EDA

October 8, 2024

[1]:
```python
import numpy as np
import pandas as pd
import os
from typing import List, Optional, Dict
import gc
import missingno as msno
import matplotlib.pyplot as plt
import math
from scipy import stats

# Set precision to 2 decimal places
pd.options.display.float_format = '{:.2f}'.format

# Suppress all warnings
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```python
def read_parquet(input_dir: str, years: Optional[List[int]] = None) ->␣
↪Dict[int, pd.DataFrame]:
    """
    Reads Parquet files for specified years from the input directory.

    Args:
        input_dir (str): The directory where Parquet files are stored.
        years (Optional[List[int]]): A list of years to read. If None, read all␣
↪available years.

    Returns:
        Dict[int, pd.DataFrame]: A dictionary with years as keys and pandas␣
↪DataFrames as values.
    """
    data_frames = {}
    available_years = [int(d.split('=')[1]) for d in os.listdir(input_dir) if d.
↪startswith('year=')]
    years_to_read = years if years is not None else available_years

    for year in years_to_read:
```

```
        year_path = os.path.join(input_dir, f'year={year}')
        if os.path.exists(year_path):
            df = pd.read_parquet(year_path)
            df['year'] = year  # Add the year column
            data_frames[year] = df
        else:
            print(f"Warning: No data found for year {year}")

    return data_frames
```

0.0.1 EDA

[3]:
```
years = [2015, 2016, 2017, 2018, 2019]
output_directory = "../data/DS/NSDUH"

# Read saved data
df = pd.concat(read_parquet(output_directory, years).values())
```

[4]:
```
shape = df.shape
missing_values = df.isnull().sum().sum()
print("Shape:", shape)
print("Missing Values:\n", missing_values)
```

```
Shape: (282768, 2814)
Missing Values:
 105689499
```

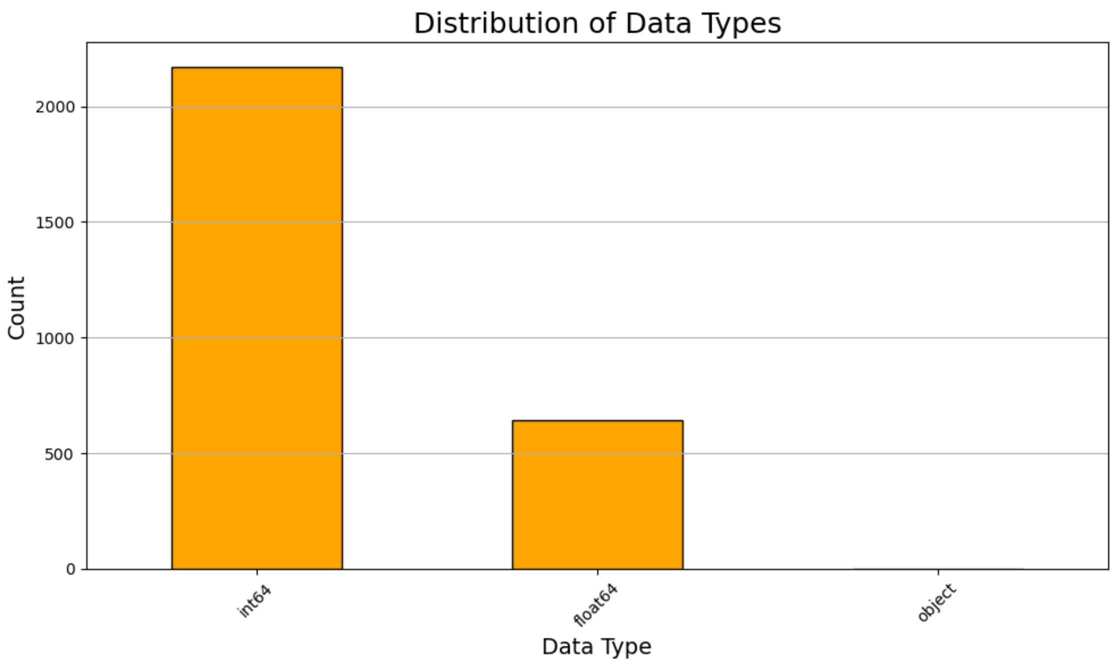
[5]:
```
# Create a summary DataFrame with data types and unique counts
summary_df = pd.DataFrame({
    'Data Type': df.dtypes,
    'Non-null Count': df.notnull().sum(),
    'Unique Count': df.nunique()
})

# Display the summary DataFrame
print(summary_df)
```

```
          Data Type  Non-null Count  Unique Count
QUESTID2      int64          282768        277519
FILEDATE     object          282768             5
CIGEVER       int64          282768             2
CIGOFRSM      int64          282768             9
CIGWILYR      int64          282768            10
…               …               …             …
OPMEDYR2    float64           56136             2
ALOPMEDYR   float64           56136             2
KRATFLG     float64           56136             2
```

```
KRATYR      float64          56136          2
KRATMON     float64          56136          2

[2814 rows x 3 columns]
```

[6]:

```python
import pandas as pd
import matplotlib.pyplot as plt

def plot_data_type_distribution(df: pd.DataFrame) -> None:
    """
    Plots a bar graph showing the distribution of different data types in the␣
 ↪DataFrame.

    Parameters:
    df (pd.DataFrame): The input dataframe.
    """
    # Count the occurrences of each data type
    data_type_counts = df.dtypes.value_counts()

    # Plotting the bar graph
    plt.figure(figsize=(10, 6))
    data_type_counts.plot(kind='bar', color='orange', edgecolor='black')
    plt.title('Distribution of Data Types', fontsize=18)
    plt.xlabel('Data Type', fontsize=14)
    plt.ylabel('Count', fontsize=14)
    plt.xticks(rotation=45)  # Rotate x labels for better readability
    plt.grid(axis='y')  # Add grid lines for better readability
    plt.tight_layout()  # Adjust layout for better fit
    plt.show()

# Example usage
# df = pd.read_csv('your_data.csv')  # Load your dataset
plot_data_type_distribution(df)
```



[7]:
```
# Get only object (string) columns
object_cols = df.select_dtypes(include='object')
print("Object Columns:")
print(object_cols.columns)
```

```
Object Columns:
Index(['FILEDATE', 'GQTYPE2'], dtype='object')
```

[8]:
```
df['GQTYPE2'].unique()
```

[8]:
```
array(['O', None, 'C'], dtype=object)
```

[9]:
```
# Calculate the percentage of zero values for each column
zero_percentage = (df == 0).mean()

# Define a threshold for sparsity, e.g., 80% zero values
sparsity_threshold = 0.8

# Identify sparse columns based on zero values
sparse_zero_columns = zero_percentage[zero_percentage > sparsity_threshold].
 ↪index.tolist()

# Combine all sparse columns
sparse_columns = list(set(sparse_zero_columns))

print("Sparse Columns:", sparse_columns)
```

