D599: Data Preparation and Exploration Task 1
Staphon Smith

Part I: Data Profiling

A1a: General Characteristics/Profile Data:
The Employee Turnover Dataset given contains 10,200 Rows and 16 Columns. The data types that are included are: Integers, Floats and Objects.

A1b: Variable Data Type and Subtype and A1c: Observable Values
I used the attribute .dtype to give me a list of the datatypes of each column and this was the result that was given:

```
EmployeeNumber                 int64
Age                            int64
Tenure                         int64
Turnover                       object
HourlyRate                     object
HoursWeekly                    int64
CompensationType               object
AnnualSalary                   float64
DrivingCommuterDistance        int64
JobRoleArea                    object
Gender                         object
MaritalStatus                  object
NumCompaniesPreviouslyWorked   float64
AnnualProfessionalDevHrs       float64
PaycheckMethod                 object
TextMessageOptIn               object
dtype: object
```

As defined by course materials/datacamp videos and the given above .dtype attributes, here are the datatypes, subtypes and first three observable values per column:

1. Employee Number
    a. Data type: Categorical - (This is a unique identifier and should be treated as a categorical variable.)
    b. Subtype: Identifier (Nominal)
    c. Observable Examples: 1, 2, 3
2. Age
    a. Data type: Numeric
    b. Subtype: Float64
    c. Observable Examples: 28,33,22
3. Tenure
    a. Data type: Numeric
    b. Subtype: Int64
    c. Observable Examples: 6,2,1
4. Turnover

    a. Data type: Text/String
    b. Subtype: Object
    c. Observable Examples: Yes, No
5. Hourly Rate
    a. Data type: Numeric
    b. Subtype: Float64
    c. Observable Examples: 24.37,22.52, 88.77
6. Hours Weekly
    a. Data type: Numeric
    b. Subtype: Int64
    c. Observable Examples:40
7. Compensation Type
    a. Data type:Text/String
    b. Subtype: Object
    c. Observable Examples: Salary
8. Annual Salary
    a. Data type: Numeric
    b. Subtype: Int64
    c. Observable Examples: ,50689.6,46841.6, 284641.6
9. Driving Commuter Distance
    a. Data type: Numeric
    b. Subtype: Int64
    c. Observable Examples: 89,35, 12
10. Job Role Area
    a. Data type:Text/String
    b. Subtype: Object
    c. Observable Examples: Research, Information Technology, Sales
11. Gender
    a. Data type:Text/String
    b. Subtype: Object
    c. Observable Examples: Female, Prefer Not to Answer, Male
12. Marital Status
    a. Data type:Text/String
    b. Subtype: Object
    c. Observable Examples:Married, Single, Divorced
13. Number of Companies Previously Worked
    a. Data type: Numeric
    b. Subtype: Float64
    c. Observable Examples: 3, 6, 1
14. Annual Professional Dev Hours
    a. Data type:Numeric
    b. Subtype: Float64
    c. Observable Examples: 7,8,N/A
15. Paycheck Method
    a. Data type:Text/String
    b. Subtype: Object
    c. Observable Examples: Mail Check, Mailed Check, Direct Deposit
16. Text Message Opt In
    a. Data type:Text/String

b. Subtype: Object
c. Observable Examples:Yes, N/A, No

Part II: Data Cleaning and Plan

B1: Dataset Quality Issues and B2: List of Quality Issues

Import data into Dataframe:

```
import pandas as pd

file_path = r"C:\Users\StaphonSmith\Desktop\Employee Turnover Dataset (1).csv"

df=pd.read_csv(file_path)
```

Issue #1 - Duplicate Entries

After importing the data into the data frame I started with the first issue, duplicate entries. I used this code below and found there were 99 Duplicate Rows.

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

```
number of duplicate rows: 99
```

Next, I printed the duplicate rows for quick inspection and then confirmed these 99 duplicates needed to be cleaned from the dataframe.

