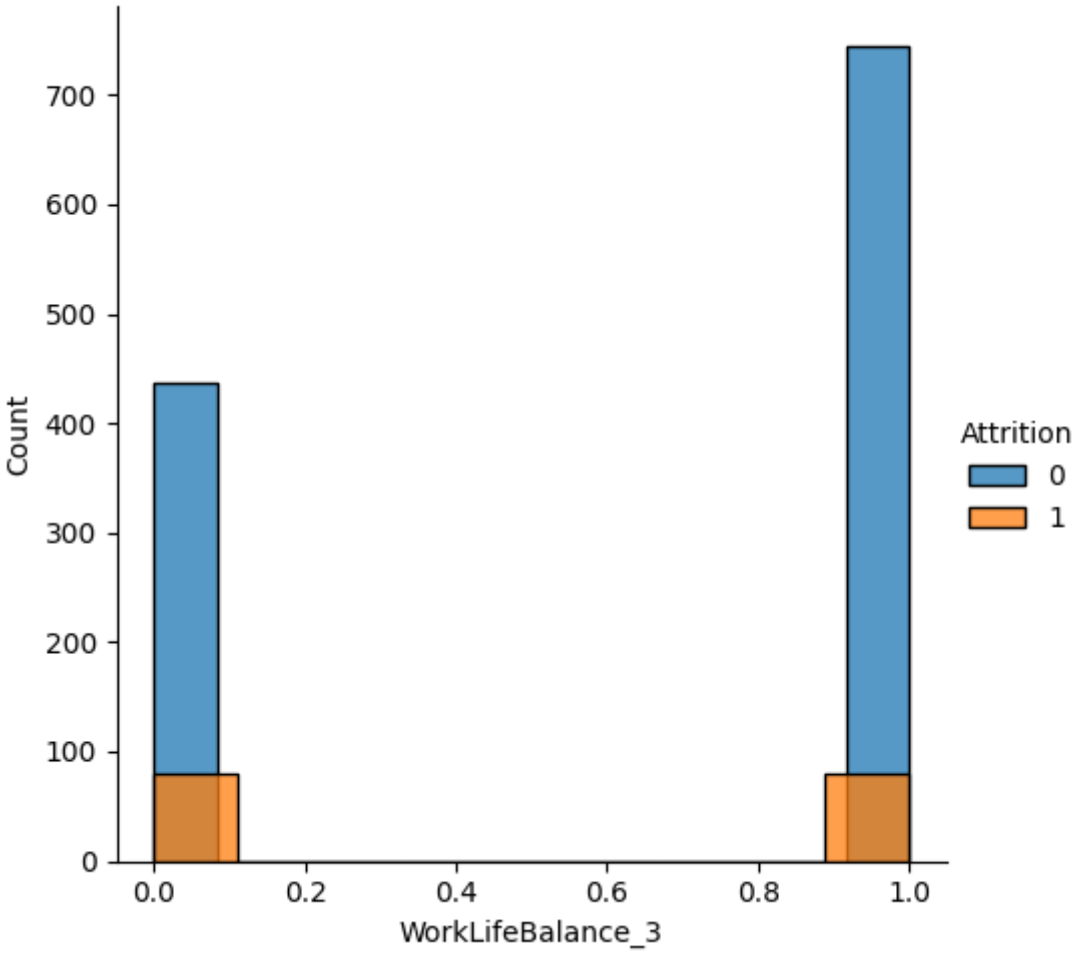

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
TrainingTimesLastYear_4
Attrition
0
1

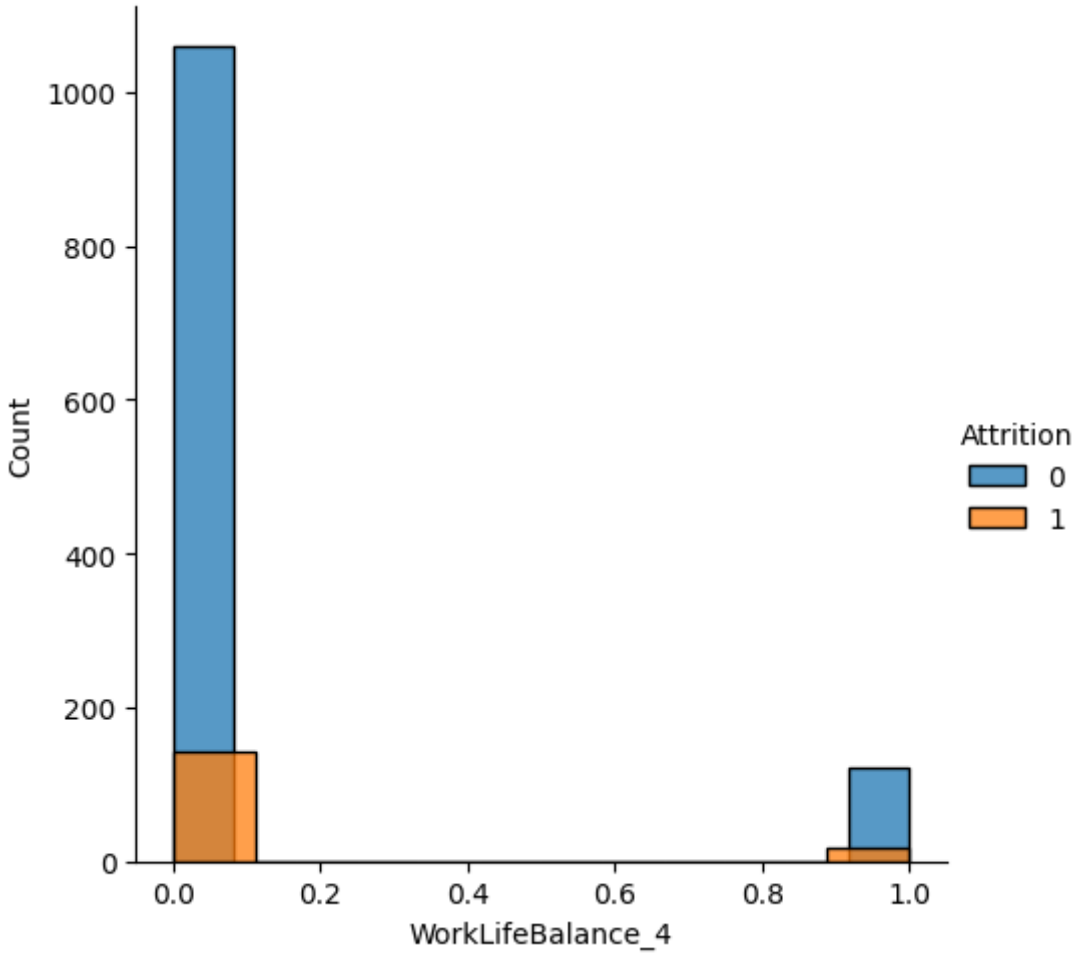

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
TrainingTimesLastYear_5
Attrition
0
1

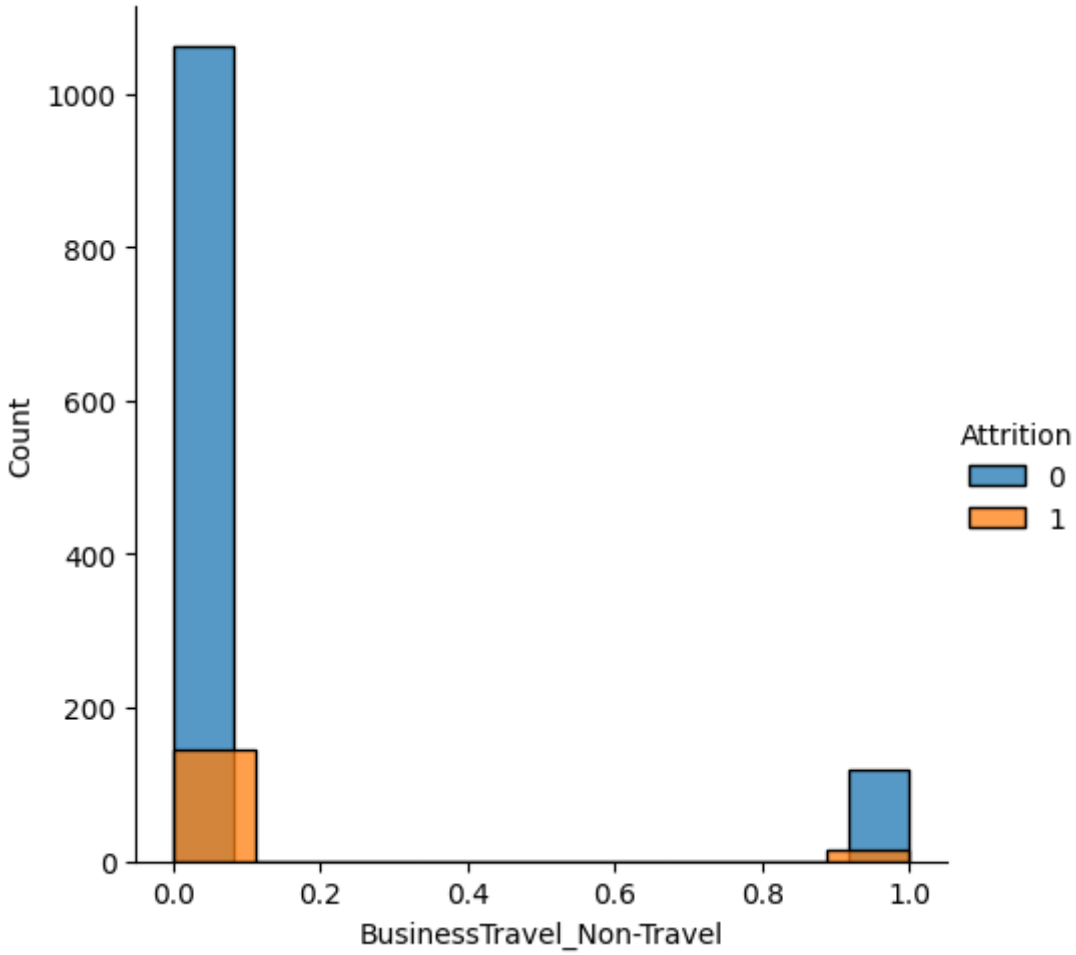

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
TrainingTimesLastYear_6
Attrition
0
1

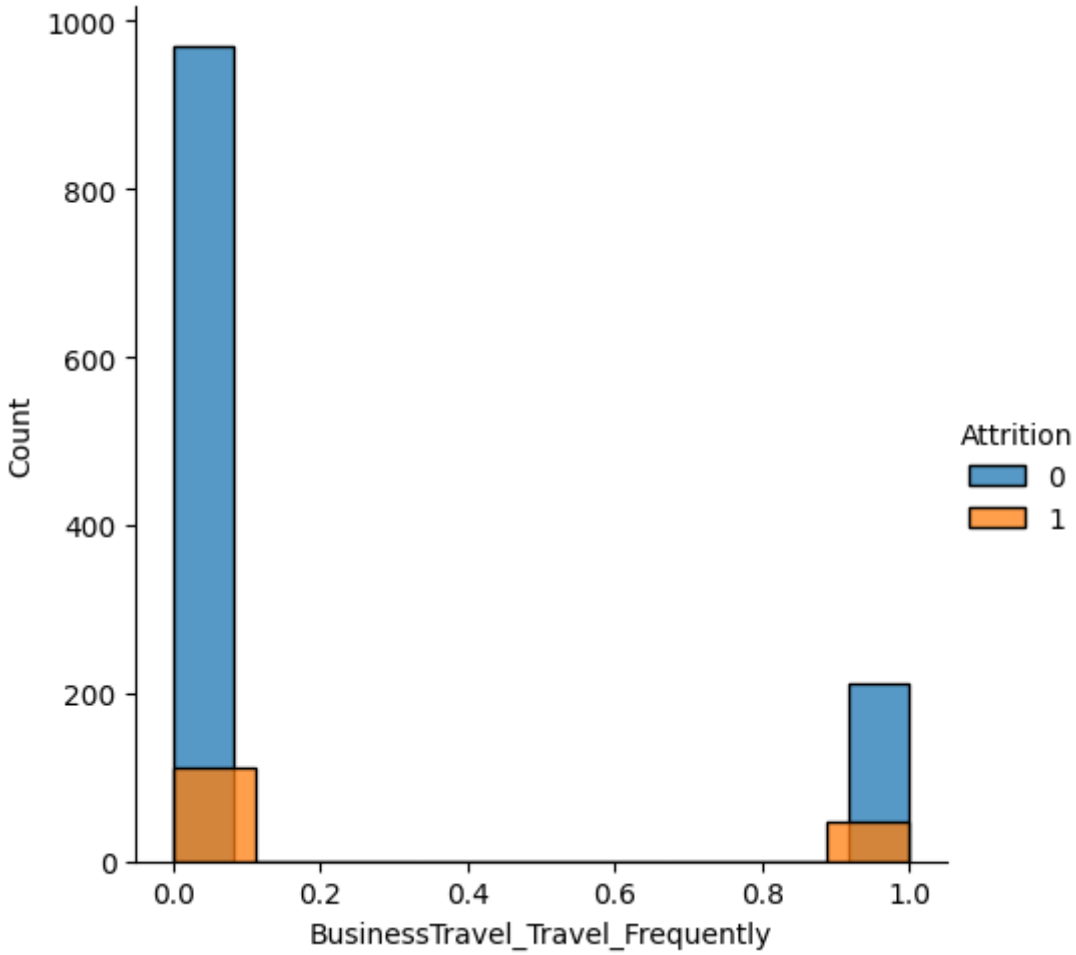

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
WorkLifeBalance_1
Attrition
0
1

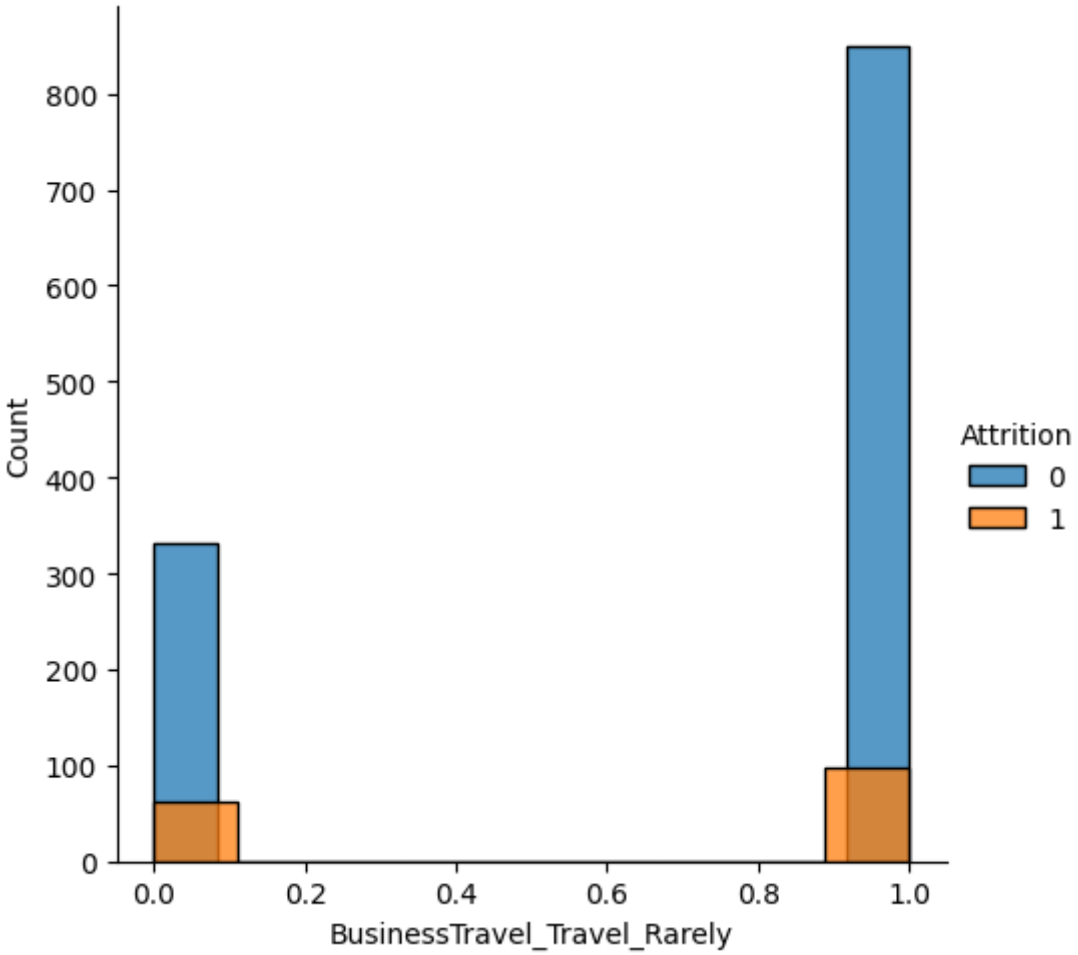

800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
WorkLifeBalance_2
Attrition
0
1

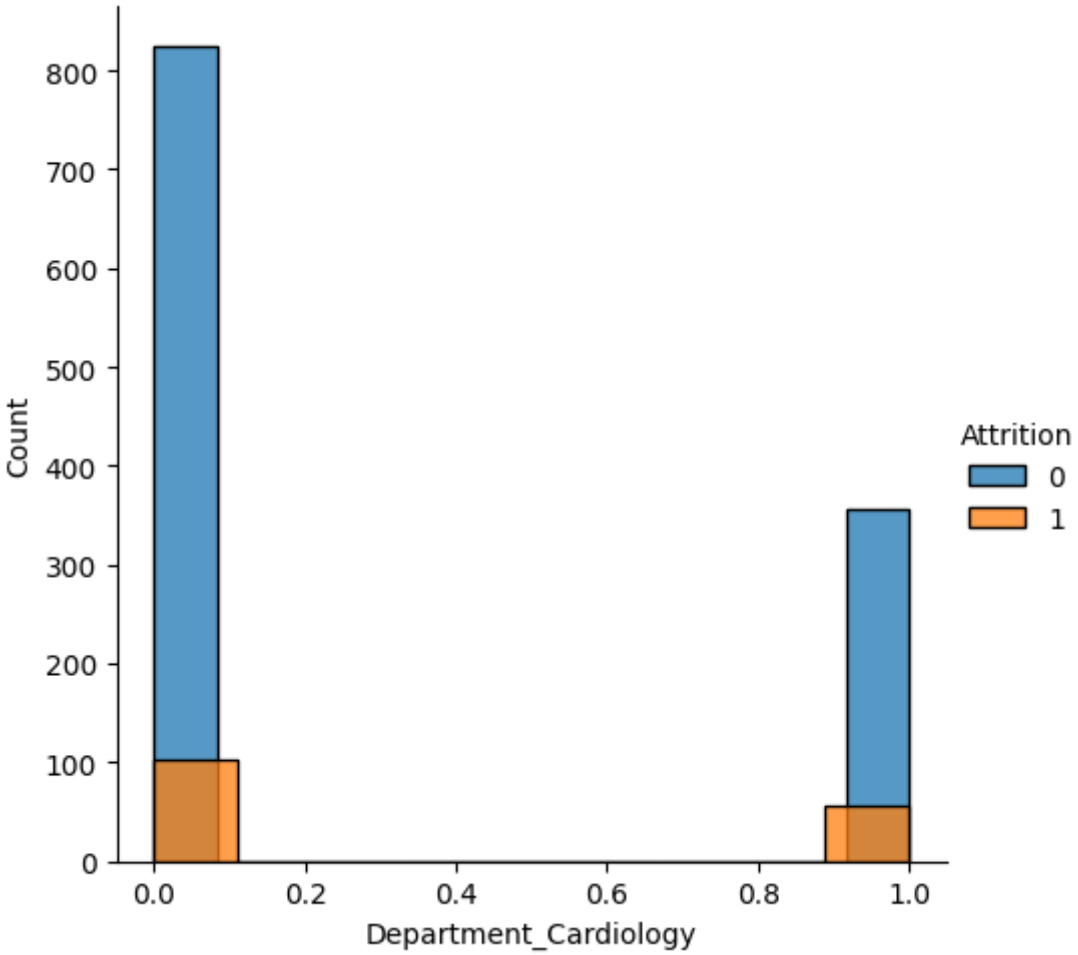

700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
WorkLifeBalance_3
Attrition
0
1

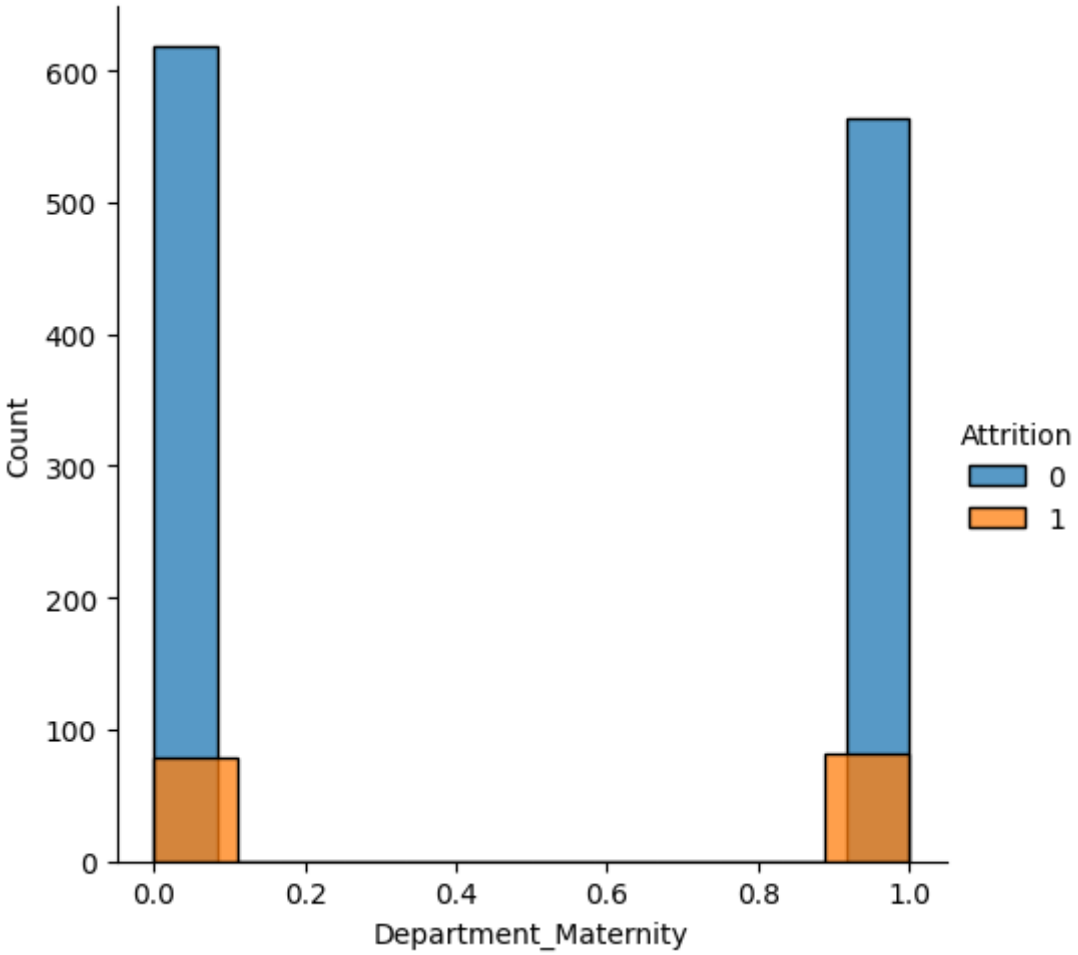

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
WorkLifeBalance_4
Attrition
0
1

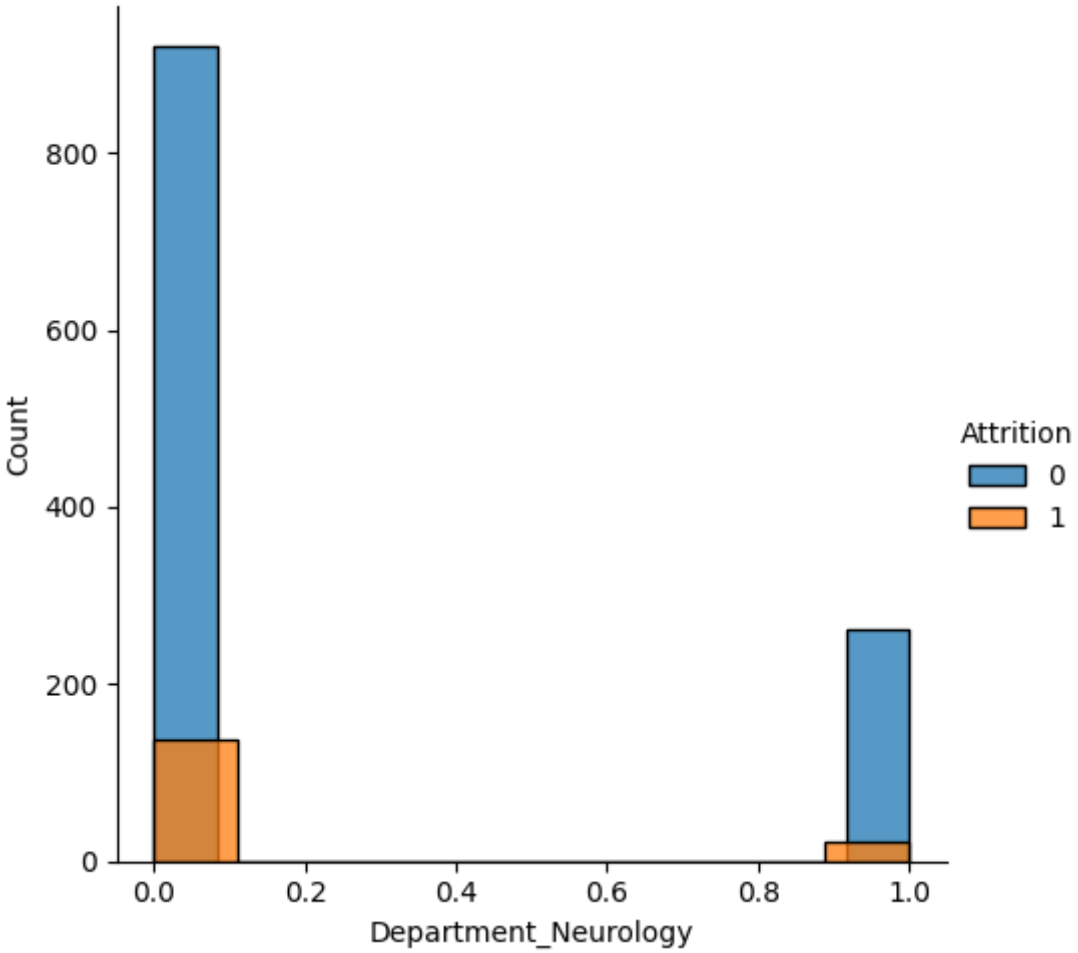

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
BusinessTravel_Non-Travel
Attrition
0
1