Sparse Columns: ['OXYCNANYYR', 'TRBENZAPYU', 'TXPDMCADAL', 'SOMAPDPYMU', 'CGRMON', 'HERSNIF2', 'TXYRHOSAL', 'DEPNDPYPNR', 'UDPYSTM', 'TXYROUTIL', 'TRAMPDPYMU', 'HYDCPDAPYU', 'COLDMONR', 'CGRYR', 'COLDYRR', 'GHBFLGR', 'AMYLNIT2', 'TXYROUTAL', 'SEDANYFLAG', 'PIPMON', 'ESZOPDPYMU', 'ZOLPPDAPYU', 'DEPNDMRJ', 'NITOXID2', 'TXYRNOSPAL', 'SEDNMYR', 'TXYRMHCOP2', 'TXYRRESAL', 'LSDFLAG', 'KETMINMON', 'TRQNMFLAG', 'ABODMRJ', 'HALLUCMON', 'SEDOTANYR2', 'TEMAPDAPYU', 'DEPNDHER', 'DEPNDALC', 'TXLTYMETH2', 'TXYRMHCIL', 'TXYSPILNAL', 'COCYR', 'DEPNDPYPSY', 'PNROTANYR2', 'NEDHER', 'SALVIAYR', 'CADRKHERN2', 'FELTMARKR2', 'KETMINFLAG', 'MJONLYMON', 'DPPYILLALC', 'TXPDSVNGIL', 'CADRKMETH2', 'TXYRSLFHP2', 'TXYRRECVD2', 'TXYRDRPRV2', 'TXYRNDALC', 'CRKFLAG', 'TXEVRRCVD2', 'HERYR', 'TXYRSPALC', 'TXYNSPILAL', 'SPPAINT2', 'TXPAYSVNG2', 'TXPAYFAML2', 'PNRNMMON', 'DEPNDPYMTH', 'BUPRPDPYMU', 'DEMEPDAPYU', 'PSYCHFLAG', 'HERFLAG', 'CDCGMO', 'ANOSTMAPYU', 'LSDYR', 'MTDNPDAPYU', 'OXYMPDPYMU', 'ECSTMOFLAG', 'METHPDAPYU', 'ETHER2', 'AMMEPDAPYU', 'DEPNDPYSED', 'TXLTYILL', 'COLDFLGR', 'MORPPDPYMU', 'CYCLPDAPYU', 'TXLTYSTIM2', 'TXPDHINSAL', 'TXPAYMILT2', 'DEPNDPYTRQ', 'TXYRMHCAL', 'UDPYILAAL', 'SVBENZPYMU', 'STMOTANYR2', 'ABUSEMRJ', 'PNRNMFLAG', 'TXYREMRAL', 'UDPYMTH', 'LORAPDAPYU', 'CADRKINHL2', 'PCPYR', 'OXYCNNMYR', 'TXYRNDILL', 'EDFAM18', 'ABUSEPYMTH', 'SALVIAFLAG', 'UDPYILAL', 'NDSSDNSP', 'MRJMON', 'TXYRRESIL', 'DEPNDPYILL', 'TXPDMCREIL', 'UDPYINH', 'FENTPDAPYU', 'HYDMPDAPYU', 'TXYRPRIAL', 'STMANYFLAG', 'PCPFLAG', 'ABUSEALC', 'TXYRHOSOV2', 'ILLMON', 'FLURPDAPYU', 'TXYROUTPT2', 'ABODHER', 'DEPNDPYINH', 'TXYSILANAL', 'TXYRRESOV2', 'LORAPDPYMU', 'TXYSPALNIL', 'TXLTYALCO2', 'TXPAYHINS2', 'DEPNDCOC', 'SOLVENT2', 'TXPDBOSSAL', 'TXYRSPILL', 'TRQOTANYR2', 'STMNMYR', 'TRIAPDAPYU', 'SMKLSSMON', 'ABUSEPYHAL', 'CDNOCGMO', 'BUPRPDAPYU', 'CLONPDAPYU', 'TRBENZPYMU', 'OXCOPDPYMU', 'TXLTYMRJH2', 'DEPNDPYSTM', 'TXPDMILTAL', 'ZALEPDAPYU', 'METHAMFLAG', 'PCPMON', 'UDPYIEM', 'UDPYTRQ', 'CADRKMARJ2', 'PSYCHMON', 'TXYRDRPAL', 'TXYRNOSPIL', 'AMPHETAPYU', 'BARBITAPYU', 'PNROTHPYMU2', 'ILLEMMON', 'TRQANYYR', 'TXYRILANAL', 'STMANYYR', 'PSILCY2', 'TXPDHINSIL', 'TXLTYTRQL2', 'CYCLPDPYMU', 'ABODALC', 'TXLTYPNRL2', 'GHBMONR', 'STMOTHPYMU2', 'MUSRLXPYMU', 'ABUSEPYTRQ', 'AMMEPDPYMU', 'ABUSEPYIEM', 'TXYRILNAL', 'TXYRPRISN2', 'HYDCPDPYMU', 'TXYRDRPIL', 'TXPDCOURIL', 'ABUSEPYPNR', 'MORPPDAPYU', 'TXYRPRIIL', 'DIFOBTLSD', 'TXYRSLFIL', 'TXYRHOSIL', 'UDPYSED', 'TXLTYHALL2', 'ABPYILLALC', 'TRQNMMON', 'OXYMPDAPYU', 'TXPDCOURAL', 'TXPDFAMLAL', 'DAMTFXFLAG', 'TXYRNDILAL', 'INHALFLAG', 'TXLTYINHL2', 'HALLUCYR', 'TRQANYFLAG', 'KETMINYR', 'SEDNMMON', 'SEDANYYR', 'TXPAYMCAD2', 'GAS2', 'ABUSEPYPSY', 'OPINMYR', 'MTDNPDPYMU', 'MUSRLXAPYU', 'GLUE2', 'MESC2', 'LGAS2', 'PROVPDPYMU', 'CADRKHALL2', 'TXYRALNIL', 'TXPDBOSSIL', 'MRJYR', 'DAMTFXYR', 'TXPAYCOUR2', 'ANOSTMPYMU', 'FTNDDNSP', 'ZOLPPDPYMU', 'CIGMON', 'SVBENZAPYU', 'METHAMMON', 'SMKLSSYR', 'CRKMON', 'MJONLYYR', 'ABUSEPYINH', 'DCIGMON', 'HERSMOK2', 'DEPNDPYHAL', 'ALPRPDAPYU', 'AMPHETPYMU', 'OTHAEROS2', 'UDPYHAL', 'PSYCHYR', 'ABODCOC', 'TXLTYSEDV2', 'STMNMFLAG', 'ABUSEPYILL', 'TRAMPDAPYU', 'TXYRALC', 'PREG', 'DNICNSP', 'TXPDSVNGAL', 'ALPRPDPYMU', 'FENTPDPYMU', 'TXLTCURRSP', 'SOMAPDAPYU', 'ABUSEPYSED', 'CLEFLU2', 'INHALYR', 'ZOHYANYYR2', 'DIAZPDAPYU', 'ANYNEEDL', 'TRQOTHPYMU2', 'COCMON', 'HALLUCFLAG', 'TXPAYPUBL2', 'DEMEPDPYMU', 'TXPDFAMLIL', 'DEPNDPYIEM', 'SALVIAMON', 'INHALMON', 'LSDMON', 'ECSTMOYR', 'METHNEEDL2', 'TXYREMRIL', 'TXLTYCOCN2', 'TXPAYBOSS2', 'TXYRILL', 'HYDMPDPYMU', 'TXPDMCADIL', 'TRQNMYR', 'OPINMMON', 'UDPYPNR', 'ESZOPDAPYU', 'NEDCOC', 'PIPFLAG', 'ILLEMYR',

'ABUSEPYSTM', 'TXPDMILTIL', 'PROVPDAPYU', 'DIAZPDPYMU', 'SEDOTHPYMU2', 'CRKYR', 'UDPYOPI', 'TXLTYHERN2', 'TXPAYMCRE2', 'PNRNMYR', 'TXYREMRGN2', 'DAMTFXMON', 'ECSTMOMON', 'CLONPDPYMU', 'APPDRGMON2', 'ABUSEHER', 'SEDNMFLAG', 'HERMON', 'HVYDRKMON', 'TXPDPUBLAL', 'SMKLSSFLAG', 'OXCOPDAPYU', 'CADRKCOCN2', 'UDPYPSY', 'GHBYRR', 'TXYRSLFAL', 'TXPDMCREAL', 'TXYRSPILAL', 'STMNMMON', 'ABUSECOC', 'METHAMYR', 'UDPYILL', 'METHPDPYMU', 'PEYOTE2', 'COCFLAG', 'AIRDUSTER2', 'TXPDPUBLIL']

[10]:
```
# Identify columns with only one unique non-NaN value
single_unique_columns = [col for col in df.columns if df[col].
 ↪nunique(dropna=True) == 1]

print("Columns with One Unique Non-NaN Value:", single_unique_columns)
```

Columns with One Unique Non-NaN Value: []

[11]:
```
# Identify columns with exactly two unique non-NaN values
two_unique_columns = [col for col in df.columns if df[col].nunique(dropna=True)␣
 ↪== 2]

print("Number of Columns with two Unique Non-NaN Value:",␣
 ↪len(two_unique_columns))

# Print the columns and their unique values
for col in two_unique_columns:
    unique_values = df[col].dropna().unique()
    # Unccoment to see all the columns
    # print(f"Column: {col}, Unique Values: {unique_values}")
```