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

```
       EmployeeNumber  Age  Tenure Turnover HourlyRate   HoursWeekly  \
10100               1   28       6      Yes     $24.37            40
10101               2   33       2      Yes     $24.37            40
10102               3   22       1       No     $22.52            40
10103               4   23       1       No     $22.52            40
10104               5   40       6       No     $88.77            40
...               ...  ...     ...      ...        ...           ...
10194              95   48      13      Yes     $85.40            40
10195              96   54      17       No     $85.40            40
10196              97   44       6       No     $71.90            40
10197              98   58      19       No     $71.90            40
10198              99   48      17      Yes     $71.33            40

      CompensationType  AnnualSalary  DrivingCommuterDistance  \
10100           Salary       50689.6                       89
10101           Salary       50689.6                       89
10102           Salary       46841.6                       35
10103           Salary       46841.6                       35
10104           Salary      284641.6                       12
...                ...           ...                      ...
10194           Salary      177632.0                       31
10195           Salary      177632.0                       31
10196           Salary      149552.0                       32
10197           Salary      149552.0                       32
10198           Salary      148075.2                       50

                 JobRoleArea             Gender MaritalStatus  \
10100               Research             Female       Married
10101               Research             Female       Married

[99 rows x 16 columns]
```

Issue #2 - Missing Values

The next issue at hand was missing values. I used ".isnull" in my python code then summed up the totals to figure out which specific columns have missing values. Below is the code along with the data received.

```python
#look for missing data
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
```

3 of the 16 columns have more than one cell missing data, these will need to be evaluated and I will create a solution to resolve this.

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

To streamline the process, I decided to check for formatting errors, inconsistencies, and outliers all at once. By examining the data distribution, it became easier to spot unusual entries, incorrect formatting, or values that didn't align with expected patterns.You can then find outliers by looking for extreme values. Obvious errors like negative distances were removed. For high but plausible values (e.g., long commute distances or large salaries), I evaluated them with domain context. I retained those that seemed realistic and removed only the most extreme, likely erroneous entries to avoid skewing the analysis. Below is a breakdown of the 16 columns I reviewed, along with the methods I used to detect and address these issues. I cleaned the data column by column, so in some cases, fixing earlier columns helped resolve problems in the later ones.

In the following Subsections I will elaborate in detail on the Code used in python and my findings (Screenshots included):

Employee Number:

Code:

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

```
EmployeeNumber
1        1
6738     1
6731     1
6732     1
6733     1
        ..
3367     1
3368     1
3369     1
3370     1
10100    1
Name: count, Length: 10100, dtype: int64
```

Findings: Every Employee Number is matched properly and nothing to adjust.

Age:

Code:

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

```
Age
39    444
37    422
36    403
38    382
40    375
43    317
44    315
48    311
46    305
56    303
42    300
41    299
60    296
58    296
47    295
54    293
61    293
59    290
53    289
45    281
57    279
49    278
51    278
52    266
50    265
55    243
32    213
30    202
34    201
```

Findings: There were no abnormalities or outliers found in the age column.

Tenure:

Code:

```
Tenure_counts = df_cleaned['Tenure'].value_counts()
print(Tenure_counts)
```

```
Tenure
1     851
10    737
5     733
7     728
6     719
8     703
9     679
3     609
2     452
4     447
14    375
15    360
20    356
19    349
13    348
16    341
18    334
11    329
12    327
17    323
Name: count, dtype: int64
```

Findings: Tenure had no abnormalities.

Turnover:

Code:

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

```
Turnover
No     5456
Yes    4644
Name: count, dtype: int64
```

Findings: No abnormalities in Turnover.

Hourly Rate:

Code:

```
# I found there was an extra space after "Hourly Rate" so i used the code below so this will not happen again.
```

```
df_cleaned.columns = df_cleaned.columns.str.strip()
```

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

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

```
HourlyRate  Counts:
HourlyRate
$34.28     11
$31.28     10
$33.66     10
$28.83      9
$33.06      9
           ..
$28.37      1
$56.02      1
$89.43      1
$88.05      1
$93.05      1
Name: count, Length: 5244, dtype: int64
```

Findings: there was a space after HourlyRate so i used the code stated above to remove that and avoid any issues further. Other than that there were no outliers or unusual data points.

Hours Weekly:

Code:

```
#Check HoursWeekly for abnormalities
HoursWeekly_counts = df_cleaned['HoursWeekly'].value_counts()
print(HoursWeekly_counts)
```

```
HoursWeekly
40    10100
Name: count, dtype: int64
```

Findings: Very standard 40 hour work weeks were found with no abnormalities.