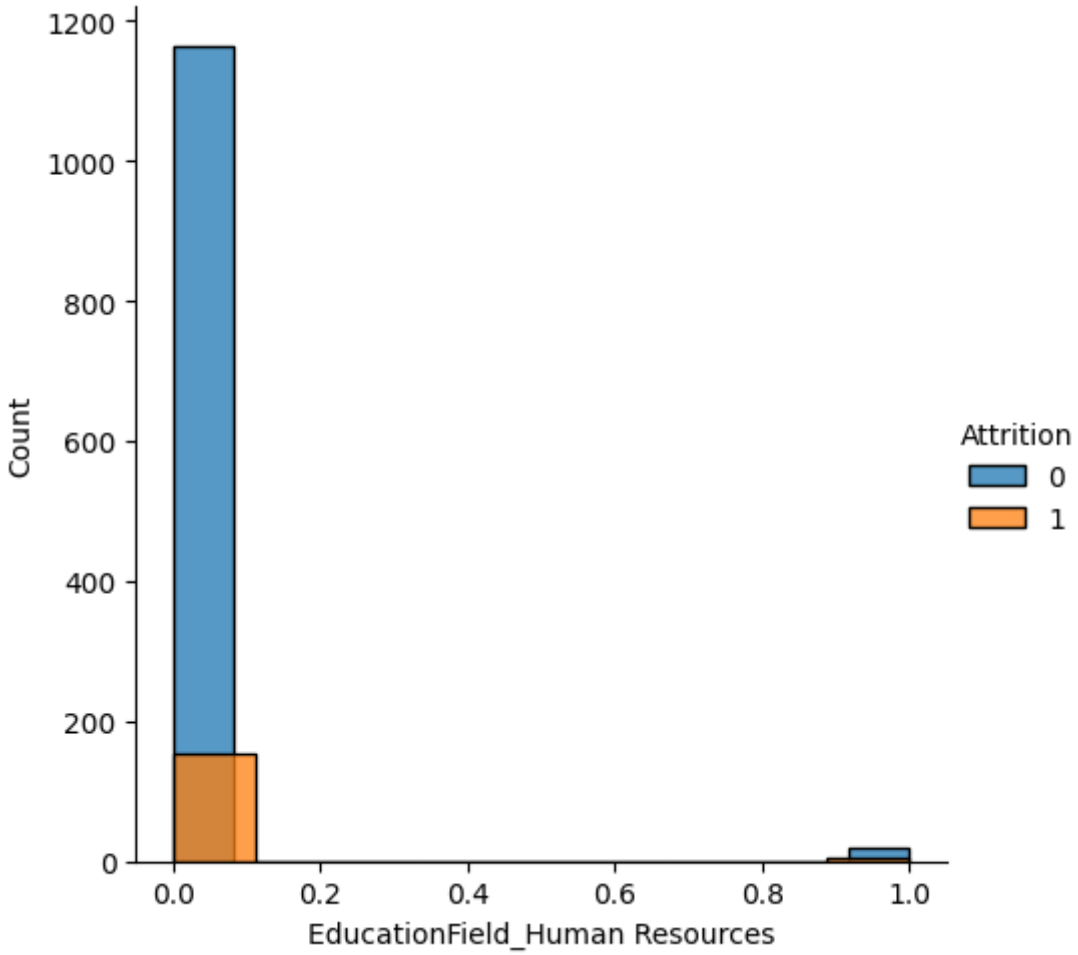

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
BusinessTravel_Travel_Frequently
Attrition
0
1

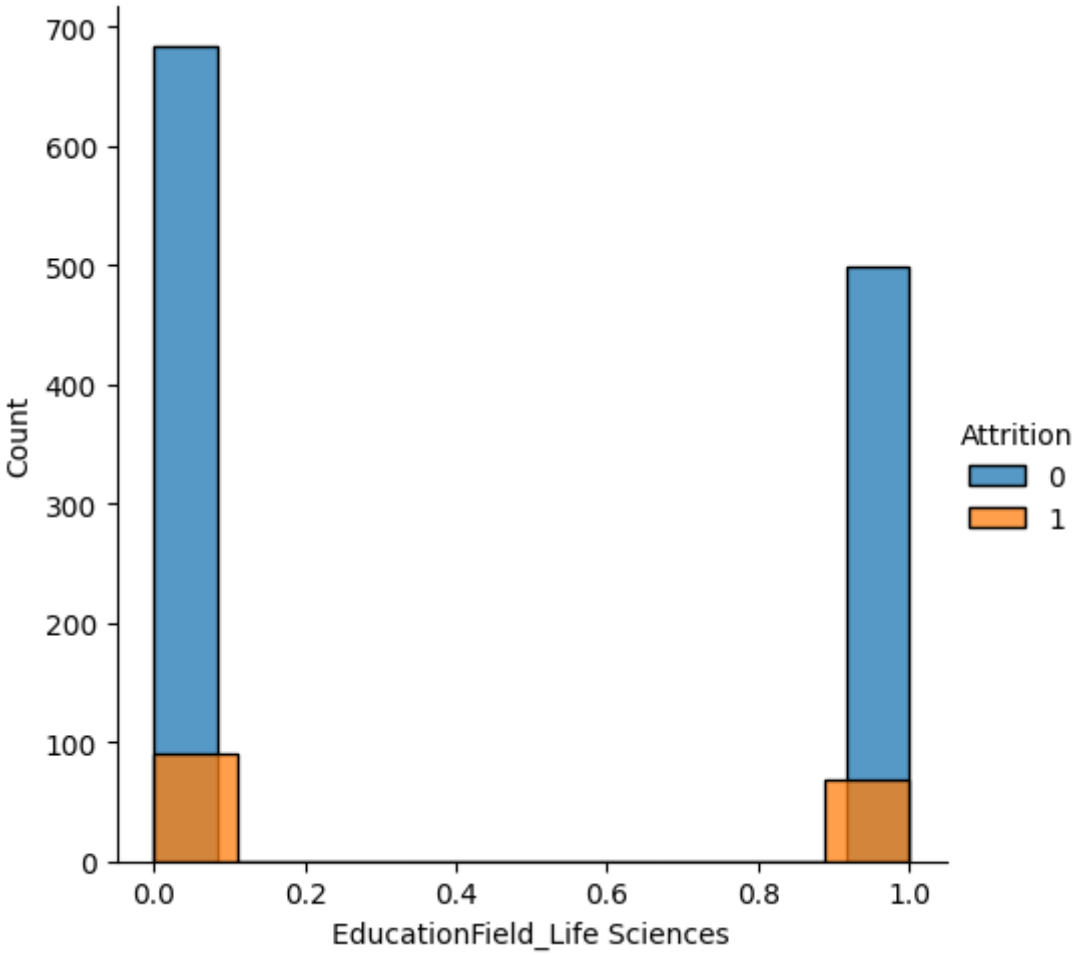

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
BusinessTravel_Travel_Rarely
Attrition
0
1

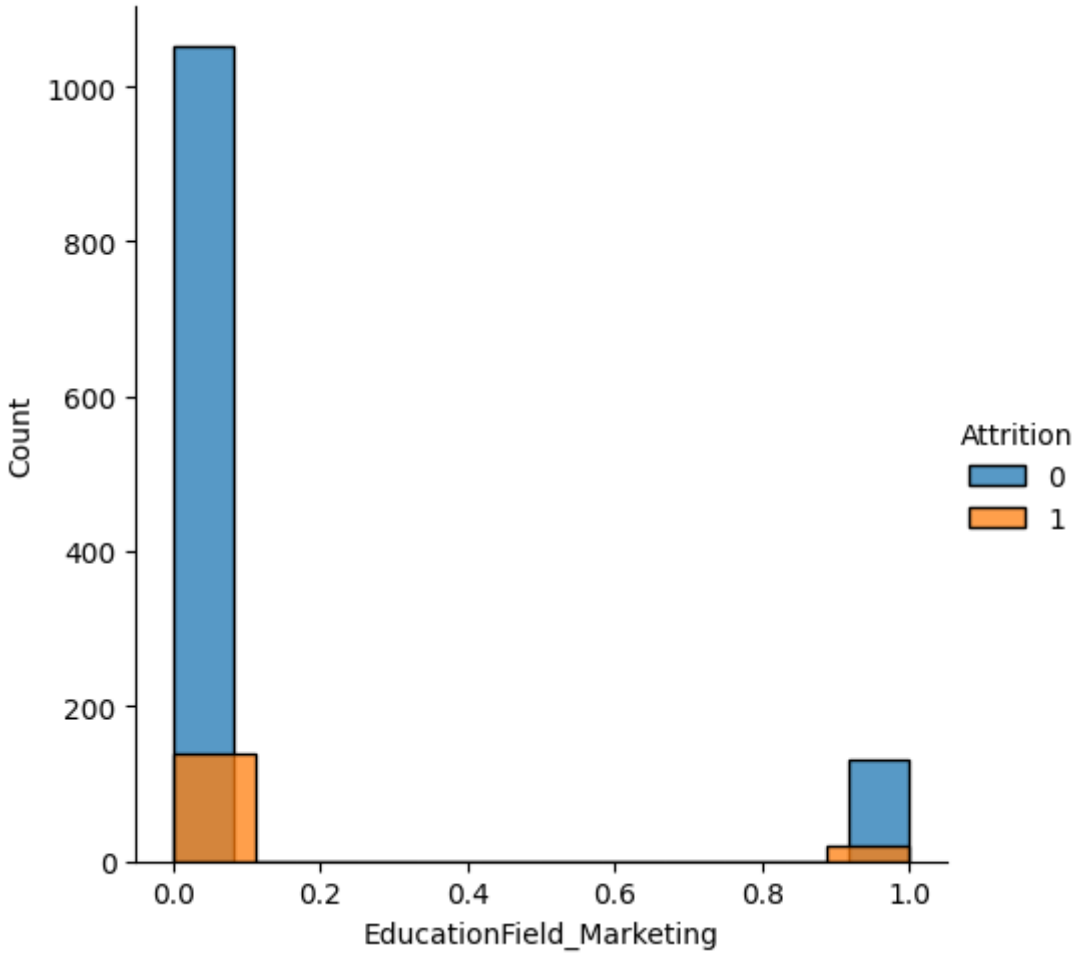

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Department_Cardiology
Attrition
0
1

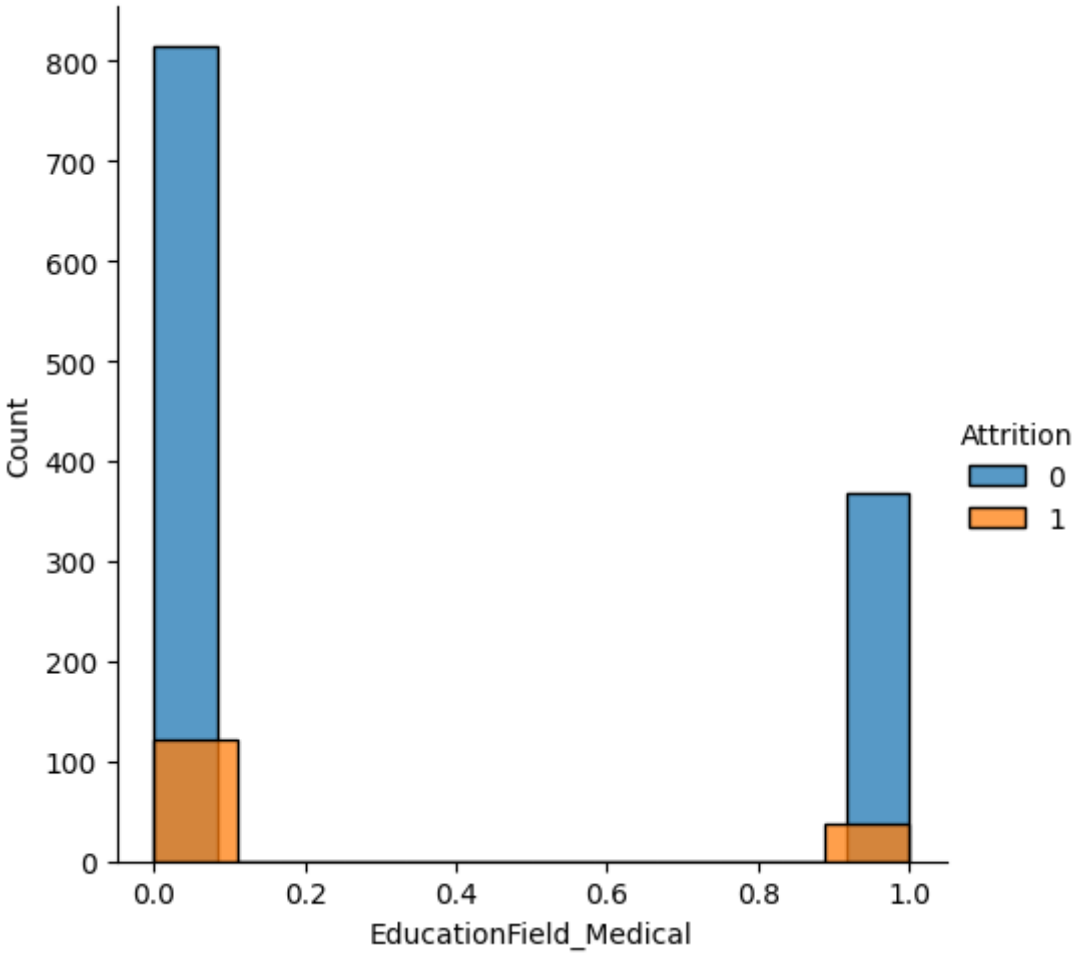

600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Department_Maternity
Attrition
0
1

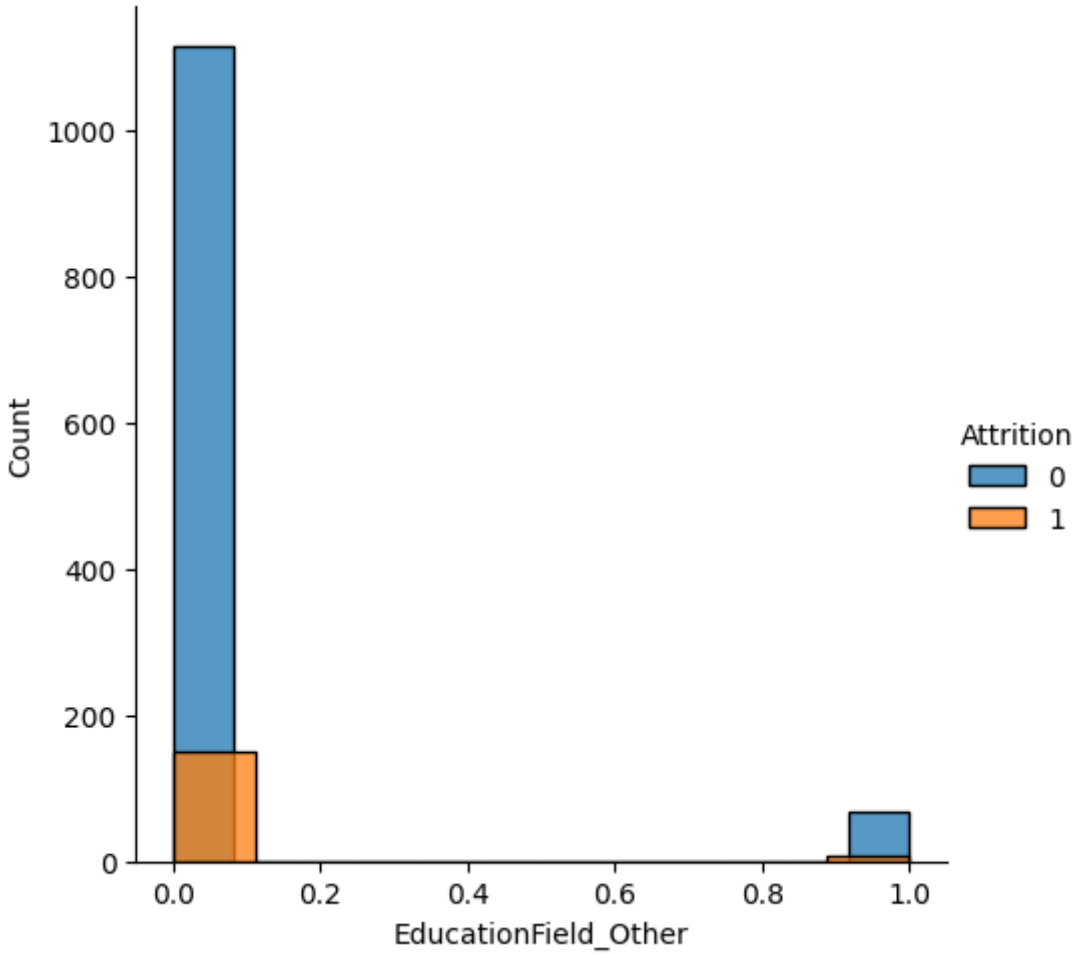

800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Department_Neurology
Attrition
0
1

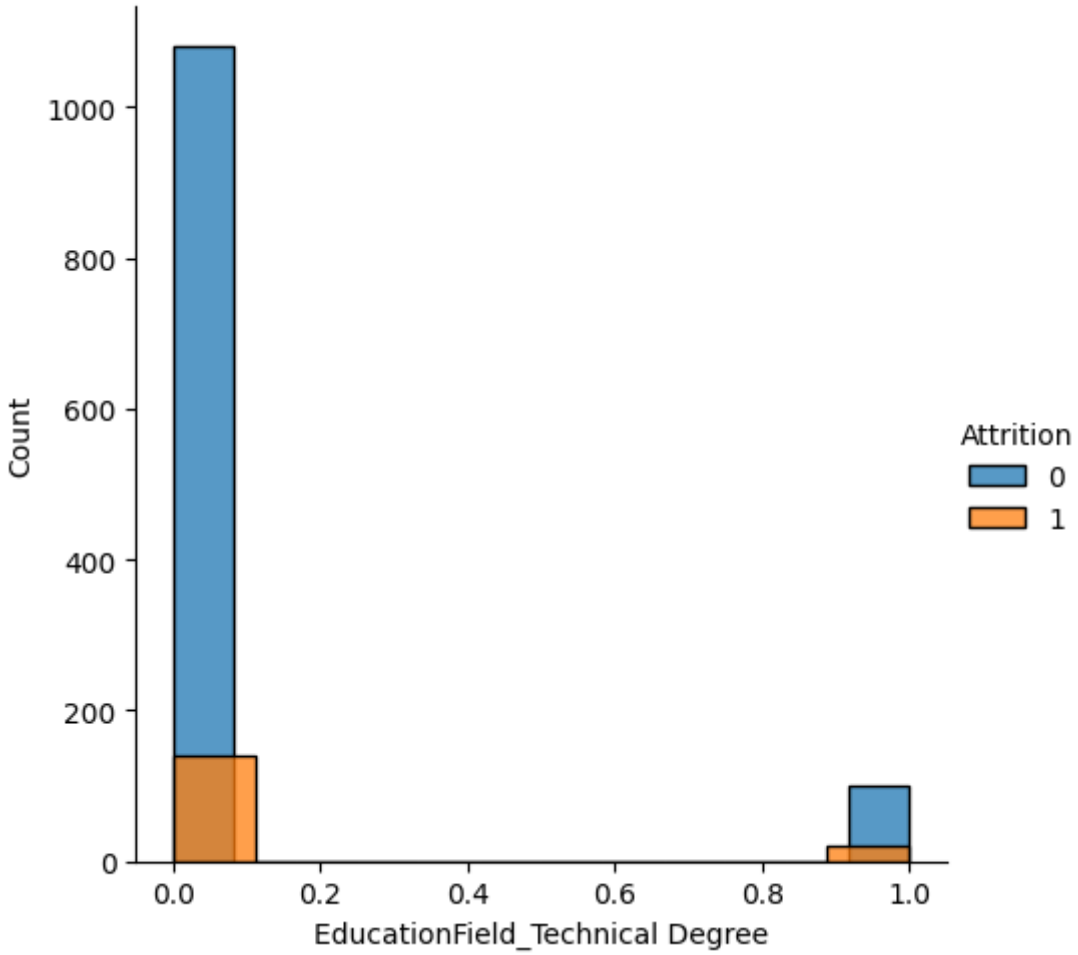

1200
1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
EducationField_Human Resources
Attrition
0
1

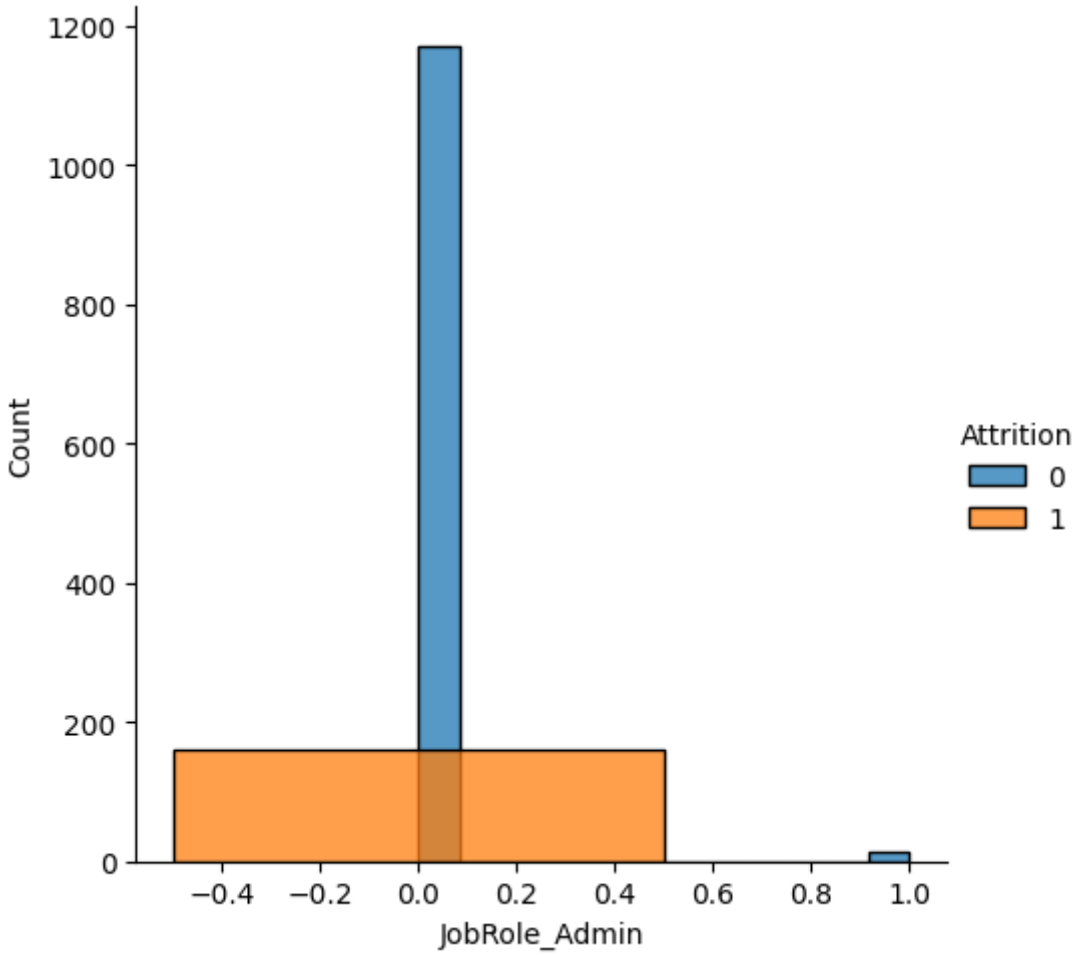

700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
EducationField_Life Sciences
Attrition
0
1