Number of Columns with two Unique Non-NaN Value: 780

[12]:
```
# Define a dictionary for replacements based on the code conventions
code_replacements = {
    91: 'Never Used',
    991: 'Never Used',
    9991: 'Never Used',
    93: 'Used Not in Period',
    993: 'Used Not in Period',
    9993: 'Used Not in Period',
    94: np.nan,  # Don't Know
    994: np.nan,
    9994: np.nan,
    97: np.nan,  # Refused
    997: np.nan,
    9997: np.nan,
    98: np.nan,  # Blank
    998: np.nan,
    9998: np.nan,
```

```
    99: np.nan,  # Legitimate Skip
    999: np.nan,
    9999: np.nan
}

# Apply the replacements to the DataFrame
df_temp = df.replace(code_replacements)

# Verify changes
print(df_temp.head())
```

```
   QUESTID2   FILEDATE  CIGEVER  CIGOFRSM  CIGWILYR      CIGTRY      CIGYFU  \
0  25095143  02/15/2018        1       NaN       NaN          16        2014
1  13005143  02/15/2018        1       NaN       NaN          15         NaN
2  67415143  02/15/2018        2       NaN       NaN  Never Used  Never Used
3  70925143  02/15/2018        2      3.00      4.00  Never Used  Never Used
4  75235143  02/15/2018        1       NaN       NaN          17         NaN

       CIGMFU      CIGREC            CIG30USE  … CASUPROB2 RCVYSUBPRB  \
0           1           2  Used Not in Period  …       NaN        NaN
1         NaN           3  Used Not in Period  …       NaN        NaN
2  Never Used  Never Used          Never Used  …       NaN        NaN
3  Never Used  Never Used          Never Used  …       NaN        NaN
4         NaN           1                  22  …       NaN        NaN

  CAMHPROB2 RCVYMHPRB ALMEDYR2 OPMEDYR2 ALOPMEDYR KRATFLG KRATYR KRATMON
0       NaN       NaN      NaN      NaN       NaN     NaN    NaN     NaN
1       NaN       NaN      NaN      NaN       NaN     NaN    NaN     NaN
2       NaN       NaN      NaN      NaN       NaN     NaN    NaN     NaN
3       NaN       NaN      NaN      NaN       NaN     NaN    NaN     NaN
4       NaN       NaN      NaN      NaN       NaN     NaN    NaN     NaN

[5 rows x 2814 columns]
```

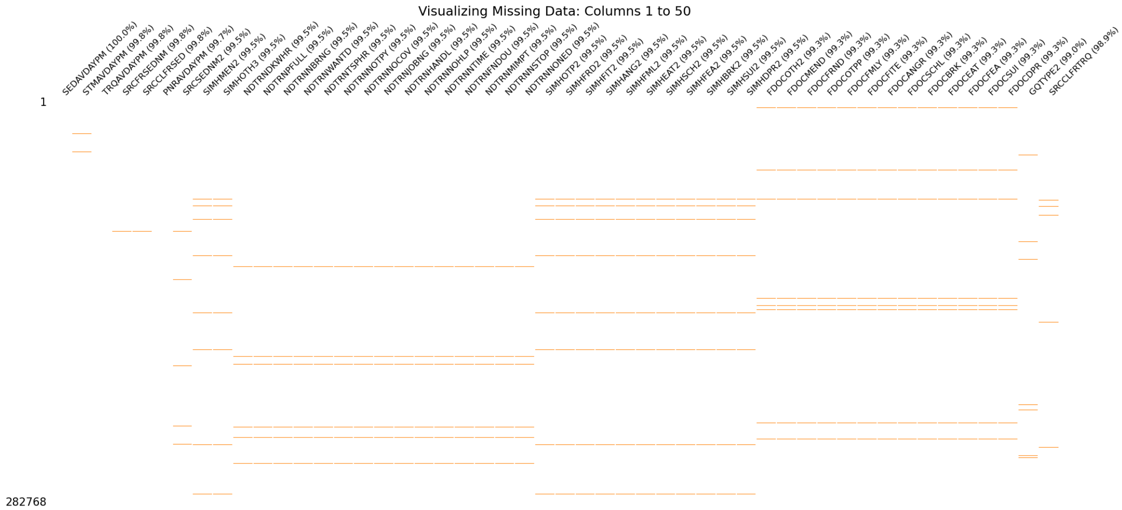
[13]:
```
df.describe()
```

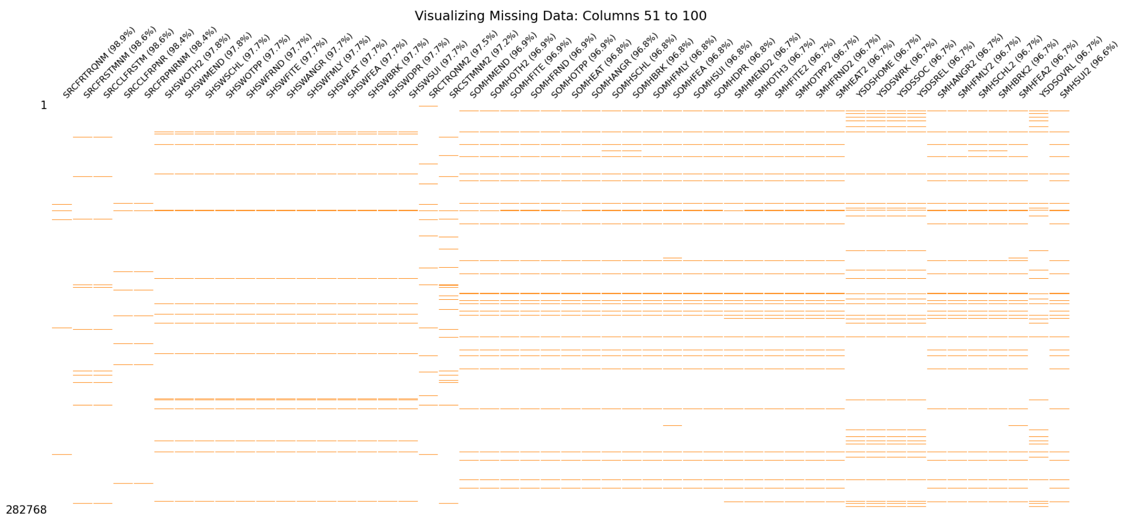
[13]:
```
          QUESTID2   CIGEVER  CIGOFRSM  CIGWILYR    CIGTRY    CIGYFU    CIGMFU  \
count  282768.00 282768.00 282768.00 282768.00 282768.00 282768.00 282768.00
mean   54378713.00      1.52     78.76     78.77    527.12   9825.70     92.95
std    25557835.04      0.50     38.94     38.92    486.98   1148.80     12.88
min    10000608.00      1.00      1.00      1.00      1.00   2013.00      1.00
25%    32135244.00      1.00     99.00     99.00     16.00   9991.00     91.00
50%    54106740.00      2.00     99.00     99.00    991.00   9991.00     91.00
75%    76090466.50      2.00     99.00     99.00    991.00   9999.00     99.00
max    99999998.00      2.00     99.00     99.00    997.00   9999.00     99.00

         CIGREC  CIG30USE    CG30EST  …  CASUPROB2  RCVYSUBPRB  CAMHPROB2  \
count 282768.00 282768.00 282768.00  …   42383.00    42361.00   42394.00
```

```
mean       48.48     79.00     92.93  …       0.11         0.08        0.24
std        44.13     27.70      4.34  …       0.31         0.27        0.43
min         1.00      1.00      1.00  …       0.00         0.00        0.00
25%         3.00     91.00     91.00  …       0.00         0.00        0.00
50%        91.00     91.00     91.00  …       0.00         0.00        0.00
75%        91.00     93.00     93.00  …       0.00         0.00        0.00
max        91.00     98.00     99.00  …       1.00         1.00        1.00

       RCVYMHPRB  ALMEDYR2  OPMEDYR2  ALOPMEDYR  KRATFLG   KRATYR  KRATMON
count   42316.00  56136.00  56136.00   56136.00 56136.00 56136.00 56136.00
mean        0.16      0.00      0.00       0.00     0.02     0.01     0.00
std         0.37      0.03      0.05       0.06     0.13     0.09     0.06
min         0.00      0.00      0.00       0.00     0.00     0.00     0.00
25%         0.00      0.00      0.00       0.00     0.00     0.00     0.00
50%         0.00      0.00      0.00       0.00     0.00     0.00     0.00
75%         0.00      0.00      0.00       0.00     0.00     0.00     0.00
max         1.00      1.00      1.00       1.00     1.00     1.00     1.00

[8 rows x 2812 columns]
```

