D599 Data Preparation and Exploration Task 1
Data Cleaning Report by Eric Williams

Part I: Data Profiling

A1a:PROFILE DATA (General characteristics)
The initial dataset is a spreadsheet with 35 columns containing company's data collected about its employees, listed in alphabetical order (except for "turnover"). There are 10,323 rows in the dataset for each column.

A1b:VARIABLE DATA TYPES and A1c:OBSERVABLE VALUES
I used the attribute .dtype to list the datatypes of each column and this was the result:

- Age float64
- Turnover 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 float64
- MonthlyRate float64
- NumCompaniesWorked float64
- Over18 object
- OverTime object
- PercentSalaryHike int64
- PerformanceRating int64
- RelationshipSatisfaction int64
- StandardHours int64
- StockOptionLevel int64
- TotalWorkingYears float64
- TrainingTimesLastYear float64
- WorkLifeBalance int64

- YearsAtCompany int64
- YearsInCurrentRole int64
- YearsSinceLastPromotion float64
- YearsWithCurrManager object

However, the datacamp videos list the three data types as numeric, text, and date. These are the datatypes, subtypes, and the first three distinct observable entries in each column in as defined by the course material and given above by the .dtype attribute:

1. Age
    a. Type: Numeric
    b. Subtype: Float64
    c. Observable examples: 33, 35, 27
2. Turnover
    a. Type: Text/String
    b. Subtype: Object
    c. Observable examples: Yes, No
3. BusinessTravel
    a. Type: Text/String
    b. Subtype: Object
    c. Observable examples: Non-Travel, Trabel_Frequently, Travel_Rarely
4. DailyRate
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 241, 679, 359
5. Department
    a. Type: Text/String
    b. Subtype: Object
    c. Observable examples: Hardware, Support, Human Resources
6. DistanceFromHome
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 16, 7, 50
7. Education
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 3, 2, 1
8. EducationField
    a. Type: Text/String
    b. Subtype: Object
    c. Observable examples: Technical Degree, Life Sciences, Human Resources
9. EmployeeCount
    a. Type: Numeric
    b. Subtype: Int64

   c. Observable examples: 1
10. EmployeeNumber
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 3505, 1129, 6305
11. EnvironmentSatisfaction
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 1, 3, 4
12. Gender
    a. Type: Text/String
    b. Subtype: Object
    c. Observable examples: Female, Male
13. HourlyRate
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 67, 122, 199
14. JobInvolvement
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 3, 2, 1
15. JobLevel
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 3, 5, 1
16. JobRole
    a. Type: Text/String
    b. Subtype: Object
    c. Observable examples: Manufacturing Director, Research Director, Sales Representative
17. JobSatisfaction
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 1, 2, 3
18. MaritalStatus
    a. Type: Text/String
    b. Subtype: Object
    c. Observable examples: Single, Married, Divorced
19. MonthlyIncome
    a. Type: Numeric
    b. Subtype: Float64
    c. Observable examples: 36809, 1690, 50883
20. MonthlyRate
    a. Type: Numeric

    b. Subtype: Float64
    c. Observable examples: 294472, 32110, 865011
21. NumCompaniesWorked
    a. Type: Numeric
    b. Subtype: Float64
    c. Observable examples: 1, 6, 8
22. Over18
    a. Type: Text/String
    b. Subtype: Object
    c. Observable examples: Y
23. OverTime
    a. Type: Text/String
    b. Subtype: Object
    c. Observable examples: Yes, No
24. PercentSalaryHike
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 18, 5, 7
25. PerformanceRating
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 1, 4, 2
26. RelationshipSatisfaction
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 1, 4, 2
27. StandardHours
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 8
28. StockOptionLevel
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 4, 1, 3
29. TotalWorkingYears
    a. Type: Numeric
    b. Subtype: Integer
    c. Observable examples: 35, 5, 10
30. TrainingTimesLastYear
    a. Type: Numeric
    b. Subtype: Integer
    c. Observable examples: 4, 1, 3
31. WorkLifeBalance
    a. Type: Numeric

b. Subtype: Int64
c. Observable examples:4, 1, 2

32. YearsAtCompany
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 13, 4, 14
33. YearsInCurrentRole
    a. Type: Numeric
    b. Subtype: Int64
    c. Observable examples: 2, 3, 7
34. YearsSinceLastPromotion
    a. Type: Numeric
    b. Subtype: Float64
    c. Observable examples: 8, 3, 4
35. YearsWithCurrManager
    a. Type: Text/String
    b. Subtype: Object
    c. Observable examples: 11, 4, 2

However, all of the floats appear to be whole numbers. Since they do not include decimals, they could easily be used as integers. Also, YearsWithCurrManager could be an integer and not a string as the .dtype attribute suggests.

Part II: Data Cleaning and Plan

B1:DATASET QUALITY ISSUES AND B2: LIST OF QUALITY ISSUES

I started by importing the data into a dataframe:

```
import pandas as pd
```

```
file_path = r"C:\Users\18014\Desktop\Masters\D599 - Data Preparation and Exploration\Employee Turnover Dataset.xlsx"

df = pd.read_excel(file_path)

print(df.dtypes)
```

Quality Issue #1 - Duplicate Entries

First, I checked to see if there were duplicated rows using the following code:

```
#Looking for duplicated data
num_duplicates = df.duplicated().sum()
print(f'Number of duplicate rows: {num_duplicates}')
```

```
Number of duplicate rows: 298
```

