Robert Pavlik

D599 Data Preparation and Exploration

TCN1 – Task 1: Data Cleaning and Profiling

Part I: Data Profiling

1. 1.
    a. The general characteristics of the data set are 10322 rows and 35 columns

```
import pandas as pd
✓ [234] < 10 ms

import numpy as np
✓ [235] < 10 ms

df = pd.read_excel('EmpTurnoverDS.xlsx')
✓ [236] 1s 502ms

df.shape
✓ [237] < 10 ms
 (10322, 35)
```

    b. The data types and subtypes for each column are in the table below.

COLUMN	DATA TYPE	DATA SUBTYPE
Age	float64	Continuous
Turnover	object	Categorical
DailyRate	object	Categorical
Department	int64	Discrete
DistanceFromHome	object	Categorical
Education	int64	Discrete
EducationField	int64	Discrete
EmployeeCount	object	Categorical
EmployeeNumber	int64	Discrete
EnviromentSatisfaction	int64	Discrete
Gender	int64	Discrete
HourlyRate	object	Categorical
JobInvolvement	int64	Discrete
JobLevel	int64	Discrete
JobRole	int64	Discrete
JobSatisfaction	object	Categorical
MaritalStatus	int64	Discrete
MonthlyIncome	object	Discrete
MonthlyRate	float64	Continuous
NumCompaniesWorked	float64	Continuous
Over18	float64	Continuous
OverTime	object	Categorical
PercentSalaryHike	object	Categorical
PerformanceRating	int64	Discrete

RelationshipSatisfaction	int64	Discrete
StandardHours	int64	Discrete
StockOptionLevel	int64	Discrete
TotalWorkingYears	int64	Discrete
TrainingTimesLastYear	float64	Continuous
WorkLifeBalance	float64	Continuous
YearsAtCompany	int64	Discrete
YearsInCurrentRole	int64	Discrete
YearsSinceLastPromotion	int64	Discrete
YearsWithCurrManager	float64	Continuous

c. Sample of Observable Data

VARIABLE	SAMPLE VALUES
Age	36.0, 24.0, 59.0
Turnover	Yes, No
BusinessTravel	Non-Travel, Travel_Rarley
DailyRate	1486, 963, 519
Department	Hardware, Support, Software
DistanceFromHome	26, 3, 50
Education	4, 1, 3
EducationField	Medical, Technical Degree, Other
EmployeeCount	1
EmployeeNumber	1253, 4494, 5958
EnviromentSatisfaction	4, 2, 3
Gender	Male, Female
HourlyRate	147, 40, 105
JobInvolvement	4, 1, 3
JobLevel	3, 1, 2
JobRole	Manager, Human Resources
JobSatisfaction	1, 2, 4
MaritalStatus	Divorced, Married, Single
MonthlyIncome	29479.0, 43656.0, 20317.0
MonthlyRate	235832.0, 1047744.0, 115815.0
NumCompaniesWorked	1.0, 3.0, 6.0
Over18	Y
OverTime	Yes, No
PercentSalarayHike	47, 48, 0
PerformanceRating	1, 4, 3
RelationshipSatisfaction	1, 4, 3
StandardHours	80
StockOptionLevel	1, 2, 3
TotalWorkingYears	35.0, 5.0, 10.0
TrainingTimesLastYear	4.0, 1.0, 3.0
WorkLifeBalance	3, 2, 1
YearsAtCompany	27, 1, 5
YearsInCurrentRole	6, 10, 13
YearsSinceLastPromotion	8.0, 3.0, 12.0
YearsWithCurrManager	5, 1, 4

```
df.dtypes
```

[240] < 10 ms

35 rows Length: 35, dtype:

	<unnamed>
Age	float64
Turnover	object
BusinessTravel	object
DailyRate	int64
Department	object
DistanceFromHome	int64
Education	int64
EducationField	object
EmployeeCount	int64
EmployeeNumber	int64
EnvironmentSatisfac…	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
RelationshipSatisfa…	int64
StandardHours	int64
StockOptionLevel	int64
TotalWorkingYears	float64
TrainingTimesLastYe…	float64
WorkLifeBalance	int64
YearsAtCompany	int64
YearsInCurrentRole	int64
YearsSinceLastPromo…	float64
YearsWithCurrManager	object

```
df.head()
```

