Project Milestone 3. DSC550-Jennifer Barrera Conde

May 17, 2024

1 Final Project

2 Course 550

3 Jennifer Barrera Conde

4 Project Milestone 1:

4.1 Description of data:

This dataset was gathered by IBM; the collected data about employee attrition and performance. I decided to focus on the data related to finance, such as monthly income and anything that could impact it, such as age, hourly rate, years with the company, years since the last promotion, and years in the current position. Another factor that could be considered is "Years with current manager"; having a healthy work environment can lead to a lower attrition rate and higher performance, although measuring such numbers can not be easily proven to have a direct impact on an employee's performance, it can be something interesting to consider. However, if one considers environmental satisfaction, job satisfaction, performance rating, and years in a current role and compares it to "Years with current manager," it may lead to an indirect relationship.

[1]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
import thinkplot
import thinkstats2
```

[3]:
```
df = pd.read_csv('WA_Fn-UseC_-HR-Employee-Attrition[1].csv')
```

[4]:
```
df.shape
```

[4]:
```
(1470, 35)
```

[5]:
```
# This is just as an example
df.head(10)
```

[5]:
```
   Age Attrition     BusinessTravel  DailyRate              Department  \
0   41       Yes      Travel_Rarely       1102                   Sales
1   49        No  Travel_Frequently        279  Research & Development
2   37       Yes      Travel_Rarely       1373  Research & Development
3   33        No  Travel_Frequently       1392  Research & Development
4   27        No      Travel_Rarely        591  Research & Development
5   32        No  Travel_Frequently       1005  Research & Development
6   59        No      Travel_Rarely       1324  Research & Development
7   30        No      Travel_Rarely       1358  Research & Development
8   38        No  Travel_Frequently        216  Research & Development
9   36        No      Travel_Rarely       1299  Research & Development

   DistanceFromHome  Education EducationField  EmployeeCount  EmployeeNumber  \
0                 1          2  Life Sciences              1               1
1                 8          1  Life Sciences              1               2
2                 2          2          Other              1               4
3                 3          4  Life Sciences              1               5
4                 2          1        Medical              1               7
5                 2          2  Life Sciences              1               8
6                 3          3        Medical              1              10
7                24          1  Life Sciences              1              11
8                23          3  Life Sciences              1              12
9                27          3        Medical              1              13

   …  RelationshipSatisfaction StandardHours  StockOptionLevel  \
0  …                         1            80                 0
1  …                         4            80                 1
2  …                         2            80                 0
3  …                         3            80                 0
4  …                         4            80                 1
5  …                         3            80                 0
6  …                         1            80                 3
7  …                         2            80                 1
8  …                         2            80                 0
9  …                         2            80                 2

   TotalWorkingYears  TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
0                  8                      0               1               6
1                 10                      3               3              10
2                  7                      3               3               0
3                  8                      3               3               8
4                  6                      3               3               2
5                  8                      2               2               7
6                 12                      3               2               1
```

```
7                   1                       2                3                1
8                  10                       2                3                9
9                  17                       3                2                7

  YearsInCurrentRole  YearsSinceLastPromotion  YearsWithCurrManager
0                   4                        0                     5
1                   7                        1                     7
2                   0                        0                     0
3                   7                        3                     0
4                   2                        2                     2
5                   7                        3                     6
6                   0                        0                     0
7                   0                        0                     0
8                   7                        1                     8
9                   7                        7                     7

[10 rows x 35 columns]
```

The dataset I chose has a total of 1471 participants and has a total of 35 variables, as stated in the previous code blocks.

[6]:
```
# I used the following to know what are my options before choosing what graphs
 ↪to make and which data to use
df.columns
```

[6]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
```

[7]:
```
df.dtypes
```

[7]:
```
Age                          int64
Attrition                   object
BusinessTravel              object
DailyRate                    int64
Department                  object
DistanceFromHome             int64
Education                    int64
EducationField              object
EmployeeCount                int64
```

```
EmployeeNumber               int64
EnvironmentSatisfaction      int64
Gender                      object
HourlyRate                   int64
JobInvolvement               int64
JobLevel                     int64
JobRole                     object
JobSatisfaction              int64
MaritalStatus               object
MonthlyIncome                int64
MonthlyRate                  int64
NumCompaniesWorked           int64
Over18                      object
OverTime                    object
PercentSalaryHike            int64
PerformanceRating            int64
RelationshipSatisfaction     int64
StandardHours                int64
StockOptionLevel             int64
TotalWorkingYears            int64
TrainingTimesLastYear        int64
WorkLifeBalance              int64
YearsAtCompany               int64
YearsInCurrentRole           int64
YearsSinceLastPromotion      int64
YearsWithCurrManager         int64
dtype: object
```