Then I inspected the duplicate rows. The entire result is long, but I was able to look at these rows in the dataset and ensure they were duplicated.

```
duplicate_rows = df[df.duplicated()]
print(duplicate_rows)
```

```
        Age Turnover BusinessTravel  DailyRate              Department  \
712    22.0       No  Travel_Rarely        611                   Sales
985    42.0       No     Non-Travel        568                   Sales
1022   28.0      Yes  Travel_Rarely        406  Research & Development
1033   29.0      Yes     Non-Travel        492                 Support
1212   41.0      Yes  Travel_Rarely        711                Hardware
...     ...      ...            ...        ...                     ...
10272  43.0       No     Non-Travel        567                 Support
10275  47.0      Yes     Non-Travel       1034                Hardware
10290  27.0       No     Non-Travel        202  Research & Development
10291  27.0       No  Travel_Rarely        649                Hardware
10319  32.0       No     Non-Travel        145                Hardware

       DistanceFromHome  Education    EducationField  EmployeeCount  \
712                  33          2             Other              1
985                  50          1     Life Sciences              1
1022                 38          1  Technical Degree              1
1033                 49          5         Marketing              1
1212                 49          5   Human Resources              1
...                 ...        ...               ...            ...
10272                43          5   Human Resources              1
10275                 2          4     Life Sciences              1
10290                16          5         Marketing              1
10291                44          4   Human Resources              1
10319                49          1             Other              1

       EmployeeNumber  ...  RelationshipSatisfaction StandardHours  \
712               416  ...                         3            80
```

There are 298 duplicate rows to clean.

Quality Issue #2 - Missing Values

To look for missing values, I had Python look for null cells using ".isnull" and summed the totals. This way I could see which columns are missing values:

```
#Looking for missing data
print(df.isnull().sum())
```

The result:

```
Age                          1
Turnover                     0
BusinessTravel               0
DailyRate                    0
Department                   0
DistanceFromHome             0
Education                    0
EducationField               2
EmployeeCount                0
EmployeeNumber               0
EnvironmentSatisfaction      0
Gender                       3
HourlyRate                   0
JobInvolvement               0
JobLevel                     0
JobRole                      0
JobSatisfaction              0
MaritalStatus                0
MonthlyIncome                1
MonthlyRate                  2
NumCompaniesWorked           1
Over18                       0
OverTime                     0
PercentSalaryHike            0
PerformanceRating            0
RelationshipSatisfaction     0
StandardHours                0
StockOptionLevel             0
TotalWorkingYears            1
TrainingTimesLastYear      418
WorkLifeBalance              0
YearsAtCompany               0
YearsInCurrentRole           0
YearsSinceLastPromotion      2
YearsWithCurrManager         0
```

9 of the 35 columns have at least one cell of missing data, but the TrainingTimesLastYear column seems to be especially problematic. In the next section, I'll discuss the solution to this problem.

Quality Issue #3, 4, and 5 - Inconsistent Entries, Formatting Errors, and Outliers

I found it easiest to check for all three of these problems at once. By listing the distribution of the data, you can see at a glance if any data entry is formatted incorrectly, entered oddly/inconsistently, or is distributed abnormally. You can then find outliers by looking for extreme values. Listed below are the 35 columns I inspected and how I evaluated them for inconsistencies, formatting, and outliers. Please note that I cleaned each column as I progressed through the list, so some errors with the later columns were eliminated during the cleaning for earlier columns.

Age

Code for inspecting:

```
#Check Age column for abnormalities
Age_counts = df_cleaned['Age'].value_counts()
print(Age_counts)
```

```
Age
48.0    252
21.0    248
57.0    247
28.0    245
60.0    242
26.0    241
41.0    240
56.0    237
31.0    237
39.0    237
50.0    234
55.0    234
49.0    232
33.0    232
24.0    231
46.0    228
51.0    228
22.0    228
23.0    227
29.0    225
27.0    225
34.0    223
18.0    223
58.0    221
54.0    220
45.0    217
19.0    216
42.0    216
25.0    216
35.0    216
43.0    215
38.0    212
30.0    212
36.0    211
40.0    211
53.0    211
47.0    210
37.0    209
52.0    207
59.0    205
20.0    200
32.0    191
44.0    183
96.0      1
12.0      1
Name: count, dtype: int64
```

Findings: It is unlikely there is an employee that is 12 or 96 years old. These should be deleted.

Turnover

Code for inspecting:

```
#Check Turnover for abnormalities
Turnover_counts = df_cleaned['Turnover'].value_counts()
print(Turnover_counts)
```

```
Turnover
No     4914
Yes    4681
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

BusinessTravel

Code for inspecting:

```
#Check BusinessTravel for abnormalities
BusinessTravel_counts = df_cleaned['BusinessTravel'].value_counts()
print(BusinessTravel_counts)
```

```
BusinessTravel
Travel_Rarely        3250
Non-Travel           3197
Travel_Frequently    3145
1                       1
-1                      1
00                      1
```

Findings: Three entries are abnormal. 1, -1, and 00 should be deleted.

DailyRate

Code for inspecting:

```
#Making sure all values in DailyRate are numeric

#First any entires that are non-numeric will be changed to NaN values. Then count how many NaN values there are to see if we caught any errors
non_numeric_entries_DR = df_cleaned[pd.to_numeric(df_cleaned['DailyRate'], errors='coerce').isna()]
print(non_numeric_entries_DR)
```

