Data Cleaning and Profiling

Task 1

Nathan Hefner

A. Profile data by doing the following:

1. Review the data dictionary in the attached "Employee Turnover Considerations and Dictionary" document and do the following:

a. Describe the general characteristics of the initial dataset (e.g., rows, columns).

The dataset is comprised of 16 Columns (Employee Number, Age, Tenure, Turnover, Hourly Rate, Hours Weekly, Compensation Type, Annual Salary, Driving Commuter Distance, Job Role Area, Gender, Marital Status, Number of Companies Previously Worked, Annual Professional Development Hours, Paycheck Method, and Text Message Opt-In) and 10,200 rows. Employee Number ensures every employee has a unique identifier to remove the possibility of removing similar employees thought to be duplicated and reduce redundancy among the dataset. The rest of the columns provide a wide array of ways to identify unique traits for the employees to find any specific reasons for early turnover or those who stay at a company for an extended period. These datasets include integer, object, and float data types.

b. Indicate the data type and data subtype for each variable.

The two types of data in data science are categorical and numeric. The subtypes for each are nominal/ordinal and discrete/continuous.

1. Employee Number
    a. Type - Numeric
    b. Subtype - Discrete
2. Age
    a. Type - Numeric
    b. Subtype - Discrete
3. Tenure
    a. Type - Numeric
    b. Subtype - Discrete
4. Turnover
    a. Type - Categorical
    b. Subtype - Nominal
5. HourlyRate
    a. Type - Numeric
    b. Subtype - Continuous
6. HoursWeekly
    a. Type - Numeric
    b. Subtype - Discrete
7. CompensationType
    a. Type - Categorical
    b. Subtype - Nominal
8. AnnualSalary

    a. Type - Numeric
    b. Subtype - Continuous
9. DrivingCommuterDistance
    a. Type - Numeric
    b. Subtype - Discrete
10. JobRoleArea
    a. Type - Categorical
    b. Subtype - Nominal
11. Gender
    a. Type - Categorical
    b. Subtype - Nominal
12. MaritalStatus
    a. Type - Categorical
    b. Subtype - Nominal
13. NumCompaniesPreviouslyWorked
    a. Type - Numeric
    b. Subtype - Discrete
14. AnnualProfessionalDevHrs
    a. Type - Numeric
    b. Subtype - Discrete
15. PaycheckMethod
    a. Type - Categorical
    b. Subtype - Nominal
16. TextMessageOptIn
    a. Type - Categorical
    b. Subtype - Nominal

c. Provide a sample of observable values for each variable.

An observable value for each variable is as follows: Employee Number will be countable, starting at one and increasing incrementally for each new employee. Age would most likely be a number from the 18-65 range. Tenure will likely be a number from 1-20. Hourly Range will be a number with a decimal starting around 20.00 and can range all the way to 100.00. Hours Weekly is a whole number value, with 40 indicating full-time employees Compensation type is categorical data as it is just salary. Annual salary is a decimal number with a wide range, such as 45489.60 to 284662.40. Driving Commuter Distance is a whole number counting how many miles employees take to arrive at work. Job Role Area is categorical and depends on your department, such as Research, Laboratory, and Information Technology. Gender is also categorical with three answers: Male, Female, and prefer not to Answer. Marital Status also has 3 answers: Married, Single, or Divorced. Number of Companies Previously Worked is a whole number starting at 0 if this is your first job and ranging upwards for each job worked. Annual Professional Development Hours is the same as it's a whole number that starts at 0 and scales upwards. Turnover, Paycheck Method, and Text Message Opt-in are either yes or no boolean values, except the paycheck method, which is either direct deposit or mailed check.

B. Inspect the dataset through data cleaning techniques for all duplicate entries, missing values, inconsistent entries, formatting errors, and outliers and do the following:

1. **Explain how you inspected the dataset for each of the quality issues listed in part B.**
2. **List your findings for each quality issue listed in part B.**

To inspect the dataset, I must first import pandas and upload the file path so the code can correctly identify, read, and analyze the correct data document. I did that with the following code:

```
file_path =r"C:\Users\Nathan\Documents\WGU\D599\Employee Turnover Dataset.csv"
df = pd.read_csv(file_path)
```

I will now show how I inspected the data for each potential issue.