[238] < 10 ms

5 rows 5 rows × 35 columns

	Age	Turnover	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	Gender	HourlyRate	Jo
0	33.0	Yes	Non-Travel	241	Hardware	16	3	Technical Degree	1	3505	1	Female	67	
1	35.0	Yes	Non-Travel	679	Support	7	2	Life Sciences	1	1129	3	Male	122	
2	27.0	Yes	Travel_Frequently	359	Hardware	50	1	Life Sciences	1	6305	4	Female	199	
3	44.0	No	Travel_Rarely	1133	Software	12	5	Life Sciences	1	4595	2	Female	150	
4	56.0	No	Travel_Rarely	118	Software	43	2	Human Resources	1	7203	2	Female	115	

Part II: Data Cleaning and Plan

B.

1. How the Data was inspected for each quality issue

- Duplicate Entries:

    - To check for duplicate entries I used the .duplicated().sum() method to give me a count of all the rows that were a complete duplicate of at least one other row.

    - Then I used the df[df.duplicated()] command to see those rows that were duplicates. Knowing that there was an EmployeeNumber column and if the row was a complete duplicate of another row, then it must have been a data error and it needed to be deleted. So I used the df.drop_duplicates(subset=[‘EmployeeNumber’], keep = ‘first’) command. This allowed me to ensure that the employee number of the duplicate row matched at least one other row and had the same employee number. The keep first ending ensures that the data row's first and original instance is kept and not deleted. After that, I ran .shape to see that rows were dropped.

```
df.duplicated().sum()
✓ [241] < 10 ms
np.int64(298)

df[df.duplicated()]
✓ [242] 16ms
```

1-10 298 rows × 35 columns

	Age	Turnover	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	Gender	HourlyRate
712	22.0	No	Travel_Rarely	611	Sales	33	2	Other	1	416	1	Female	11
985	42.0	No	Non-Travel	568	Sales	50	1	Life Sciences	1	9921	3	Female	12
1022	28.0	Yes	Travel_Rarely	406	Research & Development	38	1	Technical Degree	1	5528	4	Male	11
1033	29.0	Yes	Non-Travel	492	Support	49	5	Marketing	1	5587	4	Male	11
1212	41.0	Yes	Travel_Rarely	711	Hardware	49	5	Human Resources	1	9892	4	Male	7
1253	22.0	Yes	Travel_Frequently	1498	Research & Development	41	1	Medical	1	9919	1	Female	13
1256	36.0	Yes	Travel_Frequently	565	Hardware	9	5	Other	1	9872	3	Male	9
1424	18.0	Yes	Travel_Rarely	392	Support	45	3	Human Resources	1	5563	1	Male	16
1469	42.0	Yes	Travel_Frequently	818	Support	14	5	Medical	1	5569	2	Female	16
1512	35.0	Yes	Travel_Rarely	1400	Human Resources	25	4	Life Sciences	1	50	3	Female	15

```
df = df.drop_duplicates(subset=['EmployeeNumber'], keep='first')
✓ [243] < 10 ms

df.shape
✓ [244] < 10 ms
(9999, 35)
```

- Missing Values

    - To see how many missing values there were in the data I used the command ‘df.isnull().sum()[df.isnull().sum() > 0] ‘ that way I could just see a list of the columns that had missing values of 1 or greater. Looking at the data from that I noticed I could easily fix those issues by just taking the datatype of those columns and substituting a 0.0 for Float columns, and a ‘Unknown’ for basic object columns.