```
Empty DataFrame
Columns: [Age, Turnover, BusinessTravel, DailyRate, Department, DistanceFromHome, Education, EducationField, EmployeeCount, EmployeeNumber, EnvironmentSat
isfaction, Gender, HourlyRate, JobInvolvement, JobLevel, JobRole, JobSatisfaction, MaritalStatus, MonthlyIncome, MonthlyRate, NumCompaniesWorked, Over18,
OverTime, PercentSalaryHike, PerformanceRating, RelationshipSatisfaction, StandardHours, StockOptionLevel, TotalWorkingYears, TrainingTimesLastYear, WorkL
ifeBalance, YearsAtCompany, YearsInCurrentRole, YearsSinceLastPromotion, YearsWithCurrManager]
Index: []

[0 rows x 35 columns]
```

```
#Since the DataFrame returned empty, it must all be numeric. Now let's look for outliers
```

```
#Check DailyRate for abnormalities
DailyRate_counts = df_cleaned['DailyRate'].value_counts()
print(DailyRate_counts)
```

```
DailyRate
1146    18
946     16
1099    15
691     15
573     15
        ..
771      1
439      1
807      1
342      1
828      1
```

```
# Check DailyRate for outliers
print(df_cleaned['DailyRate'].describe())
```

```
count    9592.000000
mean      807.078294
std       405.154126
min       100.000000
25%       455.750000
50%       806.500000
75%      1161.250000
max      1500.000000
Name: DailyRate, dtype: float64
```

Findings: Min and max values look normal. No odd data. No cleaning needed.

Department

Code for inspecting:

```
# Check Department for abnormalities
department_counts = df_cleaned['Department'].value_counts()
print(department_counts)
```

```
Department
Support                   1660
Hardware                  1620
Research & Development    1610
Software                  1598
Sales                     1554
Human Resources           1550
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

DistanceFromHome

Code for inspecting:

```
#Making sure all values in DistanceFromHome are numeric

#First any entires that are non-numeric will be changed to NaN values. Then count how many NaN values there are to see if we caught any errors
non_numeric_entries_DFH = df_cleaned[pd.to_numeric(df_cleaned['DistanceFromHome'], errors='coerce').isna()]
print(non_numeric_entries_DFH)
```

```
Empty DataFrame
Columns: [Age, Turnover, BusinessTravel, DailyRate, Department, DistanceFromHome, Education, EducationField, EmployeeCount, EmployeeNumber, EnvironmentSat
isfaction, Gender, HourlyRate, JobInvolvement, JobLevel, JobRole, JobSatisfaction, MaritalStatus, MonthlyIncome, MonthlyRate, NumCompaniesWorked, Over18,
OverTime, PercentSalaryHike, PerformanceRating, RelationshipSatisfaction, StandardHours, StockOptionLevel, TotalWorkingYears, TrainingTimesLastYear, WorkL
ifeBalance, YearsAtCompany, YearsInCurrentRole, YearsSinceLastPromotion, YearsWithCurrManager]
Index: []

[0 rows x 35 columns]
```

```
#Check DistanceFromHome for abnormalities
DistanceFromHome_counts = df_cleaned['DistanceFromHome'].value_counts()
print(DistanceFromHome_counts)
```

```
DistanceFromHome
1       216
5       212
30      210
3       206
29      206
7       205
34      205
9       204
41      203
16      201
43      201
14      201
11      200
37      199
25      198
32      198
36      198
18      198
12      198
13      197
44      195
27      195
19      193
22      192
10      192
48      192
33      189
49      189
38      189
40      188
26      188
42      187
4       187
24      186
28      186
50      185
17      184
39      184
6       184
2       183
15      183
46      181
21      180
45      180
23      180
47      179
31      175
20      175
8       170
35      162
3737      1
3535      1
978       1
Name: count, dtype: int64
```

Findings: There are three very large commutes from home. Unless the CEO commutes by private jet, nobody is commuting 900 to 3,000 miles. Anything over 100 appears to be an outlier

Education

Code for inspecting:

```
#Listing unique entries in Education
Education_counts = df_cleaned['Education'].value_counts()
print(Education_counts)
```

```
Education
1    1977
2    1927
4    1921
3    1886
5    1867
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

EducationField

Code for inspecting:

```
#Listing unique entries in EducationField
EducationField_counts = df_cleaned['EducationField'].value_counts()
print(EducationField_counts)
```

```
EducationField
Marketing           1642
Medical             1624
Other               1606
Life Sciences       1598
Human Resources     1582
Technical Degree    1536
                       1
Name: count, dtype: int64
```

Findings: There's one blank entry that should be deleted.

EmployeeCount

Code for inspecting:

```
#EmployeeCount should always be 1

#Looking for rows where EmployeeCount is not equal to 1
invalid_employee_count = df_cleaned[df_cleaned['EmployeeCount'] != 1]

#Display incorrect rows
print(invalid_employee_count)
```

```
       Age Turnover     BusinessTravel  DailyRate              Department  \
140   50.0       No  Travel_Frequently        547                Hardware
7167  56.0       No         Non-Travel        105                 Support
7877  28.0      Yes         Non-Travel        929  Research & Development

      DistanceFromHome  Education   EducationField  EmployeeCount  \
140                 13          4          Medical             -1
7167                38          5            Other              3
7877                12          2  Human Resources             -1

      EmployeeNumber  ...  RelationshipSatisfaction StandardHours  \
140             8115  ...                         3            80
7167              21  ...                         1            80
7877             194  ...                         2            80

      StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  \
140                  1               23.0                    3.0
7167                 3               29.0                    2.0
7877                 3               22.0                    3.0

     WorkLifeBalance  YearsAtCompany YearsInCurrentRole  \
140                3              20                 13
7167               1              22                  5
7877               1               7                  1

      YearsSinceLastPromotion  YearsWithCurrManager
140                      12.0                     4
7167                     20.0                     7
7877                      4.0                     1