[8]:

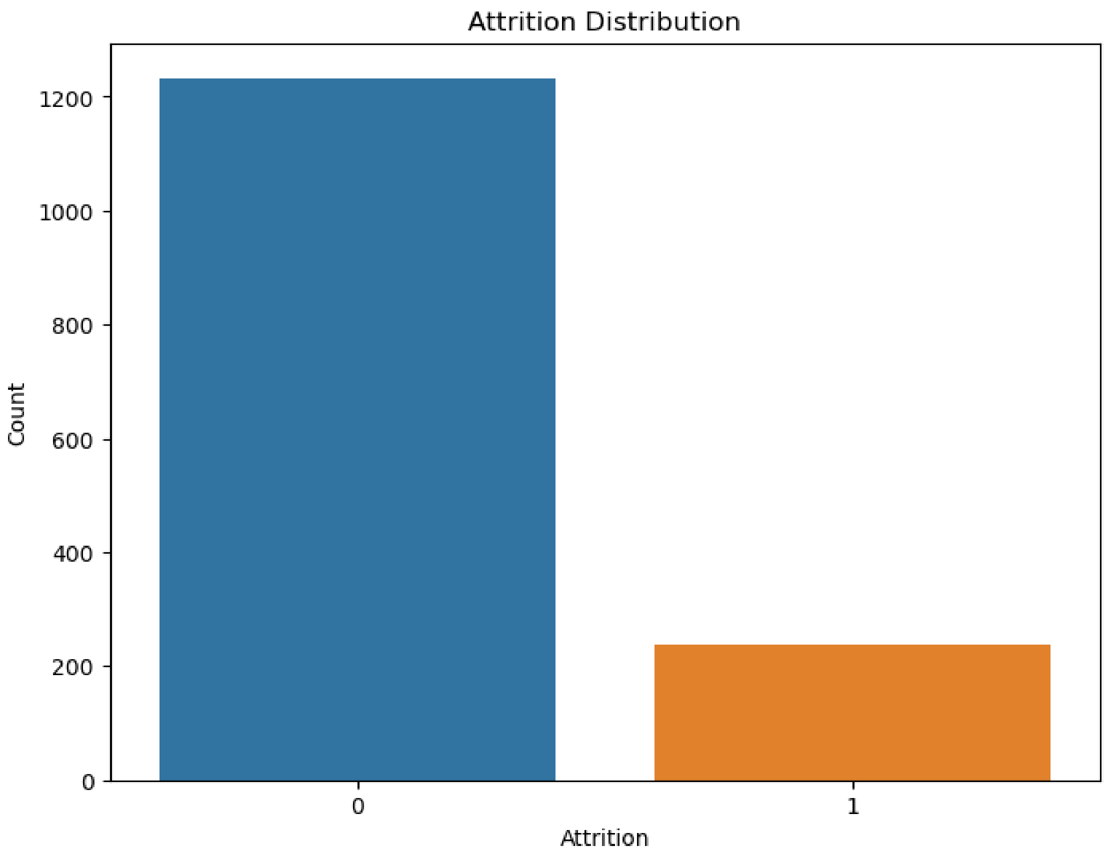
```
# Convert 'Attrition' column from object to int64
df['Attrition'] = df['Attrition'].map({'Yes': 1, 'No': 0})

# Verify the change by printing the updated data types
print(df.dtypes)
```

```
Age                          int64
Attrition                    int64
BusinessTravel              object
DailyRate                    int64
Department                  object
DistanceFromHome             int64
Education                    int64
EducationField              object
EmployeeCount                int64
EmployeeNumber               int64
EnvironmentSatisfaction      int64
Gender                      object
HourlyRate                   int64
JobInvolvement               int64
```

```
JobLevel                     int64
JobRole                     object
JobSatisfaction              int64
MaritalStatus               object
MonthlyIncome                int64
MonthlyRate                  int64
NumCompaniesWorked           int64
Over18                      object
OverTime                    object
PercentSalaryHike            int64
PerformanceRating            int64
RelationshipSatisfaction     int64
StandardHours                int64
StockOptionLevel             int64
TotalWorkingYears            int64
TrainingTimesLastYear        int64
WorkLifeBalance              int64
YearsAtCompany               int64
YearsInCurrentRole           int64
YearsSinceLastPromotion      int64
YearsWithCurrManager         int64
dtype: object
```