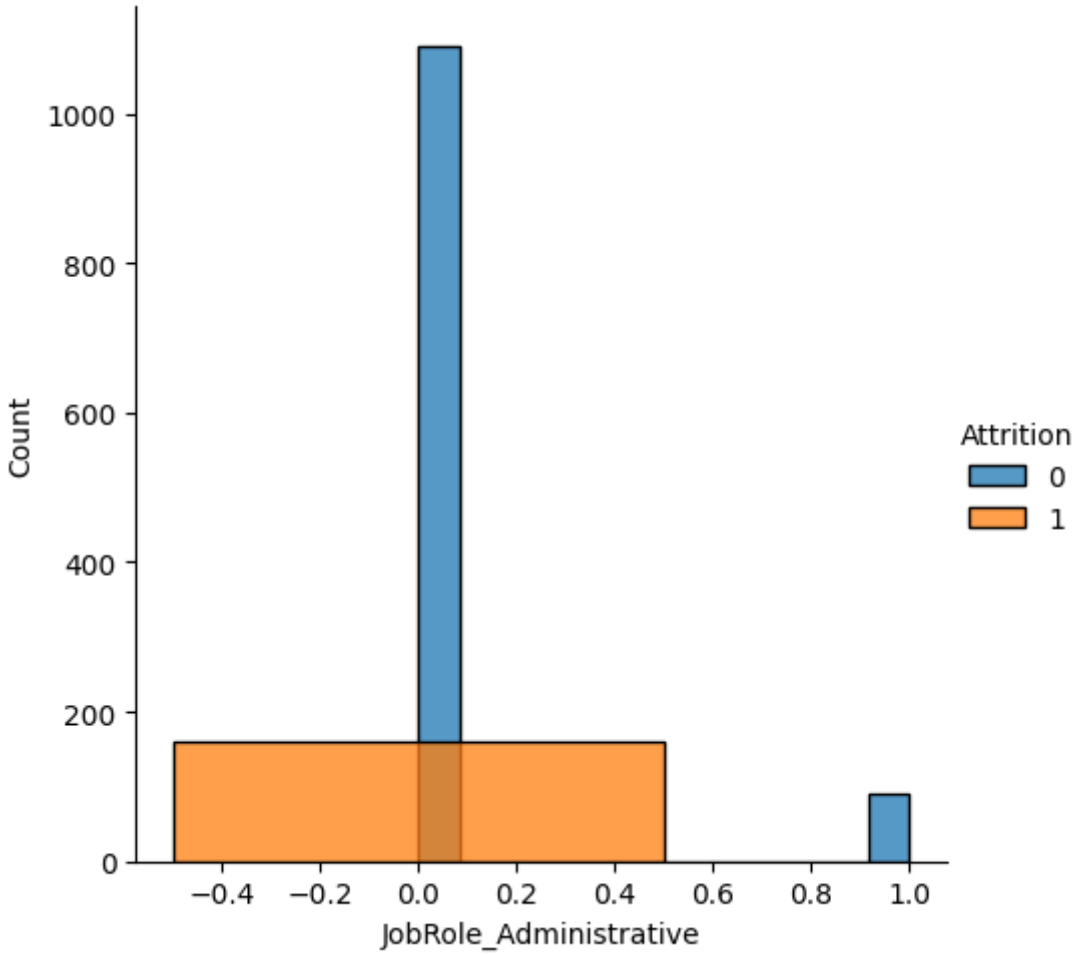

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
EducationField_Marketing
Attrition
0
1

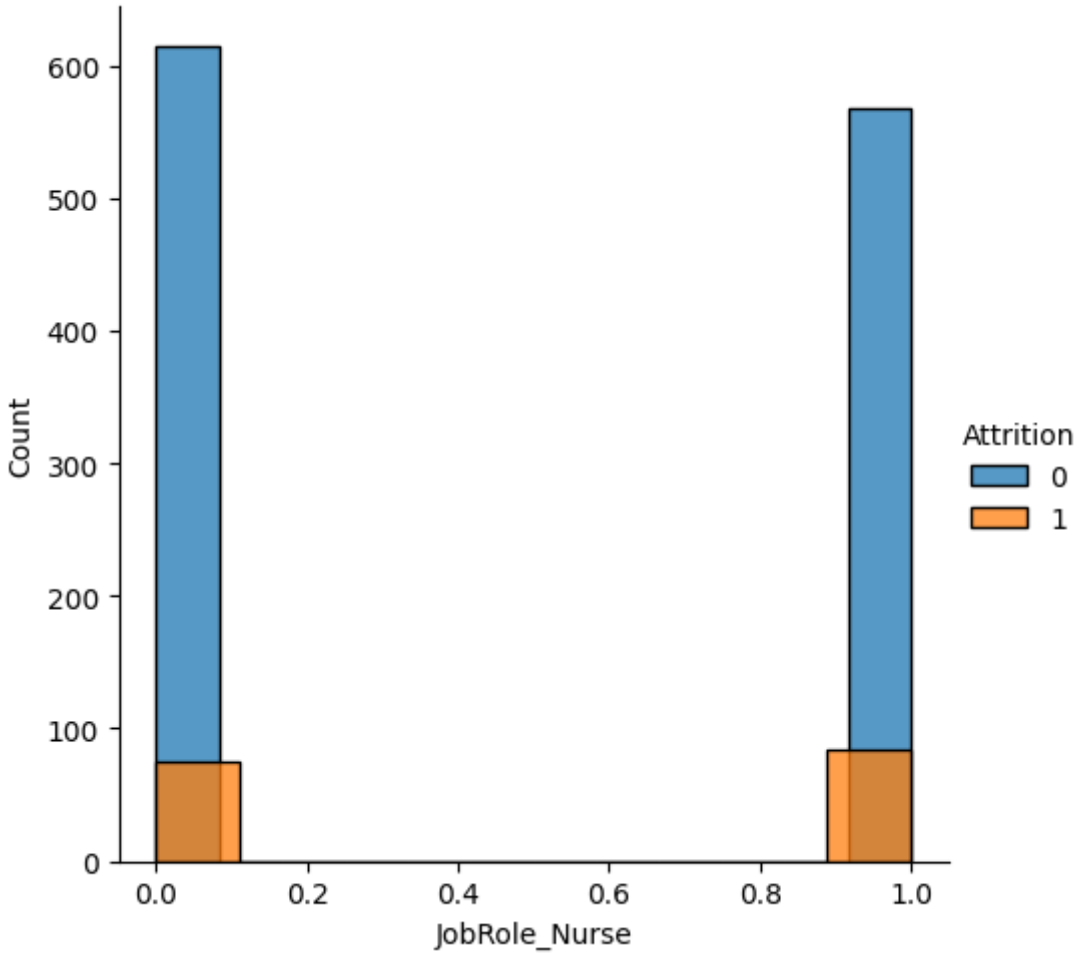

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
EducationField_Medical
Attrition
0
1

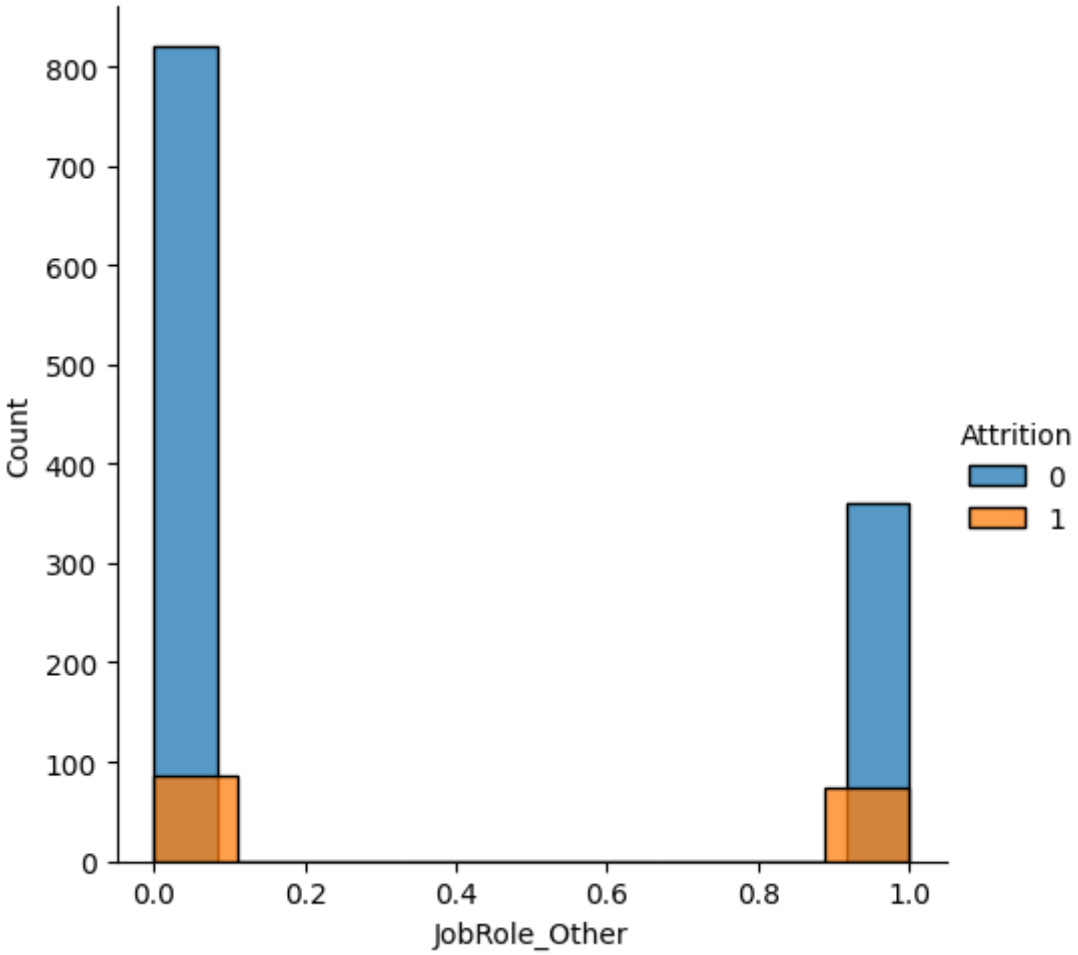

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
EducationField_Other
Attrition
0
1

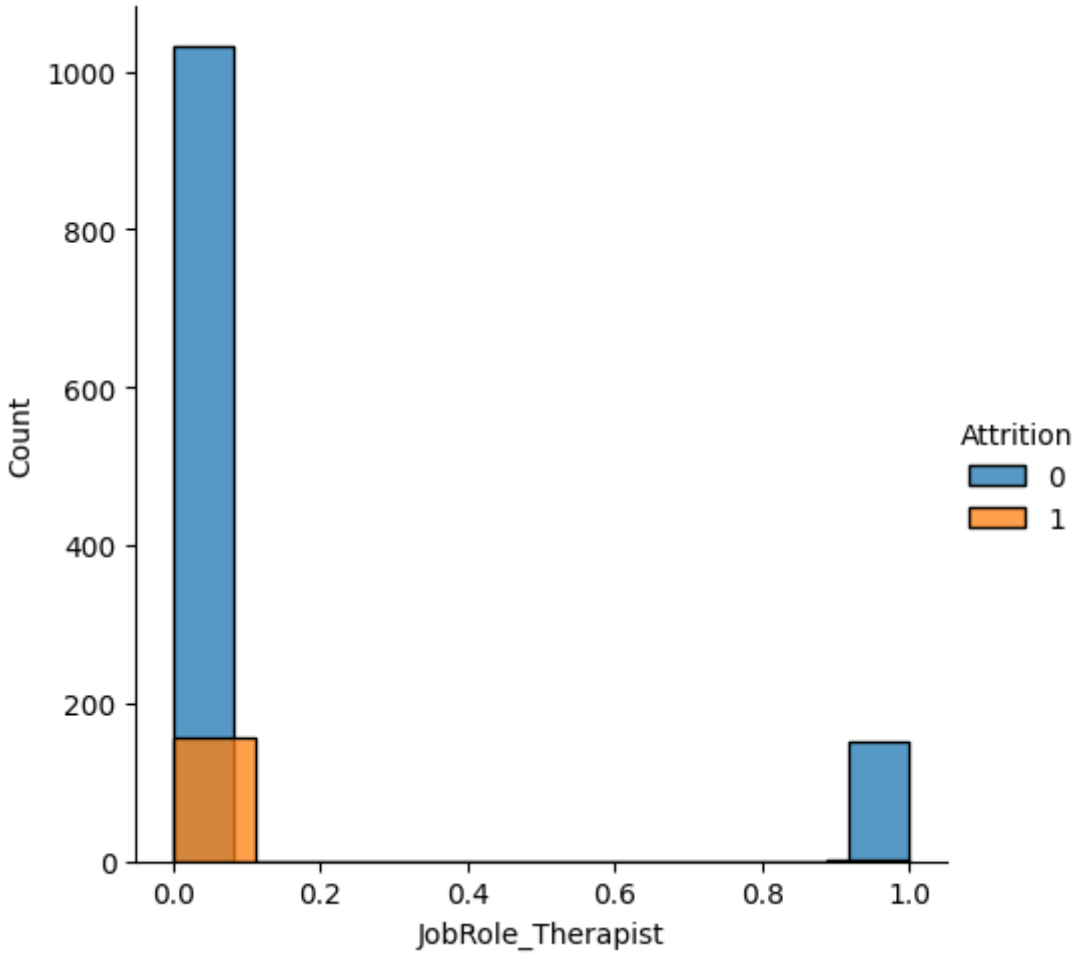

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
EducationField_Technical Degree
Attrition
0
1

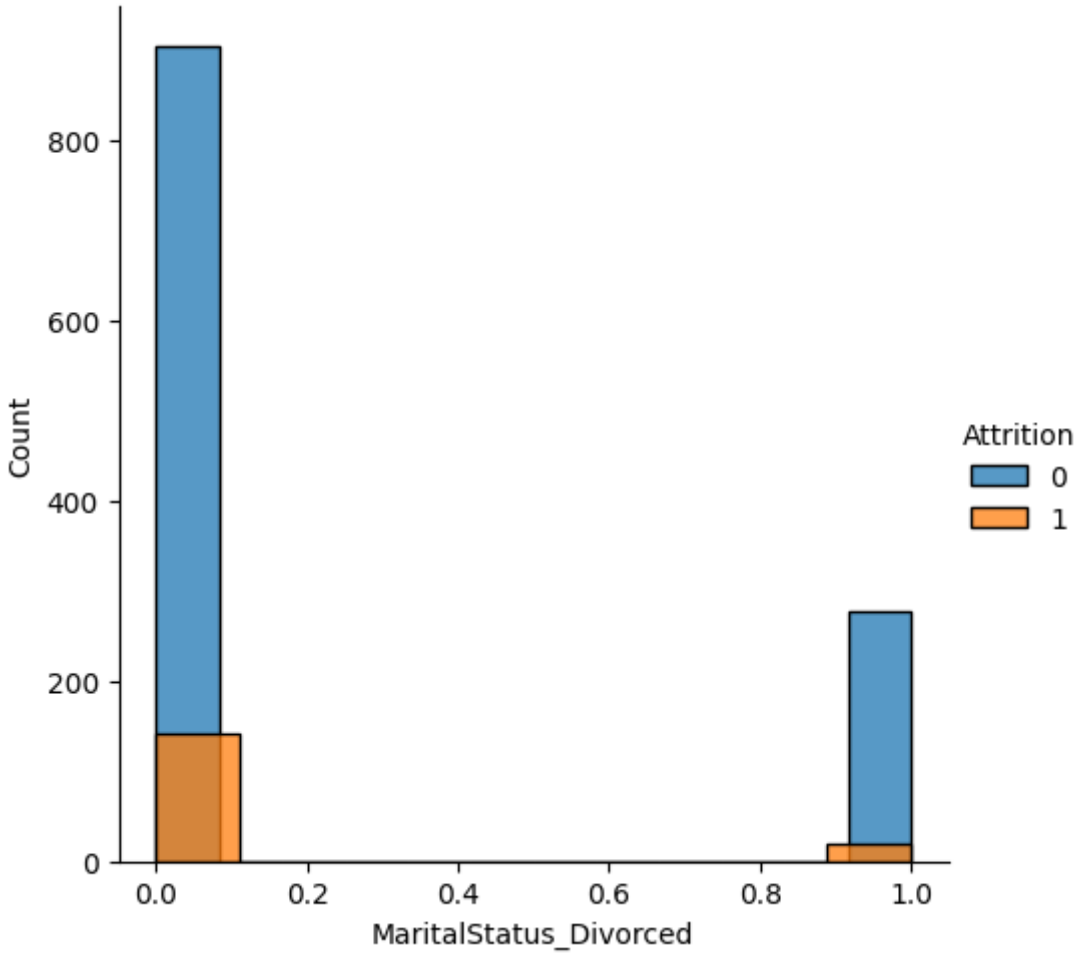

1200
1000
800
600
400
200
0
Count
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
JobRole_Admin
Attrition
0
1

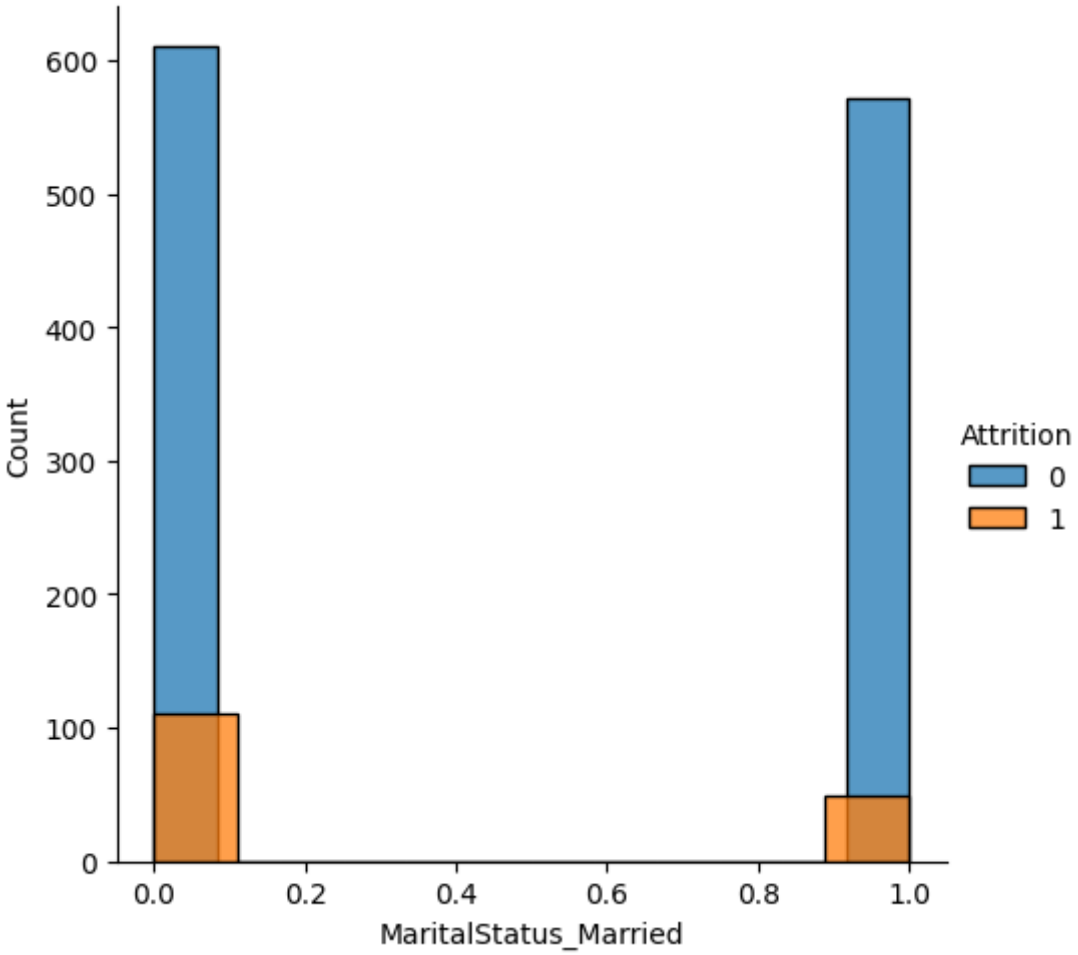

1000
800
600
400
200
0
Count
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
JobRole_Administrative
Attrition
0
1

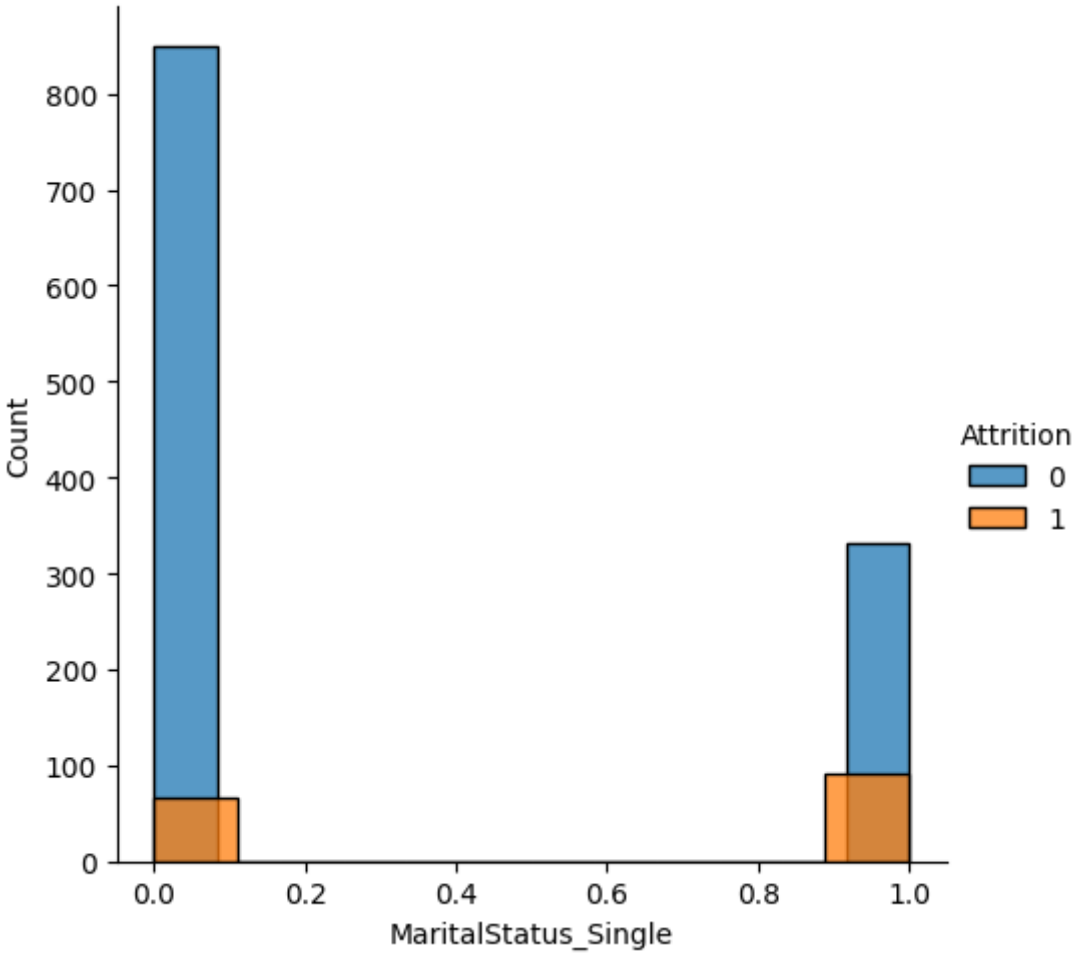

600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobRole_Nurse
Attrition
0
1

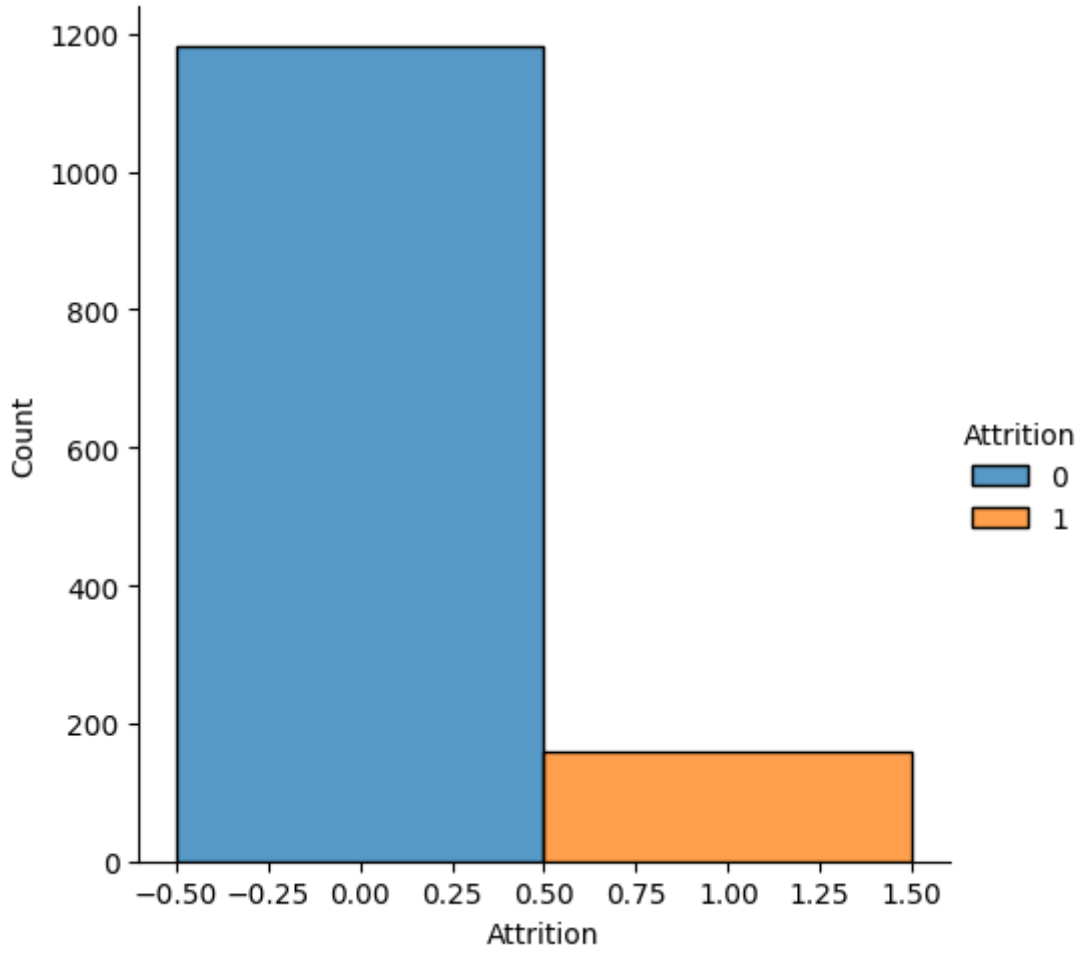

800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobRole_Other
Attrition
0
1