[3 rows x 35 columns]
```

Findings: There are three rows that need to be fixed that do not have EmployeeCount = 1.

EmployeeNumber

Code for inspecting:

```
#Making sure all values in EmployeeNumber are numeric

#First any entires that are non-numeric will be changed to NaN values. Then count how many NaN values there are to see if we caught any errors
non_numeric_entries_EN = df_cleaned[pd.to_numeric(df_cleaned['EmployeeNumber'], errors='coerce').isna()]
print(non_numeric_entries_EN)
```

```
Empty DataFrame
Columns: [Age, Turnover, BusinessTravel, DailyRate, Department, DistanceFromHome, Education, EducationField, EmployeeCount, EmployeeNumber, EnvironmentSat
isfaction, Gender, HourlyRate, JobInvolvement, JobLevel, JobRole, JobSatisfaction, MaritalStatus, MonthlyIncome, MonthlyRate, NumCompaniesWorked, Over18,
OverTime, PercentSalaryHike, PerformanceRating, RelationshipSatisfaction, StandardHours, StockOptionLevel, TotalWorkingYears, TrainingTimesLastYear, WorkL
ifeBalance, YearsAtCompany, YearsInCurrentRole, YearsSinceLastPromotion, YearsWithCurrManager]
Index: []

[0 rows x 35 columns]
```

```
#All EmployeeNumbers are numeric
```

```
#List unique entries in the EducationField column
EmployeeNumber_counts = df_cleaned['EmployeeNumber'].value_counts()
print(EmployeeNumber_counts)
```

```
EmployeeNumber
7883    2
161     2
925     1
4056    1
6353    1
       ..
9928    1
3850    1
3390    1
1280    1
5122    1
Name: count, Length: 9586, dtype: int64
```

Findings: It looks like there's two shared EmployeeNumbers. However, I'm choosing to leave this untouched because it could be that an employee left and a new employee got his old number, meaning both would be valid employees. If there were some context as to why a number was repeated, that would give more insight as to whether the data is accurate.

EnvironmentSatisfaction

Code for inspecting:

```
# Count the occurrences of each response in the EnvironmentSatisfaction column
environment_satisfaction_counts = df_cleaned['EnvironmentSatisfaction'].value_counts()

# Display the counts
print("Environment Satisfaction Counts:")
print(environment_satisfaction_counts)
```

```
Environment Satisfaction Counts:
EnvironmentSatisfaction
1    2405
2    2405
3    2402
4    2376
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

Gender

Code for inspecting:

```
# Count responses in Gender
Gender_counts = df_cleaned['Gender'].value_counts()

# Display the counts
print(Gender_counts)
```

```
Gender
Male      4803
Female    4775
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

HourlyRate

Code for inspecting:

```
#Listing unique entries in  HourlyRate
HourlyRate_counts = df_cleaned['HourlyRate'].value_counts()

# Display the counts
print("HourlyRate Counts:")
print(HourlyRate_counts)
```

```
HourlyRate Counts:
HourlyRate
140    79
170    78
194    77
87     73
195    72
       ..
56     42
193    42
156    41
131    41
158    40
Name: count, Length: 171, dtype: int64
```

Findings: Looks good! No cleaning needed.

JobInvolvement

Code for inspecting:

```
#Listing unique entries in JobInvolvement
JobInvolvement_counts = df_cleaned['JobInvolvement'].value_counts()

# Display the counts
print("JobInvolvement:")
print(JobInvolvement_counts)
```

```
JobInvolvement:
JobInvolvement
3    2416
4    2408
2    2392
1    2372
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

JobLevel

Code for inspecting:

```
#Listing unique entries in  JobLevel
JobLevel_counts = df_cleaned['JobLevel'].value_counts()

# Display the counts
print("JobLevel:")
print(JobLevel_counts)
```

```
JobLevel:
JobLevel
4    2011
2    1959
3    1905
5    1870
1    1843
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

JobRole

Code for inspecting:

```
#Listing unique entries in JobRole
JobRole_counts = df_cleaned['JobRole'].value_counts()

# Display the counts
print("JobRole:")
print(JobRole_counts)
```

```
JobRole:
JobRole
Manager                      1048
Research Director             985
Manufacturing Director        978
Healthcare Representative     961
Human Resources               952
Laboratory Technician         945
Sales Executive               944
Developer                     936
Sales Representative          921
Research Scientist            915
                                3
Name: count, dtype: int64
```

Findings: There are three blank entries that should be deleted.

JobSatisfaction

Code for inspecting:

```
# Count unique entries in JobSatisfaction
JobSatisfaction_counts = df_cleaned['JobSatisfaction'].value_counts()

# Display the counts
print("JobSatisfaction:")
print(JobSatisfaction_counts)
```

```
JobSatisfaction:
JobSatisfaction
2    2459
4    2396
1    2395
3    2335
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

MaritalStatus

Code for inspecting:

```
#Listing unique entries in MartialStatus
MaritalStatus_counts = df_cleaned['MaritalStatus'].value_counts()

# Display the counts
print("MaritalStatus:")
print(MaritalStatus_counts)
```

```
MaritalStatus:
MaritalStatus
Divorced    3317
Single      3140
Married     3128
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed. All data fit neatly into three reasonable categories and normally distributed (no outliers)

MonthlyIncome

Code for inspecting:

```
#Listing unique entries in MonthlyIncome
MonthlyIncome_counts = df_cleaned['MonthlyIncome'].value_count