[9]:

```
# Now that I know what kind of information my dataset provides, I can start␣
↪working on the graphs
# The first issue at hand is attrition
# First, let's understand the distribution of attrition
plt.figure(figsize=(8, 6))
sns.countplot(x='Attrition', data=df)
plt.title('Attrition Distribution')
plt.xlabel('Attrition')
plt.ylabel('Count')
plt.show()
```



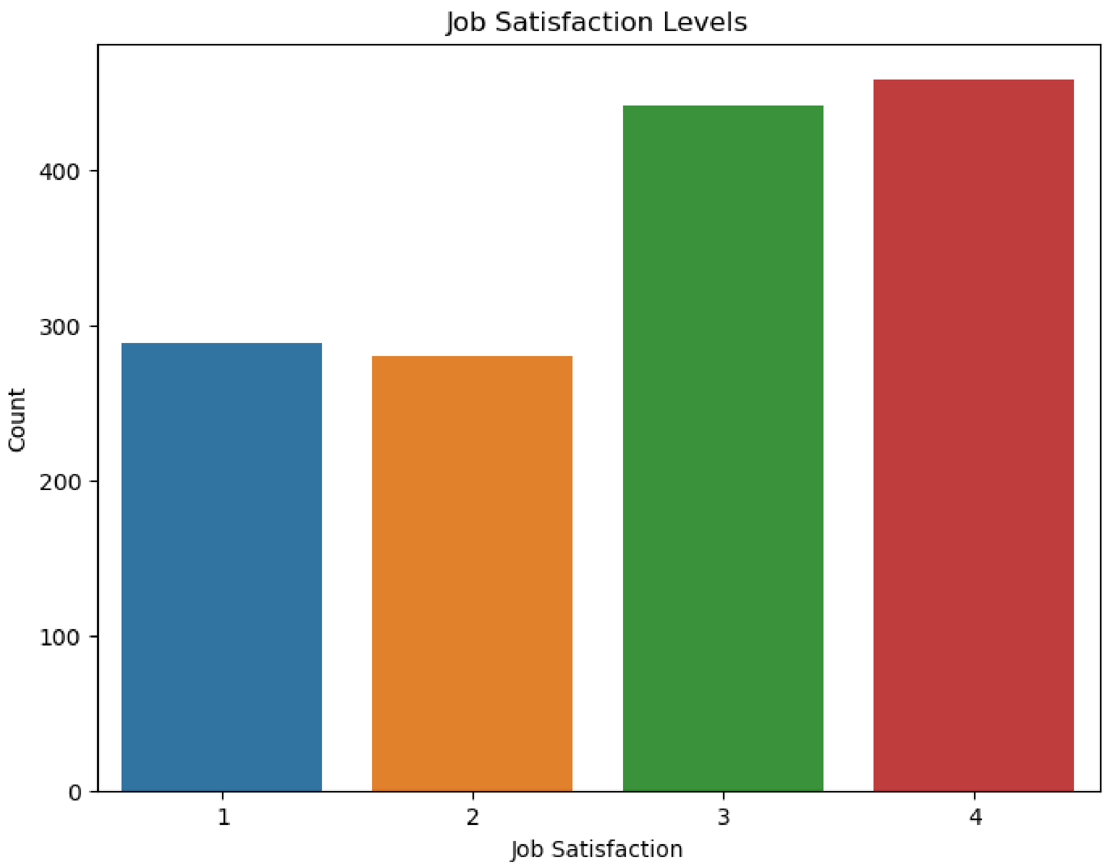
As priviously mentioned my data contained 1470 participants, from this graph we can tell that about a little over 200 participants have been accounted for attrition.