1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobRole_Therapist
Attrition
0
1


800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
MaritalStatus_Divorced
Attrition
0
1


600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
MaritalStatus_Married
Attrition
0
1


800
700
600
500
400
300
200
100
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
MaritalStatus_Single
Attrition
0
1



In [ ]:
```
# Feature selection using coefficient weights
estimator = LogisticRegression(max_iter=120)
selector = RFECV(estimator, min_features_to_select=20,
                 step=1, cv=5)
selector = selector.fit(data, np.ravel(labels))
names = selector.get_feature_names_out()

# See which features are correlated with Attrition
print(data[names].columns)
print(data[names].shape)
```

```
Index(['BusinessTravel', 'Department', 'DistanceFromHome', 'Education',
       'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement',
       'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus',
       'NumCompaniesWorked', 'OverTime', 'PercentSalaryHike',
       'PerformanceRating', 'RelationshipSatisfaction', 'Shift',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager', 'Age_Even', 'DailyRate_Even',
       'DistanceFromHome_Even', 'HourlyRate_Even', 'MonthlyIncome_Even',
       'MonthlyRate_Even', 'TotalWorkingYears_Even', 'YearsInCurrentRole_Ev
en',
       'YearsWithCurrManager_Even', 'NumCompaniesWorked_Even',
       'EnvironmentSatisfaction_1', 'EnvironmentSatisfaction_3',
       'EnvironmentSatisfaction_4', 'JobInvolvement_1', 'JobInvolvement_2',
       'JobInvolvement_3', 'JobInvolvement_4', 'JobLevel_1', 'JobLevel_2',
       'JobLevel_3', 'JobSatisfaction_1', 'JobSatisfaction_4',
       'RelationshipSatisfaction_4', 'Shift_0', 'Shift_1', 'Shift_2',
       'TrainingTimesLastYear_2', 'TrainingTimesLastYear_3',
       'WorkLifeBalance_1', 'WorkLifeBalance_3',
       'BusinessTravel_Travel_Frequently', 'BusinessTravel_Travel_Rarely',
       'Department_Cardiology', 'Department_Maternity',
       'EducationField_Marketing', 'EducationField_Medical',
       'JobRole_Therapist', 'MaritalStatus_Divorced', 'MaritalStatus_Marrie
d',
       'MaritalStatus_Single'],
      dtype='object')
(1340, 65)
```

In [ ]:
```
# Eliminate redundant features
good_data = data[names]
selected_features = mrmr.mrmr_classif(good_data,
                                      np.ravel(labels),
                                      K=20)
print(selected_features)

uncorr_data = good_data[selected_features]
uncorr_sub_data = submission_data[selected_features]
```

```
100%|██████████| 20/20 [00:00<00:00, 86.50it/s]
['OverTime', 'JobLevel_1', 'BusinessTravel_Travel_Rarely', 'JobInvolvemen
t', 'Shift_0', 'WorkLifeBalance_1', 'Age_Even', 'DistanceFromHome_Even', 'E
nvironmentSatisfaction', 'YearsInCurrentRole', 'MaritalStatus_Single', 'Tot
alWorkingYears', 'JobSatisfaction', 'JobLevel_2', 'JobInvolvement_1', 'Year
sWithCurrManager', 'JobLevel', 'EnvironmentSatisfaction_1', 'TrainingTimesL
astYear_2', 'MaritalStatus']
```

In [ ]:
```
# Save the selected data
fin_data = pd.concat([uncorr_data, labels], axis=1)
fin_data.to_csv("uncorr20_data.csv")
uncorr_sub_data.to_csv("uncorr20_sub_data.csv")
```

In [ ]:
```
# Feature engineering by making polynomial features
poly = PolynomialFeatures(degree=2)

poly_data = poly.fit_transform(uncorr_data)
```

```
poly_sub = poly.fit_transform(uncorr_sub_data)

poly_names = poly.get_feature_names_out()
# print(poly_names)

# print(poly_data.shape)

poly_data = pd.DataFrame(poly_data, columns=poly_names)
poly_sub = pd.DataFrame(poly_sub, columns=poly_names)
# print(poly_data.head())
```

In [ ]:

```
# Feature selection using coefficient weights
estimator = LogisticRegression(max_iter=120)
selector = RFECV(estimator, min_features_to_select=20,
                 step=1, cv=5)
selector = selector.fit(poly_data, np.ravel(labels))
names = selector.get_feature_names_out()

print(poly_data[names].columns)
print(poly_data[names].shape)
```

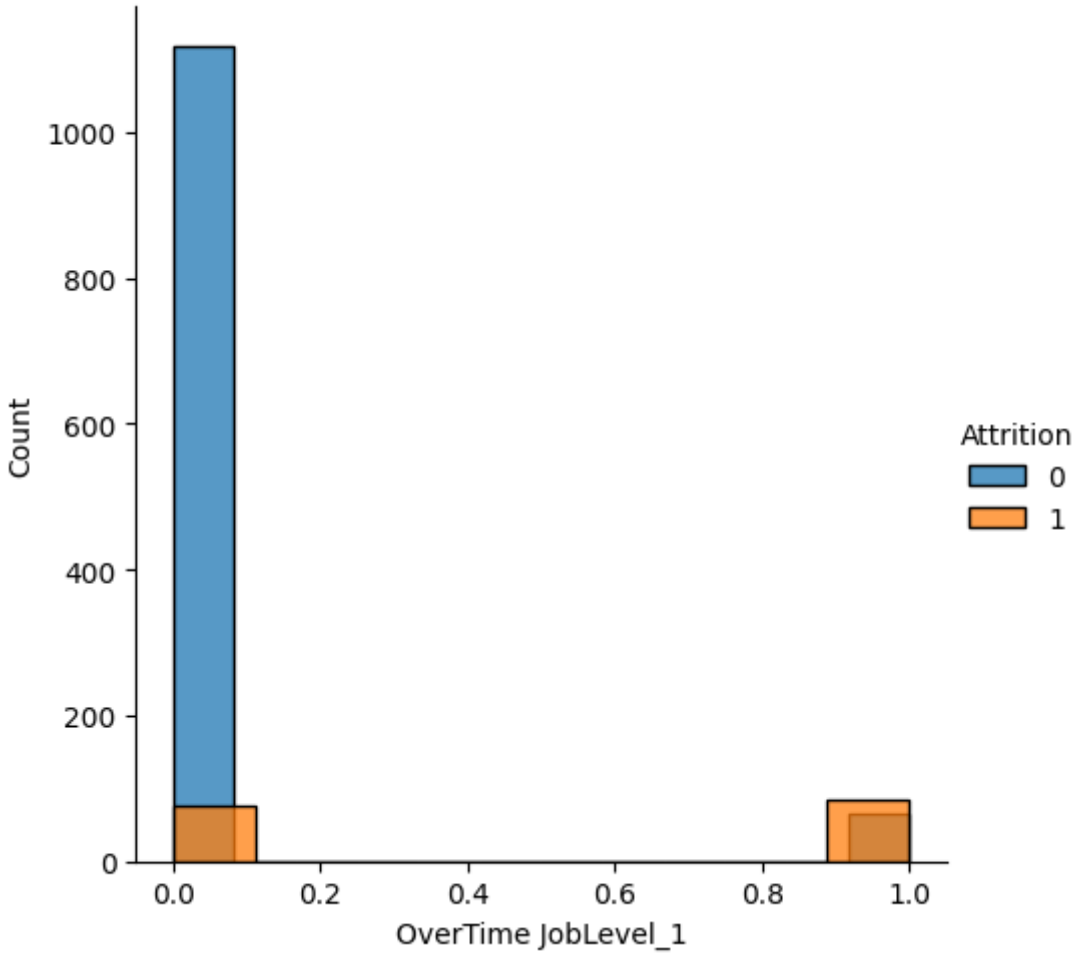
```
Index(['OverTime', 'YearsInCurrentRole', 'MaritalStatus_Single',
       'YearsWithCurrManager', 'OverTime^2', 'OverTime JobLevel_1',
       'OverTime YearsInCurrentRole', 'OverTime JobSatisfaction',
       'OverTime EnvironmentSatisfaction_1',
       'OverTime TrainingTimesLastYear_2', 'OverTime MaritalStatus',
       'JobLevel_1 WorkLifeBalance_1', 'JobLevel_1 Age_Even',
       'JobLevel_1 DistanceFromHome_Even', 'JobLevel_1 MaritalStatus_Singl
e',
       'JobLevel_1 EnvironmentSatisfaction_1',
       'BusinessTravel_Travel_Rarely WorkLifeBalance_1',
       'BusinessTravel_Travel_Rarely Age_Even',
       'BusinessTravel_Travel_Rarely DistanceFromHome_Even',
       'BusinessTravel_Travel_Rarely JobInvolvement_1',
       'BusinessTravel_Travel_Rarely JobLevel',
       'BusinessTravel_Travel_Rarely TrainingTimesLastYear_2',
       'BusinessTravel_Travel_Rarely MaritalStatus',
       'JobInvolvement MaritalStatus_Single', 'JobInvolvement JobSatisfacti
on',
       'JobInvolvement JobLevel_2', 'JobInvolvement EnvironmentSatisfaction
_1',
       'JobInvolvement MaritalStatus', 'Shift_0 WorkLifeBalance_1',
       'Shift_0 DistanceFromHome_Even', 'Shift_0 EnvironmentSatisfaction',
       'Shift_0 YearsInCurrentRole', 'Shift_0 MaritalStatus_Single',
       'Shift_0 JobInvolvement_1', 'Shift_0 YearsWithCurrManager',
       'Shift_0 EnvironmentSatisfaction_1', 'Shift_0 TrainingTimesLastYear_
2',
       'Shift_0 MaritalStatus', 'WorkLifeBalance_1 DistanceFromHome_Even',
       'WorkLifeBalance_1 YearsInCurrentRole',
       'WorkLifeBalance_1 TotalWorkingYears',
       'WorkLifeBalance_1 JobSatisfaction', 'WorkLifeBalance_1 JobLevel_2',
       'WorkLifeBalance_1 YearsWithCurrManager', 'Age_Even^2',
       'Age_Even EnvironmentSatisfaction', 'Age_Even YearsInCurrentRole',
       'Age_Even MaritalStatus_Single', 'Age_Even JobSatisfaction',
       'Age_Even JobLevel_2', 'Age_Even MaritalStatus',
       'DistanceFromHome_Even EnvironmentSatisfaction',
       'DistanceFromHome_Even JobLevel_2',
       'DistanceFromHome_Even JobInvolvement_1',
       'DistanceFromHome_Even TrainingTimesLastYear_2',
       'DistanceFromHome_Even MaritalStatus',
       'EnvironmentSatisfaction YearsInCurrentRole',
       'EnvironmentSatisfaction MaritalStatus_Single',
       'EnvironmentSatisfaction JobInvolvement_1',
       'EnvironmentSatisfaction YearsWithCurrManager',
       'EnvironmentSatisfaction TrainingTimesLastYear_2',
       'EnvironmentSatisfaction MaritalStatus',
       'YearsInCurrentRole MaritalStatus_Single',
       'YearsInCurrentRole MaritalStatus', 'MaritalStatus_Single^2',
       'MaritalStatus_Single TotalWorkingYears',
       'MaritalStatus_Single JobLevel_2',
       'MaritalStatus_Single JobInvolvement_1',
       'MaritalStatus_Single JobLevel',
       'MaritalStatus_Single EnvironmentSatisfaction_1',
       'MaritalStatus_Single MaritalStatus', 'JobSatisfaction JobLevel_2',
       'JobSatisfaction TrainingTimesLastYear_2',
       'JobSatisfaction MaritalStatus', 'JobLevel_2 JobInvolvement_1',
       'JobLevel_2 MaritalStatus', 'JobInvolvement_1 YearsWithCurrManager',
```

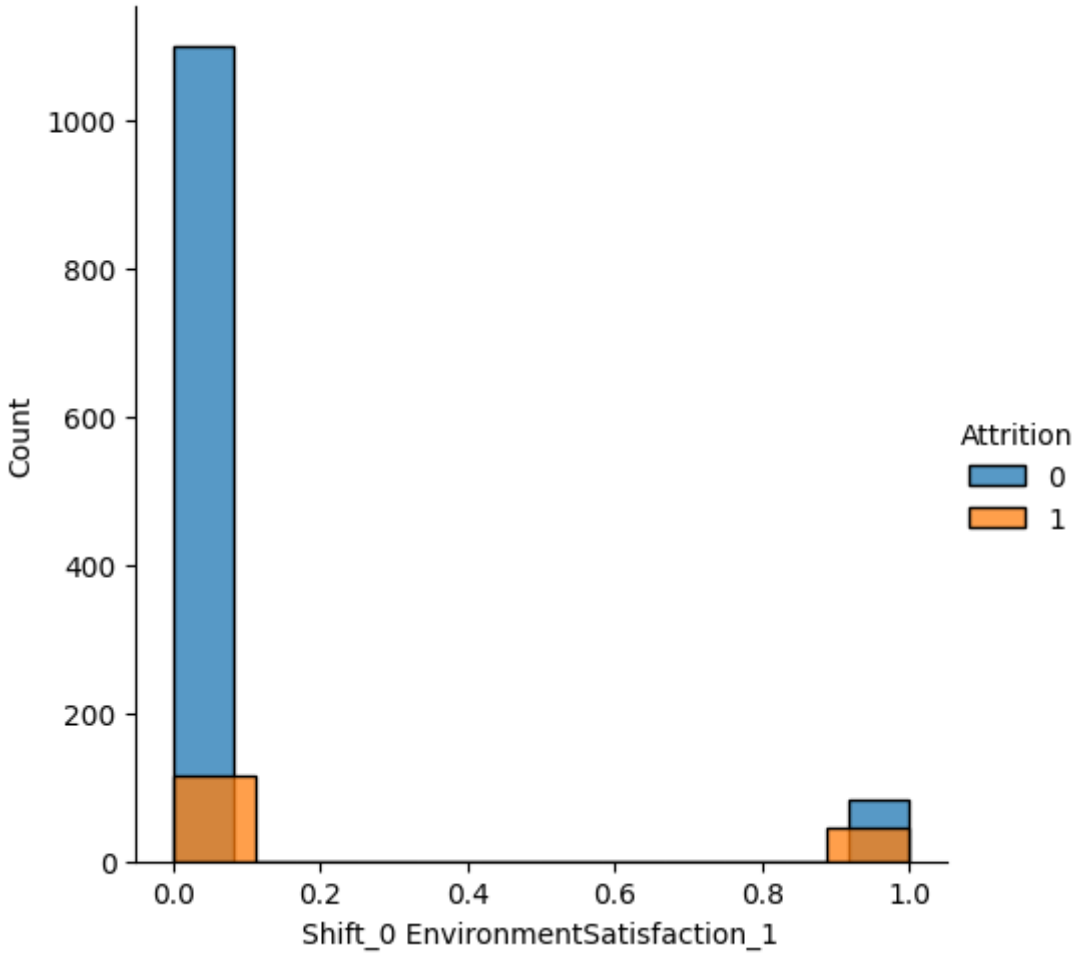
```
       'JobInvolvement_1 JobLevel', 'JobInvolvement_1 TrainingTimesLastYear
_2',
       'JobInvolvement_1 MaritalStatus', 'YearsWithCurrManager JobLevel',
       'YearsWithCurrManager EnvironmentSatisfaction_1',
       'YearsWithCurrManager MaritalStatus',
       'EnvironmentSatisfaction_1 MaritalStatus', 'TrainingTimesLastYear_2^
2',
       'MaritalStatus^2'],
      dtype='object')
(1340, 86)
```

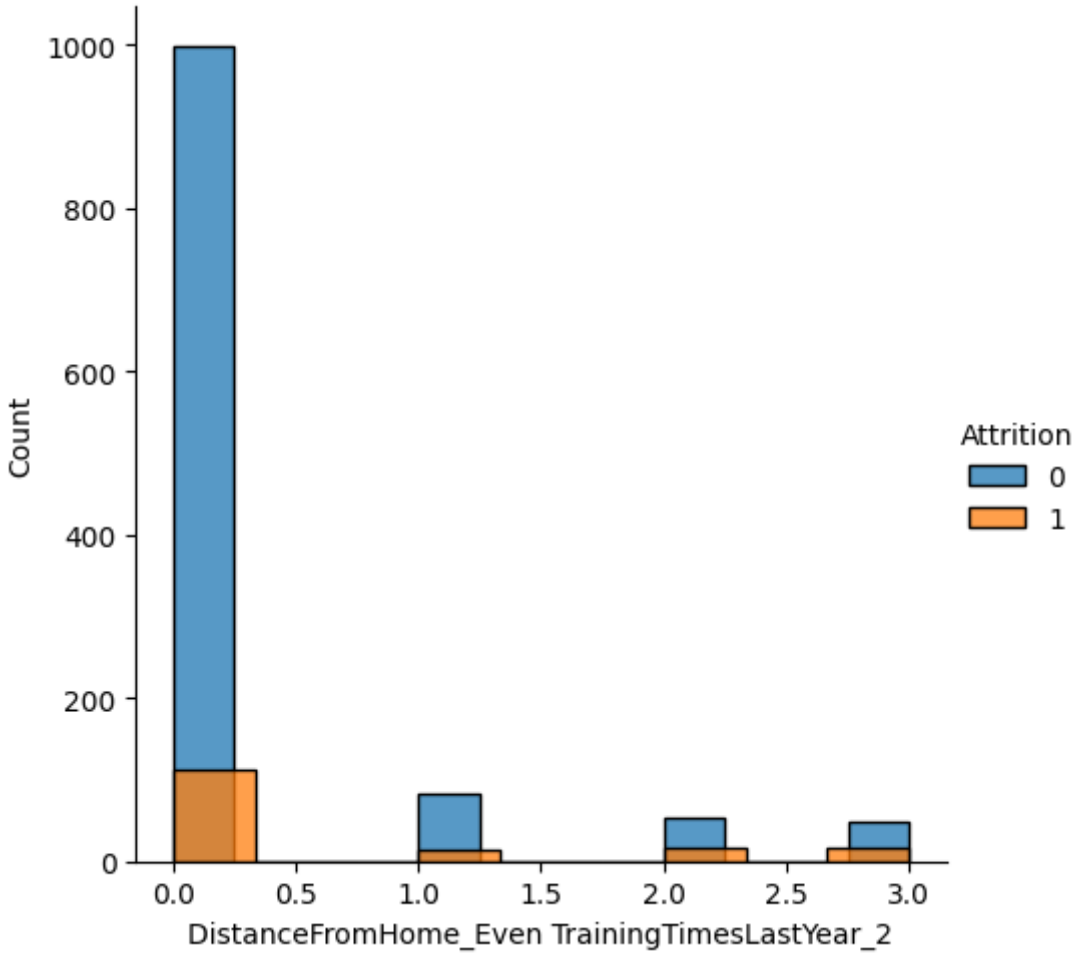
In [ ]:
```
# Eliminate redundant features
good_poly_data = poly_data[names]
selected_features = mrmr.mrmr_classif(good_poly_data,
                                      np.ravel(labels),
                                      K=20)
print(selected_features)

uncorr_poly_data = good_poly_data[selected_features]
uncorr_poly_sub_data = poly_sub[selected_features]
```

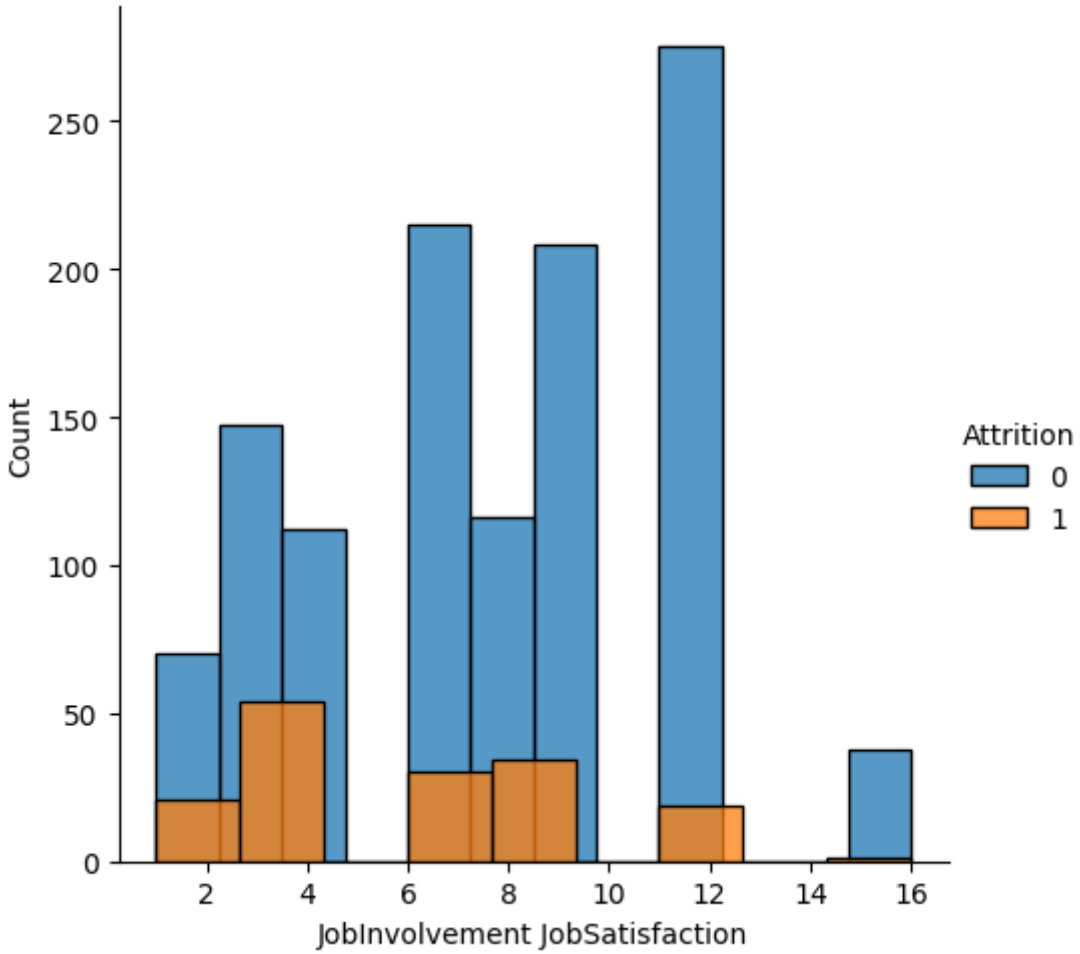
```
100%|██████████| 20/20 [00:00<00:00, 50.45it/s]
['OverTime JobLevel_1', 'Shift_0 EnvironmentSatisfaction_1', 'DistanceFromH
ome_Even TrainingTimesLastYear_2', 'JobInvolvement JobSatisfaction', 'OverT
ime MaritalStatus', 'BusinessTravel_Travel_Rarely Age_Even', 'JobLevel_1 Ma
ritalStatus_Single', 'OverTime', 'Shift_0 JobInvolvement_1', 'YearsWithCurr
Manager', 'OverTime EnvironmentSatisfaction_1', 'JobLevel_1 WorkLifeBalance
_1', 'OverTime^2', 'Age_Even EnvironmentSatisfaction', 'Shift_0 DistanceFro
mHome_Even', 'JobInvolvement JobLevel_2', 'OverTime TrainingTimesLastYear_
2', 'YearsInCurrentRole', 'MaritalStatus_Single EnvironmentSatisfaction_1',
'OverTime JobSatisfaction']
```