# Display the counts
print("MonthlyIncome:")
print(MonthlyIncome_counts)
```

```
MonthlyIncome:
MonthlyIncome
9392.0     4
34187.0    3
27263.0    3
2967.0     3
2463.0     3
          ..
28726.0    1
50291.0    1
41544.0    1
23711.0    1
29469.0    1
Name: count, Length: 8716, dtype: int64
```

```
# Looking for outliers in MonthlyIncome
print(df_cleaned['MonthlyIncome'].describe())
```

```
count     9585.000000
mean     25803.348774
std      14443.129226
min     -38005.000000
25%      13353.000000
50%      25423.000000
75%      38391.000000
max      50996.000000
Name: MonthlyIncome, dtype: float64
```

```
#Listing unique entries in MonthlyIncome that are less than 0
less_than_0 = df_cleaned[df_cleaned['MonthlyIncome'] < 0].shape[0]

print(f'Number of values in MonthlyIncome greater than 0: {less_than_0}')
```

Findings: Because monthly income can vary dramatically and can be unique or shared, the only numbers we need to be concerned about are negative numbers. There is one negative number that needs to be deleted.

MonthlyRate

Code for inspecting:

```
# MonthlyRate
MonthlyRate_counts = df_cleaned['MonthlyRate'].value_counts()
print("MonthlyRate:")
print(MonthlyRate_counts)
```

```
MonthlyRate:
MonthlyRate
526320.0    3
21078.0     2
739935.0    2
22920.0     2
870720.0    2
           ..
20133.0     1
130797.0    1
106505.0    1
334145.0    1
884070.0    1
Name: count, Length: 9446, dtype: int64
```

```
# Looking for outliers in MonthlyIncome
print(df_cleaned['MonthlyRate'].describe())
```

```
count    9.584000e+03
mean     9.140733e+07
std      8.909429e+09
min      1.270000e+03
25%      1.217768e+05
50%      3.071420e+05
75%      5.971372e+05
max      8.722149e+11
Name: MonthlyRate, dtype: float64
```

Findings: Need to delete the one outlier.

```
#Because no employee is likely to be making 8,722,149,000,000 (8 trillion dollars) in a month, this is an outlier that should be deleted.
#We can check for more outliers by checking the 10 highest results

#Sort MonthlyRatein descending order and display the top 10 rows
top_10_MonthlyRate = df_cleaned[['MonthlyRate']].sort_values(by='MonthlyRate', ascending=False).head(10)

print(top_10_MonthlyRate)
```

```
      MonthlyRate
2900  8.722149e+11
3466  1.523280e+06
8264  1.522920e+06
2190  1.514100e+06
4291  1.496430e+06
4048  1.492170e+06
1112  1.491360e+06
7497  1.491120e+06
4855  1.484400e+06
1056  1.482120e+06
```

NumCompaniesWorked

Code for inspecting:

```
#Listing unique entries in NumCompaniesWorked
NumCompaniesWorked_counts = df_cleaned['NumCompaniesWorked'].value_counts()
print("NumCompaniesWorked:")
print(NumCompaniesWorked_counts)
```

```
NumCompaniesWorked:
NumCompaniesWorked
2.0    1099
8.0    1079
7.0    1077
1.0    1072
0.0    1064
4.0    1062
6.0    1060
3.0    1059
5.0    1011
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

Over18

Code for inspecting:

```
#Listing unique entries in Over18
Over18_counts = df_cleaned['Over18'].value_counts()
print(Over18_counts)
```

```
Over18
Y    9583
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

OverTime

Code for inspecting:

```
#Listing unique entries in OverTime
OverTime_counts = df_cleaned['OverTime'].value_counts()
print(OverTime_counts)
```

```
OverTime
No     4806
Yes    4777
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

PercentSalaryHike

Code for inspecting:

```
#Listing unique entries in  PercentSalaryHike
PercentSalaryHike_counts = df_cleaned['PercentSalaryHike'].value_counts()
print("PercentSalaryHike:")
print(PercentSalaryHike_counts)
```

```
PercentSalaryHike:
PercentSalaryHike
30    219
44    212
40    209
11    209
32    206
19    206
6     205
4     204
9     204
41    203
46    200
42    200
43    198
26    198
27    198
33    197
20    197
3     197
25    197
8     197
28    195
10    195
2     194
36    192
15    191
45    191
18    190
24    190
5     189
48    188
0     187
49    187
35    186
13    186
34    185
47    184
21    184
22    183
7     183
12    182
37    182
38    181
23    180
31    179
14    179
29    178
39    175
16    175
17    173
1     163
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

PerformanceRating

Code for inspecting:

```
#Listing unique entries in PerformanceRating
PerformanceRating_counts = df_cleaned['PerformanceRating'].value_counts()
print(PerformanceRating_counts)
```

```
PerformanceRating
1    2445
4    2419
3    2405
2    2314
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

RelationshipSatisfaction

Code for inspecting:

```
#Listing unique entries in RelationshipSatisfaction
RelationshipSatisfaction_counts = df_cleaned['RelationshipSatisfaction'].value_counts()
print("RelationshipSatisfaction:")
print(RelationshipSatisfaction_counts)
```

```
RelationshipSatisfaction:
RelationshipSatisfaction
2    2444
3    2392
1    2380
4    2367
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

StandardHours

Code for inspecting:

```
#Listing unique entries in StandardHours
StandardHours_counts = df_cleaned['StandardHours'].value_counts()
print(StandardHours_counts)
```

```
StandardHours
80    9583
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

StockOptionLevel

Code for inspecting:

```
#Listing unique entries in StockOptionLevel
StockOptionLevel_counts = df_cleaned['StockOptionLevel'].value_counts()
print("StockOptionLevel:")
print(StockOptionLevel_counts)
```

```
StockOptionLevel:
StockOptionLevel
2    2450
4    2415
1    2391
3    2327
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

<u>TotalWorkingYears</u>
Code for inspecting:

```
#Listing unique entries in TotalWorkingYears
TotalWorkingYears_counts = df_cleaned['TotalWorkingYears'].value_counts()
print(TotalWorkingYears_counts)
```

```
TotalWorkingYears
 3.0      288
 32.0     269
 28.0     266
 16.0     260
 30.0     253
 26.0     252
 40.0     252
 21.0     249
 4.0      249
 13.0     249
 12.0     247
 18.0     247
 8.0      246
 24.0     245
 20.0     245
 29.0     243
 23.0     243
 17.0     242
 36.0     241
 34.0     241
 37.0     240
 35.0     239
 38.0     237
 19.0     235
 22.0     233
 10.0     233
 33.0     232
 6.0      231
 9.0      231
 11.0     229
 5.0      227
 39.0     227
 14.0     227
 1.0      226
 2.0      226
 31.0     223
 15.0     221
 7.0      217
 27.0     215
 25.0     205
-1.0        1
 222.0      1
Name: count, dtype: int64
```

Findings: Clearly nobody has been working for -1 year or 220 years. These rows will be deleted.

TrainingTimesLastYear

Code for inspecting:

```
#Listing unique entries in TrainingTimesLastYear
TrainingTimesLastYear_counts = df_cleaned['TrainingTimesLastYear'].value_counts()
print(TrainingTimesLastYear_counts)
```

```
TrainingTimesLastYear
4.0    1633
1.0    1616
2.0    1610
6.0    1579
3.0    1577
5.0    1566
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

WorkLifeBalance

Code for inspecting:

```
#Listing unique entries in WorkLifeBalance
WorkLifeBalance_counts = df_cleaned['WorkLifeBalance'].value_counts()
print(WorkLifeBalance_counts)
```

```
WorkLifeBalance
1    2435
2    2402
4    2384
3    2360
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

YearsAtCompany

Code for inspecting:

```
#Listing unique entries in YearsAtCompany
YearsAtCompany_counts = df_cleaned['YearsAtCompany'].value_counts()
print(YearsAtCompany_counts)
```

```
YearsAtCompany
1     1018
2      757
3      705
4      590
5      518
6      462
7      441
8      401
9      376
10     362
11     324
12     304
14     282
13     281
15     251
17     237
16     220
19     204
18     196
21     169
20     157
22     148
23     147
26     116
24     112
25     104
27     101
29      95
28      90
31      78
30      75
33      62
32      49
34      38
35      35
36      27
37      24
38      11
39       8
40       6
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

YearsInCurrentRole

Code for inspecting:

```
#Listing unique entries in YearsInCurrentRole
YearsInCurrentRole_counts = df_cleaned['YearsInCurrentRole'].value_counts()
print(YearsInCurrentRole_counts)
```

```
YearsInCurrentRole
1     2412
2     1328
3      949
4      775
5      610
6      501
7      418
8      355
9      313
10     269
11     244
12     212
13     161
14     146
15     119
16     118
17      99
18      92
20      63
19      59
21      55
22      49
23      44
26      28
25      28
24      28
27      26
31      18
28      15
30      13
29      11
32       8
33       5
34       4
35       3
37       3
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

YearsSinceLastPromotion

Code for inspecting:

```
#Listing unique entries in YearsSinceLastPromotion
YearsSinceLastPromotion_counts = df_cleaned['YearsSinceLastPromotion'].value_counts()
print(YearsSinceLastPromotion_counts)
```

```
YearsSinceLastPromotion
1.0     2395
2.0     1342
3.0      952
4.0      773
5.0      590
6.0      477
7.0      426
8.0      354
9.0      326
10.0     253
11.0     248
12.0     196
13.0     172
14.0     156
15.0     128
16.0     100
17.0      97
18.0      86
19.0      84
20.0      71
22.0      52
21.0      45
23.0      39
26.0      37
28.0      30
24.0      30
27.0      27
25.0      23
29.0      19
30.0      16
31.0      12
34.0       7
32.0       7
33.0       3
36.0       3
35.0       2
37.0       2
38.0       1
Name: count, dtype: int64
```

Findings: Looks good! No cleaning needed.

YearsWithCurrManager

Code for inspecting:

```
#Listing unique entries in YearsWithCurrManager
YearsWithCurrManager_counts = df_cleaned['YearsWithCurrManager'].value_counts()
print(YearsWithCurrManager_counts)
```

```
YearsWithCurrManager
1        2430
2        1352
3         985
4         777
5         578
6         492
7         408
8         361
9         287
10        245
11        221
12        213
13        185
14        149
16        135
15        120
18         89
17         86
19         75
21         54
22         54
20         53
23         39
24         35
29         24
27         24
25         23
26         21
28         18
30         14
33          7
34          7
31          6
32          5
na          2
37          2
35          2
36          1
38          1
-1000       1
Name: count, dtype: int64
```

Findings: Need to remove the last row. Working with a manager for -1000 years doesn't make sense.

C1:DATASET MODIFICATION and C2:DATA CLEANING TECHNIQUES

Here I will list the issues I found and the code I used to correct them, as well as the checks I used to ensure the data had been altered properly. I will also include why the cleaning method was helpful and necessary.

Quality Issue #1 - Duplicate Entries

The issue: There were 298 duplicate rows to delete. Because they are duplicates, they are not useful.

The solution:

```
# Drop all duplicate rows
df_cleaned = df.drop_duplicates()
```

```
#Checking again for duplicated data to ensure they were deleted
num_duplicates = df_cleaned.duplicated().sum()
print(f'Number of duplicate rows: {num_duplicates}')
```

```
Number of duplicate rows: 0
```

Quality Issue #2 - Missing Values

9 of the 35 columns have at least one cell of missing data. TrainingTimesLastYear column seems to be especially problematic.

The solution: Because the number of times trained could very well impact be a factor in worker retention, without this data or an explanation for why it is empty, it is not usable. I deleted the rows that were missing data.