Duplicates

I will use the duplicated() function to count all complete duplicates and find how many duplicates there are. Below is the code used.

```
num_duplicates = df.duplicated().sum()
print(f'Number of Duplicate Rows: {num_duplicates}')
```

```
Number of Duplicate Rows: 99
```

This dataset has 99 duplicate rows to be deleted.

Missing Values

I will use the isnull() function to count all missing values within the dataset. Below is the code.

```
print(df.isnull().sum())
```

```
EmployeeNumber                  0
Age                             0
Tenure                          0
Turnover                        0
HourlyRate                      0
HoursWeekly                     0
CompensationType                0
AnnualSalary                    0
DrivingCommuterDistance         0
JobRoleArea                     0
Gender                          0
MaritalStatus                   0
NumCompaniesPreviouslyWorked  665
AnnualProfessionalDevHrs     1969
PaycheckMethod                  0
TextMessageOptIn             2266
dtype: int64
```

Three columns are missing a substantial amount of data.

Inconsistent Entries, Formatting Errors, Outliers

The simplest way to inspect each of these columns for quality issues would be to inspect every column one at a time. This will allow us to check for inconsistent entries by seeing which values do not fit with the rest, formatting errors to see which values were mistyped or are inconsistent, and outliers by seeing if any extreme values do not make sense. Please note I have not dropped the duplicate rows yet also, I skipped Employee Number as we already identified duplicates.

Age

```
age_counts = df['Age'].value_counts()
print(age_counts)
```

```
Age
39    448
37    428
36    409
38    386
40    382
44    322
43    317
48    315
46    306
56    304
41    300
42    300
58    297
47    297
54    296
60    296
61    293
59    292
53    291
45    284
57    281
49    278
51    278
52    267
50    267
55    245
32    218
30    211
34    206
33    192
35    188
31    186
22    103
24     97
21     93
26     91
27     90
29     89
25     89
23     85
28     82
```

This column appears to be clean as the lowest age is 21 and the oldest age is 61, which are both sensible.

Tenure

```
tenure_counts = df['Tenure'].value_counts()
print(tenure_counts)
```

```
Tenure
1     865
10    741
5     739
7     734
6     730
8     714
9     688
3     617
2     456
4     454
14    376
15    361
20    358
19    350
13    349
16    347
18    337
11    329
17    327
12    327
```

This column appears clean as every value is reasonable.

Turnover

```
turnover_counts = df['Turnover'].value_counts()
print(turnover_counts)
```

```
Turnover
No     5509
Yes    4690
Name: count, dtype: int64
```

This column is clean as the only values are Yes and No.

HourlyRate

For this column I first removed the $ sign to make it easier to analyze.

```
df['HourlyRate'] = df['HourlyRate'].replace({'\$': ''}, regex=True).astype(float)

print(df['HourlyRate'])
```

```
0        24.37
1        24.37
2        22.52
3        22.52
4        88.77
         ...
10194    85.40
10195    85.40
10196    71.90
10197    71.90
10198    71.33
```

I then check if the entire column is now numeric.

```
is_numeric = pd.to_numeric(df['HourlyRate'], errors='coerce').notna().all()

print(is_numeric)
```

```
True
```

I now check the values of the column to check for oddities and any outliers.

```
hourlyrate_counts = df['HourlyRate'].value_counts()
print(hourlyrate_counts)
```

```
HourlyRate
26.28    11
34.28    11
31.39    10
31.28    10
33.66    10
         ..
21.89     1
68.92     1
92.46     1
53.20     1
28.06     1
```

```
print(df['HourlyRate'].describe())
```

```
count    10199.000000
mean        52.792995
std         23.941940
min         17.210000
25%         30.955000
50%         48.830000
75%         73.980000
max         98.070000
```

```
negative_count = (df['HourlyRate'] < 0).sum()

print(f"Number of negative values: {negative_count}")
```

```
Number of negative values: 0
```

Data looks clean no need to change anything.