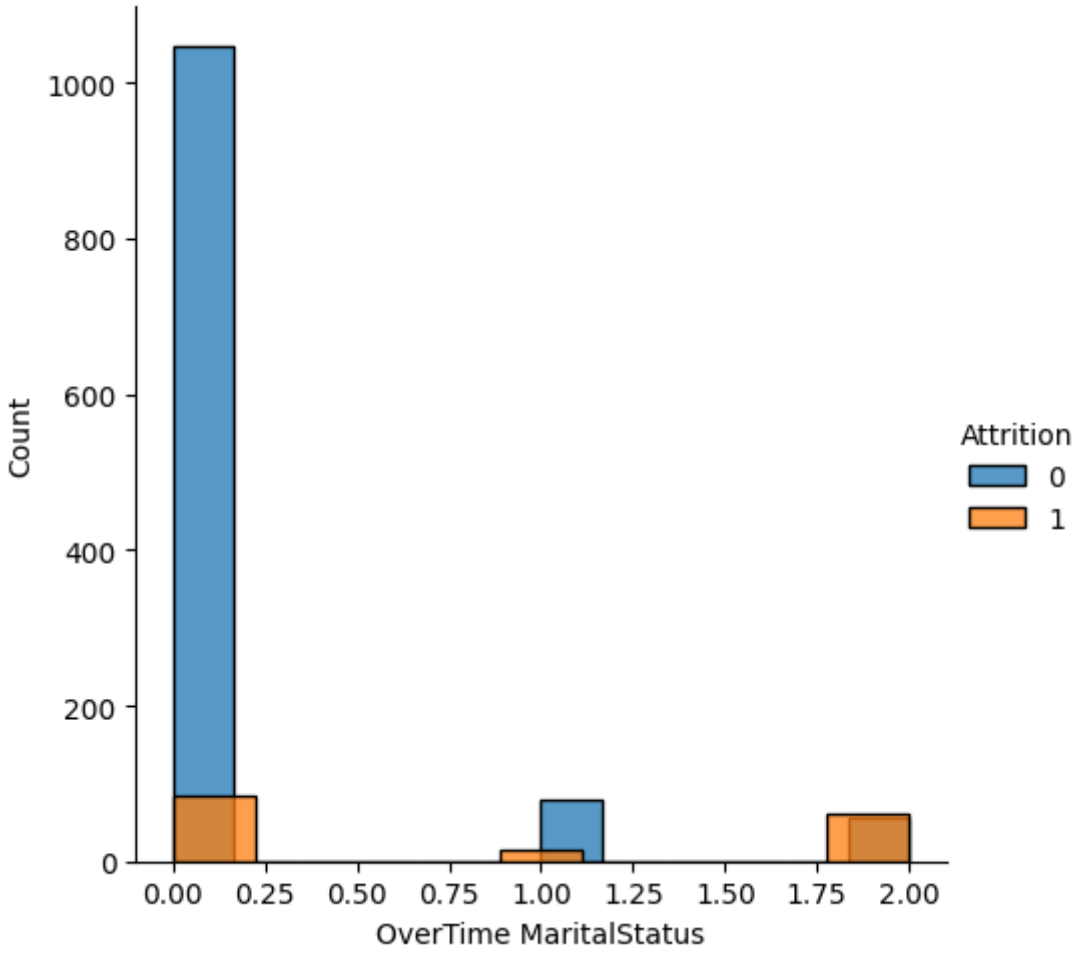
In [ ]:
```
# Generate histograms to check usefulness of features selected
fin_poly_data = pd.concat([uncorr_poly_data, labels], axis=1)
for c in fin_poly_data.columns:
    sns.FacetGrid(fin_poly_data,
                  hue="Attrition",
                  height= 5).map(sns.histplot,c).add_legend()
```

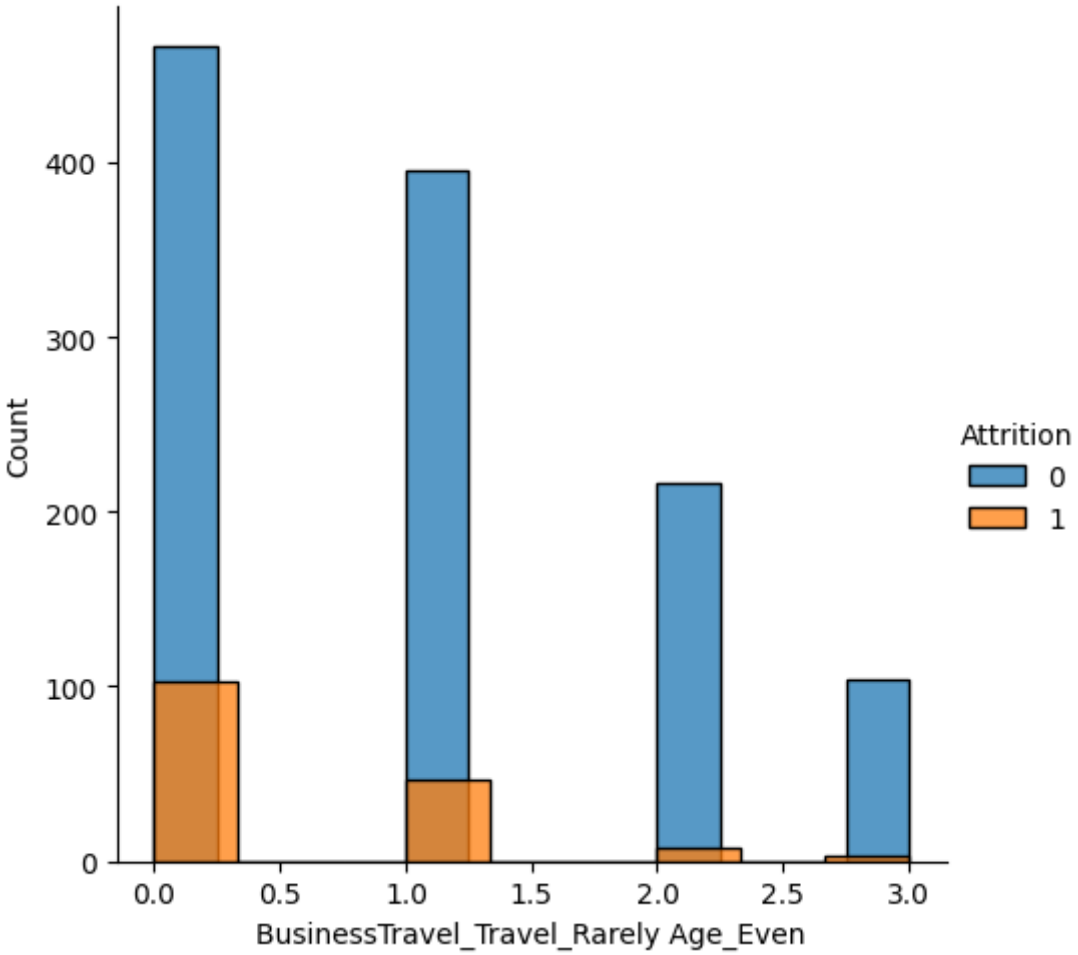

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
OverTime JobLevel_1
Attrition
0
1

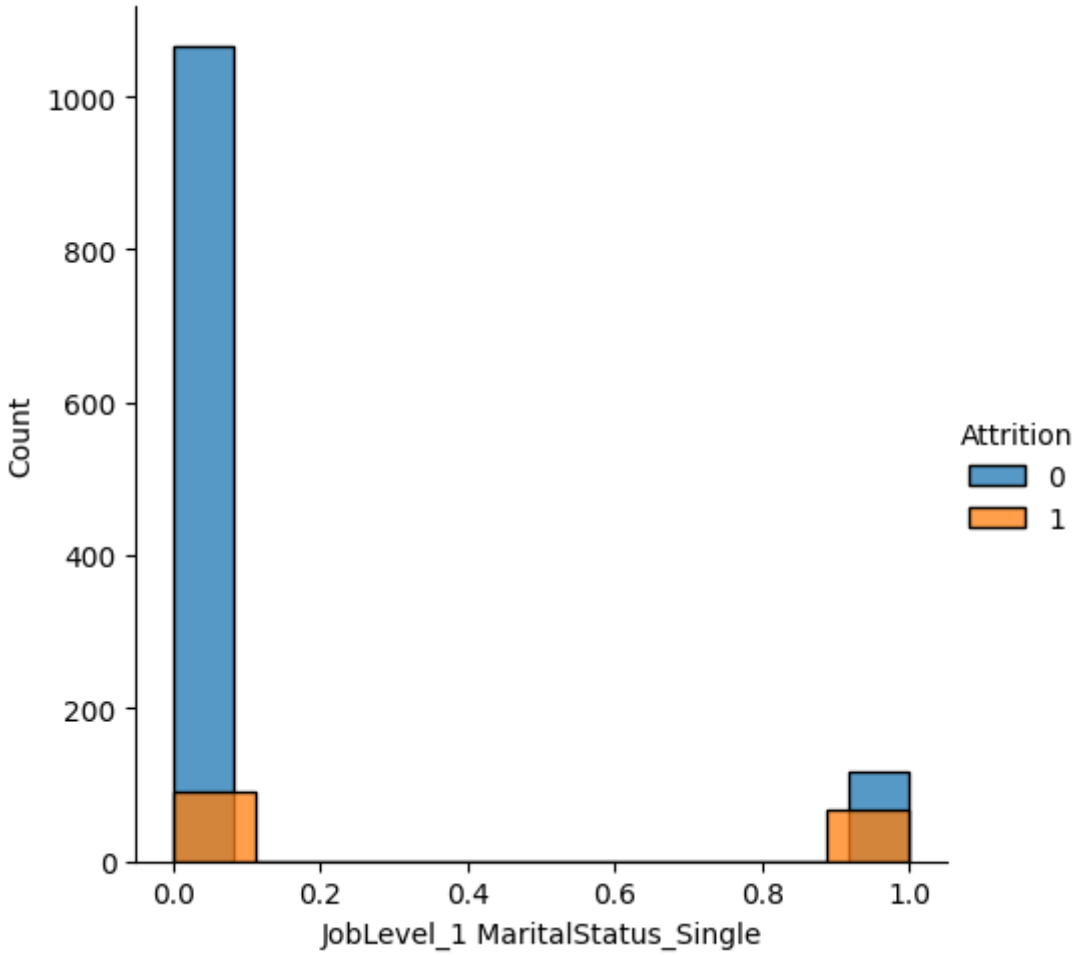

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Shift_0 EnvironmentSatisfaction_1
Attrition
0
1

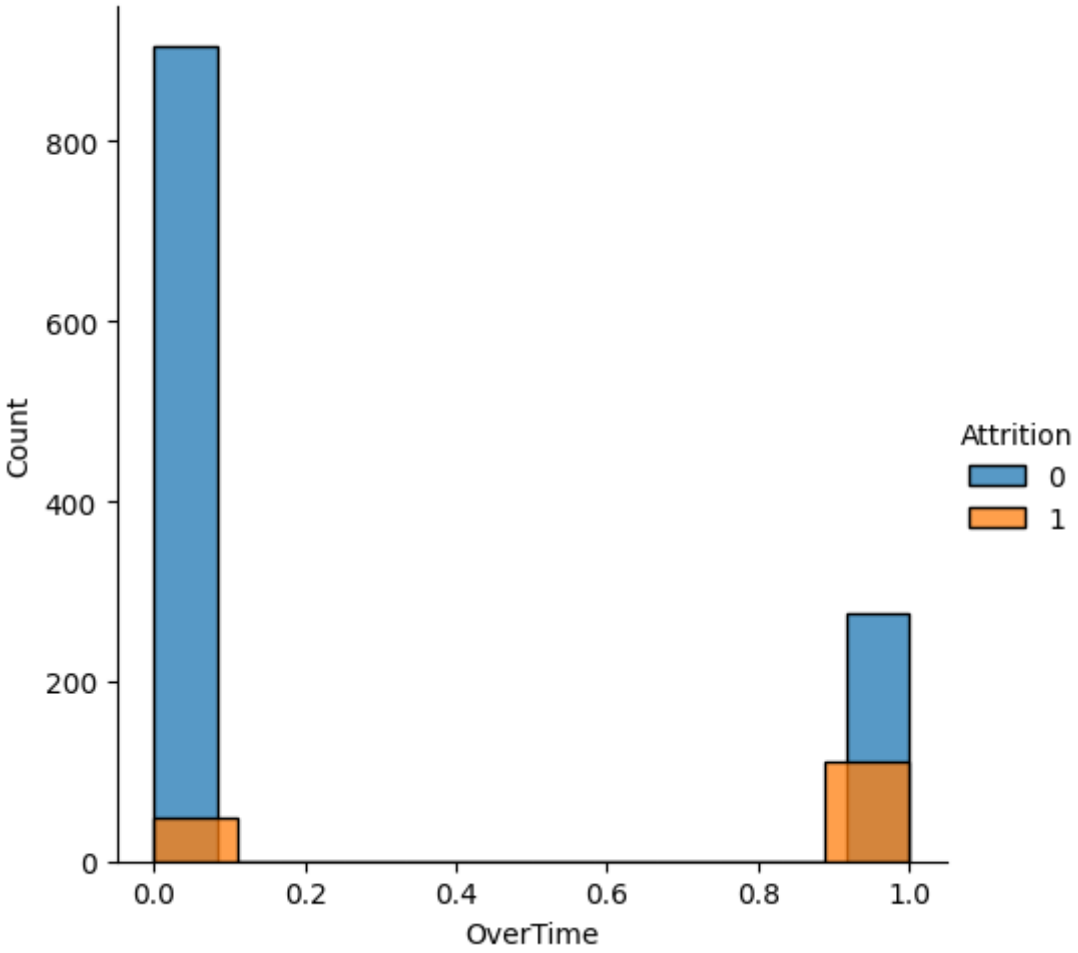

1000
800
600
400
200
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
DistanceFromHome_Even TrainingTimesLastYear_2
Attrition
0
1

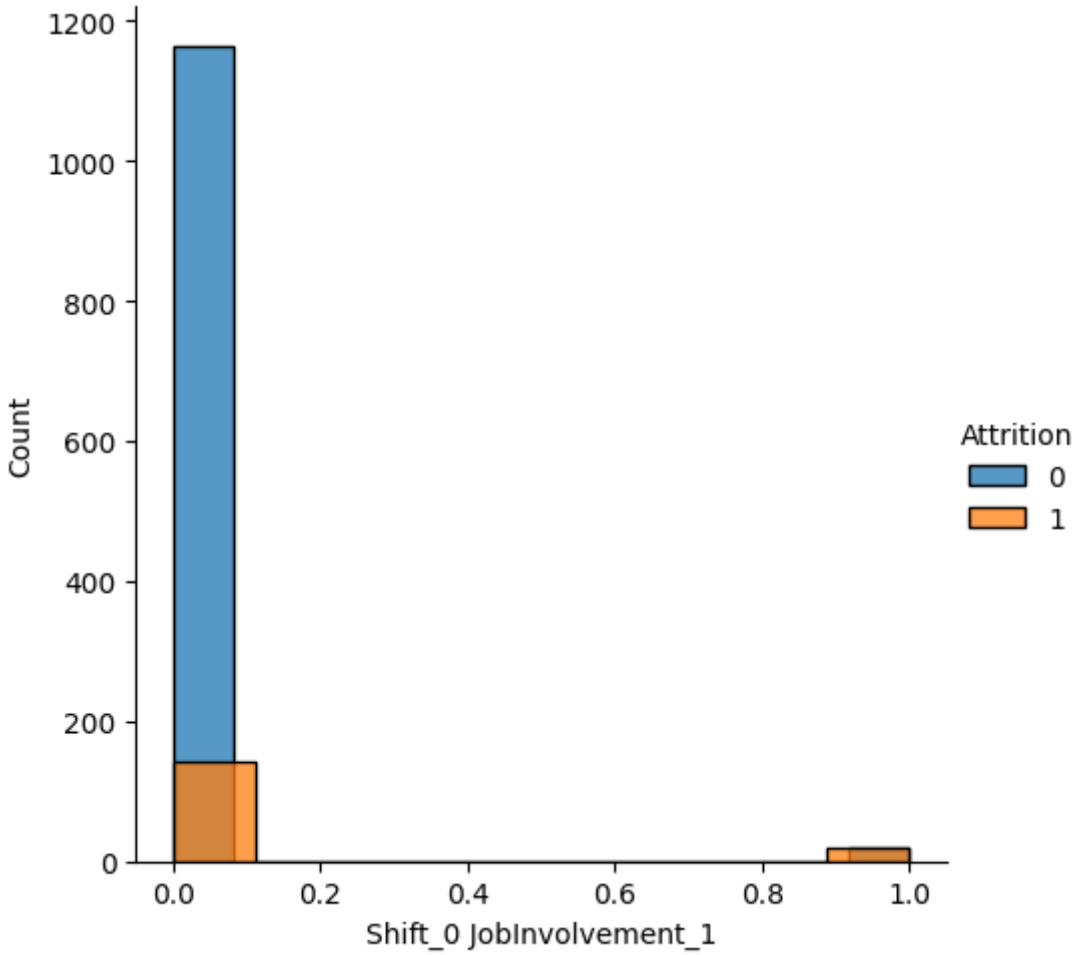

250
200
150
100
50
0
Count
2
4
6
8
10
12
14
16
JobInvolvement JobSatisfaction
Attrition
0
1

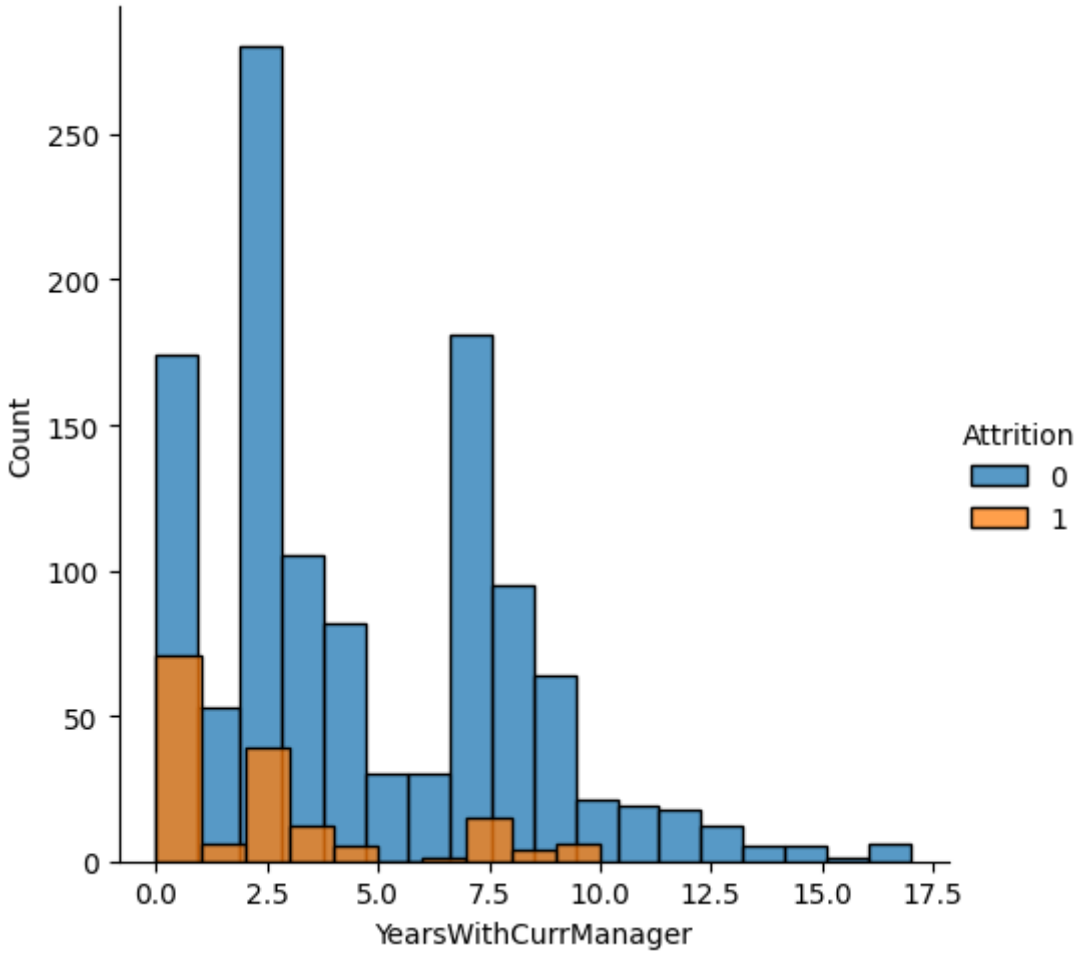

1000
800
600
400
200
0
Count
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
OverTime MaritalStatus
Attrition
0
1

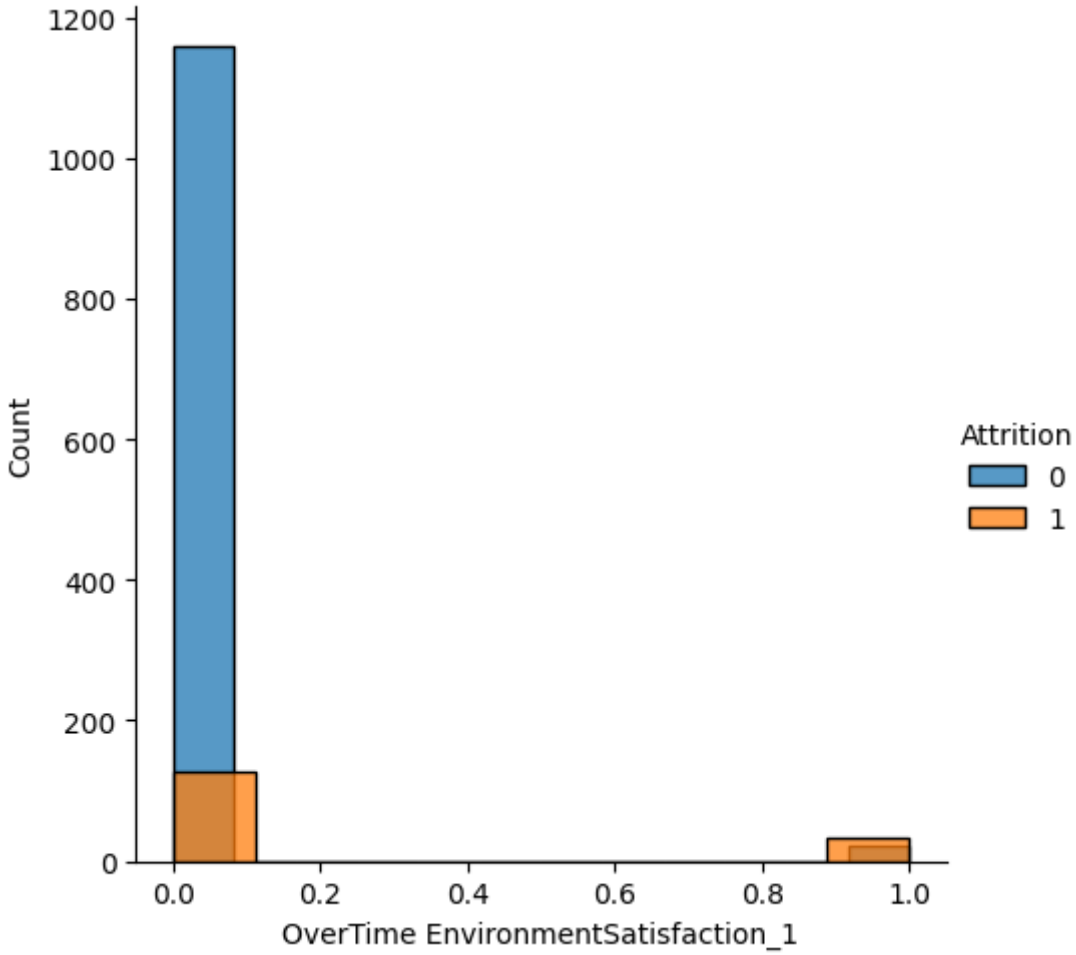

Count
400
300
200
100
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
BusinessTravel_Travel_Rarely Age_Even
Attrition
0
1

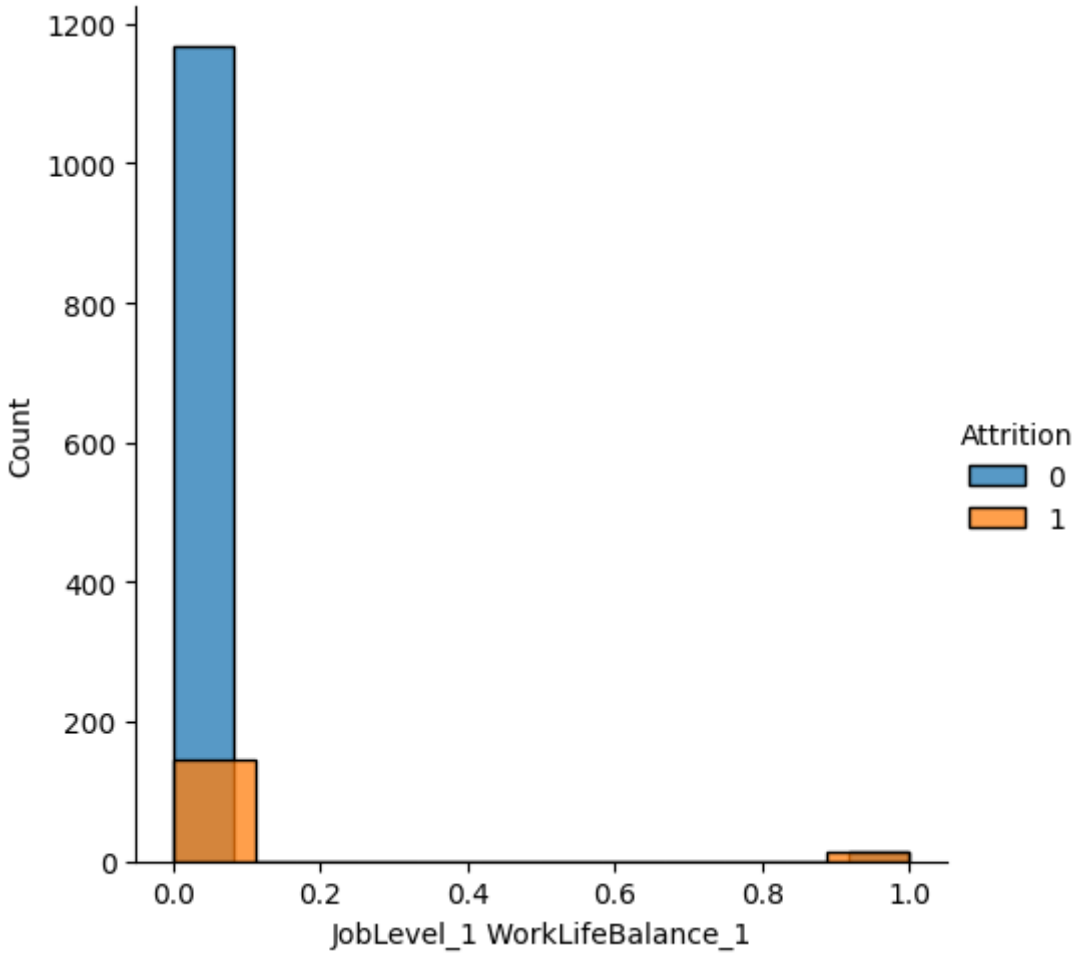

1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobLevel_1 MaritalStatus_Single
Attrition
0
1