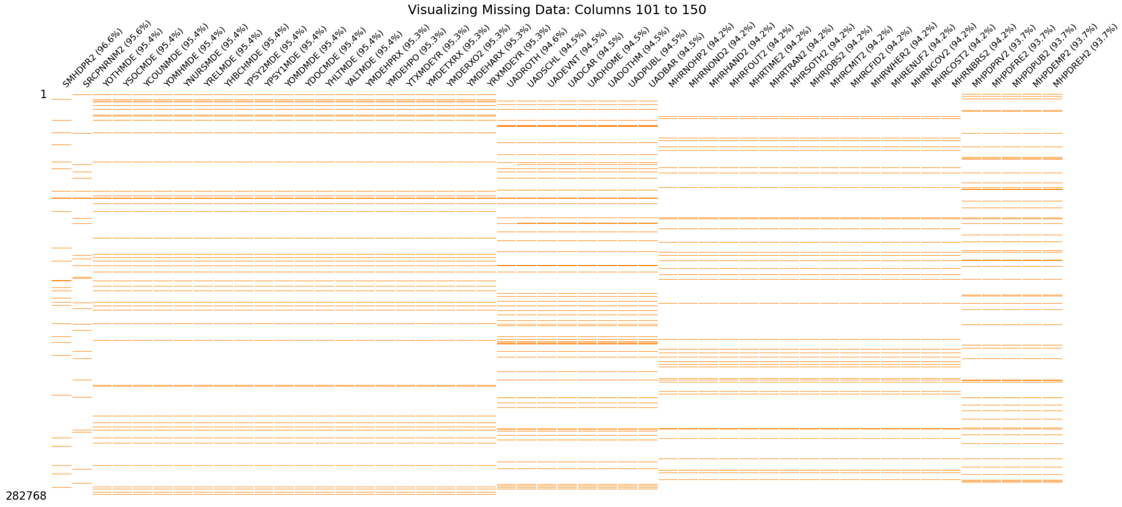
[14]:
```python
def visualize_missing_columns_in_chunks(df: pd.DataFrame, threshold: float = 20.
↪0, chunk_size: int = 50) -> None:
    """
    Visualizes the missing values for columns that have more than the given␣
↪percentage of missing data using `msno.matrix`,
    iterating over the columns in chunks, and customizes the color to orange␣
↪with missing percentages in column names.

    Parameters:
    df (pd.DataFrame): The input dataframe.
    threshold (float): The percentage threshold to filter columns. Default is␣
↪20%.
    chunk_size (int): The number of columns to visualize per iteration. Default␣
↪is 50.
    """
    # Calculate the percentage of missing values for each column
    missing_percentage = (df.isnull().sum() / len(df)) * 100

    # Filter columns that have more than the specified percentage of missing␣
↪values
    columns_with_missing = missing_percentage[missing_percentage > threshold].
↪sort_values(ascending=False)

    # Create a dictionary to rename the columns by adding the missing␣
↪percentage to the name
```

```
    renamed_columns = {col: f"{col} ({missing_percentage[col]:.1f}%)" for col␣
↪in columns_with_missing.index}

    # Create a DataFrame copy with renamed columns for those with missing␣
↪values above the threshold
    df_with_missing_renamed = df[columns_with_missing.index].
↪rename(columns=renamed_columns)

    # Determine the number of iterations needed
    num_chunks = math.ceil(len(columns_with_missing) / chunk_size)

    # Iterate and visualize columns in chunks
    for i in range(num_chunks):
        start_idx = i * chunk_size
        end_idx = min((i + 1) * chunk_size, len(columns_with_missing))

        # Get the subset of renamed columns for the current chunk
        chunk_columns = list(renamed_columns.values())[start_idx:end_idx]

        # Use msno.matrix directly without plt.figure() to suppress the figure␣
↪message
        msno.matrix(df_with_missing_renamed[chunk_columns], fontsize=12,␣
↪sparkline=False, color=(1.0, 0.5, 0.0))  # RGB color for orange

        # Add a title showing the range of columns being visualized and set a␣
↪bigger font size
        plt.title(f"Visualizing Missing Data: Columns {start_idx + 1} to␣
↪{end_idx}", fontsize=18)  # Adjusted title size

        # Display the plot
        plt.show()


visualize_missing_columns_in_chunks(df, threshold=10.0, chunk_size=50)
```

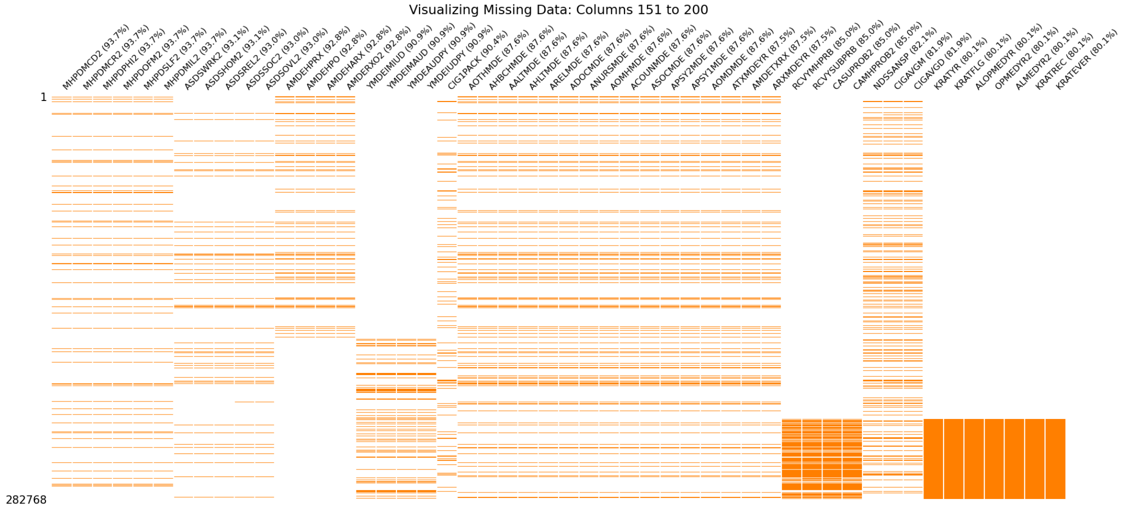

Visualizing Missing Data: Columns 1 to 50
SEDAVDAYPM (100.0%)
STMAVDAYPM (99.8%)
TRQAVDAYPM (99.8%)
SRCFRSEDNM (99.8%)
SRCCLFRSED (99.8%)
PNRAVDAYPM (99.7%)
SRCSEDNM2 (99.5%)
SIMHMEN2 (99.5%)
SIMHOTH3 (99.5%)
NDTRNDKWHR (99.5%)
NDTRNPFULL (99.5%)
NDTRNNBRNG (99.5%)
NDTRNWANTD (99.5%)
NDTRNTSPHR (99.5%)
NDTRNNOTPY (99.5%)
NDTRNNOCOV (99.5%)
NDTRNJOBNG (99.5%)
NDTRNHANDL (99.5%)
NDTRNNOHLP (99.5%)
NDTRNNTIME (99.5%)
NDTRNFNDOU (99.5%)
NDTRNMIMPT (99.5%)
NDTRNNSTOP (99.5%)
NDTRNNONED (99.5%)
SIMHOTP2 (99.5%)
SIMHFRD2 (99.5%)
SIMHFIT2 (99.5%)
SIMHANG2 (99.5%)
SIMHFML2 (99.5%)
SIMHEAT2 (99.5%)
SIMHSCH2 (99.5%)
SIMHFEA2 (99.5%)
SIMHBRK2 (99.5%)
SIMHSUI2 (99.5%)
SIMHDPR2 (99.5%)
FDOCOTH2 (99.3%)
FDOCMEND (99.3%)
FDOCFRND (99.3%)
FDOCOTPP (99.3%)
FDOCFMLY (99.3%)
FDOCFITE (99.3%)
FDOCANGR (99.3%)
FDOCSCHL (99.3%)
FDOCBRK (99.3%)
FDOCEAT (99.3%)
FDOCFEA (99.3%)
FDOCSUI (99.3%)
FDOCDPR (99.3%)
GQTYPE2 (99.0%)
SRCCLFRTRQ (98.9%)
1
282768