Compensation Type:

Code:

```
#CompensationType for Errors/abnormalities
CompensationType_counts = df_cleaned['CompensationType'].value_counts()
print(CompensationType_counts)
```

```
CompensationType
Salary    10100
Name: count, dtype: int64
```

Findings: No issues found with Compensation Type.

Annual Salary:

Code:

```
#AnnualSalary for abnormalities
AnnualSalary_counts = df_cleaned['AnnualSalary'].value_counts()
print(AnnualSalary_counts)
```

```
AnnualSalary
76294.4     9
64896.0     8
54350.4     7
53414.4     7
67392.0     7
           ..
49254.4     1
156707.2    1
37086.4     1
202571.2    1
333544.0    1
Name: count, Length: 5538, dtype: int64
```

Findings: No issues with Annual Salary, given a normal company has wide ranges in pay for certain employees.

Driving Commuting Distance:

Code:

```
#Check DrivingCommuterDistance for abnormalities
DrivingCommuterDistance_counts = df_cleaned['DrivingCommuterDistance'].value_counts()
print(DrivingCommuterDistance_counts)
```

```
DrivingCommuterDistance
 33     184
 250    177
 28     157
 27     125
 42     113
       ...
-125     19
-275     17
 125     17
 910      4
 950      2
Name: count, Length: 120, dtype: int64
```

```
#there are a lot of outliers and commutes that are unfeasible. i would have taken out 250+ miles however there is many pec
```

```
# Remove rows where DrivingCommuterDistance is greater than 250 and less than 0
df_cleaned = df_cleaned[(df_cleaned['DrivingCommuterDistance'] <= 250) & (df_cleaned['DrivingCommuterDistance'] >= 0)]
```


```
# Remove rows where DrivingCommuterDistance is greater than 250 and less than 0
df_cleaned = df_cleaned[(df_cleaned['DrivingCommuterDistance'] <= 250) & (df_cleaned['DrivingCommuterDistance'] >= 0)]
```


```
#Reused prior code to see if outliers have been deleted.
DrivingCommuterDistance_counts = df_cleaned['DrivingCommuterDistance'].value_counts()
print(DrivingCommuterDistance_counts)
```

```
DrivingCommuterDistance
33     184
250    177
28     157
27     125
42     113
      ...
90      62
98      58
96      57
91      51
125     17
Name: count, Length: 100, dtype: int64
```

Findings: Found there were a few outliers that had negative distances Obvious errors like negative distances were removed. For high but plausible values (e.g., long commute distances or large salaries), I evaluated them with domain context. I retained those that seemed realistic and removed only the most extreme, likely erroneous entries to avoid skewing the analysis.

<u>Job Role Area:</u>

Code:

```
#Listing unique entries in JobRoleArea
JobRoleArea_counts = df_cleaned['JobRoleArea'].value_counts()

# Display the counts
print("JobRoleArea:")
print(JobRoleArea_counts)
```

```
JobRoleArea:
JobRoleArea
Research                  1733
Sales                     1718
Marketing                  943
Manufacturing              895
Healthcare                 890
Laboratory                 870
Human Resources            765
Information Technology     736
InformationTechnology       73
HumanResources              43
Information_Technology      36
Human_Resources             27
Name: count, dtype: int64
```

Findings: Nothing abnormal.

Gender:
Code:

```
#Listing unique entries in Gender
Gender_counts = df_cleaned['Gender'].value_counts()

# Display the counts
print(Gender_counts)
```

```
Gender
Female                  4964
Male                    3640
Prefer Not to Answer     125
Name: count, dtype: int64
```

Findings: Nothing Abnormal.

Marital Status:
Code:

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

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

```
MaritalStatus:
MaritalStatus
Single      2939
Married     2924
Divorced    2866
Name: count, dtype: int64
```

Findings: Nothing abnormal and no outliers.

Number of Companies Previously Worked:

Code:

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

```
NumCompaniesPreviouslyWorked:
NumCompaniesPreviouslyWorked
1.0    1289
2.0    1263
3.0    1259
6.0     914
4.0     873
5.0     846
7.0     588
8.0     565
9.0     561
Name: count, dtype: int64
```

Findings: Nothing Abnormal.