    - Then it came down to missing data in the columns that I felt just couldn’t be replaced with a 0. So, for those columns, I decided to use an interpolate method to fill in the missing data. Which is a method used to estimate missing values based on existing data points.

```
df.isnull().sum()[df.isnull().sum() > 0]
✓ [245] < 10 ms
```

8 rows Length: 8, dtype: int64

	<unnamed>
EducationField	1
Gender	3
MonthlyIncome	1
MonthlyRate	2
NumCompaniesWorked	1
TotalWorkingYears	1
TrainingTimesLastYear	410
YearsSinceLastPromotion	2

```
df.loc[:, 'TrainingTimesLastYear'] = df['TrainingTimesLastYear'].fillna(0.0)
df.loc[:, 'NumCompaniesWorked'] = df['NumCompaniesWorked'].fillna(0.0)
✓ [246] < 10 ms
```

```
df.loc[:, 'EducationField'] = df['EducationField'].fillna('Unknown')
df.loc[:, 'Gender'] = df['Gender'].fillna('Unknown')
✓ [247] < 10 ms
```

```
df.loc[:, 'MonthlyIncome'] = df['MonthlyIncome'].interpolate()
df.loc[:, 'MonthlyRate'] = df['MonthlyRate'].interpolate()
df.loc[:, 'TotalWorkingYears'] = df['TotalWorkingYears'].interpolate()
df.loc[:, 'YearsSinceLastPromotion'] = df['YearsSinceLastPromotion'].interpolate()
✓ [248] < 10 ms
```

```
df.isnull().sum()[df.isnull().sum() > 0]
✓ [249] < 10 ms
 Series([], dtype: int64)
```

- Inconsistent Entries and Formatting Errors

  - For inconsistent and Formatting Errors, I started by iterating through the columns and listing all the unique values that each column has (The ones that were 'object'). This gave me a greater overview of what wording each column used and I could easily see any errors. For three of the columns, I decided to use a dictionary replace method to correct the errors, for example, replacing 1 and -1 with 'Unknown'. With this method, I could easily take all the errors I didn't want and replace them with wording that worked better for the dataset. Then there was a column that was listed as an object but should have been a numeric value. So, I had to first take any object values and replace 'na' with NaN using a method in NumPy. Then I was able to convert that column to numeric using Panadas, and then convert any value less than 0 to NaN. Finally, I set this column to the float data type.

```python
inconsistent_entries = {}
for column in df.select_dtypes(include=['object']).columns:
    unique_values = df[column].unique()
    inconsistent_entries[column] = df[column].unique()
    print(f"{column}: {(unique_values)}")
```
✓ [250] < 10 ms

```
Turnover: ['Yes' 'No']
BusinessTravel: ['Non-Travel' 'Travel_Frequently' 'Travel_Rarely' 1 -1 '00' ' ']
Department: ['Hardware' 'Support' 'Software' 'Human Resources' 'Sales'
 'Research & Development']
EducationField: ['Technical Degree' 'Life Sciences' 'Human Resources' 'Other' 'Medical'
 'Marketing' ' ' 'Unknown']
Gender: ['Female' 'Male' 'Unknown']
JobRole: ['Manufacturing Director' 'Research Director' 'Sales Representative'
 'Developer' 'Laboratory Technician' 'Human Resources'
 'Healthcare Representative' 'Research Scientist' 'Manager'
 'Sales Executive' ' ']
MaritalStatus: ['Married' 'Single' 'Divorced']
Over18: ['Y']
OverTime: ['Yes' 'No']
YearsWithCurrManager: [11 4 2 1 7 3 8 12 6 16 9 5 27 13 21 22 18 29 15 14 10 19 20 23 24 17 30
 28 31 26 34 25 36 32 37 33 'na' 38 35 -1000]
```

```python
df.loc[:, 'BusinessTravel'] = df['BusinessTravel'].replace({1: 'Unknown', -1: 'Unknown', '00': 'Unknown', ' ': 'Unknown'})
df.loc[:, 'EducationField'] = df['EducationField'].replace({' ': 'Unknown'})
df.loc[:, 'JobRole'] = df['JobRole'].replace({' ': 'Unknown'})
```
✓ [251] < 10 ms

```python
df.loc[df['YearsWithCurrManager'] == 'na', 'YearsWithCurrManager'] = np.nan
```
✓ [252] < 10 ms

```python
df.loc[:, 'YearsWithCurrManager'] = pd.to_numeric(df['YearsWithCurrManager'], errors='coerce')
```
✓ [253] < 10 ms

```python
df.loc[df['YearsWithCurrManager'] < 0, 'YearsWithCurrManager'] = np.nan
```
✓ [254] < 10 ms

```python
df.loc[:, 'YearsWithCurrManager'] = df['YearsWithCurrManager'].astype(float)
```
✓ [255] < 10 ms