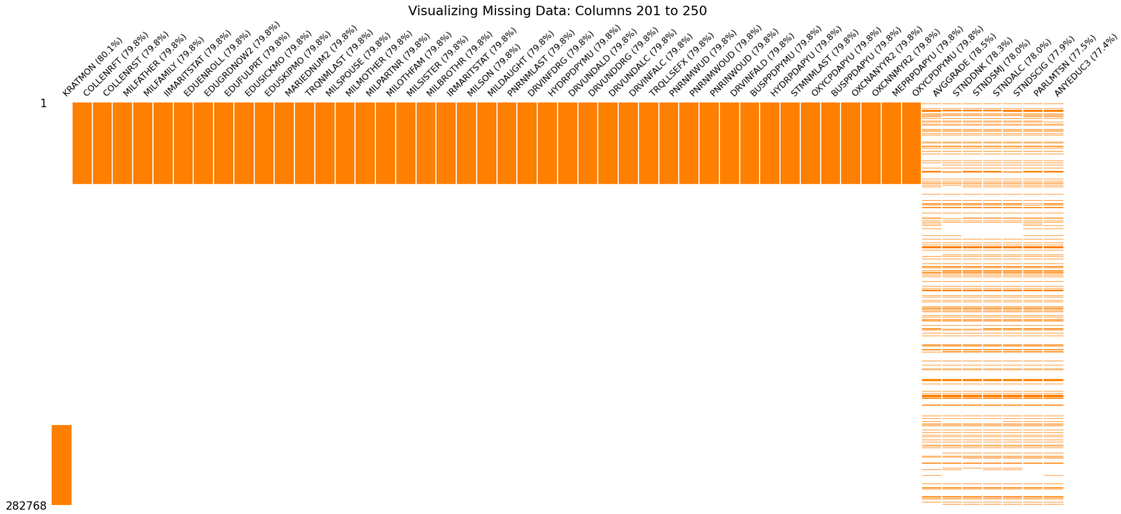

Visualizing Missing Data: Columns 51 to 100
SRCFRTRQNM (98.9%)
SRCFRSTMNM (98.6%)
SRCCLFRSTM (98.6%)
SRCCLFRPNR (98.4%)
SRCFRPNRNM (98.4%)
SHSWOTH2 (97.8%)
SHSWMEND (97.8%)
SHSWSCHL (97.7%)
SHSWOTPP (97.7%)
SHSWFRND (97.7%)
SHSWFITE (97.7%)
SHSWANGR (97.7%)
SHSWFMLY (97.7%)
SHSWEAT (97.7%)
SHSWFEA (97.7%)
SHSWBRK (97.7%)
SHSWDPR (97.7%)
SHSWSUI (97.7%)
SRCTRQNM2 (97.5%)
SRCSTMNM2 (97.2%)
SOMHMEND (96.9%)
SOMHOTH2 (96.9%)
SOMHFITE (96.9%)
SOMHFRND (96.9%)
SOMHOTPP (96.9%)
SOMHEAT (96.8%)
SOMHANGR (96.8%)
SOMHSCHL (96.8%)
SOMHBRK (96.8%)
SOMHFMLY (96.8%)
SOMHFEA (96.8%)
SOMHSUI (96.8%)
SOMHDPR (96.8%)
SMHMEND2 (96.7%)
SMHOTH3 (96.7%)
SMHFITE2 (96.7%)
SMHOTPP2 (96.7%)
SMHFRND2 (96.7%)
SMHEAT2 (96.7%)
YSDSHOME (96.7%)
YSDSWRK (96.7%)
YSDSSOC (96.7%)
YSDSREL (96.7%)
SMHANGR2 (96.7%)
SMHFMLY2 (96.7%)
SMHSCHL2 (96.7%)
SMHBRK2 (96.7%)
SMHFEA2 (96.7%)
YSDSOVRL (96.7%)
SMHSUI2 (96.6%)
1
282768

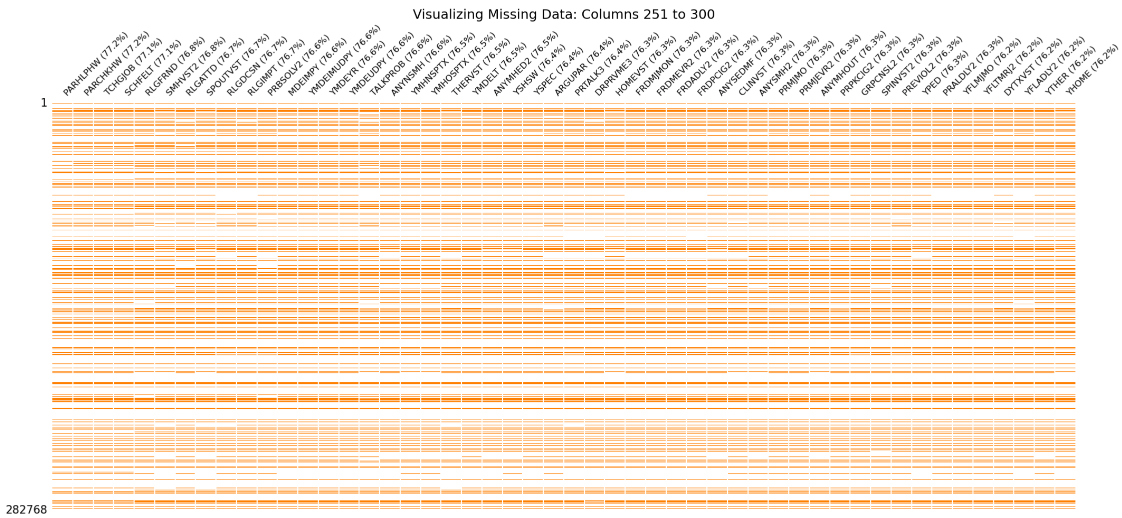
Visualizing Missing Data: Columns 101 to 150

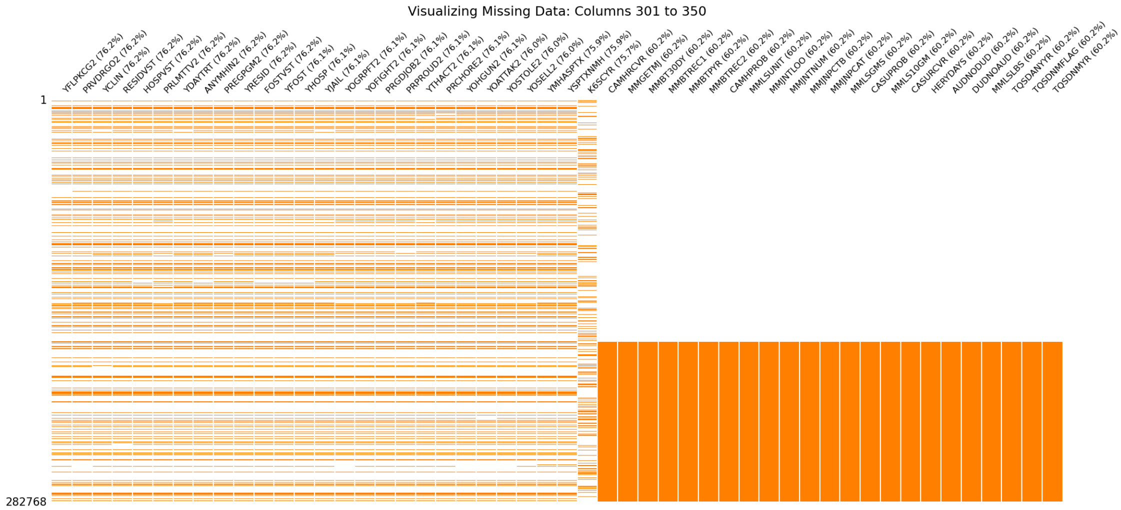


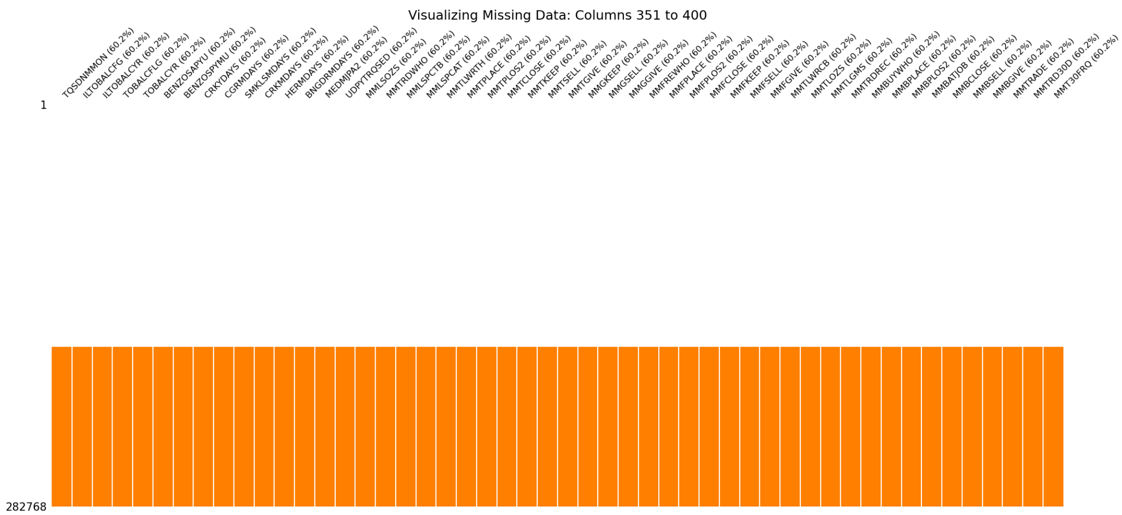
Visualizing Missing Data: Columns 151 to 200