Annual Prof. Dev Hours:

Code:

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

```
AnnualProfessionalDevHrs:
AnnualProfessionalDevHrs
14.0    368
10.0    368
23.0    365
25.0    363
13.0    352
5.0     346
6.0     344
22.0    340
12.0    339
9.0     334
19.0    332
7.0     330
18.0    329
15.0    329
17.0    326
11.0    325
8.0     324
24.0    317
21.0    314
20.0    310
16.0    305
Name: count, dtype: int64
```

Findings: Nothing Abnormal.

Pay Check Method:

Code:

```
PaycheckMethod_counts = df_cleaned['PaycheckMethod'].value_counts()
print("PaycheckMethod:")
print(PaycheckMethod_counts)
```

```
PaycheckMethod:
PaycheckMethod
Mail Check         4248
Mailed Check       2103
DirectDeposit       861
Direct_Deposit      799
Mail_Check          482
Direct Deposit      193
MailedCheck          43
Name: count, dtype: int64
```

```
#there is some errors in just putting data into two categories so I will fix that below.
```

```python
# Clean and standardize the PaycheckMethod column
df_cleaned.loc[:, 'PaycheckMethod'] = df_cleaned['PaycheckMethod'].replace({
    'direct deposit': 'Direct Deposit',
    'check': 'Check'
})


# Map cleaned entries to standard categories
df_cleaned.loc[:, 'PaycheckMethod'] = df_cleaned['PaycheckMethod'].replace({
    'direct deposit': 'Direct Deposit',
    'check': 'Check',
    'Check': 'Check',
    'Direct deposit': 'Direct Deposit'
})

# Check result
print(df_cleaned['PaycheckMethod'].value_counts())
```

```
PaycheckMethod
Mailed Check      6876
Direct Deposit    1853
Name: count, dtype: int64
```

Findings: There were some issues with the spacing however no real issues. To clean up the data a bit and get rid of inconsistencies I grouped them properly into “Mailed Check” and “Direct Deposit” as shown above.

Text Messages Opt In:

Code:

```python
TextMessageOptIn_counts = df_cleaned['TextMessageOptIn'].value_counts()
print("TextMessageOptIn:")
print(TextMessageOptIn_counts)
```

```
TextMessageOptIn:
TextMessageOptIn
Yes    6285
No      455
Name: count, dtype: int64
```

Findings: Nothing Abnormal.

C1:DATASET MODIFICATION and C2:DATA CLEANING TECHNIQUES

I will outline the issues identified in the dataset, the code used to address them, and the verification steps taken to confirm the corrections. Additionally, I will explain the importance and necessity of each cleaning method applied

Quality Issue #1 - Duplicate Entries

Problem: There were 99 duplicate rows to delete. Because they are duplicates, they are not useful.

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

Problem: 3 of the 16 columns have at least one cell of missing data.

The solution: Since the 3 columns that are missing values are plausible to have a 0 as an accurate measuring and to keep the bulk of that data i chose to fill the N/A’s with 0 to maintain the data. As for Text Message Opt In i chose to change the N/A’s to no as that would be a default answer.

```python
# Fill missing values with 0 for numeric columns
df['NumCompaniesPreviouslyWorked'] = df['NumCompaniesPreviouslyWorked'].fillna(0)
df['AnnualProfessionalDevHrs'] = df['AnnualProfessionalDevHrs'].fillna(0)

# Fill missing values with 'No' for categorical column
df['TextMessageOptIn'] = df['TextMessageOptIn'].fillna('No')

# Confirm no missing values remain
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    0
AnnualProfessionalDevHrs        0
PaycheckMethod                  0
TextMessageOptIn                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: 10199 rows
Cleaned DataFrame: 10100 rows
```

We kept 10,100 rows out of 10,199 after changing the 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.

Hourly Rate:

Problem: There was an extra space that was causing formatting errors.