- Outliers

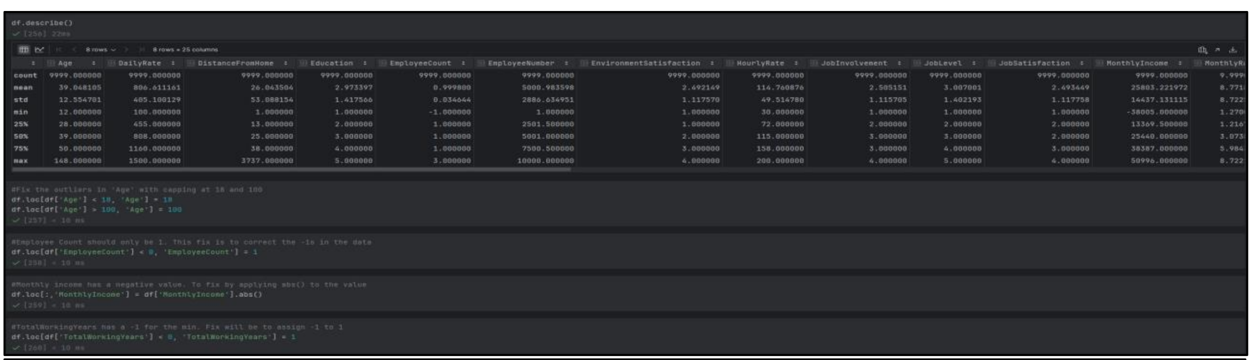
    - The next objective was to identify and deal with outliers. I started with using the '.describe method'. This gave me a statistical look at each column. Listing the mean, std, min, 25%, 50%, 75%, and max values for the data. From there I took into account the min and max values for each column to see if I could notice anything that stood out. The first thing I came to was the 'Age' column. It looked to have some outliers, min was 12 and max was 140. Common sense would tell you that no one under the age of 18 was working for this company and 140 is a little to old. So I just simply set anything less than 18 to 18 and anything over 100 to 100.

    - Next the column Employee Count which should just be a value of 1 had a min of -1 and a max of 1. So, to fix this I just replaced all values less than 0 with a 1.

    - Monthly Income had a min that was negative. This appeared to just be a formatting error. So, to correct this I just used the absolute value method on the values in the column to remove any negatives.

    - Total Years appeared to have a negative value as well. So again, I just replaced all values less than 0 with 1.

    - Lastly, the biggest challenge of the outliers was the 'MonthlyRate' column it was showing a its values in scientific notation, meaning there was a huge outlier in the data. First, I looked at what the max value was for the column ('872,214,872,214'). Assuming this wasn't what someone would make a month, I decided to check for any more large values by iterating through the column and listing values over 2000000. I noticed two values were way over that value. So, to fix this I decided to create a variable that took the mean of all the values in the column except for the two noted outliers. I then set those two outlier values to the mean of the rest of the column. This ensured that if that data is used later in analysis no unreal values should skew the data.



```
df.describe()
✓ [256] 22ms
```