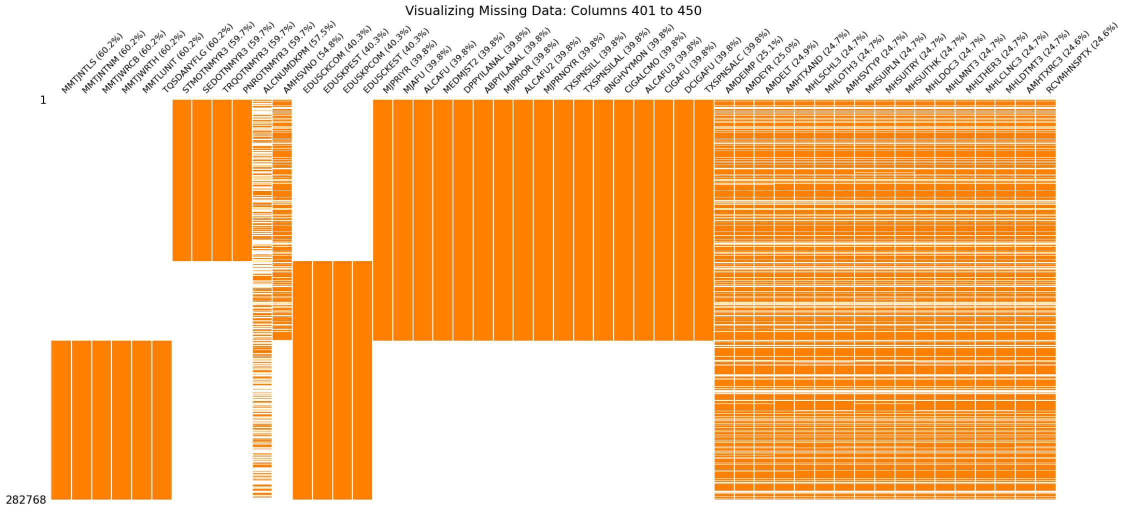


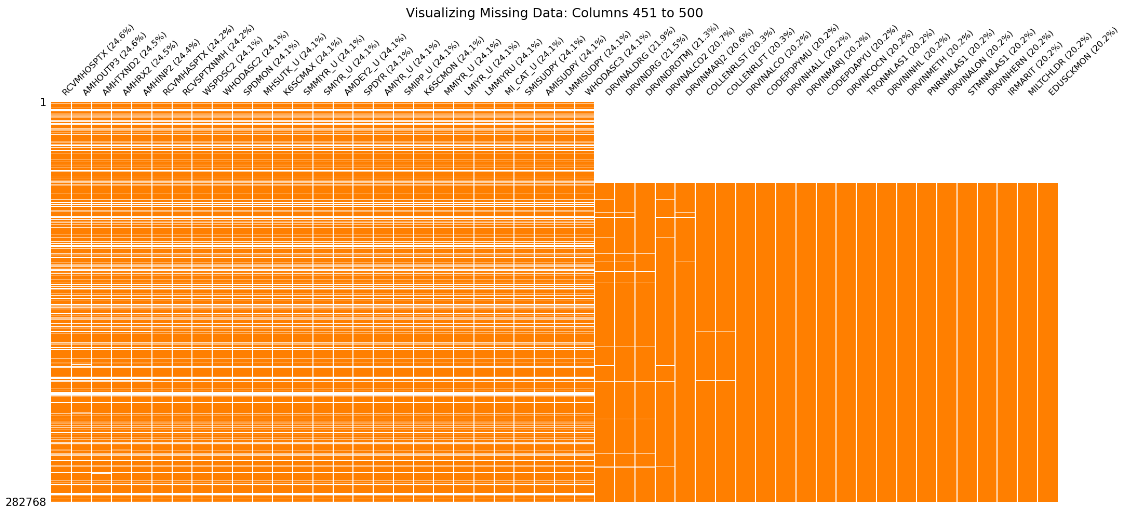

Visualizing Missing Data: Columns 201 to 250
KRATMON (80.1%)
COLLENRFT (79.8%)
COLLENRST (79.8%)
MILFATHER (79.8%)
MILFAMILY (79.8%)
IIMARITSTAT (79.8%)
EDUENROLL (79.8%)
EDUGRDNOW2 (79.8%)
EDUFULPRT (79.8%)
EDUSICKMO (79.8%)
EDUSKIPMO (79.8%)
MARIEDNUM2 (79.8%)
TRQNMLAST (79.8%)
MILSPOUSE (79.8%)
MILMOTHER (79.8%)
MILPARTNR (79.8%)
MILOTHFAM (79.8%)
MILSISTER (79.8%)
MILBROTHR (79.8%)
IRMARITSTAT (79.8%)
MILSON (79.8%)
MILDAUGHT (79.8%)
PNRNMLAST (79.8%)
DRVINFDRG (79.8%)
HYDRPDPYMU (79.8%)
DRVUNDALD (79.8%)
DRVUNDDRG (79.8%)
DRVUNDALC (79.8%)
DRVINFALC (79.8%)
TRQLLSEFX (79.8%)
PNRNMWUD (79.8%)
PNRNMWOUD (79.8%)
PNRINWOUD (79.8%)
DRVINFALD (79.8%)
BUSPPDPYMU (79.8%)
HYDRPDAPYU (79.8%)
STMNMLAST (79.8%)
OXYCPDAPYU (79.8%)
BUSPPDAPYU (79.8%)
OXCNANYYR2 (79.8%)
OXCNNMYR2 (79.8%)
MEPRPDAPYU (79.8%)
OXYCPDPYMU (79.8%)
AVGGRADE (78.5%)
STNDDNK (78.3%)
STNDSMJ (78.0%)
STNDALC (78.0%)
STNDSCIG (77.9%)
PARLMTSN (77.5%)
ANYEDUC3 (77.4%)
1
282768