```python
# Drop all rows with any missing values
df_cleaned = df_cleaned.dropna()

#Looking for missing data
print(df_cleaned.isnull().sum())
```

```
Age                         0
Turnover                    0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
Over18                      0
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StandardHours               0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
dtype: int64
```

```python
# Check the number of rows before and after dropping missing values
print(f"Original DataFrame: {df.shape[0]} rows")
print(f"Cleaned DataFrame: {df_cleaned.shape[0]} rows")
```

```
Original DataFrame: 10322 rows
Cleaned DataFrame: 9597 rows
```

We still have 96% of our data in our data frame and now there are no missing values.

Quality Issue #3, 4, and 5 - Inconsistent Entries, Formatting Errors, Outliers

Here are the variables I found problems with after fixing the duplicate and missing values.

Age

The issue: It is unlikely there is an employee that is 12 or 96 years old. These should be deleted.

The solution:

```
# Filter df_cleaned to keep only the rows where Age is between 18 and 80
df_cleaned = df_cleaned[(df_cleaned['Age'] >= 18) & (df_cleaned['Age'] <= 80)]

# Reuse code from above to check if the age filtering worked
Age_counts = df_cleaned['Age'].value_counts()
print(Age_counts)
```

```
Age
48.0    252
21.0    248
57.0    247
28.0    245
60.0    242
26.0    241
41.0    240
56.0    237
31.0    237
39.0    237
50.0    234
55.0    234
49.0    232
33.0    232
24.0    231
22.0    228
46.0    228
51.0    228
23.0    227
29.0    225
27.0    225
18.0    223
34.0    223
58.0    221
54.0    220
45.0    217
19.0    216
25.0    216
35.0    216
42.0    216
43.0    215
38.0    212
30.0    212
40.0    211
53.0    211
36.0    211
47.0    210
37.0    209
52.0    207
59.0    205
20.0    200
32.0    191
44.0    183
Name: count, dtype: int64
```

As you can see, the problematic data has been removed.

BusinessTravel

The issue: Three entries are abnormal. 1, -1, and 00 do not make sense. These rows should be deleted.

The solution:

```
# Defining the unwanted values
unwanted_values = [1, -1, '00']

# Removing the rows where BusinessTravel is in the unwanted values
df_cleaned = df_cleaned[~df_cleaned['BusinessTravel'].isin(unwanted_values)]

#Reuse code from above to see if the abnormalities are still in the data
BusinessTravel_counts = df_cleaned['BusinessTravel'].value_counts()
print(BusinessTravel_counts)
```

```
BusinessTravel
Travel_Rarely        3250
Non-Travel           3197
Travel_Frequently    3145
Name: count, dtype: int64
```

As you can see, the problematic data has been removed.

DistanceFromHome

The issue: There are three very large commutes from home. Unless the CEO commutes by private jet, nobody is commuting 900 to 3,000 miles. Anything over 100 appears to be an outlier.

The solution:

```
# Remove rows where DistanceFromHome is greater than 100
df_cleaned = df_cleaned[df_cleaned['DistanceFromHome'] <= 100]

# Reuse code from above to see if the outliers were deleted
DistanceFromHome_counts = df_cleaned['DistanceFromHome'].value_counts()
print(DistanceFromHome_counts)
```

```
DistanceFromHome
1     216
5     212
30    210
3     206
29    206
7     205
34    205
9     204
41    203
14    201
16    201
43    201
11    200
37    199
25    198
18    198
36    198
32    198
12    198
13    197
44    195
27    195
19    193
10    192
48    192
22    192
33    189
38    189
49    189
40    188
26    188
4     187
42    187
24    186
28    186
50    185
39    184
6     184
17    184
15    183
2     183
46    181
21    180
23    180
45    180
47    179
31    175
20    175
8     170
35    162
Name: count, dtype: int64
```

As you can see, the problematic data has been removed.

EducationField

The issue: There's one blank entry that should be deleted to ensure uniformity in the data.

The solution:

```
# Remove rows where EducationField is an empty string
df_cleaned = df_cleaned[df_cleaned['EducationField'] != ' ']

# Use the same code from earlier to confirm the blank data is gone
EducationField_counts = df_cleaned['EducationField'].value_counts()
print(EducationField_counts)
```

```
EducationField
Marketing           1642
Medical             1624
Other               1606
Life Sciences       1598
Human Resources     1582
Technical Degree    1536
Name: count, dtype: int64
```

As you can see, the problematic data has been removed.

EmployeeCount

The issue: There are three rows that need to be fixed that do not have EmployeeCount = 1.

The solution:

```
#EmployeeCount should always be 1

#Looking for rows where EmployeeCount is not equal to 1
invalid_employee_count = df_cleaned[df_cleaned['EmployeeCount'] != 1]

#Display incorrect rows
print(invalid_employee_count)
```

```
#There are three rows that need to be fixed

# Set all EmployeeCount values to 1
df_cleaned['EmployeeCount'] = 1

#Run the same code from earlier to ensure there are no Employee Count not equal to 1
invalid_employee_count = df_cleaned[df_cleaned['EmployeeCount'] != 1]