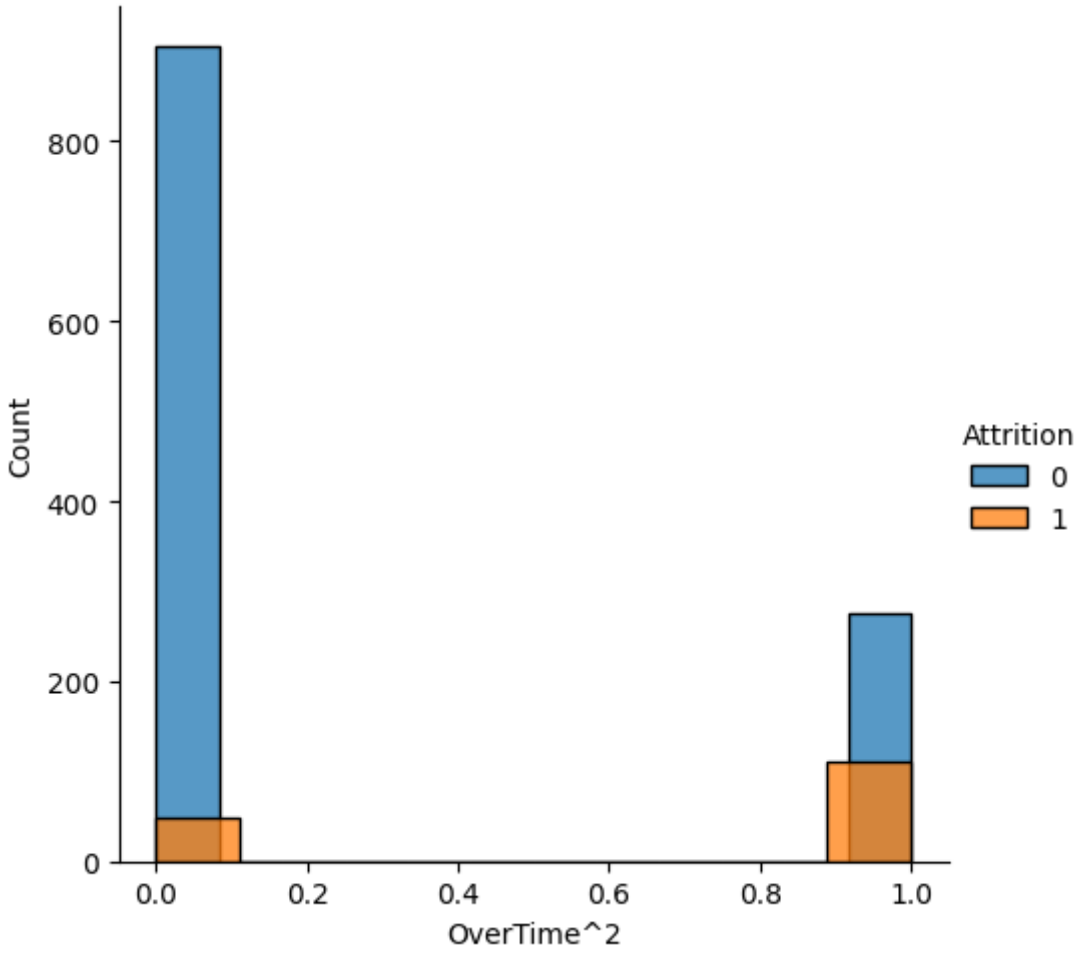

800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
OverTime
Attrition
0
1

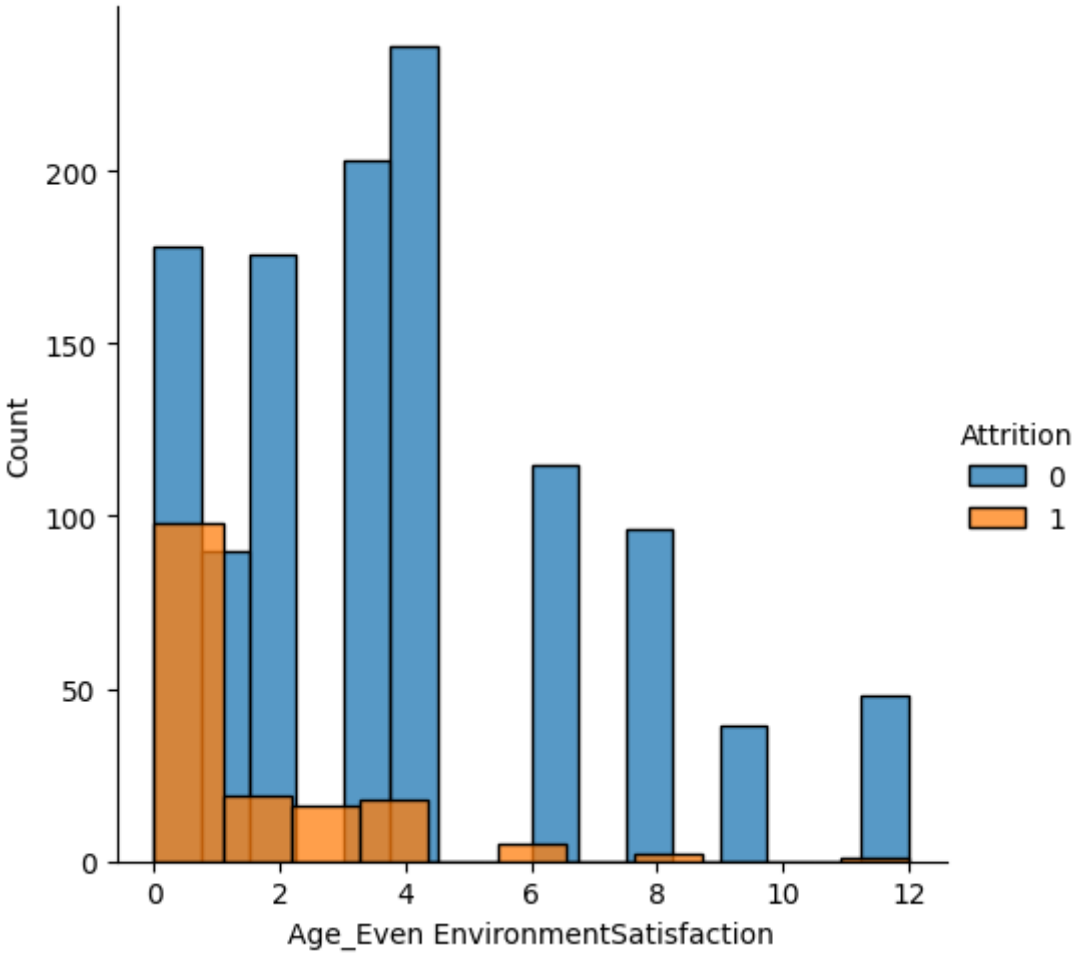

1200
1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Shift_0 JobInvolvement_1
Attrition
0
1

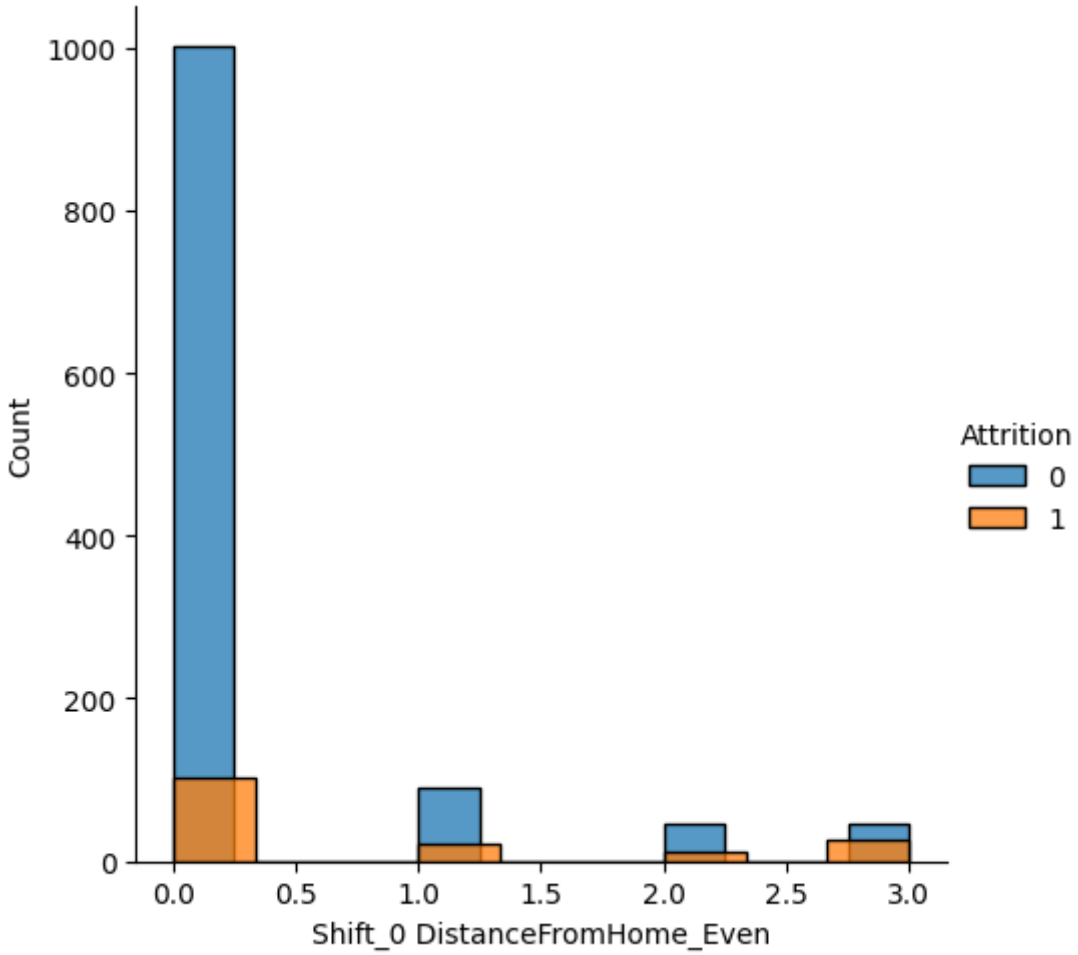

Count
250
200
150
100
50
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
YearsWithCurrManager
Attrition
0
1

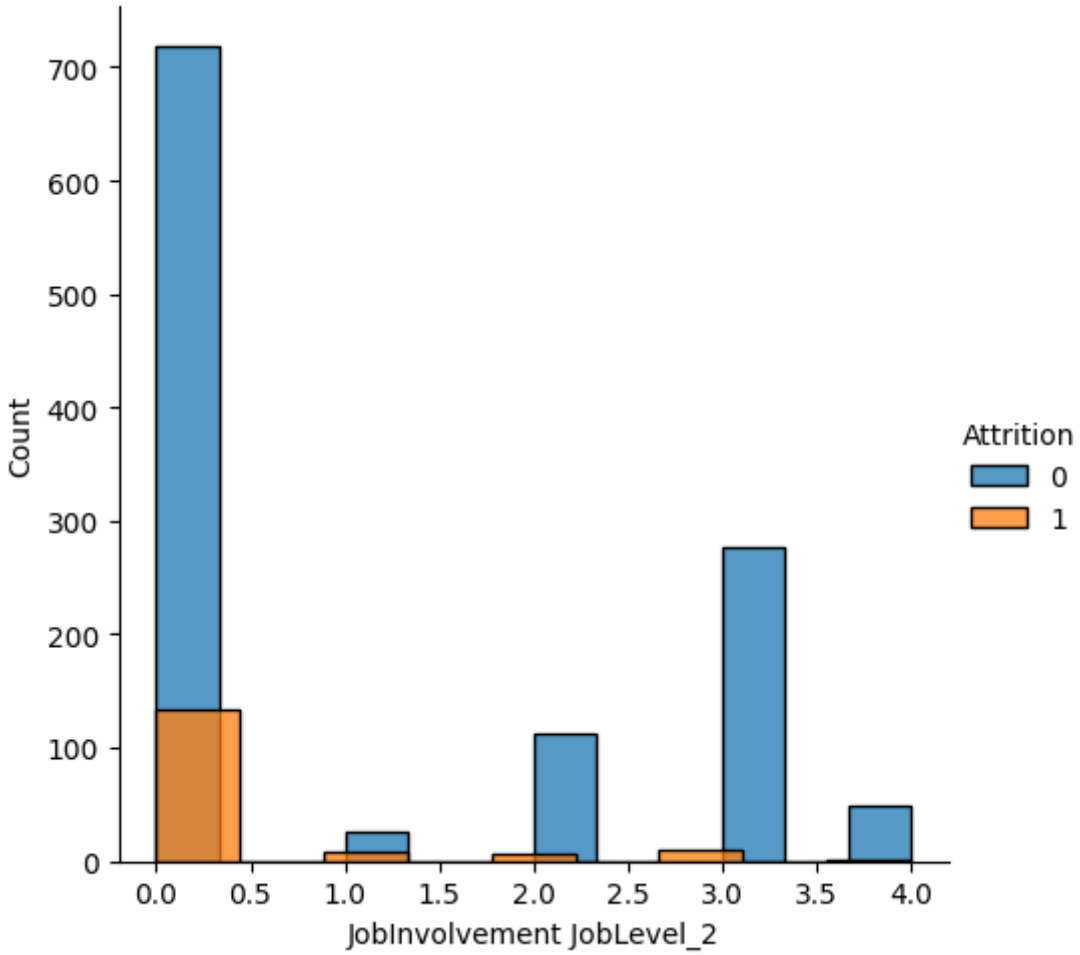

1200
1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
OverTime EnvironmentSatisfaction_1
Attrition
0
1

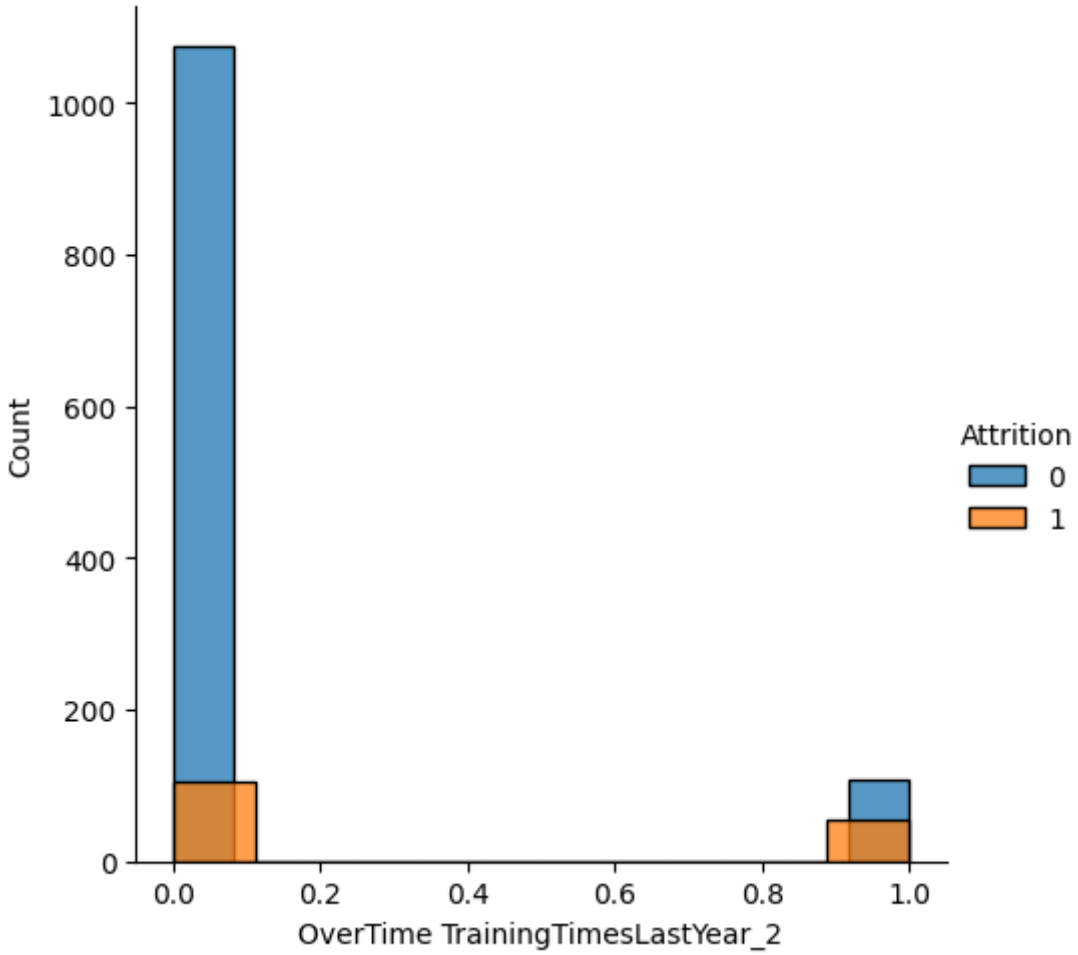

1200
1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
JobLevel_1 WorkLifeBalance_1
Attrition
0
1

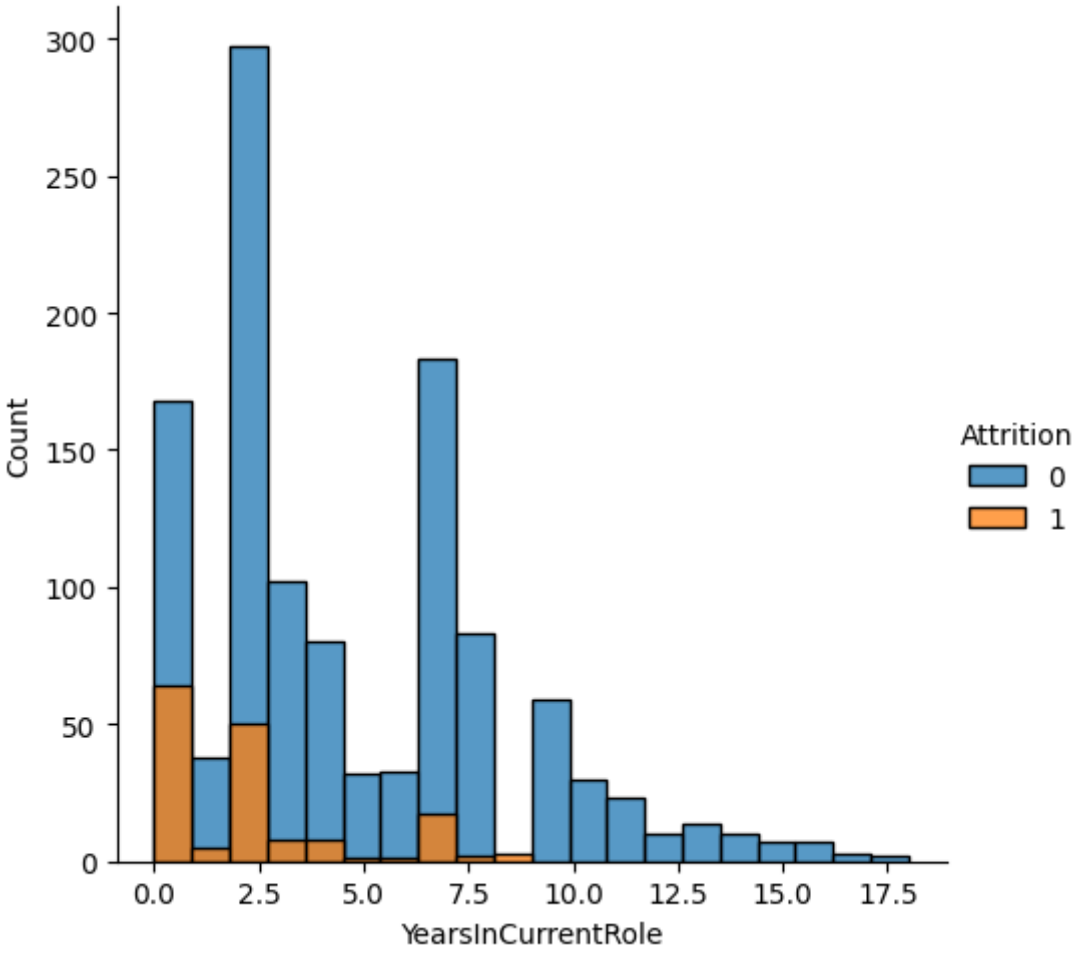

800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
OverTime^2
Attrition
0
1

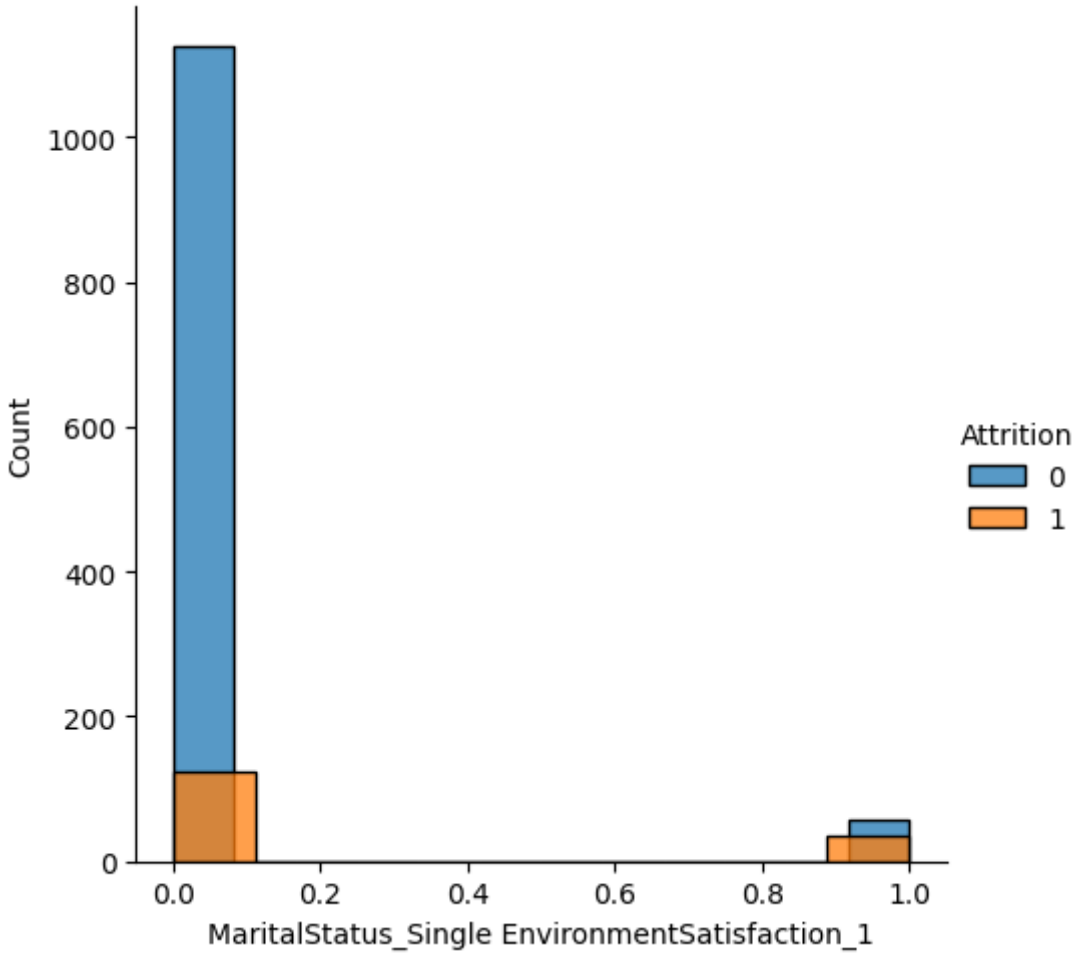

Count
200
150
100
50
0
0
2
4
6
8
10
12
Age_Even EnvironmentSatisfaction
Attrition
0
1

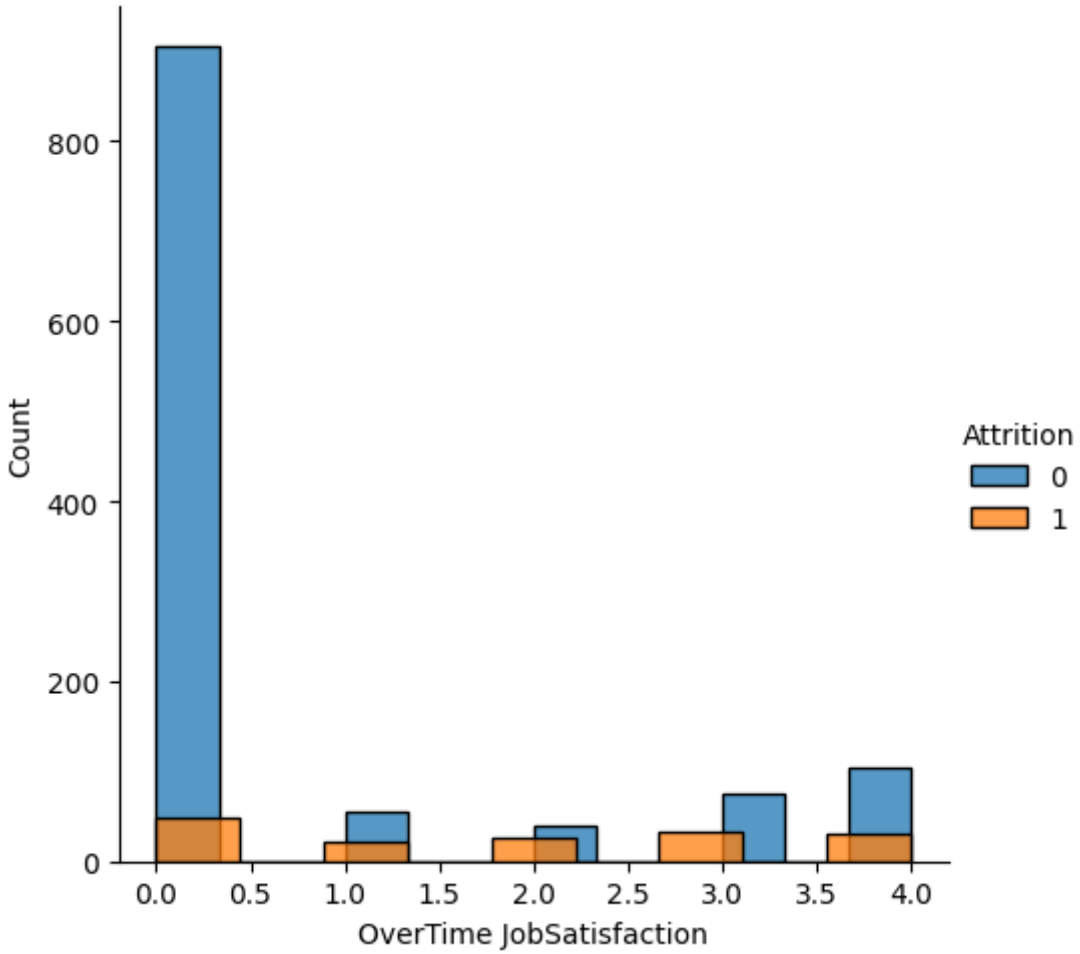

1000
800
600
400
200
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Shift_0 DistanceFromHome_Even
Attrition
0
1

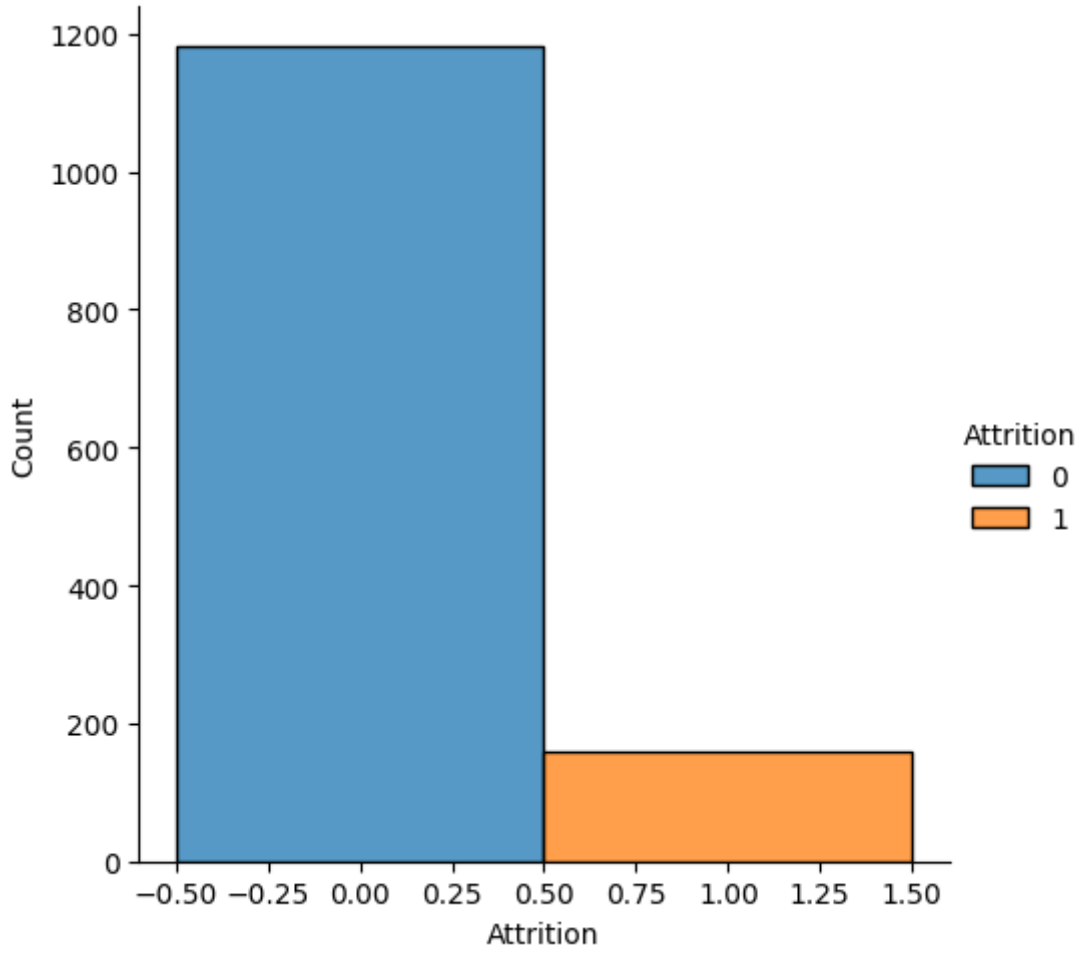

700
600
500
400
300
200
100
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
JobInvolvement JobLevel_2
Attrition
0
1


1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
OverTime TrainingTimesLastYear_2
Attrition
0
1


300
250
200
150
100
50
0
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
YearsInCurrentRole
Attrition
0
1


1000
800
600
400
200
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
MaritalStatus_Single EnvironmentSatisfaction_1
Attrition
0
1


800
600
400
200
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
OverTime JobSatisfaction
Attrition
0
1



In [ ]:
```
# save the final poly data
fin_poly_data = pd.concat([uncorr_poly_data, labels], axis=1)
fin_poly_data.to_csv("uncorr20_poly_data.csv")
uncorr_poly_sub_data.to_csv("uncorr20_poly_sub_data.csv")
```

In [ ]:
```
### Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
import seaborn as sns
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import f1_score, confusion_matrix
from sklearn.utils import resample
```

In [ ]:
```
### Import data
data = pd.read_csv("uncorr20_data.csv")
# data = pd.read_csv("uncorr20_poly_data.csv")

submission_data = pd.read_csv("uncorr20_sub_data.csv")
# submission_data = pd.read_csv("uncorr20_poly_sub_data.csv")

# separate into X and Y
y = data.pop("Attrition")

# store column names
columns = data.columns

# set aside test data
train_X, test_X, train_Y, test_Y = train_test_split(data, y,
                                                    stratify=y,
                                                    test_size=0.2,
                                                    random_state=0,
                                                    shuffle=True)

# turn into np array
train_X, train_Y = np.array(train_X), np.array(train_Y)
test_X, test_Y = np.array(test_X), np.array(test_Y)
submission_data = np.array(submission_data)
```

In [ ]:
```
print(submission_data.shape)
print(data.shape)
```

```
(336, 21)
(1340, 21)
```

In [ ]:
```
def get_f1(model, X, y):
    preds = model.predict(X)
    f1 = f1_score(y, preds)
    return f1
```

In [ ]:
```
# Baseline
adaboost = AdaBoostClassifier()

start = time.time()
adaboost.fit(train_X, train_Y)
stop = time.time()
```

```
print("Time to train: ", str(stop-start))

print(adaboost.get_params())

train_f1 = get_f1(adaboost, train_X, train_Y)
print("Train f1: ", train_f1)

test_f1 = get_f1(adaboost, test_X, test_Y)
print("Test f1: ", test_f1)
```

```
Time to train:  0.04631495475769043
{'algorithm': 'SAMME.R', 'base_estimator': None, 'learning_rate': 1.0, 'n_e
stimators': 50, 'random_state': None}
Train f1:  0.7136563876651982
Test f1:  0.6545454545454547
```

In [ ]:
```
# Upsample
X = pd.concat([pd.DataFrame(train_X), pd.DataFrame(train_Y)], axis=1)
new_cols = np.append(np.array(columns), ["Attrition"])
X.columns = new_cols

not_attr = X[X.Attrition==0]
attr = X[X.Attrition==1]

attr_upsampled = resample(attr,
                          replace=True, # sample with replacement
                          n_samples=int(np.round(len(not_attr)/1.75)), # num
                          random_state=0)

upsampled = pd.concat([not_attr, attr_upsampled])

train_Y_up = np.array(upsampled.pop("Attrition"))
train_X_up = np.array(upsampled)
```

In [ ]:
```
# Up Baseline
adaboost = AdaBoostClassifier(n_estimators=30,
                              learning_rate=1)
adaboost.fit(train_X_up, train_Y_up)

train_f1 = get_f1(adaboost, train_X_up, train_Y_up)
print("Train f1: ", train_f1)

test_f1 = get_f1(adaboost, test_X, test_Y)
print("Test f1: ", test_f1)
```

```
Train f1:  0.865616311399444
Test f1:  0.5866666666666667
```

In [ ]:
```
# Use upsampled data
# train_X, train_Y = train_X_up, train_Y_up
```

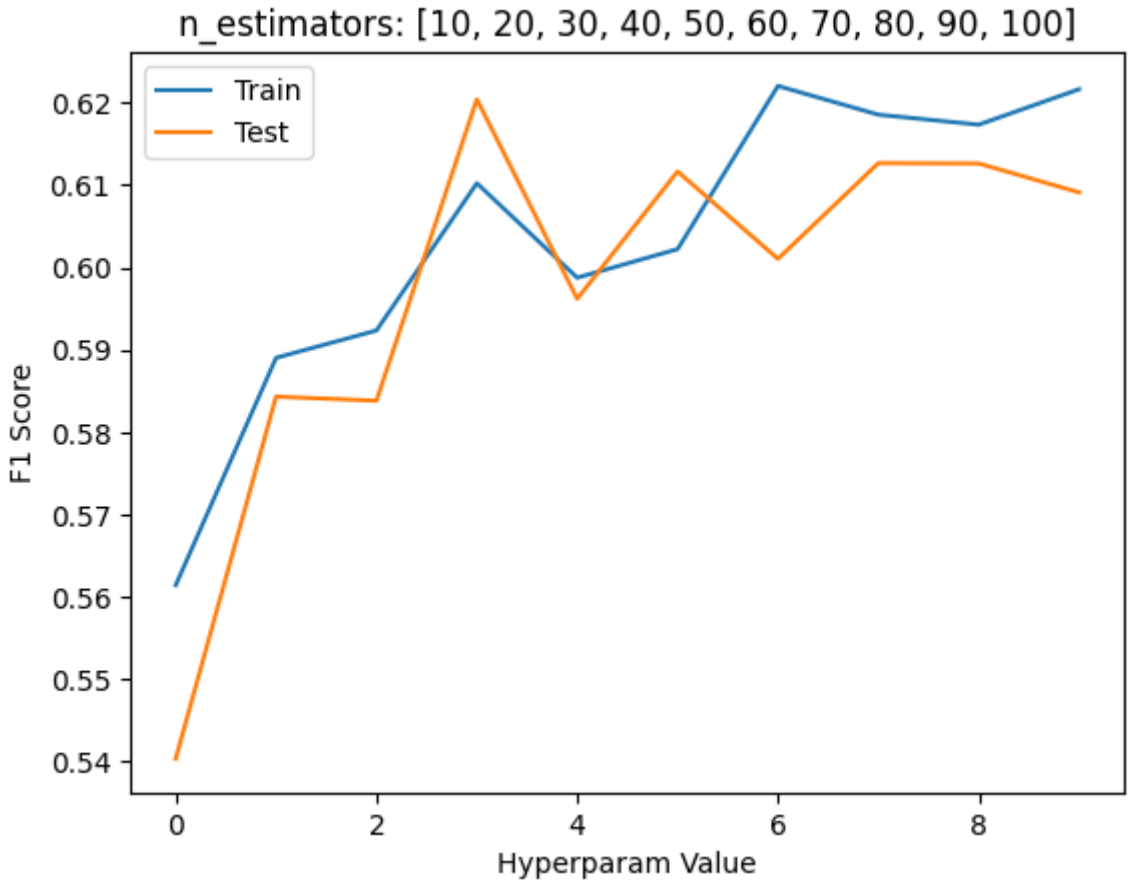
In [ ]:
```
# Hyperparameter Tuning

# Define the parameter grid
param_grid = {
    "n_estimators": [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
```

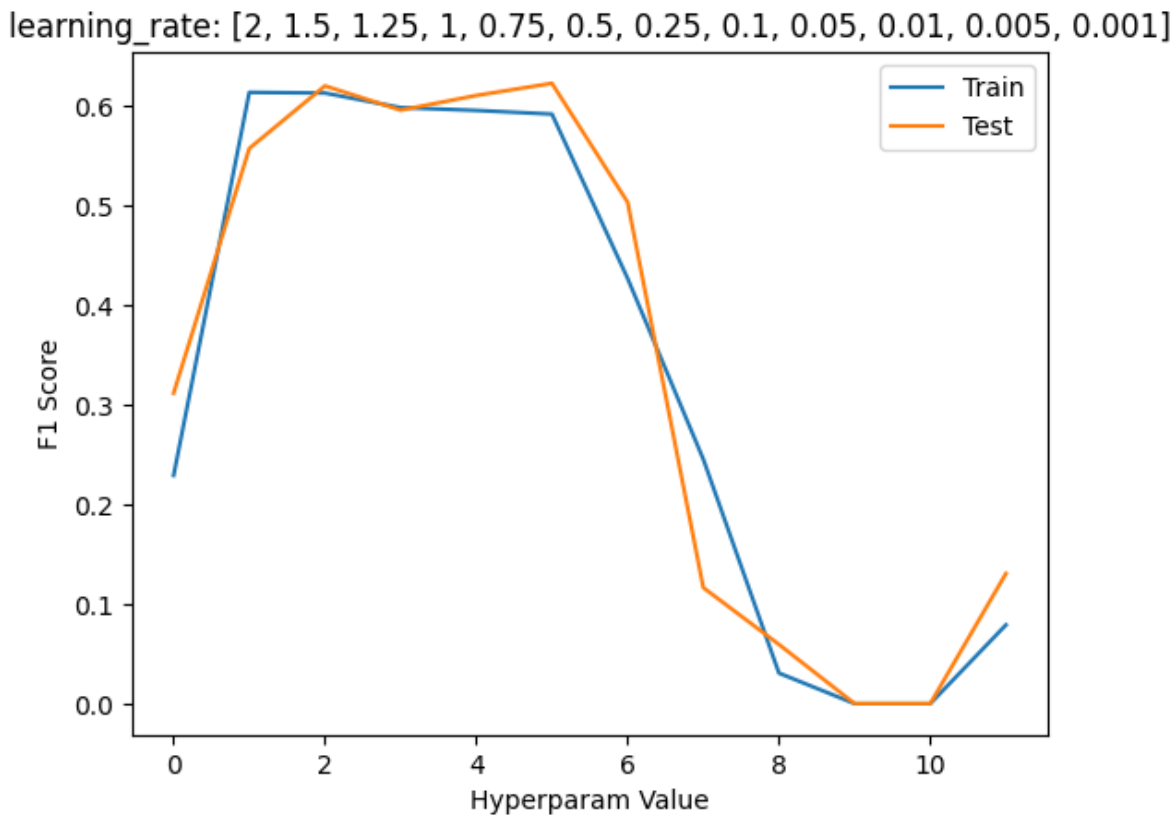
```
    "learning_rate": [2, 1.5, 1.25, 1, 0.75, 0.5, 0.25, 0.1,
                      0.05, 0.01, 0.005, 0.001],
    "base_estimator": [DecisionTreeClassifier(max_depth=1),
                       DecisionTreeClassifier(max_depth=2),
                       DecisionTreeClassifier(max_depth=3),
                       DecisionTreeClassifier(max_depth=4)],
}

train_scores, test_scores = {}, {}      # k: paramter being tuned; v: scores

for k, v in param_grid.items():
    print(k)

    train, test = [], []
    for v_i in v:
        NUM_SPLITS = 3
        cv_train = np.empty(NUM_SPLITS)
        cv_test = np.empty(NUM_SPLITS)
        cv = StratifiedKFold(n_splits=NUM_SPLITS)

        for idx, (train_idx, test_idx) in enumerate(cv.split(train_X, train_
            X_train, X_test = train_X[train_idx], train_X[test_idx]
            y_train, y_test = train_Y[train_idx], train_Y[test_idx]

            adaboost = AdaBoostClassifier(**{k:v_i})
            adaboost.fit(X_train, y_train)

            train_f1 = get_f1(adaboost, X_test, y_test)
            test_f1 = get_f1(adaboost, test_X, test_Y)

            cv_train[idx] = train_f1
            cv_test[idx] = test_f1

        train.append(np.mean(cv_train))
        test.append(np.mean(cv_test))

    train_scores[k] = train
    test_scores[k] = test
```

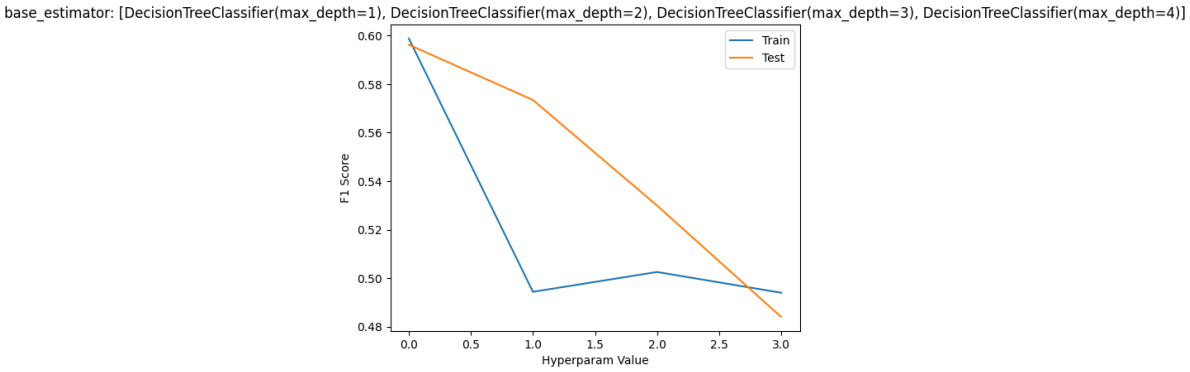
```
n_estimators
learning_rate
base_estimator
```

In [ ]:

```
for k in train_scores.keys():
    plt.figure()
    plt.plot(list(range(len(train_scores[k]))), train_scores[k])
    plt.plot(list(range(len(train_scores[k]))), test_scores[k])
    plt.title(k + ": " + str(param_grid[k]))
    plt.xlabel("Hyperparam Value")
    plt.ylabel("F1 Score")
    plt.legend(["Train", "Test"])
    plt.show()
```


n_estimators: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Train
Test
F1 Score
0.62
0.61
0.60
0.59
0.58
0.57
0.56
0.55
0.54
0
2
4
6
8
Hyperparam Value


learning_rate: [2, 1.5, 1.25, 1, 0.75, 0.5, 0.25, 0.1, 0.05, 0.01, 0.005, 0.001]
Train
Test
F1 Score
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8
10
Hyperparam Value



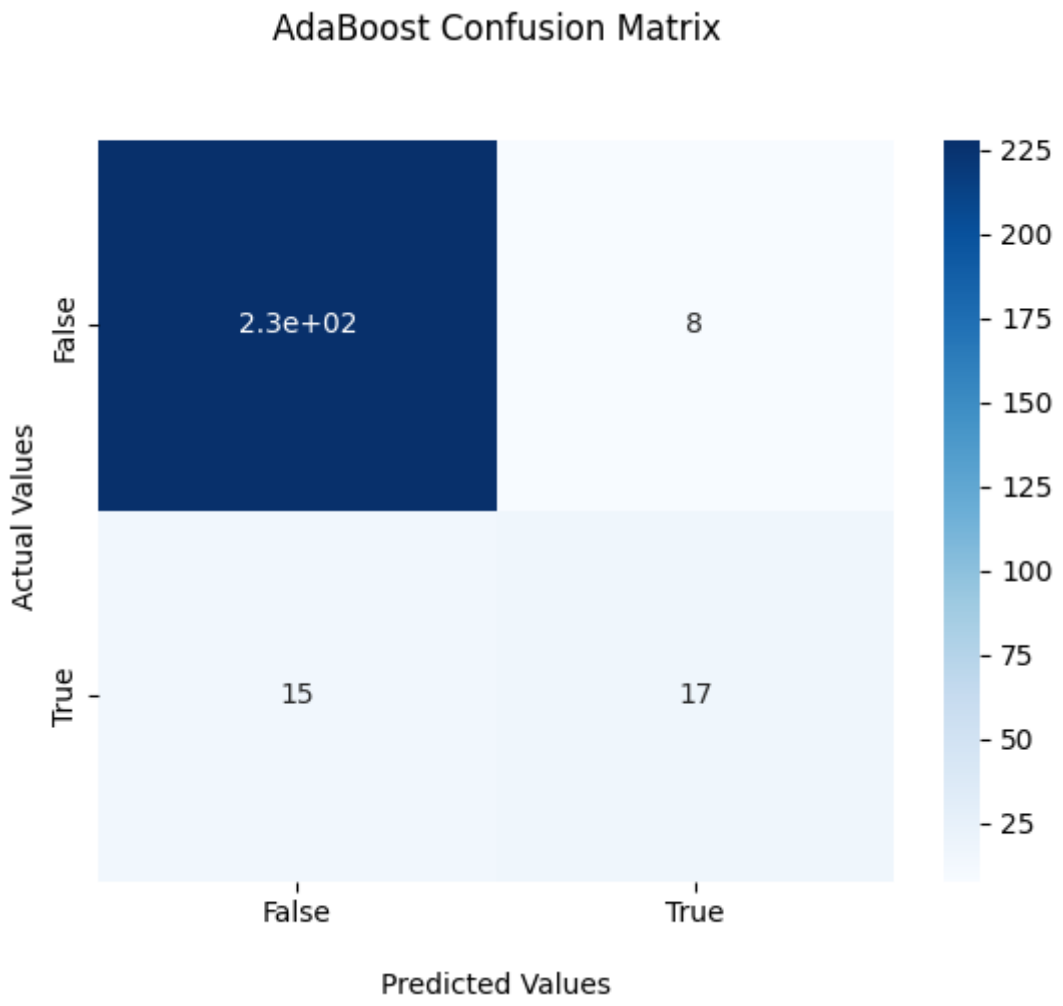
In [ ]:
```
# Get best hyperparameters
best_params = {}
for k,v in train_scores.items():
    best_params[k] = param_grid[k][v.index(max(v))]
print(best_params)
```

```
{'n_estimators': 70, 'learning_rate': 1.5, 'base_estimator': DecisionTreeCl
assifier(max_depth=1)}
```

In [ ]:
```
# Train model with best hyperparameters
ada_tuned = AdaBoostClassifier(**best_params)
ada_tuned.fit(train_X, train_Y)

train_f1 = get_f1(ada_tuned, train_X, train_Y)
print("Train f1: ", train_f1)

test_f1 = get_f1(ada_tuned, test_X, test_Y)
print("Test f1: ", test_f1)
```

```
Train f1:  0.7672413793103449
Test f1:  0.5964912280701754
```

In [ ]:
```
# Make a confusion matrix
c_matrix = confusion_matrix(test_Y, ada_tuned.predict(test_X))
ax = sns.heatmap(c_matrix, annot=True, cmap='Blues')
ax.set_title('AdaBoost Confusion Matrix\n\n')
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ')
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



In [ ]:
```
sub_preds = ada_tuned.predict(submission_data)

print( sum(sub_preds) / len(sub_preds))
print( sum(train_Y) / len(train_Y))
print( sum(test_Y) / len(test_Y))
```

```
0.08630952380952381
0.11847014925373134
0.11940298507462686
```

In [ ]:
```
# get predictions for submission
sub_preds = ada_tuned.predict(submission_data)
ids = list(range(0, len(sub_preds)))

output_data = pd.DataFrame({"Id": ids, "Predicted": sub_preds})
output_data = output_data.set_index("Id")

output_data.to_csv("ada_submission.csv")
```

In [ ]:
```
### Imports
import pandas as pd
import numpy as np
import lightgbm as lgb
import time
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import f1_score, confusion_matrix
from sklearn.utils import resample
```

In [ ]:
```
### Import data
data = pd.read_csv("uncorr20_data.csv")
# data = pd.read_csv("uncorr20_poly_data.csv")

submission_data = pd.read_csv("uncorr20_sub_data.csv")
# submission_data = pd.read_csv("uncorr20_poly_sub_data.csv")

# separate into X and Y
y = data.pop("Attrition")

# store column names
columns = data.columns

# set aside test data
train_X, test_X, train_Y, test_Y = train_test_split(data, y,
                                                    stratify=y,
                                                    test_size=0.2,
                                                    random_state=0,
                                                    shuffle=True)

# turn into np array
train_X, train_Y = np.array(train_X), np.array(train_Y)
test_X, test_Y = np.array(test_X), np.array(test_Y)
submission_data = np.array(submission_data)
```

Establish a baseline by training a LGBM classifier with no changes

In [ ]:
```
# General f1 score function
def get_f1(model, X, y):
    preds = model.predict(X)
    f1 = f1_score(y, preds)
    return f1
```

In [ ]:
```
# function for lgb to evaluate by f1
def lgb_f1_score(y_hat, data):
    y_true = data #.get_label()
    y_hat = np.round(y_hat)
    return 'f1', f1_score(y_true, y_hat), True
```

In [ ]:
```
# Baseline
# train model
lgb_model = lgb.LGBMClassifier()
```

```
start = time.time()
lgb_model.fit(train_X, train_Y, eval_metric=lgb_f1_score)
stop = time.time()
print("Time to train: ", str(stop-start))

# get test f1
model_f1_score = get_f1(lgb_model, train_X, train_Y)
print("Train f1 score: ", model_f1_score)

model_f1_score = get_f1(lgb_model, test_X, test_Y)
print("Test f1 score: ", model_f1_score)
```

```
Time to train:  0.5555188655853271
Train f1 score:  1.0
Test f1 score:  0.6206896551724138
```

In [ ]:
```
# Upsample
X = pd.concat([pd.DataFrame(train_X), pd.DataFrame(train_Y)], axis=1)
new_cols = np.append(np.array(columns), ["Attrition"])
X.columns = new_cols

not_attr = X[X.Attrition==0]
attr = X[X.Attrition==1]

attr_upsampled = resample(attr,
                          replace=True, # sample with replacement
                          n_samples=int(np.round(len(not_attr)/1.75)), # nub
                          random_state=0)

upsampled = pd.concat([not_attr, attr_upsampled])

train_Y_up = np.array(upsampled.pop("Attrition"))
train_X_up = np.array(upsampled)
```

In [ ]:
```
# Upsample Baseline

# train model
lgb_model = lgb.LGBMClassifier()
lgb_model.fit(train_X_up, train_Y_up, eval_metric=lgb_f1_score)

# get test f1
model_f1_score = get_f1(lgb_model, train_X_up, train_Y_up)
print("Train f1 score: ", model_f1_score)

model_f1_score = get_f1(lgb_model, test_X, test_Y)
print("Test f1 score: ", model_f1_score)
```

```
Train f1 score:  1.0
Test f1 score:  0.6451612903225806
```

In [ ]:
```
# Use upsampled data
train_X, train_Y = train_X_up, train_Y_up
```

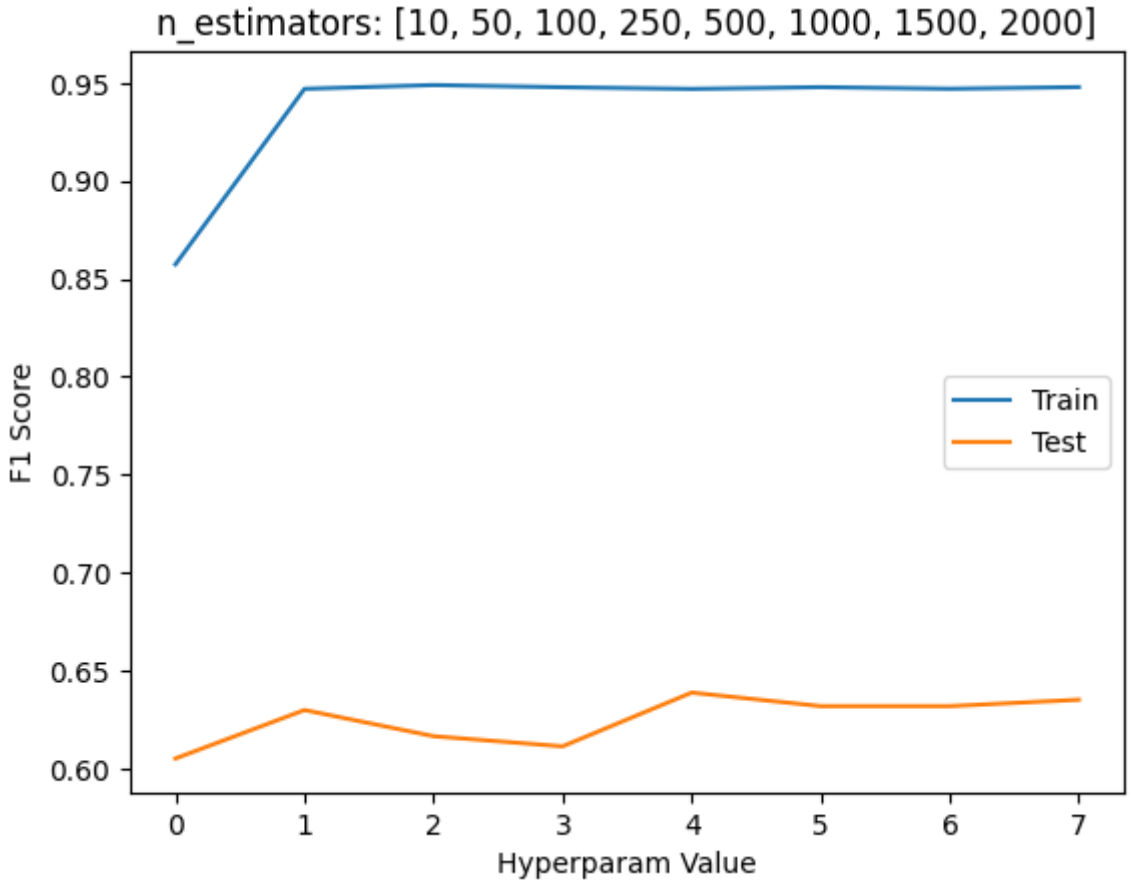
In [ ]:
```
# Hyperparameter Tuning
```

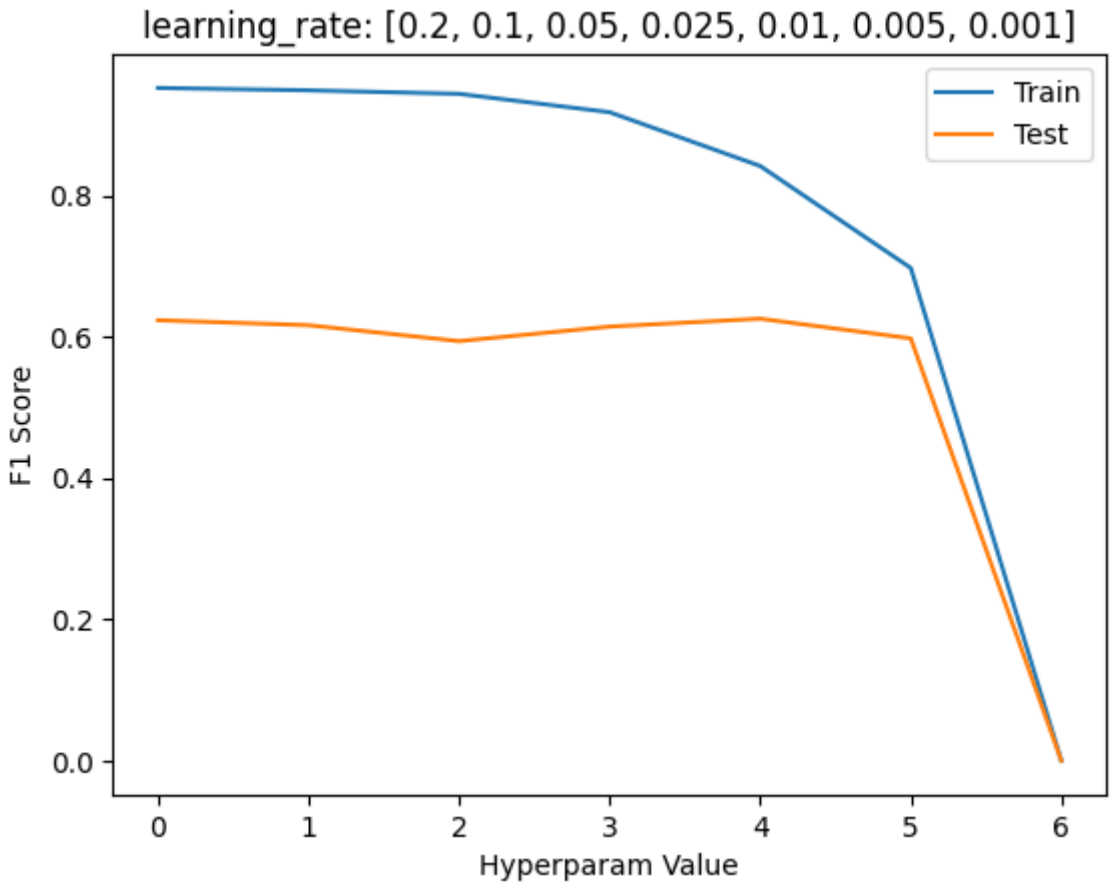
```
# Define the parameter grid
param_grid = {
    "n_estimators": [10, 50, 100, 250,
                     500, 1000, 1500, 2000],
    "learning_rate": [0.2, 0.1, 0.05, 0.025,
                      0.01, 0.005, 0.001],
    "num_leaves": [2, 5, 10, 25, 50,
                   100, 250, 500, 1000],
    "max_depth": [1, 2, 3, 4, 5, 6, 7, 8, None],
    "scale_pos_weight": [0.1, 0.5, 1, 2, 3, 4, 5]
}

train_scores, test_scores = {}, {}      # k: paramter being tuned; v: scores

for k, v in param_grid.items():
    print(k)

    train, test = [], []
    for v_i in v:
        NUM_SPLITS = 3
        cv_train = np.empty(NUM_SPLITS)
        cv_test = np.empty(NUM_SPLITS)
        cv = StratifiedKFold(n_splits=NUM_SPLITS)

        for idx, (train_idx, test_idx) in enumerate(cv.split(train_X, train_
            X_train, X_test = train_X[train_idx], train_X[test_idx]
            y_train, y_test = train_Y[train_idx], train_Y[test_idx]


            lgb_model = lgb.LGBMClassifier(**{k:v_i})
            lgb_model.fit(X_train, y_train,
                          eval_metric=lgb_f1_score)

            train_f1 = get_f1(lgb_model, X_test, y_test)
            test_f1 = get_f1(lgb_model, test_X, test_Y)

            cv_train[idx] = train_f1
            cv_test[idx] = test_f1

        train.append(np.mean(cv_train))
        test.append(np.mean(cv_test))

    train_scores[k] = train
    test_scores[k] = test
```

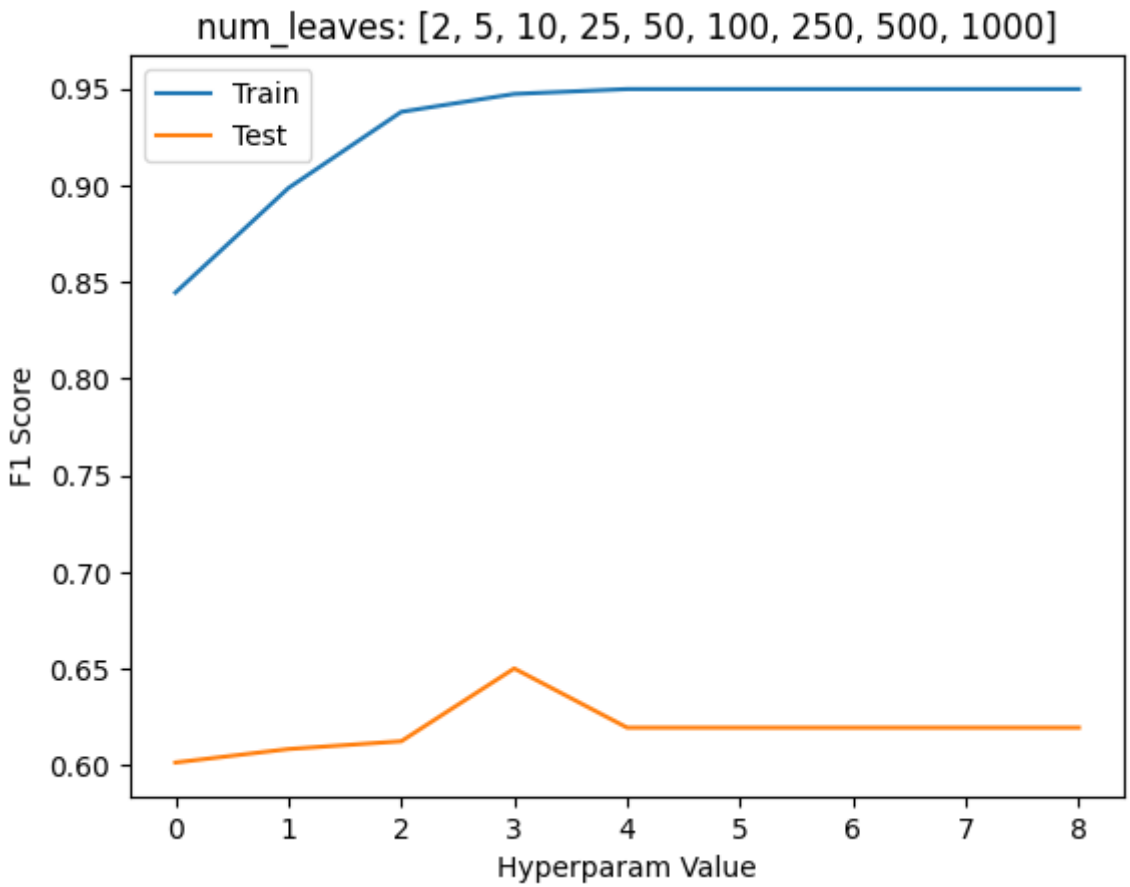
```
n_estimators
learning_rate
num_leaves
max_depth
scale_pos_weight
```

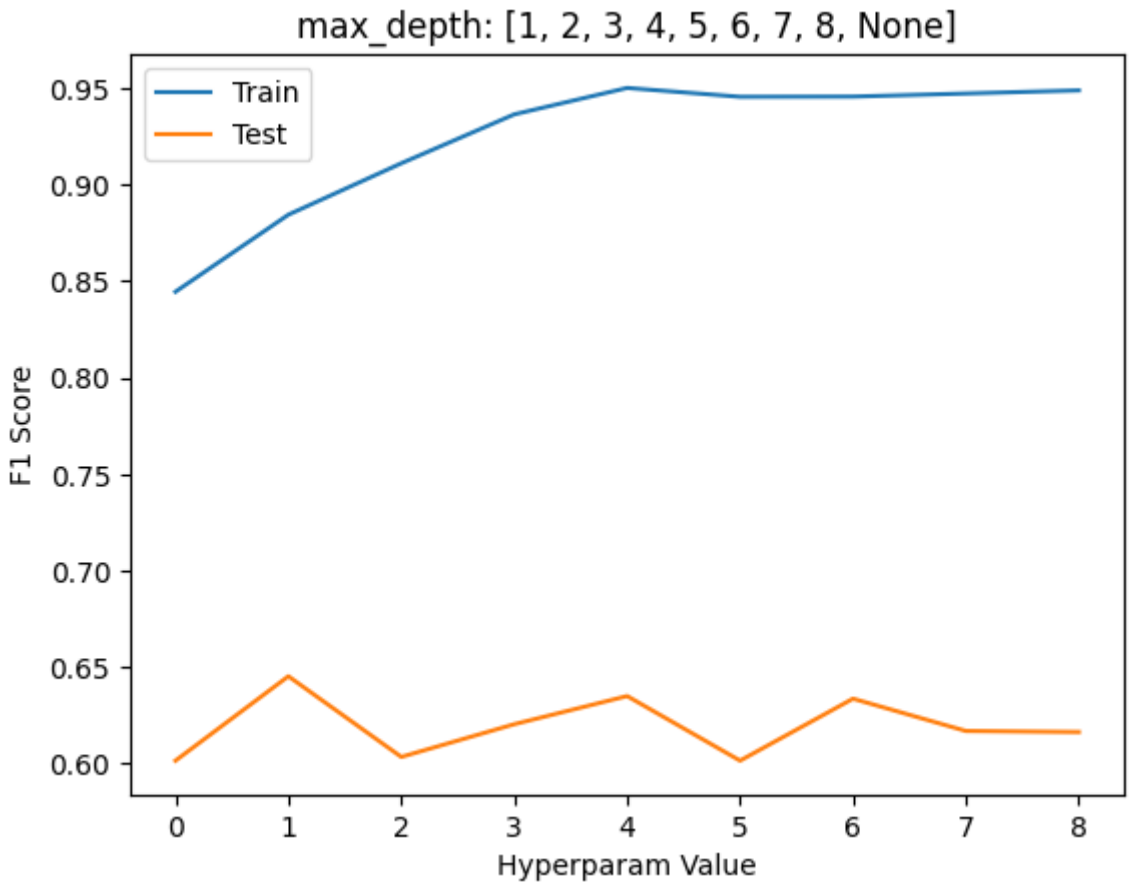
In [ ]:
```
for k in train_scores.keys():
    plt.figure()
    plt.plot(list(range(len(train_scores[k]))), train_scores[k])
    plt.plot(list(range(len(train_scores[k]))), test_scores[k])
    plt.title(k + ": " + str(param_grid[k]))
```

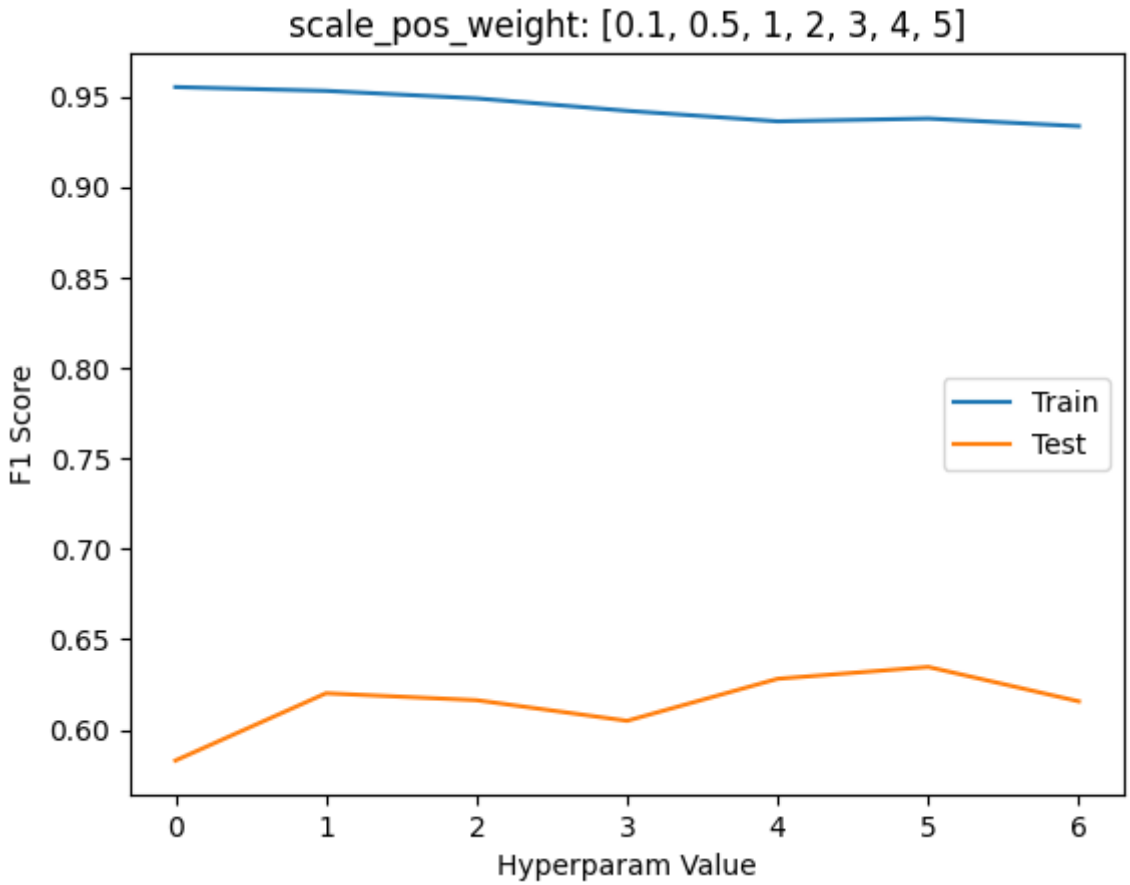
```
    plt.xlabel("Hyperparam Value")
    plt.ylabel("F1 Score")
    plt.legend(["Train", "Test"])
    plt.show()
```




learning_rate: [0.2, 0.1, 0.05, 0.025, 0.01, 0.005, 0.001]
Train
Test
F1 Score
0.0
0.2
0.4
0.6
0.8
0
1
2
3
4
5
6
Hyperparam Value


num_leaves: [2, 5, 10, 25, 50, 100, 250, 500, 1000]
Train
Test
F1 Score
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
0
1
2
3
4
5
6
7
8
Hyperparam Value





In [ ]:
```
# Get best hyperparameters
```

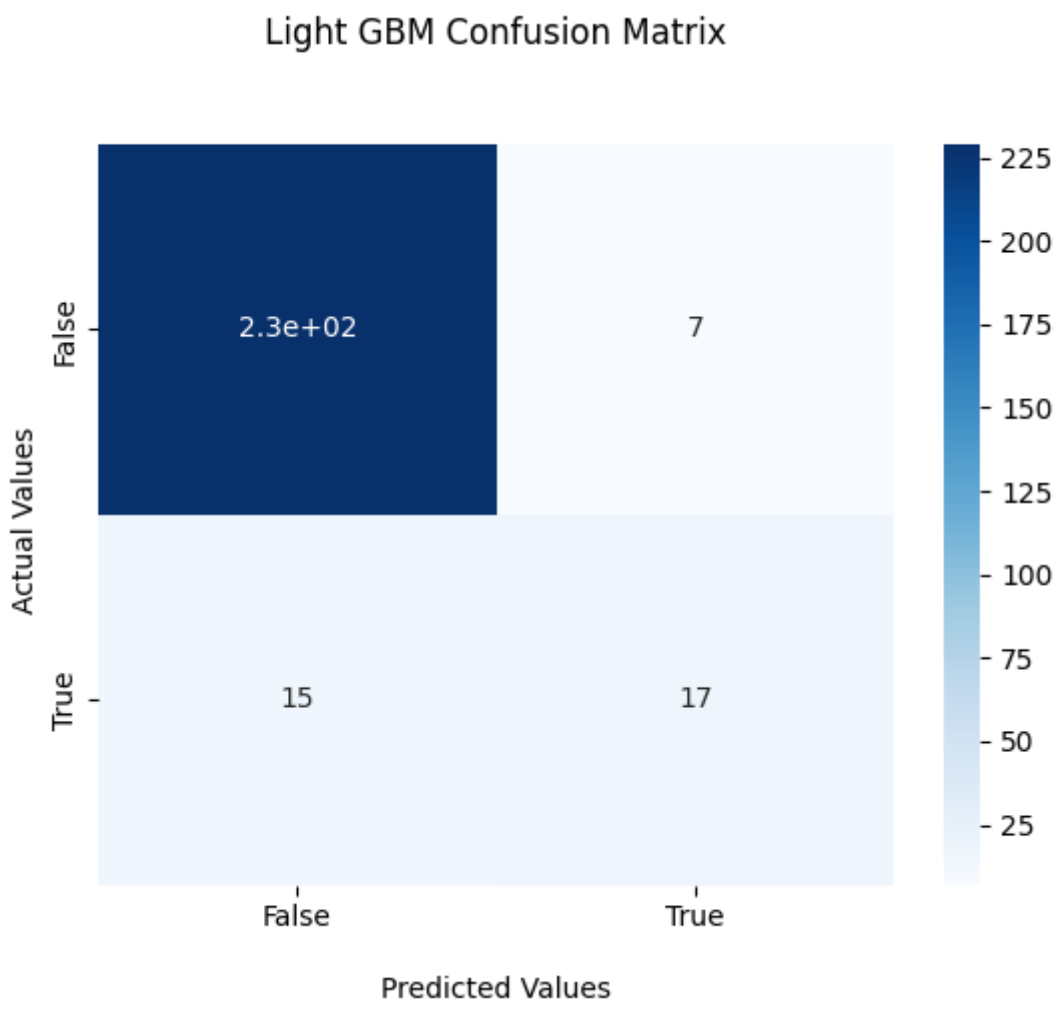
```
best_params = {}
for k,v in train_scores.items():
    best_params[k] = param_grid[k][v.index(max(v))]
print(best_params)
```

```
{'n_estimators': 100, 'learning_rate': 0.2, 'num_leaves': 50, 'max_depth':
5, 'scale_pos_weight': 0.1}
```

In [ ]:
```
# Train model with best hyperparameters
gbm_tuned = lgb.LGBMClassifier(**best_params)
gbm_tuned.fit(train_X, train_Y,
              eval_metric=lgb_f1_score)

train_f1 = get_f1(gbm_tuned, train_X, train_Y)
print("Train f1: ", train_f1)

test_f1 = get_f1(gbm_tuned, test_X, test_Y)
print("Test f1: ", test_f1)
```

```
Train f1:  1.0
Test f1:  0.6071428571428571
```

In [ ]:
```
# Make a confusion matrix
c_matrix = confusion_matrix(test_Y, gbm_tuned.predict(test_X))
ax = sns.heatmap(c_matrix, annot=True, cmap='Blues')
ax.set_title('Light GBM Confusion Matrix\n\n')
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ')
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])
plt.show()
```



In [ ]:
```
sub_preds = gbm_tuned.predict(submission_data)

print( sum(sub_preds) / len(sub_preds))
print( sum(train_Y) / len(train_Y))
print( sum(test_Y) / len(test_Y))
```

```
0.07142857142857142
0.36363636363636365
0.11940298507462686
```

In [ ]:
```
# get predictions for submission
sub_preds = gbm_tuned.predict(submission_data)
ids = list(range(0, len(sub_preds)))

output_data = pd.DataFrame({"Id": ids, "Predicted": sub_preds})
output_data = output_data.set_index("Id")

output_data.to_csv("gbm_submission.csv")
```

9 Logistics

The report is due on Dec 9th. Good luck!