[ ]:

```

```

[10]:

```
# Now, let's visualize the job satisfaction levels
plt.figure(figsize=(8, 6))
sns.countplot(x='JobSatisfaction', data=df)
plt.title('Job Satisfaction Levels')
plt.xlabel('Job Satisfaction')
plt.ylabel('Count')
plt.show()
```



According to IBM: 1 'Low' 2 'Medium' 3 'High' 4 'Very High'

From this graph we can assume that more than hald of the employees are actually satisfied with their jobs.

[ ]:

```

```

[11]:

```
# Another factor at Disney is age, many of their employees come as part of the
↪Disney College program
# Which tend to be some of the youngest employees
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Age', bins=30, kde=True)
plt.title('Age Distribution of Employees')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

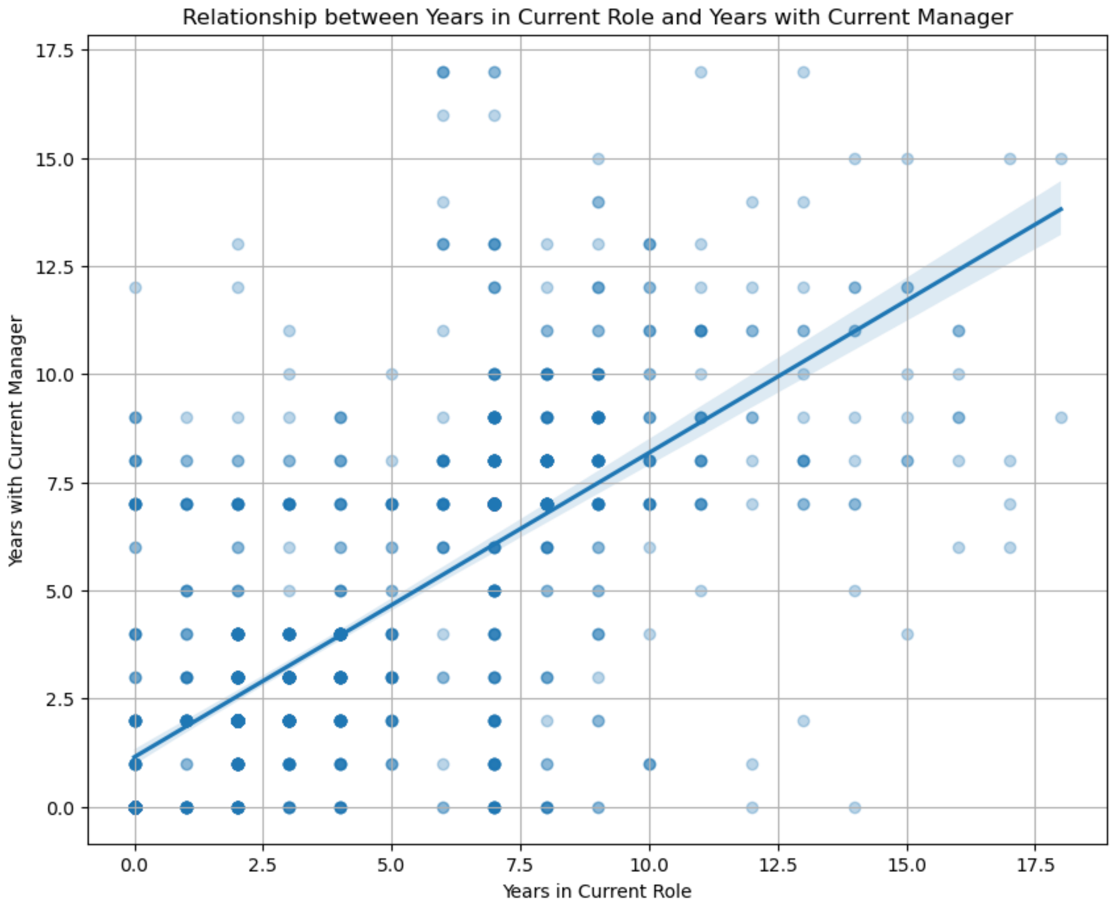
Although this data is not related to Disney, and IBM's dataset is being used as a mock test the patterns of age group do match those at some Disney locations.

[ ]:

```
```

[12]:

```
# Create a scatter plot with linear regression line
plt.figure(figsize=(10, 8))
sns.regplot(x='YearsInCurrentRole', y='YearsWithCurrManager', data=df,␣
 ↪scatter_kws={'alpha':0.3})
plt.title('Relationship between Years in Current Role and Years with Current␣
 ↪Manager')
plt.xlabel('Years in Current Role')
plt.ylabel('Years with Current Manager')
plt.grid(True)
plt.show()
```



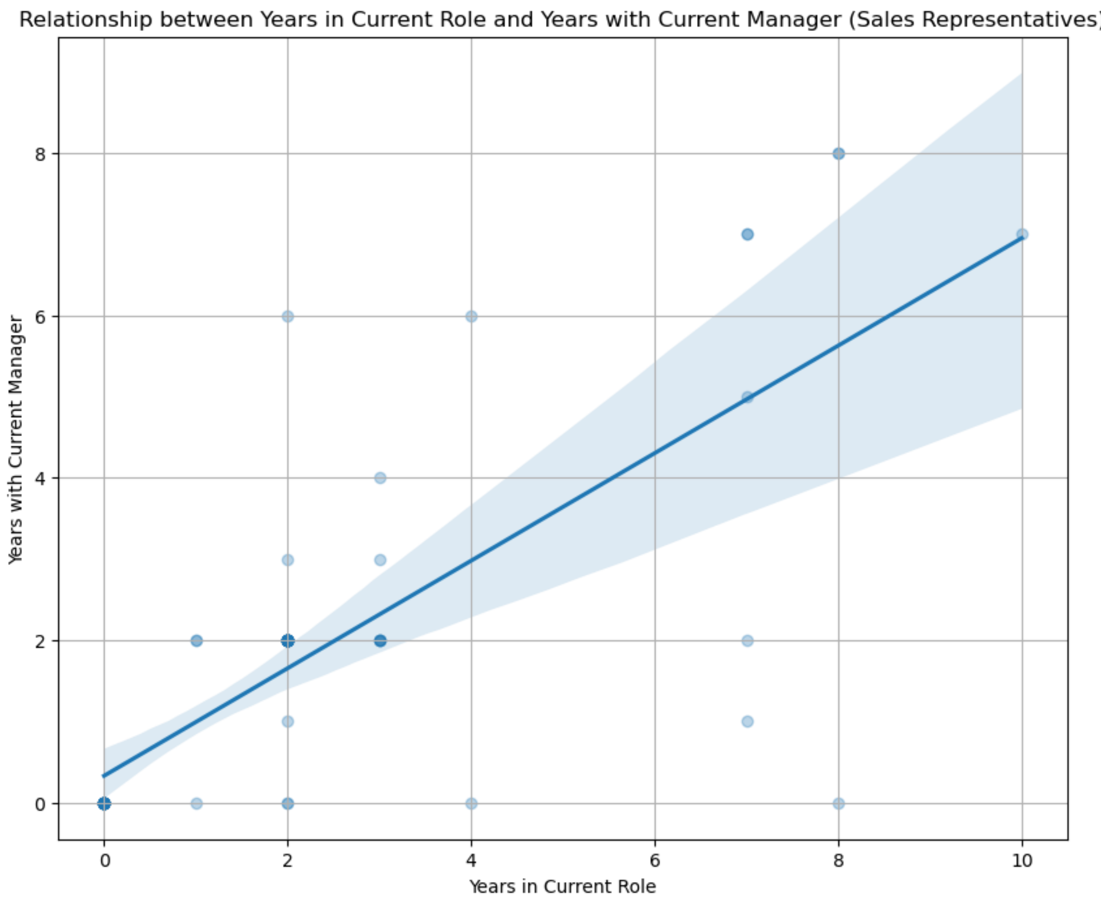
There is no clear consistency in the graph, this could also be due to so many roles being part of the dataset collected by IBM, let us take a look at the roles collected by IBM and try again

[13]:
```
# What are the job roles?
job_roles_counts = df['JobRole'].value_counts()

print("Job Roles and Frequencies:")
print(job_roles_counts)
```

```
Job Roles and Frequencies:
JobRole
Sales Executive              326
Research Scientist           292
Laboratory Technician        259
Manufacturing Director       145
Healthcare Representative    131
Manager                      102
Sales Representative          83
Research Director             80
```