Solution:

```
# I found there was an extra space after "Hourly Rate" so i used the code below so this will not happen aga
```

```
df_cleaned.columns = df_cleaned.columns.str.strip()
```

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

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

```
HourlyRate  Counts:
HourlyRate
$34.28     11
$31.28     10
$33.66     10
$28.83      9
$33.06      9
           ..
$28.37      1
$56.02      1
$89.43      1
$88.05      1
$93.05      1
Name: count, Length: 5244, dtype: int64
```

Problematic data has been resolved.

Driving Commuter Distance:

Problem: there were outliers in this section where people were commuting over 250+ miles for work.also some negative mileage that would be impossible were also a problem.

Solution:

```
#Check DrivingCommuterDistance for abnormalities
DrivingCommuterDistance_counts = df_cleaned['DrivingCommuterDistance'].value_counts()
print(DrivingCommuterDistance_counts)
```

```
DrivingCommuterDistance
 33     184
 250    177
 28     157
 27     125
 42     113
       ...
-125     19
-275     17
 125     17
 910      4
 950      2
Name: count, Length: 120, dtype: int64
```

```
#there are a lot of outliers and commutes that are unfeasible. i would have taken out 250+ miles however there is many people who c
```

```
# Remove rows where DrivingCommuterDistance is greater than 250 and less than 0
df_cleaned = df_cleaned[(df_cleaned['DrivingCommuterDistance'] <= 250) & (df_cleaned['DrivingCommuterDistance'] >= 0)]
```

I chose to clean those outliers from the data frame. Typically I would think 250 miles is quite far as well but didn't really qualify as an outlier as there were 177 others that also do that same commute.

```
#Reused prior code to see if outliers have been deleted.
DrivingCommuterDistance_counts = df_cleaned['DrivingCommuterDistance'].value_counts()
print(DrivingCommuterDistance_counts)
```

```
DrivingCommuterDistance
33     184
250    177
28     157
27     125
42     113
      ...
90      62
98      58
96      57
91      51
125     17
Name: count, Length: 100, dtype: int64
```

Problematic data has been resolved.

Paycheck Method

Problem: There were Inconsistent entries/formatting errors for the paycheck method.

Solution:

```
PaycheckMethod_counts = df_cleaned['PaycheckMethod'].value_counts()
print("PaycheckMethod:")
print(PaycheckMethod_counts)
```

```
PaycheckMethod:
PaycheckMethod
Mail Check        4248
Mailed Check      2103
DirectDeposit      861
Direct_Deposit     799
Mail_Check         482
Direct Deposit     193
MailedCheck         43
Name: count, dtype: int64
```

I chose to group them into 2 sections: “Direct Deposit” and “Mailed Check” for simplicity and better congruence.

```python
# Clean and standardize the PaycheckMethod column
df_cleaned.loc[:, 'PaycheckMethod'] = df_cleaned['PaycheckMethod'].replace({
    'direct deposit': 'Direct Deposit',
    'check': 'Check'
})


# Map cleaned entries to standard categories
df_cleaned.loc[:, 'PaycheckMethod'] = df_cleaned['PaycheckMethod'].replace({
    'direct deposit': 'Direct Deposit',
    'check': 'Check',
    'Check': 'Check',
    'Direct deposit': 'Direct Deposit'
})

# Check result
print(df_cleaned['PaycheckMethod'].value_counts())
```

```
PaycheckMethod
Mailed Check      6876
Direct Deposit    1853
Name: count, dtype: int64
```

Problematic data has been Resolved.

C3: TECHNIQUE ADVANTAGES

I decided to remove the incomplete data instead of filling it in with averages or placeholders like null values. This approach ensures that the dataset is clean and consistent, which makes it easier to work with during analysis. We won't run into issues caused by missing or estimated values, and we can trust that all the information we're using is accurate and not based on assumptions.

C4:TECHNIQUE LIMITATIONS

By choosing to delete the missing data rather than substitute it with average values or nulls, we ended up with a smaller dataset. This might affect the overall outcome of our analysis, as those removed data points could have influenced the results. Additionally, if the missing values followed a specific pattern or were concentrated in a certain group, eliminating them entirely might introduce bias and leave out important trends.

D1-4: PANOPTO VIDEO & DOCUMENTS

These will be submitted together.

E: Sources

No sources were used besides WGU official course materials