HoursWeekly

```
hoursweekly_counts = df['HoursWeekly'].value_counts()
print(hoursweekly_counts)
```

```
HoursWeekly
40    10199
```

There is no need for cleaning all values are the same.

CompensationType

```
compensationtype_counts = df['CompensationType'].value_counts()
print(compensationtype_counts)
```

```
CompensationType
Salary    10199
Name: count, dtype: int64
```

There is no need for cleaning all values are the same.

AnnualSalary

I first check if every value is numeric in this column.

```
is_numeric = pd.to_numeric(df['AnnualSalary'], errors='coerce').notna().all()

print(is_numeric)
```

```
True
```

Which they are. I then check the stats of this column.

```
print(df['AnnualSalary'].describe()
```

```
count     10199.000000
mean     120947.568526
std       77566.715759
min      -33326.400000
25%       63252.800000
50%      101566.400000
75%      153878.400000
max      339950.400000
Name: AnnualSalary, dtype: float64
```

There are negative values that certainly should not be for salary values. I investigate further to see how many values are inputted incorrectly.

```
negative_count = (df['AnnualSalary'] < 0).sum()

print(f"Number of negative values: {negative_count}")
```

```
Number of negative values: 57
```

There are 57 values that will need correcting.

DrivingCommuterDistance

```
pd.set_option('display.max_rows', None)

drivingcommuterdistance_counts = df['DrivingCommuterDistance'].value_counts()
print(drivingcommuterdistance_counts)
```

```
DrivingCommuterDistance
 33     190
 250    177
 28     161
 27     125
 42     115
 77     110
-5      104
 76     104
 89     104
 2      103
 87     102
 44     102
 17     101
 78     101
 64     100
 24     100
 35     100
```

34	89
52	89
5	88
32	88
16	88
43	88
53	87
-12	87
67	87
79	86
62	86
83	86
6	86
51	86
9	85
39	85
20	85
73	85
-2	85
26	85
69	85
66	83
59	83
45	83
37	83
61	82
15	82
58	82
3	82
38	82
63	82
-3	81
12	81
68	81
40	81
72	81
-1	81
65	81
-11	81
-4	80
4	80
11	78
-13	78
48	78
7	78
-9	78
49	77
30	77

```
 29      77
 14      76
 80      76
 82      76
 85      75
 81      75
 74      73
 21      72
 92      71
 97      69
 94      67
 23      67
 93      67
 99      66
 95      66
 90      62
 98      58
 96      57
 91      51
 322     22
-125     19
-275     17
 125     17
 910      8
 950      2
```

This is too much data to examine, so instead, we will count how many values are outside of an acceptable range (0-100).

```
negative_count = (df['DrivingCommuterDistance'] < 0).sum()

print(f"Number of negative values: {negative_count}")
```

```
Number of negative values: 1351
```

There are 1351 negative values that will need to be made positive.

```
high_value_count = (df['DrivingCommuterDistance'] > 100).sum()

print(f"Number of high values: {high_value_count}")
```

```
Number of high values: 226
```

These values will need to be changed as they're outliers, but we will worry about that in the data cleaning section.

JobRoleArea

```
jobrolearea_counts = df['JobRoleArea'].value_counts()
print(jobrolearea_counts)
```

```
JobRoleArea
Research                  2025
Sales                     2007
Marketing                 1105
Manufacturing             1039
Laboratory                1021
Healthcare                1008
Human Resources            909
Information Technology     857
InformationTechnology       80
Information_Technology      56
HumanResources              51
Human_Resources             41
```

This column will need to be cleaned as there are multiple variations of the same answers.

Gender

```
gender_counts = df['Gender'].value_counts()
print(gender_counts)
```

```
Gender
Female                 5812
Male                   4232
Prefer Not to Answer    155
```

This column is clean no need to change anything.