8 rows × 25 columns

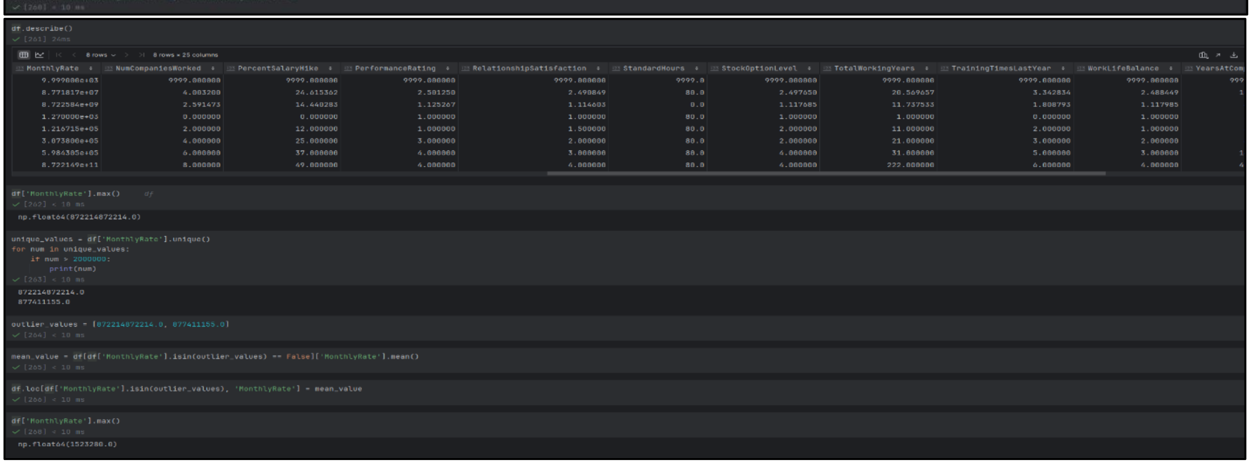
	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	HourlyRate	JobInvolvement	JobLevel	JobSatisfaction	MonthlyIncome
count	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000
mean	39.048105	806.611161	26.043504	2.973397	0.999800	5000.983598	2.492149	114.760876	2.505151	3.007001	2.493449	25803.221972
std	12.554781	405.100129	53.888154	1.417566	0.034644	2886.634951	1.117570	49.514780	1.115705	1.402193	1.117758	14437.131115
min	12.000000	100.000000	1.000000	1.000000	-1.000000	1.000000	1.000000	30.000000	1.000000	1.000000	1.000000	-38005.000000
25%	28.000000	455.000000	13.000000	2.000000	1.000000	2501.500000	1.000000	72.000000	2.000000	2.000000	2.000000	13369.500000
50%	39.000000	808.000000	25.000000	3.000000	1.000000	5001.000000	2.000000	115.000000	3.000000	3.000000	2.000000	25440.000000
75%	50.000000	1160.000000	38.000000	4.000000	1.000000	7500.500000	3.000000	158.000000	3.000000	4.000000	3.000000	38387.000000
max	148.000000	1500.000000	3737.000000	5.000000	3.000000	10000.000000	4.000000	200.000000	4.000000	5.000000	4.000000	50996.000000

```
#Fix the outliers in 'Age' with capping at 18 and 100
df.loc[df['Age'] < 18, 'Age'] = 18
df.loc[df['Age'] > 100, 'Age'] = 100
✓ [257] < 10 ms

#Employee Count should only be 1. This fix is to correct the -1s in the data
df.loc[df['EmployeeCount'] < 0, 'EmployeeCount'] = 1
✓ [258] < 10 ms

#Monthly income has a negative value. To fix by applying abs() to the value
df.loc[:,'MonthlyIncome'] = df['MonthlyIncome'].abs()
✓ [259] < 10 ms

#TotalWorkingYears has a -1 for the min. Fix will be to assign -1 to 1
df.loc[df['TotalWorkingYears'] < 0, 'TotalWorkingYears'] = 1
✓ [260] < 10 ms
```



```
df.describe()
✓ [261] 24ms
```

8 rows × 25 columns

MonthlyRate	NumCompaniesWorked	PercentSalaryHike	PerformanceRating	RelationshipSatisfaction	StandardHours	StockOptionLevel	TotalWorkingYears	TrainingTimesLastYear	WorkLifeBalance
9.999000e+03	9999.000000	9999.000000	9999.000000	9999.000000	9999.0	9999.000000	9999.000000	9999.000000	9999.000000
8.771817e+07	4.003200	24.615362	2.501250	2.498849	80.0	2.497650	20.569657	3.342834	2.488449
8.722584e+09	2.591473	14.440283	1.125267	1.114603	0.0	1.117685	11.737533	1.808793	1.117985
1.270000e+03	0.000000	0.000000	1.000000	1.000000	80.0	1.000000	1.000000	0.000000	1.000000
1.216715e+05	2.000000	12.000000	1.000000	1.500000	80.0	2.000000	11.000000	2.000000	1.000000
3.073800e+05	4.000000	25.000000	3.000000	2.000000	80.0	2.000000	21.000000	3.000000	2.000000
5.984305e+05	6.000000	37.000000	4.000000	3.000000	80.0	4.000000	31.000000	5.000000	3.000000
8.722149e+11	8.000000	49.000000	4.000000	4.000000	80.0	4.000000	222.000000	6.000000	4.000000

```
df['MonthlyRate'].max()
✓ [262] < 10 ms
np.float64(872214872214.0)

unique_values = df['MonthlyRate'].unique()
for num in unique_values:
    if num > 2000000:
        print(num)
✓ [263] < 10 ms
872214872214.0
877411155.0

outlier_values = [872214872214.0, 877411155.0]
✓ [264] < 10 ms

mean_value = df[df['MonthlyRate'].isin(outlier_values) == False]['MonthlyRate'].mean()
✓ [265] < 10 ms

df.loc[df['MonthlyRate'].isin(outlier_values), 'MonthlyRate'] = mean_value
✓ [266] < 10 ms

df['MonthlyRate'].max()
✓ [268] < 10 ms
np.float64(1523280.0)
```