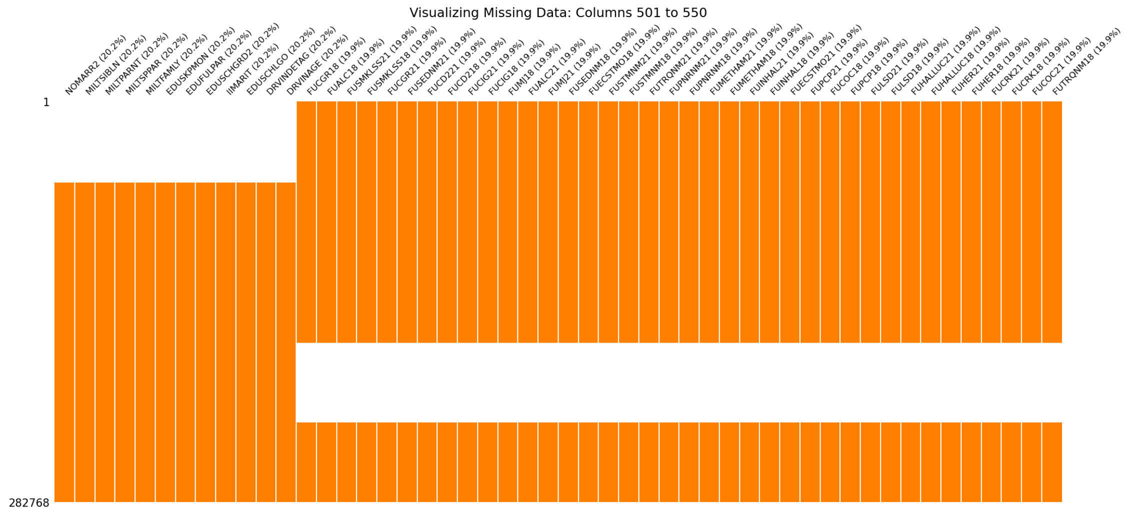

Visualizing Missing Data: Columns 251 to 300
PARHLPHW (77.2%)
PARCHKHW (77.2%)
TCHGJOB (77.1%)
SCHFELT (77.1%)
RLGFRND (76.8%)
SMHVST2 (76.8%)
RLGATTD (76.7%)
SPOUTVST (76.7%)
RLGDCSN (76.7%)
RLGIMPT (76.7%)
PRBSOLV2 (76.6%)
MDEIMPY (76.6%)
YMDEIMUDPY (76.6%)
YMDEYR (76.6%)
YMDEUDPY (76.6%)
TALKPROB (76.6%)
ANYNSMH (76.6%)
YMHNSPTX (76.5%)
YMHOSPTX (76.5%)
THERVST (76.5%)
YMDELT (76.5%)
ANYMHED2 (76.5%)
YSHSW (76.4%)
YSPEC (76.4%)
ARGUPAR (76.4%)
PRTALK3 (76.4%)
DRPRVME3 (76.3%)
HOMEVST (76.3%)
FRDMJMON (76.3%)
FRDMEVR2 (76.3%)
FRDADLY2 (76.3%)
FRDPCIG2 (76.3%)
ANYSEDMF (76.3%)
CLINVST (76.3%)
ANYSMH2 (76.3%)
PRMJMO (76.3%)
PRMJEVR2 (76.3%)
ANYMHOUT (76.3%)
PRPKCIG2 (76.3%)
GRPCNSL2 (76.3%)
SPINVST2 (76.3%)
PREVIOL2 (76.3%)
YPED (76.3%)
PRALDLY2 (76.3%)
YFLMJMO (76.2%)
YFLTMRJ2 (76.2%)
DYTXVST (76.2%)
YFLADLY2 (76.2%)
YTHER (76.2%)
YHOME (76.2%)
1
282768


Visualizing Missing Data: Columns 301 to 350
YFLPKCG2 (76.2%)
PRVDRGO2 (76.2%)
YCLIN (76.2%)
RESIDVST (76.2%)
HOSPVST (76.2%)
PRLMTTV2 (76.2%)
YDAYTRT (76.2%)
ANYMHIN2 (76.2%)
PREGPGM2 (76.2%)
YRESID (76.2%)
FOSTVST (76.2%)
YFOST (76.1%)
YHOSP (76.1%)
YJAIL (76.1%)
YOGRPFT2 (76.1%)
YOFIGHT2 (76.1%)
PRGDJOB2 (76.1%)
PRPROUD2 (76.1%)
YTHACT2 (76.1%)
PRCHORE2 (76.1%)
YOHGUN2 (76.1%)
YOATTAK2 (76.0%)
YOSTOLE2 (76.0%)
YOSELL2 (76.0%)
YMHASPTX (75.9%)
YSPTXNMH (75.9%)
K6SCYR (75.7%)
CAMHRCVR (60.2%)
MMGETMJ (60.2%)
MMBT30DY (60.2%)
MMBTREC1 (60.2%)
MMBTPYR (60.2%)
MMBTREC2 (60.2%)
CAMHPROB (60.2%)
MMLSUNIT (60.2%)
MMJNTLOO (60.2%)
MMJNTNUM (60.2%)
MMJNPCTB (60.2%)
MMJNPCAT (60.2%)
MMLSGMS (60.2%)
CASUPROB (60.2%)
MMLS10GM (60.2%)
CASURCVR (60.2%)
HERYDAYS (60.2%)
AUDNODUD (60.2%)
DUDNOAUD (60.2%)
MMLSLBS (60.2%)
TQSDANYYR (60.2%)
TQSDNMFLAG (60.2%)
TQSDNMYR (60.2%)
1
282768


Visualizing Missing Data: Columns 351 to 400
TQSDNMMON (60.2%)
ILTOBALCFG (60.2%)
ILTOBALCYR (60.2%)
TOBALCFLG (60.2%)
TOBALCYR (60.2%)
BENZOSAPYU (60.2%)
BENZOSPYMU (60.2%)
CRKYDAYS (60.2%)
CGRMDAYS (60.2%)
SMKLSMDAYS (60.2%)
CRKMDAYS (60.2%)
HERMDAYS (60.2%)
BNGDRMDAYS (60.2%)
MEDMJPA2 (60.2%)
UDPYTRQSED (60.2%)
MMLSOZS (60.2%)
MMTRDWHO (60.2%)
MMLSPCTB (60.2%)
MMLSPCAT (60.2%)
MMTLWRTH (60.2%)
MMTPLACE (60.2%)
MMTPLOS2 (60.2%)
MMTCLOSE (60.2%)
MMTKEEP (60.2%)
MMTSELL (60.2%)
MMTGIVE (60.2%)
MMGKEEP (60.2%)
MMGSELL (60.2%)
MMGGIVE (60.2%)
MMFREWHO (60.2%)
MMFPLACE (60.2%)
MMFPLOS2 (60.2%)
MMFCLOSE (60.2%)
MMFKEEP (60.2%)
MMFSELL (60.2%)
MMFGIVE (60.2%)
MMTLWRCB (60.2%)
MMTLOZS (60.2%)
MMTLGMS (60.2%)
MMTRDREC (60.2%)
MMBUYWHO (60.2%)
MMBPLACE (60.2%)
MMBPLOS2 (60.2%)
MMBATJOB (60.2%)
MMBCLOSE (60.2%)
MMBSELL (60.2%)
MMBGIVE (60.2%)
MMTRADE (60.2%)
MMTRD30D (60.2%)
MMT30FRQ (60.2%)
1
282768


Visualizing Missing Data: Columns 401 to 450
MMTJNTLS (60.2%)
MMTJNTNM (60.2%)
MMTJWRCB (60.2%)
MMTJWRTH (60.2%)
MMTLUNIT (60.2%)
TQSDANYFLG (60.2%)
STMOTNMYR3 (59.7%)
SEDOTNMYR3 (59.7%)
TRQOTNMYR3 (59.7%)
PNROTNMYR3 (59.7%)
ALCNUMDKPM (57.5%)
AMHSVNO (54.8%)
EDUSCKCOM (40.3%)
EDUSKPEST (40.3%)
EDUSKPCOM (40.3%)
EDUSCKEST (40.3%)
MJPRIYR (39.8%)
MJAFU (39.8%)
ALCAFU (39.8%)
MEDMJST2 (39.8%)
DPPYILANAL (39.8%)
ABPYILANAL (39.8%)
MJPRIOR (39.8%)
ALCAFU2 (39.8%)
MJPRNOYR (39.8%)
TXSPNSILL (39.8%)
TXSPNSILAL (39.8%)
BNGHVYMON (39.8%)
CIGALCMO (39.8%)
ALCAFU3 (39.8%)
CIGAFU (39.8%)
DCIGAFU (39.8%)
TXSPNSALC (39.8%)
AMDEIMP (25.1%)
AMDEYR (25.0%)
AMDELT (24.9%)
AMHTXAND (24.7%)
MHLSCHL3 (24.7%)
MHLOTH3 (24.7%)
AMHSVTYP (24.7%)
MHSUIPLN (24.7%)
MHSUITRY (24.7%)
MHSUITHK (24.7%)
MHLDOC3 (24.7%)
MHLMNT3 (24.7%)
MHLTHER3 (24.7%)
MHLCLNC3 (24.7%)
MHLDTMT3 (24.7%)
AMHTXRC3 (24.6%)
RCVMHNSPTX (24.6%)
1
282768


Visualizing Missing Data: Columns 451 to 500
RCVMHOSPTX (24.6%)
AMHOUTP3 (24.6%)
AMHTXND2 (24.5%)
AMHRX2 (24.5%)
AMHINP2 (24.4%)
RCVMHASPTX (24.2%)
RCVSPTXNMH (24.2%)
WSPDSC2 (24.1%)
WHODASC2 (24.1%)
SPDMON (24.1%)
MHSUTK_U (24.1%)
K6SCMAX (24.1%)
SMMIYR_U (24.1%)
SMIYR_U (24.1%)
AMDEY2_U (24.1%)
SPDYR (24.1%)
AMIYR_U (24.1%)
SMIPP_U (24.1%)
K6SCMON (24.1%)
MMIYR_U (24.1%)
LMIYR_U (24.1%)
LMMIYRU (24.1%)
MI_CAT_U (24.1%)
SMISUDPY (24.1%)
AMISUDPY (24.1%)
LMMISUDPY (24.1%)
WHODASC3 (24.1%)
DRVINALDRG (21.9%)
DRVINDRG (21.5%)
DRVINDROTMJ (21.3%)
DRVINALCO2 (20.7%)
DRVINMARJ2 (20.6%)
COLLENRLST (20.3%)
COLLENRLFT (20.3%)
DRVINALCO (20.2%)
CODEPDPYMU (20.2%)
DRVINHALL (20.2%)
DRVINMARJ (20.2%)
CODEPDAPYU (20.2%)
DRVINCOCN (20.2%)
TRQNMLAS1 (20.2%)
DRVININHL (20.2%)
DRVINMETH (20.2%)
PNRNMLAS1 (20.2%)
DRVINALON (20.2%)
STMNMLAS1 (20.2%)
DRVINHERN (20.2%)
IRMARIT (20.2%)
MILTCHLDR (20.2%)
EDUSCKMON (20.2%)
1
282768



0.0.2 Observations from `msno.matrix()` Visualization

- **High Missing Values**:
  - There are **1,458 columns** with more than **10% missing values**.
- **Patterns of Missingness**:
  - We can observe clear patterns of missing data where **certain columns have data present in earlier years but are missing in later years**, and conversely, **some columns are missing in earlier years but have data available in later years**.
  - This indicates that the missingness is **non-random** and could be related to the **time period** or changes in data collection processes over time.
- **Next Steps**:
  - Consider potential imputation strategies, especially for time-based patterns.
  - If necessary, drop columns with high missing percentages or those that do not contribute significantly to the analysis.

0.1 Data Cleaning Steps

0.2 1. Handling Sparse Columns

- **Goal**: Identify columns with sparse data and remove or address them.
- **Method**: Columns with more than 10% missing values were considered sparse.

0.3 2. Removing Columns with Large Missing Values

- **Goal**: Improve data quality by eliminating columns with excessive missing data.
- **Method**: Any column with a significant portion (over 10%) of missing values was identified from the dataset.

0.4 3. Filtering for Necessary Columns

- **Goal**: Retain only the relevant columns for analysis.

- **Method**: Non-essential columns were filtered out, leaving only those pertinent to the analysis and model-building steps.

0.5 4. Type Correction

- **Goal**: Ensure that each column's data type is appropriate for its contents.
- **Method**: The data types of each column were reviewed and corrected where necessary. For example, converting strings representing dates into datetime objects, or strings containing numeric data into integers/floats.

0.6 5. Duplicate Record Removal

- **Goal**: Eliminate redundant entries that could distort analysis.
- **Method**: Identify and remove duplicate rows to ensure each record is unique.

0.7 6. Handling Missing Data

- **Goal**: Handle/impute missing Data
- **Method**: Imputed Missing data.

0.8 6. Data Range validation and cleaning.

- **Goal**: Handle Data Range validation.
- **Method**: using assertions(if statements)

0.9 7. Substituting Values with Desired Entries

- **Goal**: Substituting Values with Desired Entries
- **Method**: Used .replace functions

0.9.1 Note: We have performed these cleanings based on EDA and specific requirements for each hypothesis. These cleaning steps are done across different notebooks.

0.9.2 EDA for the Online Gaming Study - Anxiety Dataset

[15]:
```
gaming_dat = pd.read_csv(r"../data/GamingStudy_data.csv")
```

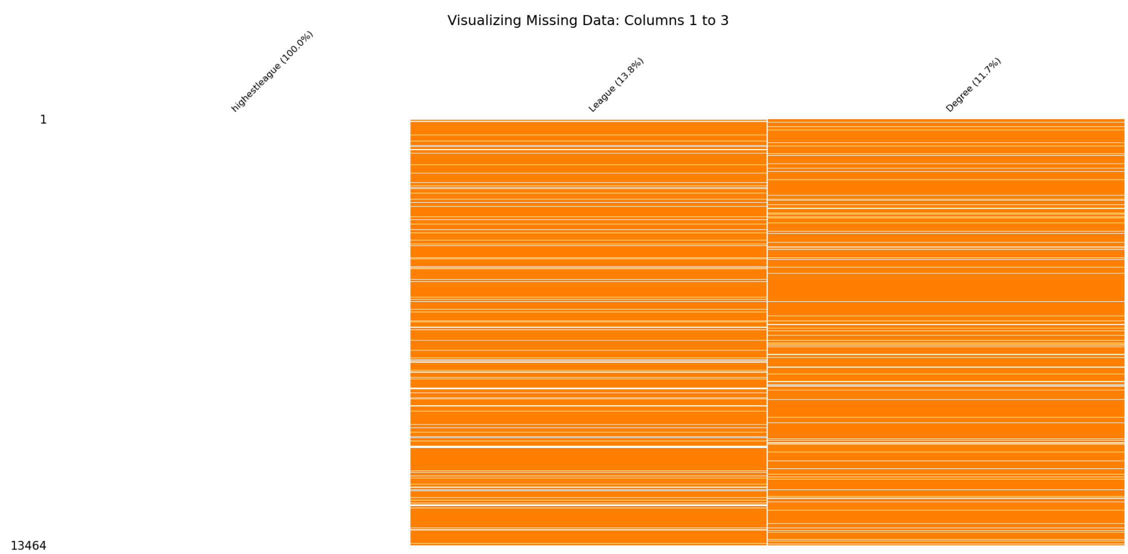
[16]:
```
gaming_dat.describe()
```

[16]:

	S. No.	Timestamp	GAD1	GAD2	GAD3	GAD4	GAD5 \
count	13464.00	13464.00	13464.00	13464.00	13464.00	13464.00	13464.00
mean	7096.84	42054.84	0.86	0.67	0.97	0.72	0.49
std	4114.48	0.27	0.93	0.92	0.98	0.92	0.84
min	1.00	42052.00	0.00	0.00	0.00	0.00	0.00
25%	3532.75	42054.72	0.00	0.00	0.00	0.00	0.00
50%	7087.50	42054.80	1.00	0.00	1.00	0.00	0.00
75%	10654.25	42054.93	1.00	1.00	2.00	1.00	1.00
max	14250.00	42058.36	3.00	3.00	3.00	3.00	3.00

```
           GAD6      GAD7      SWL1  …    SPIN13    SPIN14    SPIN15    SPIN16  \
count 13464.00 13464.00 13464.00  … 13277.00 13308.00 13317.00 13317.00
mean      0.91      0.59      3.72  …      0.54      1.25      1.41      0.62
std       0.93      0.89      1.74  …      0.94      1.21      1.35      0.96
min       0.00      0.00      1.00  …      0.00      0.00      0.00      0.00
25%       0.00      0.00      2.00  …      0.00      0.00      0.00      0.00
50%       1.00      0.00      4.00  …      0.00      1.00      1.00      0.00
75%       1.00      1.00      5.00  …      1.00      2.00      2.00      1.00
max       3.00      3.00      7.00  …      4.00      4.00      4.00      4.00

         SPIN17  Narcissism       Age     GAD_T     SWL_T    SPIN_T
count 13289.00    13441.00 13464.00 13464.00 13464.00 12814.00
mean      0.94        2.03     20.93      5.21     19.79     19.85
std       1.18        1.06      3.30      4.71      7.23     13.47
min       0.00        1.00     18.00      0.00      5.00      0.00
25%       0.00        1.00     18.00      2.00     14.00      9.00
50%       0.00        2.00     20.00      4.00     20.00     17.00
75%       2.00        3.00     22.00      8.00     26.00     28.00
max       4.00        5.00     63.00     21.00     35.00     68.00

[8 rows x 39 columns]
```

[17]:
```
visualize_missing_columns_in_chunks(gaming_dat, threshold=10.0, chunk_size=50)
```



0.9.3 Observations for Online Gaming Study - Anxiety Dataset

- Only two columns in this dataset have more than 10% of missing data.