MaritalStatus

```
maritalstatus_counts = df['MaritalStatus'].value_counts()
print(maritalstatus_counts)
```

```
MaritalStatus
Married     3439
Single      3422
Divorced    3338
Name: count, dtype: int64
```

This column is clean no need to change anything.

NumCompaniesPreviouslyWorked

```
numcompaniespreviouslyworked_counts = df['NumCompaniesPreviouslyWorked'].value_counts()
print(numcompaniespreviouslyworked_counts)
```

```
NumCompaniesPreviouslyWorked
1.0    1517
2.0    1487
3.0    1464
6.0    1061
4.0    1021
5.0     994
7.0     687
8.0     655
9.0     648
```

Data looks clean no need to change anything, however this is one of the columns that have missing values, so we will need to find a solution for that later.

AnnualProfessionalDevHrs

```
annualprofessionaldevhrs_counts = df['AnnualProfessionalDevHrs'].value_counts()
print(annualprofessionaldevhrs_counts)
```

```
AnnualProfessionalDevHrs
23.0    431
10.0    423
25.0    413
14.0    413
13.0    410
6.0     410
5.0     399
11.0    399
7.0     393
12.0    393
9.0     393
22.0    388
18.0    387
15.0    384
19.0    384
17.0    379
8.0     376
24.0    373
20.0    362
16.0    361
21.0    359
```

Data looks clean no need to change anything, however this is one of the columns that have missing values, so we will need to find a solution for that later.

PaycheckMethod

```
paycheckmethod_counts = df['PaycheckMethod'].value_counts()
print(paycheckmethod_counts)
```

```
PaycheckMethod
Mail Check        4986
Mailed Check      2441
DirectDeposit      992
Direct_Deposit     958
Mail_Check         547
Direct Deposit     226
MailedCheck         49
```

This column has many variations of the same answers. That will need to be changed to improve consistency and reduce redundancy.

TextMessageOptIn

```
textmessageoptin_counts = df['TextMessageOptIn'].value_counts()
print(textmessageoptin_counts)
```

```
TextMessageOptIn
Yes    7390
No      543
```

Data looks clean no need to change anything, however this is one of the columns that have missing values, so we will need to find a solution for that later.

C. Discuss which data cleaning techniques you used to correct all the data quality issues you identified by doing the following:

1. **Describe how you modified the dataset after identifying each quality issue listed in part B.**
2. **Discuss why you chose the specific data cleaning techniques you used to clean the quality issues listed in part B.**

Duplicates

This can easily be achieved with the drop_duplicates() function. I chose this function because, luckily, the data has Employee Number, which acts as a unique identifier for every employee, meaning we will not need to deal with partial duplicates, therefore the function above will be the most efficient way of cleaning this data.

```
df_cleaned = df.drop_duplicates()
```

```
num_duplicates = df_cleaned.duplicated().sum()
print(f'Number of Duplicate Rows: {num_duplicates}')
```

```
Number of Duplicate Rows: 0
```

All duplicate rows dropped, leaving us with 10100 rows now.

Missing Values, Inconsistent Entries, Formatting Errors, Outliers

At this point, I will once again go through the columns one at a time to tackle their specific issues.

AnnualSalary

This column had negative values, which should not occur for salary values. I believe the negative was a misinput and should not be there therefore, all values should be positive. To do this, I use the abs() function on the column, eliminating all negative numbers.

```
df_cleaned = df_cleaned.copy()
df_cleaned['AnnualSalary'] = df_cleaned['AnnualSalary'].abs()
```

```
negative_count = (df_cleaned['AnnualSalary'] < 0).sum()

print(f"Number of negative values: {negative_count}")
```

```
Number of negative values: 0
```

DrivingCommuterDistance

This column had negative numbers and outliers, as anything over 100 seems to be too extreme of a realistic value.