```
Human Resources              52
Name: count, dtype: int64
```

[14]:
```
# Filter the dataset to include only employees with the job role 'Sales␣
↪Representative'
sales_rep_df = df[df['JobRole'] == 'Sales Representative']

# Create a scatter plot with linear regression line for Sales Representatives
plt.figure(figsize=(10, 8))
sns.regplot(x='YearsInCurrentRole', y='YearsWithCurrManager',␣
↪data=sales_rep_df, scatter_kws={'alpha': 0.3})
plt.title('Relationship between Years in Current Role and Years with Current␣
↪Manager (Sales Representatives)')
plt.xlabel('Years in Current Role')
plt.ylabel('Years with Current Manager')
plt.grid(True)
plt.show()
```



Now that we selected a role and tried again we can actually see a positive correlation between years in current role, and years with current manager based on "Sales Representative" as the only job role

I am currently happy with the results for my first Project Milestone, it gives me more ideas as to how it could go from now. Should I do the same step but focusing in a different job role? Should I continue only with sales representative? What questions could I answer next to prove my hypothesis that a manager could be a reason for higher or lower rates of attrition?

[ ]:

```

```

5 Project Milestone 2:

5.0.1 Drop unnecessary features:

[15]:

```
to_drop = ['EmployeeCount', 'EmployeeNumber', 'StandardHours']
df_cleaned = df.drop(to_drop, axis=1)
print ("The current unnecessary features are: ", (to_drop))

print ("Preview of the cleaned data: ", (df_cleaned.head(2)))
```

```
The current unnecessary features are:  ['EmployeeCount', 'EmployeeNumber',
'StandardHours']
Preview of the cleaned data:     Age  Attrition     BusinessTravel  DailyRate
Department  \
0   41          1      Travel_Rarely       1102                   Sales
1   49          0  Travel_Frequently        279  Research & Development

   DistanceFromHome  Education EducationField  EnvironmentSatisfaction  \
0                 1          2  Life Sciences                        2
1                 8          1  Life Sciences                        3

   Gender  …  PerformanceRating  RelationshipSatisfaction  StockOptionLevel  \
0  Female  …                  3                         1                 0
1    Male  …                  4                         4                 1

  TotalWorkingYears  TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
0                 8                      0               1               6
1                10                      3               3              10

   YearsInCurrentRole  YearsSinceLastPromotion YearsWithCurrManager
0                   4                        0                    5
1                   7                        1                    7

[2 rows x 32 columns]
```

This method allows you to selectively remove columns from a DataFrame based on specific criteria or a predefined list of column names (to_drop). Adjust the to_drop list based on the columns you want to exclude from your DataFrame (df).

Doing steps as such can also save in memory and processing power as it stops the necessity of using more resources than needed.

[ ]:

5.0.2 Perform any data extraction/selection steps:

[16]:
```
# I opted for filtering as a test, where I selected 'Sales'
sales_data = df_cleaned[df_cleaned['Department'] == 'Sales']
num_rows_sales_data = len(sales_data)
print("Number of participants in sales_data after filtering:",␣
↪num_rows_sales_data)
```

```
Number of participants in sales_data after filtering: 446
```

This step filters the data frame to include only rows where the value in the "Department" column is "Sales". The boolean runs through the chart classifying True and False for each row and only keeps the True. This means that out of a total of 1470 participants only 446 of those participants where "True" to the previously created Filter.

[ ]:

5.0.3 Transform features if necessary:

[17]:
```
# Encode categorical variables using One-Hot
categorical_cols = ['BusinessTravel', 'Department', 'EducationField', 'Gender',␣
↪'JobRole', 'MaritalStatus', 'OverTime']
df_encoded = pd.get_dummies(df_cleaned, columns=categorical_cols,␣
↪drop_first=True)
```

The variable 'df_encoded' holds the new DataFrame after applying one-hot encoding to the specified categorical columns (categorical_cols) in df_cleaned. Each categorical column is replaced with multiple binary (0 or 1) columns representing the categories within that column.

This process is commonly used in machine learning workflows to prepare categorical data for modeling, where numeric inputs are required. One-hot encoding ensures that categorical variables are represented appropriately for analysis and modeling purposes.

[ ]:

5.0.4 Engineer new useful features:

[18]:
```
df_encoded['YearsInCurrentRoleRatio'] = df_encoded['YearsInCurrentRole'] /␣
↪df_encoded['YearsAtCompany']
```

Example/Test:

[19]:
```
# Assuming df_encoded is a DataFrame with columns 'YearsInCurrentRole' and␣
↪'YearsAtCompany'
data = {
```