C.

1. In the above section ('B') for each issue I listed I also explained, how I went about modifying the data to correct these issues.

2. Why did I choose the specific data cleaning technique?

    a. Duplicate Values
        i. For duplicates, I noticed there was a column called 'EmployeeNumber.' Knowing this is a unique number given to each employee, and a complete row duplicate was just an error. So, the drop method would eliminate the duplicate rows without harming the data.
    b. Missing Values
        i. There were a few issues that needed to be addressed regarding missing values. I choose to use '.fillna(0.0)' for fields such as 'TrainingTimesLastYear' and 'NumCompaniesWorked' since there would be no way to accurately imply a value for these two fields. In the same sense I took the fields 'EducationField' and 'Gender' with the value 'Unknown', these missing values would not benefit from trying to analyze what their missing value could be.
        ii. Then it came to 4 columns with missing values that would benefit from actual values being filled in, rather than just '.fillna'. To resolve these issues, I decided to use the '.interpolate()' method of resolving missing values. Interpolation is a way to leverage the continuity of the data, by preserving underlying trends, avoiding the introduction of bias, and maintaining the statistical properties of the dataset.
    c. Inconsistent Entries
        i. Once I identified all the columns that had inconsistent entries, I looked at the main unique values of each column and came up with a value that would better be suited to fill in the inconsistencies.
        ii. I used a dictionary replace method to take values that were inconsistent with the columns and replaced those values with 'Unknown'.
        iii. There was one column that was listed as an object column but mainly contained int64(float64) values. So, I decided to correct this column first I had to change a value in the column that was listed as 'na' and convert it to a 'nan' value. Then I was able to make the column numeric, then with that in place, I was able to take any value that was less than 0 to also be 'nan'. Lastly, I changed the column, and all the values contained within to a '(float)' value.
    d. Outliers
        i. Lastly to deal with outliers I took a few of the columns and dealt with outliers by simply capping the values since the outliers didn't fit in within the scope of the rest of the data, setting an outlier to be absolute value since a negative value seemed to be more of an error than outlier, lastly taking a negative value in a column and setting the value to 1 since again a negative value did not fit into the scope of the data for that column.
        ii. The odd column of 'MonthlyRate' was the most difficult outlier to deal with. I choose to identify the highest outliers in that column and then set their values to the mean of the rest of the values. The original values were so extreme that they could only of been entered in error. Without being able to contact the company's HR department to correct the actual data, setting it to a mean value would get the values to be in scope with the rest of the values in that column.

3. Advantages of Data Cleaning Approach
    a. Through my data cleaning approach, I believe the main advantages were Efficiency and Data Integrity.
    b. Addressing duplicates aided in minimizing data redundancies, while using targeted imputations for missing values preserved the dataset’s structure, and reduced data loss
    c. Retaining original values for non-problematic columns maintained the dataset's overall accuracy and minimized the introduction of new biases.
4. Limitations of the Data Cleaning Approach
    a. Using such things as mean imputation assumes that the missing values are random, which may not be true and could introduce slight biases into the dataset.
    b. Assigning ‘Unknown’ might obscure actual trends, making it harder to interpret categorical distributions accurately if missing data patterns were meaningful.