#Display incorrect rows
print(invalid_employee_count)
```

```
Empty DataFrame
```

JobRole

The Issue: There are three blank entries that should be deleted to ensure uniformity in the data.

The solution:

```
# Remove rows where JobRole is an empty string
df_cleaned = df_cleaned[df_cleaned['JobRole'] != ' ']

# Do the same code from earlier to confirm the blank data is gone
JobRole_counts = df_cleaned['JobRole'].value_counts()

# Display the counts
print("JobRole:")
print(JobRole_counts)
```

```
JobRole:
JobRole
Manager                      1048
Research Director             985
Manufacturing Director        978
Healthcare Representative     961
Human Resources               952
Laboratory Technician         945
Sales Executive               944
Developer                     936
Sales Representative          921
Research Scientist            915
Name: count, dtype: int64
```

As you can see, the problematic data has been removed.

MonthlyIncome

The issue: Because monthly income can vary dramatically and can be unique or shared, the only numbers we need to be concerned about are negative numbers. There is one negative number that needs to be deleted because it cannot be reasonably guessed or replaced.

The solution:

```
# Filter df_cleaned to keep only rows where MonthlyIncome is positive
df_cleaned = df_cleaned[(df_cleaned['MonthlyIncome'] >= 0)]

#To be sure it worked, let's reuse our code for checking for negative values
# Count how many values in MonthlyIncome are less than 0
less_than_0 = df_cleaned[df_cleaned['MonthlyIncome'] < 0].shape[0]

print(f'Number of values in MonthlyIncome less than 0: {less_than_0}')
```

```
Number of values in MonthlyIncome greater than 0: 0
```

As you can see, the problematic data has been removed.

MonthlyRate

The issue: Need to delete the one outlier. Having a datapoint so large can skew our data.

The solution:

```
# Remove rows where MonthlyRate is greater than 2,000,000
df_cleaned = df_cleaned[df_cleaned['MonthlyRate'] <= 2000000]
```

```
#Reuse code from earlier to make sure the outlier was deleted

#Sort MonthlyRatein descending order and display the top 10 rows
top_10_MonthlyRate = df_cleaned[['MonthlyRate']].sort_values(by='MonthlyRate', ascending=False).head(10)

print(top_10_MonthlyRate)
```

```
      MonthlyRate
3466    1523280.0
8264    1522920.0
2190    1514100.0
4291    1496430.0
4048    1492170.0
1112    1491360.0
7497    1491120.0
4855    1484400.0
1056    1482120.0
7797    1478043.0
```

As you can see, the problematic data has been removed.

TotalWorkingYears

The issue: Clearly nobody has been working for -1 year or 220 years. These rows will be deleted because they cannot be accurately replaced.

The solution:

```
# Filter df_cleaned to keep only rows where TotalWorkingYears is between 0 and 100
df_cleaned = df_cleaned[(df_cleaned['TotalWorkingYears'] >= 0) & (df_cleaned['TotalWorkingYears'] <= 100)]

#Reuse code from above to see if the outliers are gone
TotalWorkingYears_counts = df_cleaned['TotalWorkingYears'].value_counts()
print(TotalWorkingYears_counts)
```

```
TotalWorkingYears
3.0     288
32.0    269
28.0    266
16.0    260
30.0    253
26.0    252
40.0    252
13.0    249
21.0    249
4.0     249
12.0    247
18.0    247
8.0     246
24.0    245
20.0    245
23.0    243
29.0    243
17.0    242
34.0    241
36.0    241
37.0    240
35.0    239
38.0    237
19.0    235
22.0    233
10.0    233
33.0    232
6.0     231
9.0     231
11.0    229
39.0    227
14.0    227
5.0     227
1.0     226
2.0     226
31.0    223
15.0    221
7.0     217
27.0    215
25.0    205
Name: count, dtype: int64
```

As you can see, the problematic data has been removed.

YearsWithCurrManager

The issue: Need to remove the last row. Working with a manager for -1000 years doesn't make sense.

The solution:

```
# Remove rows where YearsWithCurrManager is -1000
df_cleaned = df_cleaned[(df_cleaned['YearsWithCurrManager'] != 'na') & (df_cleaned['YearsWithCurrManager'] != -1000)]

YearsWithCurrManager_counts = df_cleaned['YearsWithCurrManager'].value_counts()
print(YearsWithCurrManager_counts)
```

```
YearsWithCurrManager
1     2430
2     1352
3      985
4      777
5      578
6      492
7      408
8      361
9      287
10     245
11     221
12     213
13     185
14     149
16     135
15     120
18      89
17      86
19      75
22      54
21      54
20      53
23      39
24      35
29      24
27      24
25      23
26      21
28      18
30      14
34       7
33       7
31       6
32       5
37       2
35       2
36       1
38       1
Name: count, dtype: int64
```

As you can see, the problematic data has been removed.

C3:TECHNIQUE ADVANTAGES

Because I chose to err on the side of deleting the data rather than replacing it with the mean or a null value:

1. The data is uniform and complete. This will make it easy to analyze and we will not have to deal with null or missing values when doing calculations and analysis, simplifying processes later.
2. The data does not contain any guesses or incorrect data. We can be certain every datapoint in the data set is accurate.

C4:TECHNIQUE LIMITATIONS

Because I chose to err on the side of deleting the data rather than replacing it with the mean or a null value:

1. There is less data overall. The missing data points might have changed our analysis if they instead were changed to the mean or a null value.
2. If there was a particular reason why some of the data had a problem or a missing value, that trend/population will be entirely absent from the analysis. This could create a bias in our data.

Sources

No sources were used besides WGU official course materials.