```
    'YearsInCurrentRole': [3, 2, 5, 1, 4],
    'YearsAtCompany': [5, 3, 8, 2, 6]
}

df_encoded = pd.DataFrame(data)

# Calculate the ratio of YearsInCurrentRole to YearsAtCompany
df_encoded['YearsInCurrentRoleRatio'] = df_encoded['YearsInCurrentRole'] /␣
 ↪df_encoded['YearsAtCompany']

print("DataFrame with Ratio Column:")
print(df_encoded)
```

```
DataFrame with Ratio Column:
   YearsInCurrentRole  YearsAtCompany  YearsInCurrentRoleRatio
0                   3               5                 0.600000
1                   2               3                 0.666667
2                   5               8                 0.625000
3                   1               2                 0.500000
4                   4               6                 0.666667
```

This kind of feature engineering can be useful for creating new insights or metrics from existing columns in a DataFrame. The 'YearsInCurrentRoleRatio' column provides a normalized measure of how long employees have stayed in their current roles relative to their total tenure at the company, which can be valuable for certain analyses or modeling tasks.

[ ]:

5.0.5 More Testing:

Testing transformed features:

[20]:

```
# Assuming df_cleaned is a DataFrame containing categorical columns
data = {
    'BusinessTravel': ['Travel_Rarely', 'Travel_Frequently', 'Non-Travel'],
    'Department': ['Sales', 'Research & Development', 'Sales'],
    'EducationField': ['Life Sciences', 'Medical', 'Life Sciences'],
    'Gender': ['Female', 'Male', 'Male'],
    'JobRole': ['Sales Executive', 'Research Scientist', 'Sales␣
 ↪Representative'],
    'MaritalStatus': ['Single', 'Married', 'Single'],
    'OverTime': ['Yes', 'No', 'Yes'],
    'Attrition': ['Yes', 'No', 'Yes']
}

df_cleaned = pd.DataFrame(data)

categorical_cols = ['BusinessTravel', 'Department', 'EducationField', 'Gender',␣
 ↪'JobRole', 'MaritalStatus', 'OverTime', 'Attrition']
```

```
# Apply one-hot encoding to specified categorical columns
df_encoded = pd.get_dummies(df_cleaned, columns=categorical_cols,␣
↪drop_first=True)

print("Original DataFrame:")
print(df_cleaned)

print("\nEncoded DataFrame:")
print(df_encoded)
```

```
Original DataFrame:
      BusinessTravel              Department EducationField  Gender  \
0      Travel_Rarely                   Sales  Life Sciences  Female
1  Travel_Frequently  Research & Development        Medical    Male
2         Non-Travel                   Sales  Life Sciences    Male

                JobRole MaritalStatus OverTime Attrition
0       Sales Executive        Single      Yes       Yes
1    Research Scientist       Married       No        No
2  Sales Representative        Single      Yes       Yes

Encoded DataFrame:
   BusinessTravel_Travel_Frequently  BusinessTravel_Travel_Rarely  \
0                             False                          True
1                              True                         False
2                             False                         False

   Department_Sales  EducationField_Medical  Gender_Male  \
0              True                   False        False
1             False                    True         True
2              True                   False         True

   JobRole_Sales Executive  JobRole_Sales Representative  \
0                     True                         False
1                    False                         False
2                    False                          True

   MaritalStatus_Single  OverTime_Yes  Attrition_Yes
0                  True          True           True
1                 False         False          False
2                  True          True           True
```

I decided to run one more test where the df_encoded DataFrame showcases the transformation of categorical data into a format suitable for machine learning algorithms that require numeric inputs. Each category within the specified columns is represented by a binary (0 or 1) indicator column in the encoded DataFrame (df_encoded).

[ ]:

```

```

Testing for Missing Values:

[21]:

```
# Apply fillna with column means
df_encoded.fillna(df_encoded.mean(), inplace=True)

# Step 1: Inspect DataFrame before and after operation
print("DataFrame before filling NaNs:")
print(df_encoded)  # Display DataFrame before filling NaNs