First we will turn all values to positive.

```
df_cleaned = df_cleaned.copy()
df_cleaned['DrivingCommuterDistance'] = df_cleaned['DrivingCommuterDistance'].abs()
```

```
negative_count = (df_cleaned['DrivingCommuterDistance'] < 0).sum()

print(f"Number of negative values: {negative_count}")
```

```
Number of negative values: 0
```

Now, we will examine the dataset again using the describe function.

```
print(df_cleaned['DrivingCommuterDistance'].describe())
```

```
count    10100.000000
mean        48.646535
std         48.108708
min          0.000000
25%         15.000000
50%         42.000000
75%         71.000000
max        950.000000
```

This gives us valuable insights and shows some extreme values. We will dig deeper and use the unique function to find out more.

```
drivingcommuterdistance_unique = df_cleaned['DrivingCommuterDistance'].unique()
print(drivingcommuterdistance_unique)
```

```
[ 89  35  12   0  76  15   2  36  60  14  75   5 910  28   4  33  79  50
  13  57  82  42  64   8  77   1  24  67  41  62  47   7  31  32  56  22
  51  58  34  37  16  26  74  10  84  44  61  87  46  52  73  53  85  66
  80  55 950  70  17 250  43  49  81  86   3  48  45  68  20  38  21  11
  65  78  69   6  25  54  83  27   9  39  40  29  71  30  63  72 125  88
  59  23 275 322  99  91  94  97  93  92  90  96  95  98]
```

We see there are numerous values that seem extreme for commuting distance and nonsensical. We will use the 1.5 IQR rule for outliers outside of Q3 to clean this data. The rule is simple it follows this equation: Q3 + 1.5•IQR. We know the IQR by subtracting Q1 from Q3, which is 71 - 15 = 56 (We know these values from the describe() function we used earlier). We will use Python to solve the rest of the equation.

```
print((56*1.5)+71)
```

```
155.0
```

This value will be our maximum, and every value above that will be replaced with 155. We will use the apply function. I use it because it sets a rule for the entire column with the argument we provide.

```
df_cleaned['DrivingCommuterDistance'] = df_cleaned['DrivingCommuterDistance'].apply(lambda x: 155 if x > 155 else x)
```

```
drivingcommuterdistance_unique = df_cleaned['DrivingCommuterDistance'].unique()
print(drivingcommuterdistance_unique)
```

```
[ 89  35  12   0  76  15   2  36  60  14  75   5 155  28   4  33  79  50
  13  57  82  42  64   8  77   1  24  67  41  62  47   7  31  32  56  22
  51  58  34  37  16  26  74  10  84  44  61  87  46  52  73  53  85  66
  80  55  70  17  43  49  81  86   3  48  45  68  20  38  21  11  65  78
  69   6  25  54  83  27   9  39  40  29  71  30  63  72 125  88  59  23
  99  91  94  97  93  92  90  96  95  98]
```

We can now see every extreme outlier has been replaced with a more logical answer.

JobRoleArea

This column had inconsistent entries, specifically with Information Technology and Human Resources. They both had three different value types that should be collapsed into one. I will use the replace function to achieve this.

```
df_cleaned['JobRoleArea'] = df_cleaned['JobRoleArea'].replace({
    "InformationTechnology": "Information Technology",
    "Information_Technology": "Information Technology",
    "HumanResources": "Human Resources",
    "Human_Resources": "Human Resources"
})
```

```
jobrolearea_unique = df_cleaned['JobRoleArea'].unique()
print(jobrolearea_unique)
```

```
['Research' 'Information Technology' 'Sales' 'Human Resources'
 'Laboratory' 'Manufacturing' 'Healthcare' 'Marketing']
```

The replace function is helpful here as it can identify what needs to be replaced and what to replace it with.

After using the function, the results show there are no longer duplicate or inconsistent entries.

NumCompaniesPreviouslyWorked

This column has missing values labeled as “N/A” for this column in particular, I believe the “N/A” should be replaced with 0 as it’s possible for this job to be an employee’s first, there are no 0 values as of now, and it will make the column numeric therefore being more consistent and cleaner.

First, we will replace all “N/A” values with 0.

```
df_cleaned['NumCompaniesPreviouslyWorked'] = df_cleaned['NumCompaniesPreviouslyWorked'].replace(["N/A"], 0).fillna(0)
```

```
numcompaniespreviouslyworked_unique = df_cleaned['NumCompaniesPreviouslyWorked'].unique()
print(numcompaniespreviouslyworked_unique)
```

```
[3. 6. 1. 7. 0. 2. 5. 4. 8. 9.]
```

Using the replace() function once again, we can change those values to 0, therefore eliminating empty cells and creating a more consistent column.

AnnualProfessionalDevHrs

The same exact thought process is used for this column: there are no 0 values, which would make sense for the missing data cells, therefore we will do the same process to replace them with 0 instead.

```
df_cleaned['AnnualProfessionalDevHrs'] = df_cleaned['AnnualProfessionalDevHrs'].replace(["N/A"], 0).fillna(0)
```

```
annualprofessionaldevhrs_unique = df_cleaned['AnnualProfessionalDevHrs'].unique()
print(annualprofessionaldevhrs_unique)
```

```
[ 7.  8.  0. 19. 23. 25.  9.  6.  5. 15. 16. 24. 10. 20. 17. 21. 12. 22.
 11. 14. 13. 18.]
```

PaycheckMethod

This column has the same problems as JobRoleArea, so we will use the same process to rectify the data.

```
df_cleaned['PaycheckMethod'] = df_cleaned['PaycheckMethod'].replace({
    "Mail Check": "Mailed Check",
    "Mail_Check": "Mailed Check",
    "MailedCheck": "Mailed Check",
    "DirectDeposit": "Direct Deposit",
    "Direct_Deposit": "Direct Deposit"
})
```

```
paycheckmethod_unique = df_cleaned['PaycheckMethod'].unique()
print(paycheckmethod_unique)
```

```
['Mailed Check' 'Direct Deposit']
```

Now the data only has two entries.

TextMessageOptIn

This column also had missing values, since the data type is categorical we can first check which value has the highest count. I did this by using the value_counts() function.

```
textmessageoptin_unique = df_cleaned['TextMessageOptIn'].value_counts()
print(textmessageoptin_unique)
```

```
TextMessageOptIn
Yes    7299
No      543
Name: count, dtype: int64
```

We see that the answer Yes heavily outweighs the answer No, so we can safely impute the data by adding Yes to all the missing values.

I did this with the replace function, as it can detect empty cells and replace them with a new string text.

```
df_cleaned['TextMessageOptIn'] = df_cleaned['TextMessageOptIn'].replace('', 'Yes').fillna('Yes')
```

```
textmessageoptin_count = df_cleaned['TextMessageOptIn'].value_counts()
print(textmessageoptin_count)
```

```
TextMessageOptIn
Yes    9557
No      543
```

Now, every row has a value. We have officially cleaned this dataset.

3. Describe two or more advantages to your data cleaning approach specified in part C1.

One advantage of my data cleaning approach is that using df.drop duplicates() helps reduce redundancy and improve data integrity. These values are no longer counted twice, inflating their actual value and possibly tampering with the model's results for calculating or finding hidden trends.

Another advantage is using the replace function to standardize the values in the cells. This helps keep the data consistent and uniform and helps prevent mismatches, especially in number-only columns.

4. Discuss two or more limitations to your data cleaning approach specified in part C1.

One limitation of my approach is for distance commuting column I dropped values outside of my calculated range and replaced them with the maximum value (155). This may skew the data or introduce bias because we may have inadvertently eliminated values showing a hidden trend or reason for such high values.

Another limitation is that the df.drop_duplicates() may miss slight variations of duplicates. With this dataset, it was simple as we could use the unique identifier employee number, but this may not always be the case, and we may have to broaden our reach with partial duplicates.

Sources

No sources were used except for WGU material.