# Step 2: Check for NaN values after operation
nan_counts = df_encoded.isnull().sum()
print("\nNumber of NaN values after filling with column means:")
print(nan_counts)  # Display counts of NaN values in each column

# Step 3: Verify filling with column means
column_means = df_encoded.mean()
print("\nColumn Means:")
print(column_means)  # Display means of each column
```

```
DataFrame before filling NaNs:
   BusinessTravel_Travel_Frequently  BusinessTravel_Travel_Rarely  \
0                             False                          True
1                              True                         False
2                             False                         False

   Department_Sales  EducationField_Medical  Gender_Male  \
0              True                   False        False
1             False                    True         True
2              True                   False         True

   JobRole_Sales Executive  JobRole_Sales Representative  \
0                     True                         False
1                    False                         False
2                    False                          True

   MaritalStatus_Single  OverTime_Yes  Attrition_Yes
0                  True          True           True
1                 False         False          False
2                  True          True           True

Number of NaN values after filling with column means:
BusinessTravel_Travel_Frequently    0
BusinessTravel_Travel_Rarely        0
Department_Sales                    0
EducationField_Medical              0
Gender_Male                         0
```

```
JobRole_Sales Executive          0
JobRole_Sales Representative     0
MaritalStatus_Single             0
OverTime_Yes                     0
Attrition_Yes                    0
dtype: int64

Column Means:
BusinessTravel_Travel_Frequently    0.333333
BusinessTravel_Travel_Rarely        0.333333
Department_Sales                    0.666667
EducationField_Medical              0.333333
Gender_Male                         0.666667
JobRole_Sales Executive             0.333333
JobRole_Sales Representative        0.333333
MaritalStatus_Single                0.666667
OverTime_Yes                        0.666667
Attrition_Yes                       0.666667
dtype: float64
```

By following these steps and examining the DataFrame and its characteristics, I can confirm whether the fillna() operation was successful in replacing NaN values with column means in df_encoded. This verification process ensures the integrity and quality of the dataset after performing data imputation.

[ ]:

5.0.6 Create dummy variables if necessary:

Currently there is no need for any more dummy variables to be created.

[ ]:

6 Project Milestone 3:

6.0.1 Data Preparation:

Ensure the dataset is ready for modeling. This includes handling missing values, encoding categorical variables, and splitting into features and target variable.

In this step I want to focus on Attrition.

[22]:

```
print(df_encoded.columns)
```

```
Index(['BusinessTravel_Travel_Frequently', 'BusinessTravel_Travel_Rarely',
       'Department_Sales', 'EducationField_Medical', 'Gender_Male',
       'JobRole_Sales Executive', 'JobRole_Sales Representative',
       'MaritalStatus_Single', 'OverTime_Yes', 'Attrition_Yes'],
      dtype='object')
```

[23]:
```python
# Assuming df_encoded is the DataFrame after feature engineering and encoding
X = df_encoded.drop(columns=['Attrition_Yes'])  # Features
y = df_encoded['Attrition_Yes']  # Target variable
```

[ ]:
```python

```

6.0.2 Model Selection:

For a classification problem like predicting employee attrition, common models include Logistic Regression, Decision Trees, Random Forest, etc. I chose Logistic Regression for its simplicity and interpretability.

[24]:
```python
from sklearn.linear_model import LogisticRegression

# Initialize the Logistic Regression model
model = LogisticRegression()
```

[ ]:
```python

```

6.0.3 Model Training:

Splitting the data into training and testing sets.

[25]:
```python
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train the model on the training data
model.fit(X_train, y_train)
```

[25]:
```
LogisticRegression()
```

[ ]:
```python

```

6.0.4 Model Evaluation:

Select appropriate evaluation metrics. For this binary classification problem, I'll use accuracy, precision, recall, and F1-score.

[26]:
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

# Predict on the testing data
y_pred = model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
```

```
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
```

[ ]:

6.1 Overview/Conclusion:

Accuracy, Precision, Recall, and F1-score are all reported as 1.0, which indicates perfect performance. However, this may not always be the case in real-world scenarios. It's essential to interpret these metrics in conjunction with the context of the problem and the dataset.

These metrics are fundamental tools for assessing how well a model performs its classification task. They help in understanding the model's behavior and identifying areas for improvement if necessary.

[ ]: