Advanced python project based on principal component analysis

Guided by:

Netali Agarwal (Manager Data Science at Commonwealth Bank) and

Kunal Naik (Senior Data Scientist at Dell Technologies)

Submitted by:

Utkarsh Ajay Gupta

Towards the partial fulfilment of

"Data Science Certification Program"

at AltUni by InsideIIM.com

Contents

Problem Statement 3
Methodology 3
Objective 3
Steps 3
Exploratory Data Analysis & feature engineering 4
    Importing necessary libraries and data files 4
    Performing Exploratory Data Analysis (EDA) 4
    Feature engineering - outlier treatment 5
    Feature engineering - Scaling 14
Principal component analysis 15
    Plotting the Principal components 15
    Feature Selection: 17
Conclusion: 18
References: 18

Problem Statement

PCA FH (FT): Primary census abstract for female headed households excluding institutional households (India & States/UTs - District Level), Scheduled tribes - 2011

PCA for Female Headed Household Excluding Institutional Household

The Indian Census has the reputation of being one of the best in the world. The first Census in India was conducted in the year 1872. This was conducted at different points of time in different parts of the country. In 1881 a Census was taken for the entire country simultaneously. Since then, Census has been conducted every ten years, without a break. Thus, the Census of India 2011 was the fifteenth in this unbroken series since 1872, the seventh after independence and the second census of the third millennium and twenty first century. The census has been uninterruptedly continued despite several adversities like wars, epidemics, natural calamities, political unrest, etc.

The Census of India is conducted under the provisions of the Census Act 1948 and the Census Rules, 1990. The Primary Census Abstract which is important publication of 2011 Census gives basic information on Area, Total Number of Households, Total Population, Scheduled Castes, Scheduled Tribes Population, Population in the age group 0-6, Literates, Main Workers and Marginal Workers classified by the four broad industrial categories, namely, (i) Cultivators, (ii) Agricultural Laborers, (iii) Household Industry Workers, and (iv) Other Workers and also non-workers. The characteristics of the Total Population include Scheduled Castes, Scheduled Tribes, Institutional and Houseless Population and are presented by sex and residence (rural-urban). Census 2011 covered 35 States/Union Territories containing 640 districts which in turn contained 5,924 sub-districts, 7,935 towns and 6,40,867 villages.

The data collected has so many variables making it difficult to find useful details without using Data Science Techniques. You are tasked to perform detailed EDA and identify Optimum Principal Components that explains the most variance in data. (Use sklearn only).

Data file - PCA India Data Census.xlsx

Methodology

- The coding has been done using python programming language.
- Python libraries like pandas, numpy, matplotlib, sklearn and seaborn have been extensively used throughout the code.
- Jupyter notebook on a windows 11 operating system has been used as the coding platform.

Objective

To perform principal component analysis on the given data by finding the components that show the most variance and ultimately scoop out a new dataset out of the given dataset.

Steps

1. Exploratory data analysis & feature engineering
2. Principal component analysis

Exploratory Data Analysis & feature engineering

Importing necessary libraries and data files

```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
```

Pandas is a Python library for data manipulation and analysis, providing data structures like dataframes for working with structured data. It simplifies tasks like data loading, cleaning, and aggregation, making it a powerful tool for data exploration and manipulation.
NumPy is a fundamental library for numerical computing, introducing efficient array operations and mathematical functions in Python. It is widely used in scientific and data analysis tasks, enabling vectorized operations on data, making it faster and more memory-efficient.

Seaborn is a data visualization library built on Matplotlib, offering a high-level interface for creating aesthetically pleasing statistical graphics. It simplifies the creation of various data visualizations and is especially useful for exploring and presenting data patterns.

Scikit-Learn (sklearn) is a machine learning library in Python, providing tools for a wide range of machine learning tasks, including classification, regression, and clustering. It simplifies the implementation of machine learning algorithms and model evaluation, making it a go-to choice for many data scientists and machine learning practitioners.

Matplotlib is a comprehensive Python library for creating 2D plots and charts, offering customization and fine-tuning options. It is a versatile tool for creating static or interactive visualizations for a variety of data analysis and presentation needs.

Reading the data file "PCA India Data Census.xlsx" and putting it into a dataframe named 'df'.

```
df = pd.read_excel('PCA India Data_Census.xlsx')
```

Performing Exploratory Data Analysis (EDA)

EDA on the data to know about its configuration, checking for any missing data, duplicate data, knowing some statistics about the data like mean, median, etc.

```
df.head()

df.info()

df.isnull().sum()

dups = df.duplicated()

print('No. of duplicate rows are %d' % (dups.sum()))

df.describe()
```

df.head(): Displays the first few rows of the DataFrame to give a quick overview of its data and structure.

df.info(): Provides a concise summary of the DataFrame, including the data types, non-null counts, and memory usage, which is useful for understanding the dataset.

df.isnull().sum(): Counts and displays the number of missing (null) values in each column of the DataFrame, helping identify missing data.

df.duplicated(): Identifies and counts duplicate rows in the DataFrame, and prints the total number of duplicate rows found.

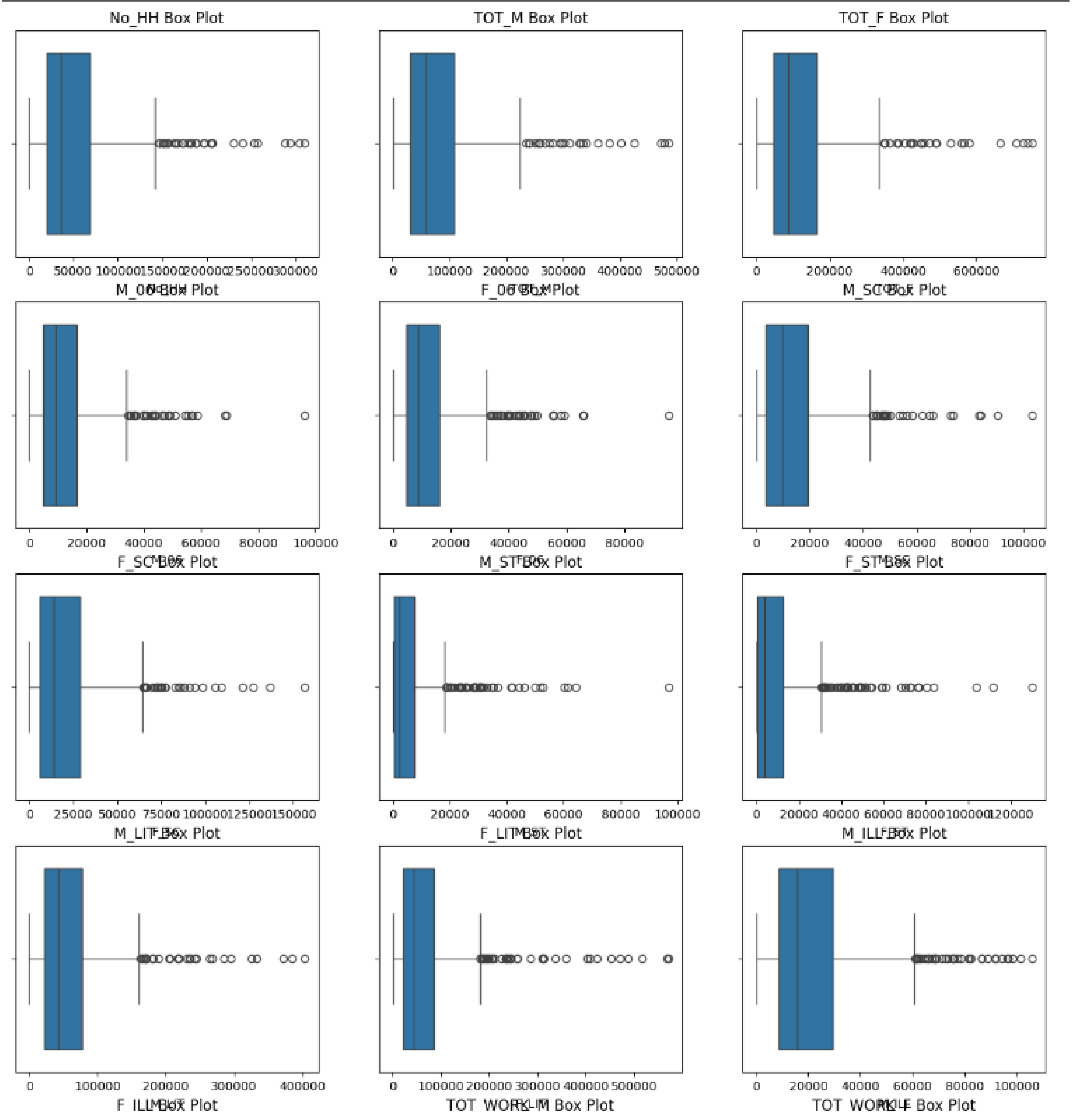
df.describe(): Generates descriptive statistics for the numeric columns in the DataFrame, including count, mean, standard deviation, minimum, maximum, and quartiles, to summarize the data's central tendency and spread.

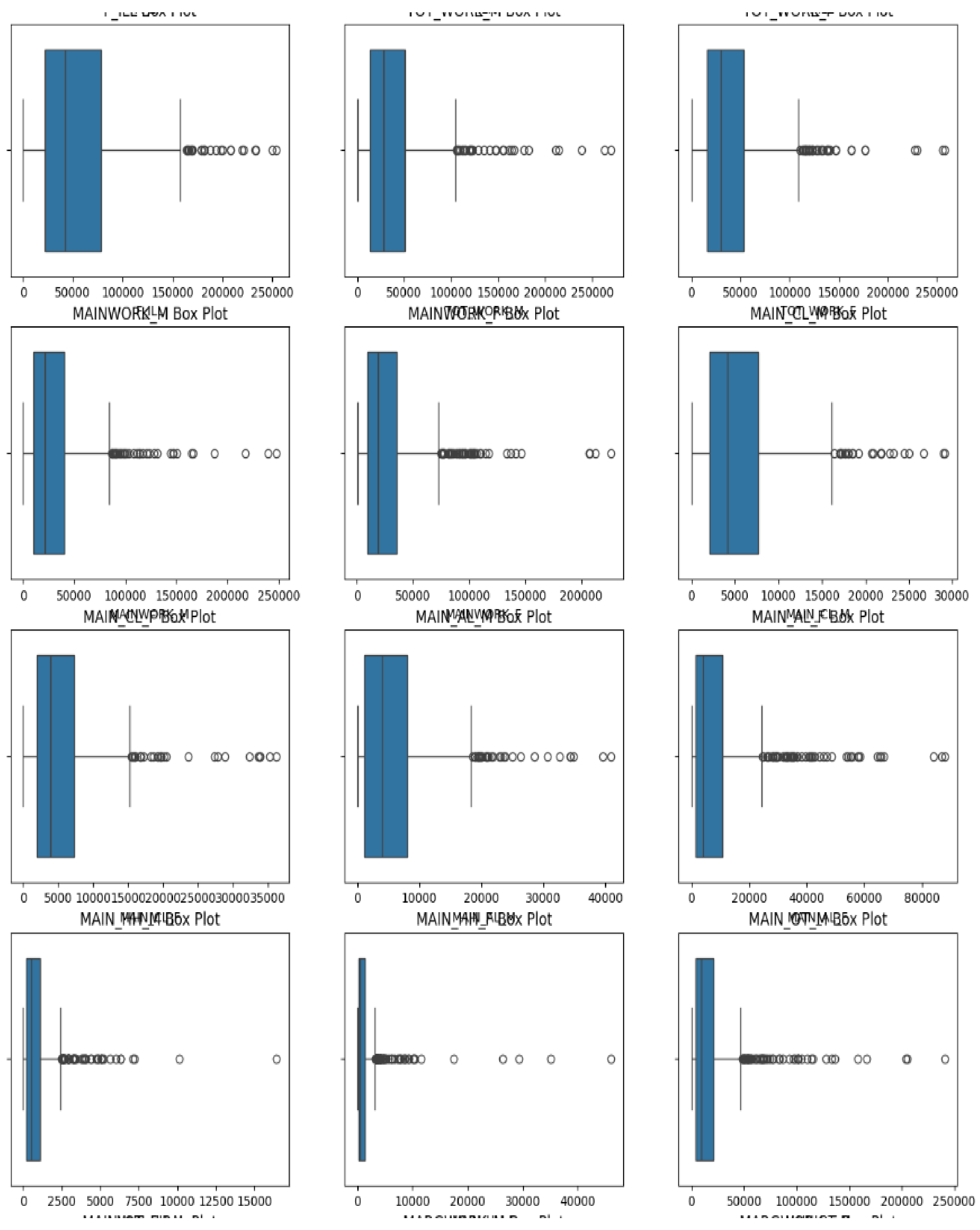
Feature engineering - outlier treatment

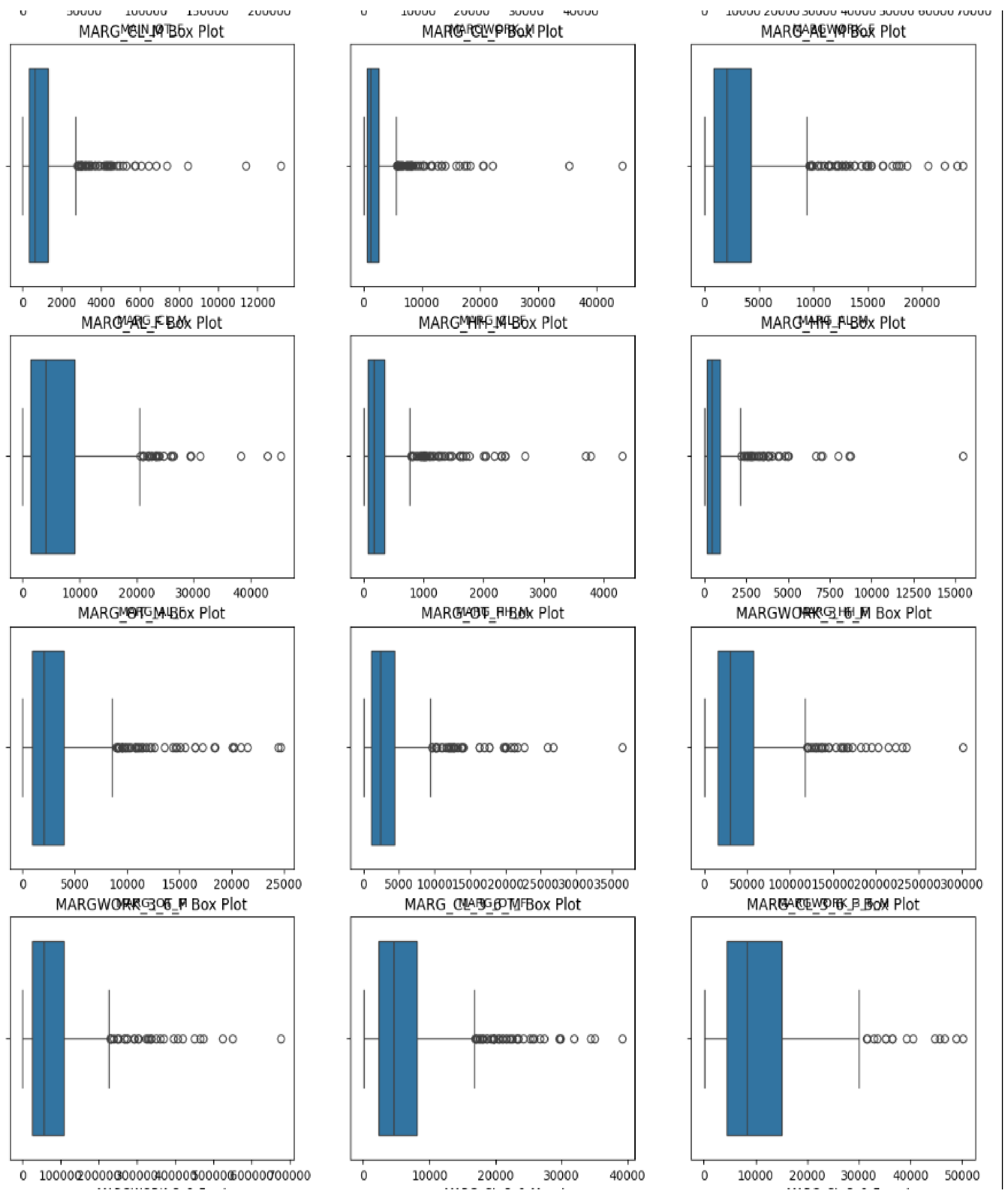
We have plotted boxplots using seaborn library of all variables in the dataset to check for outliers

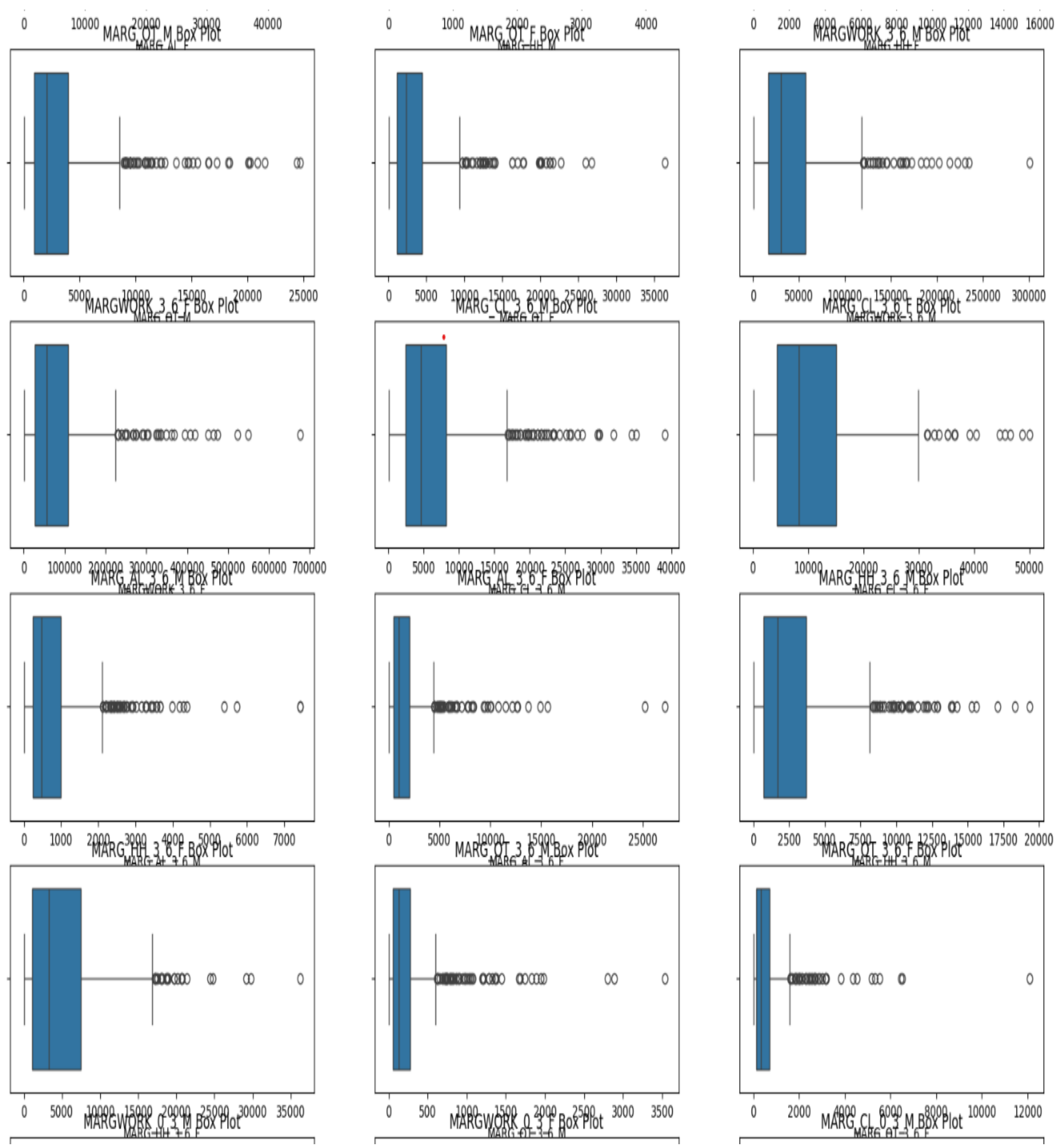
```
subplots_per_row = 3
num_rows = (len(df.columns) - 1) // subplots_per_row + 1
fig, axes = plt.subplots(num_rows, subplots_per_row, figsize=(15, 4 * num_rows))
axes = axes.flatten()
for i, column in enumerate(df.columns):
  sns.boxplot(x=df[column], ax=axes[i]) axes[i].set_title(f'{column} Box Plot')
  plt.show()
```

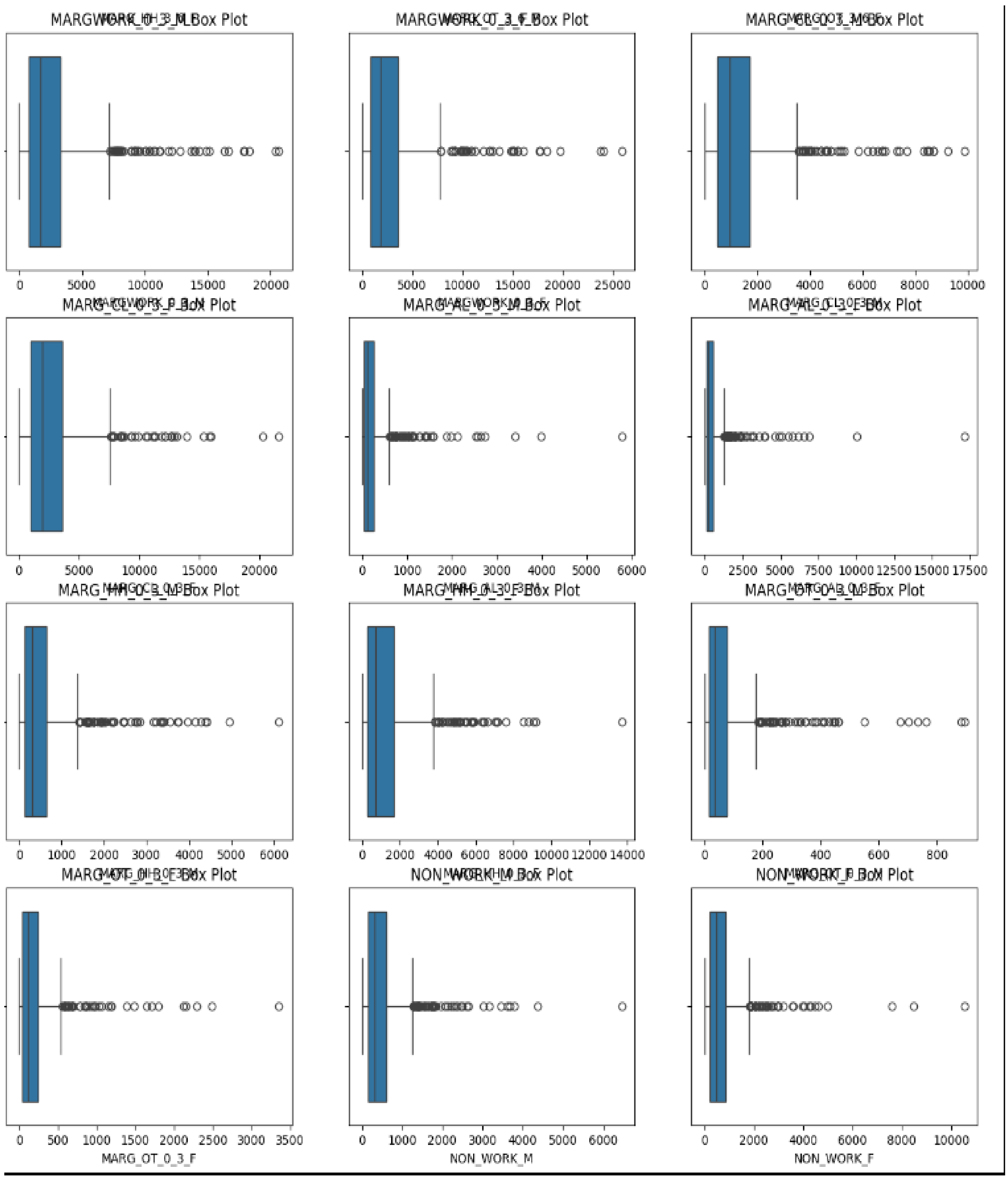
Below mentioned is the output received, of all the continuous variables in the dataset.




F_ILL Box Plot
F_ILL
TOT_WORK_M Box Plot
TOT_WORK_M
TOT_WORK_F Box Plot
TOT_WORK_F
MAINWORK_M Box Plot
MAINWORK_M
MAINWORK_F Box Plot
MAINWORK_F
MAIN_CL_M Box Plot
MAIN_CL_M
MAIN_CL_F Box Plot
MAIN_CL_F
MAIN_AL_M Box Plot
MAIN_AL_M
MAIN_AL_F Box Plot
MAIN_AL_F
MAIN_HH_M Box Plot
MAIN_HH_M
MAIN_HH_F Box Plot
MAIN_HH_F
MAIN_OT_M Box Plot
MAIN_OT_M
0 50000 100000 150000 200000 250000
0 5000 10000 15000 20000 25000 30000
0 5000 10000 15000 20000 25000 30000 35000
0 10000 20000 30000 40000
0 20000 40000 60000 80000
0 2500 5000 7500 10000 12500 15000


MARG_CL_M Box Plot
MARG_CL_F Box Plot
MARG_AL_M Box Plot
MARG_AL_F Box Plot
MARG_HH_M Box Plot
MARG_HH_F Box Plot
MARG_OT_M Box Plot
MARG_OT_F Box Plot
MARGWORK_3_6_M Box Plot
MARGWORK_3_6_F Box Plot
MARG_CL_3_6_M Box Plot
MARG_CL_3_6_F Box Plot
MAIN_OT_F
MARGWORK_M
MARGWORK_F
MARG_CL_M
MARG_CL_F
MARG_AL_M
MARG_AL_F
MARG_HH_M
MARG_HH_F
MARG_OT_M
MARG_OT_F
MARGWORK_3_6_M


MARG_OT_M Box Plot
MARG_OT_F Box Plot
MARGWORK_3_6_M Box Plot
MARGWORK_3_6_F Box Plot
MARG_CL_3_6_M Box Plot
MARG_CL_3_6_F Box Plot
MARG_AL_3_6_M Box Plot
MARG_AL_3_6_F Box Plot
MARG_HH_3_6_M Box Plot
MARG_HH_3_6_F Box Plot
MARG_OT_3_6_M Box Plot
MARG_OT_3_6_F Box Plot
MARGWORK_0_3_M Box Plot
MARGWORK_0_3_F Box Plot
MARG_CL_0_3_M Box Plot



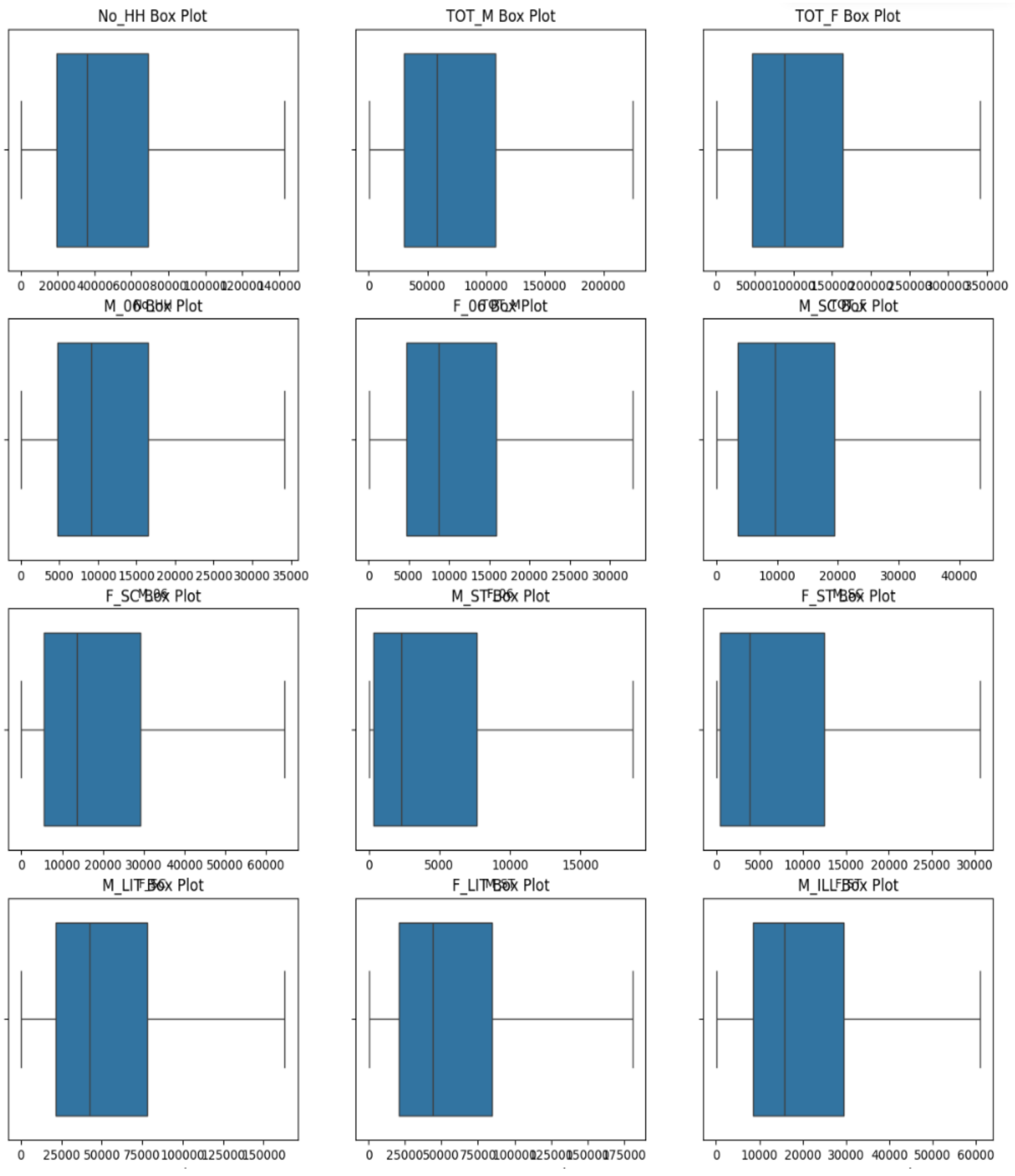
All the above plots depict that there are a lot of outliers in every variable of the dataset and so we perform outlier treatment by defining a function for outlier treatment

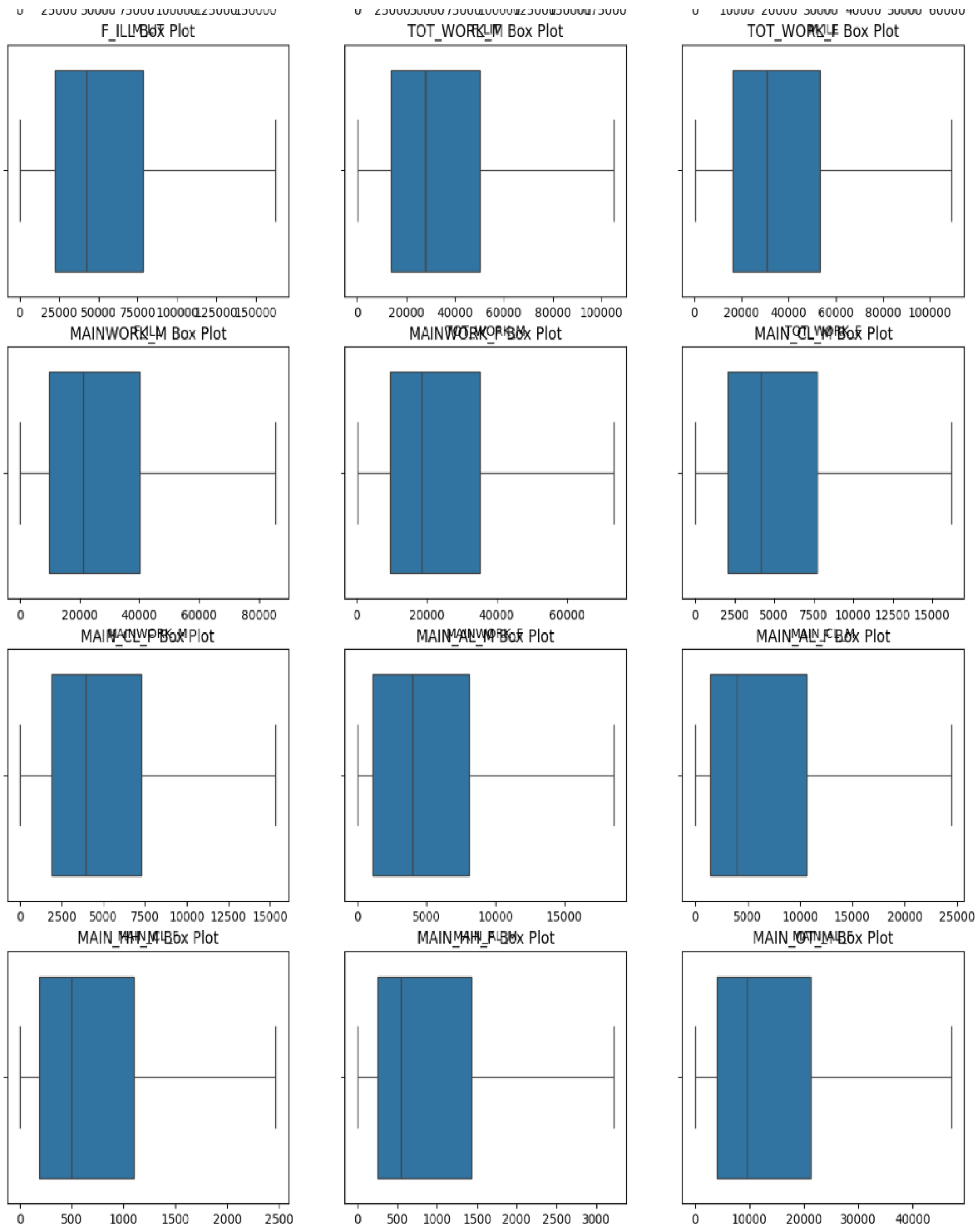
```
def treat_outliers(column):
  q1 = column.quantile(0.25)
  q3 = column.quantile(0.75)
  iqr = q3 - q1
  lower_bound = q1 - 1.5 * iqr
  upper_bound = q3 + 1.5 * iqr
  column = np.where(column > upper_bound, upper_bound, column)
  column = np.where(column < lower_bound, lower_bound, column)
  return column
```

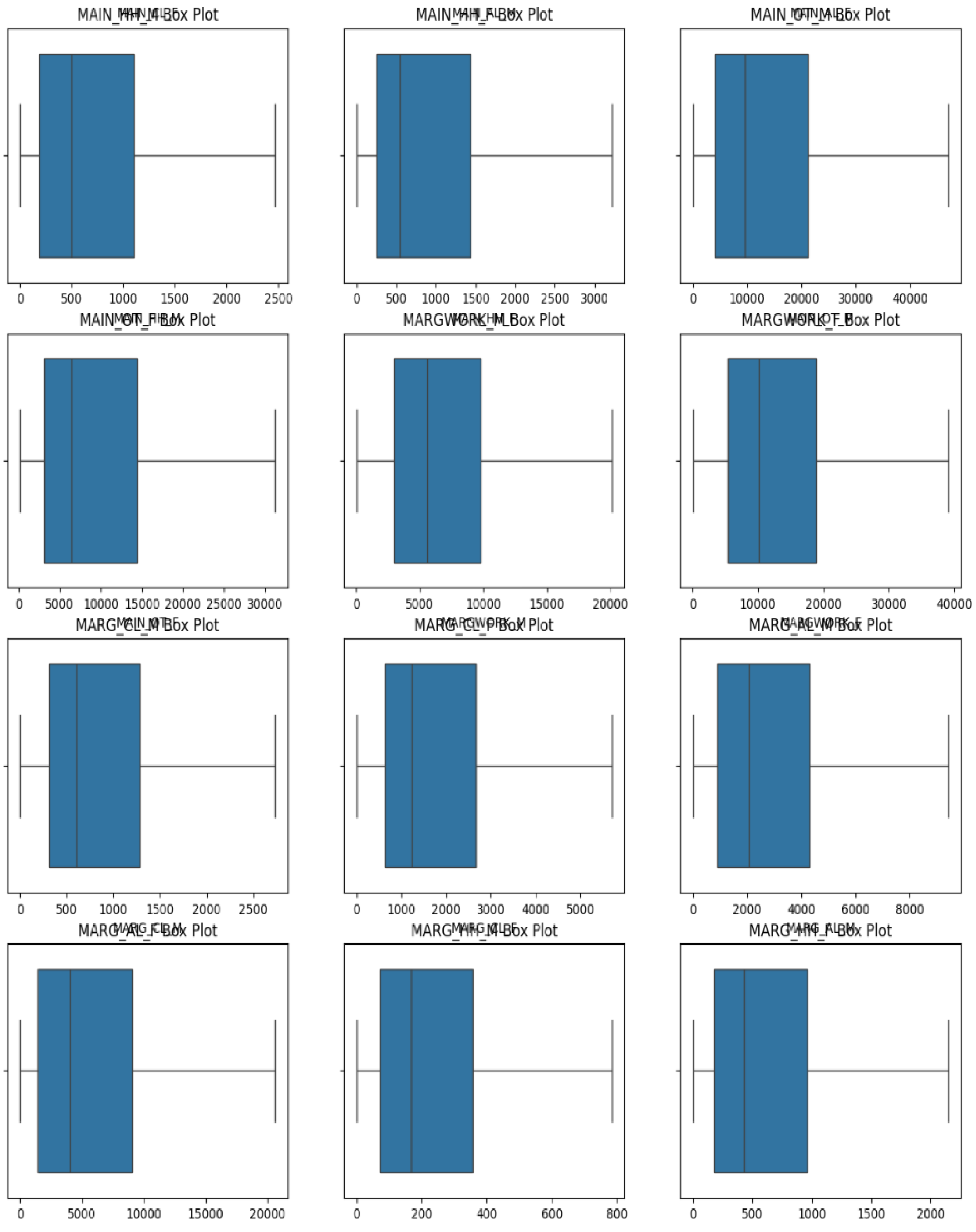
Here we determine the inter quartile range and remove all the data that lies outside the lower bound and upper bound by running the function for our dataframe.

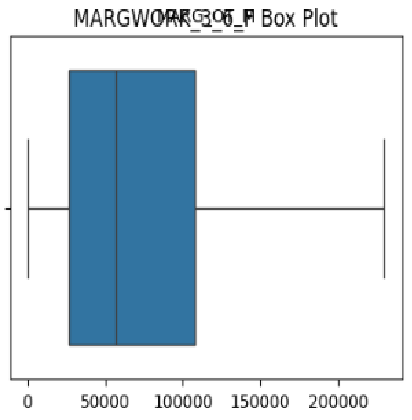
```
for column in df.select_dtypes(include=['number']):
  df[column] = treat_outliers(df[column])
```

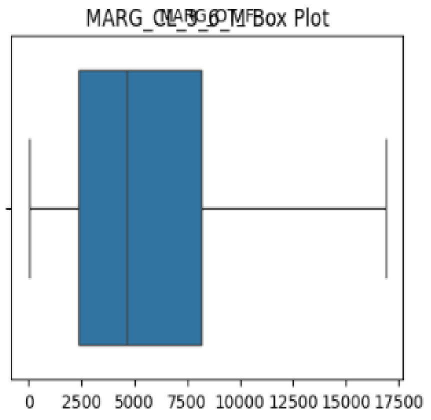
Now if we recheck the box plots we'll be able to see all the outliers treated.

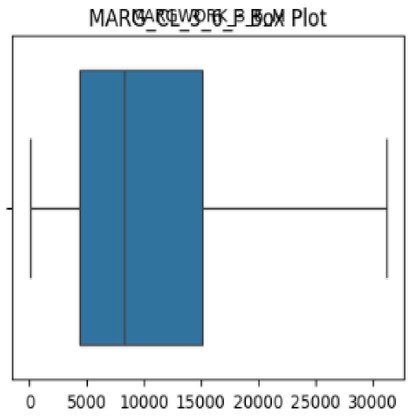


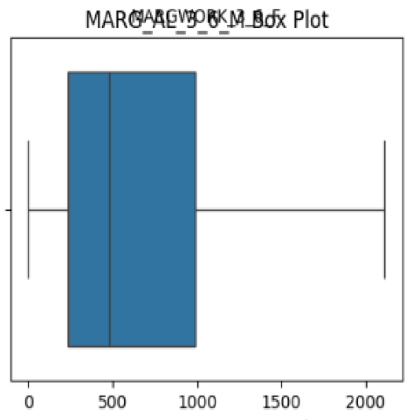

0 25000 50000 75000 100000 125000 150000
0 25000 50000 75000 100000 125000 150000 175000
0 10000 20000 30000 40000 50000 60000
M_ILLIT
F_ILLIT
M_ILLIT
F_ILLIT Box Plot
0 25000 50000 75000 100000 125000 150000
F_ILLIT
TOT_WORK_M Box Plot
0 20000 40000 60000 80000 100000
TOT_WORK_M
TOT_WORK_F Box Plot
0 20000 40000 60000 80000 100000
TOT_WORK_F
MAINWORK_M Box Plot
0 20000 40000 60000 80000
MAINWORK_M
MAINWORK_F Box Plot
0 20000 40000 60000
MAINWORK_F
MAIN_CL_M Box Plot
0 2500 5000 7500 10000 12500 15000
MAIN_CL_M
MAIN_CL_F Box Plot
0 2500 5000 7500 10000 12500 15000
MAIN_CL_F
MAIN_AL_M Box Plot
0 5000 10000 15000
MAIN_AL_M
MAIN_AL_F Box Plot
0 5000 10000 15000 20000 25000
MAIN_AL_F
MAIN_HH_M Box Plot
0 500 1000 1500 2000 2500
MAIN_HH_M
MAIN_HH_F Box Plot
0 500 1000 1500 2000 2500 3000
MAIN_HH_F
MAIN_OT_M Box Plot
0 10000 20000 30000 40000

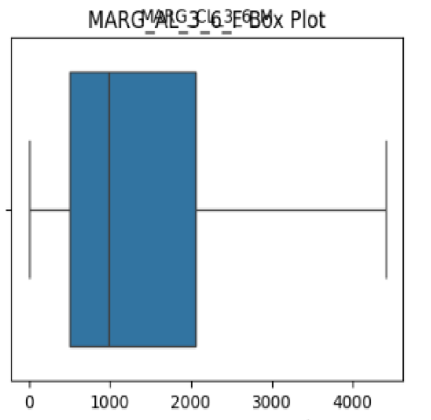

MAIN_HH_M Box Plot
0 500 1000 1500 2000 2500
MAIN_HH_F Box Plot
0 500 1000 1500 2000 2500 3000
MAIN_OT_M Box Plot
0 10000 20000 30000 40000
MAIN_OT_F Box Plot
0 5000 10000 15000 20000 25000 30000
MARGWORK_M Box Plot
0 5000 10000 15000 20000
MARGWORK_F Box Plot
0 10000 20000 30000 40000
MARG_CL_M Box Plot
0 500 1000 1500 2000 2500
MARG_CL_F Box Plot
0 1000 2000 3000 4000 5000
MARG_AL_M Box Plot
0 2000 4000 6000 8000
MARG_AL_F Box Plot
0 5000 10000 15000 20000
MARG_HH_M Box Plot
0 200 400 600 800
MARG_HH_F Box Plot
0 500 1000 1500 2000

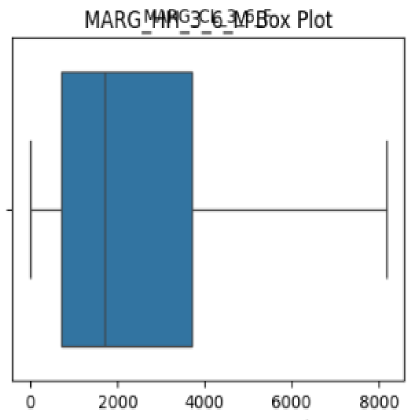

MARGWORK_3_6_F Box Plot
MARG_OT_F
0
50000
100000
150000
200000

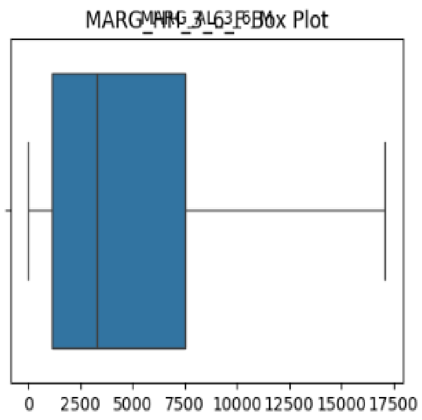

MARG_CL_3_6_M Box Plot
MARG_OT_M
0
2500
5000
7500
10000
12500
15000
17500

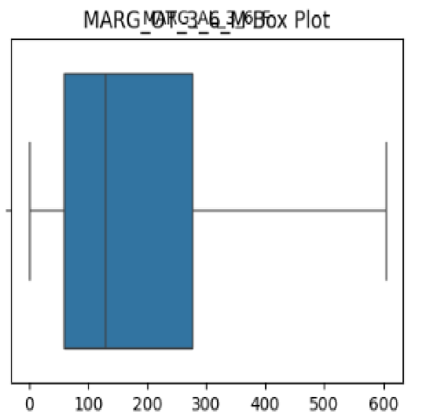

MARG_CL_3_6_F Box Plot
MARGWORK_3_6_M
0
5000
10000
15000
20000
25000
30000

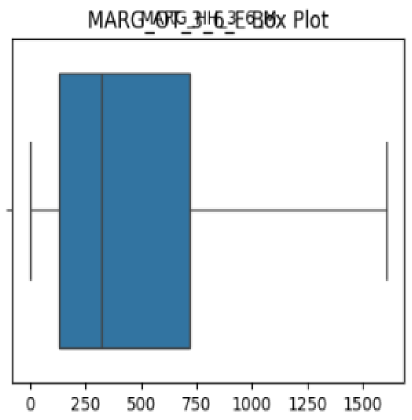

MARG_AL_3_6_M Box Plot
MARGWORK_3_6_F
0
500
1000
1500
2000

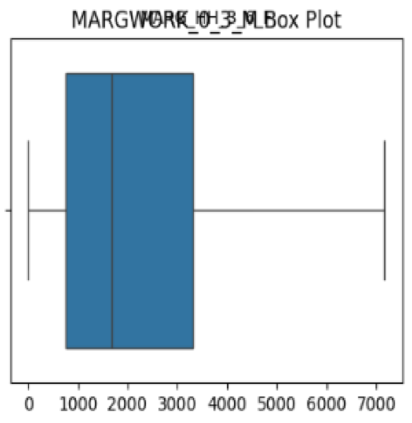

MARG_AL_3_6_F Box Plot
MARG_CL_3_6_M
0
1000
2000
3000
4000

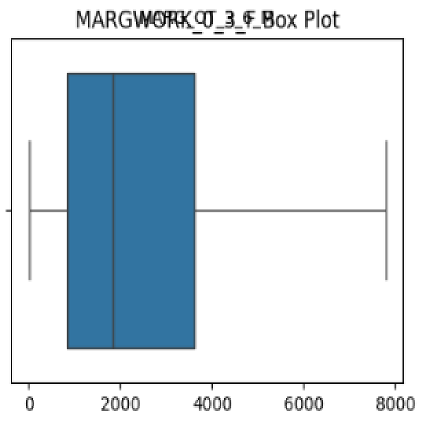

MARG_HH_3_6_M Box Plot
MARG_CL_3_6_F
0
2000
4000
6000
8000

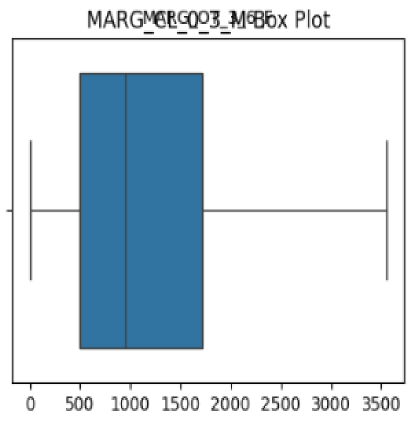

MARG_HH_3_6_F Box Plot
MARG_AL_3_6_M
0
2500
5000
7500
10000
12500
15000
17500

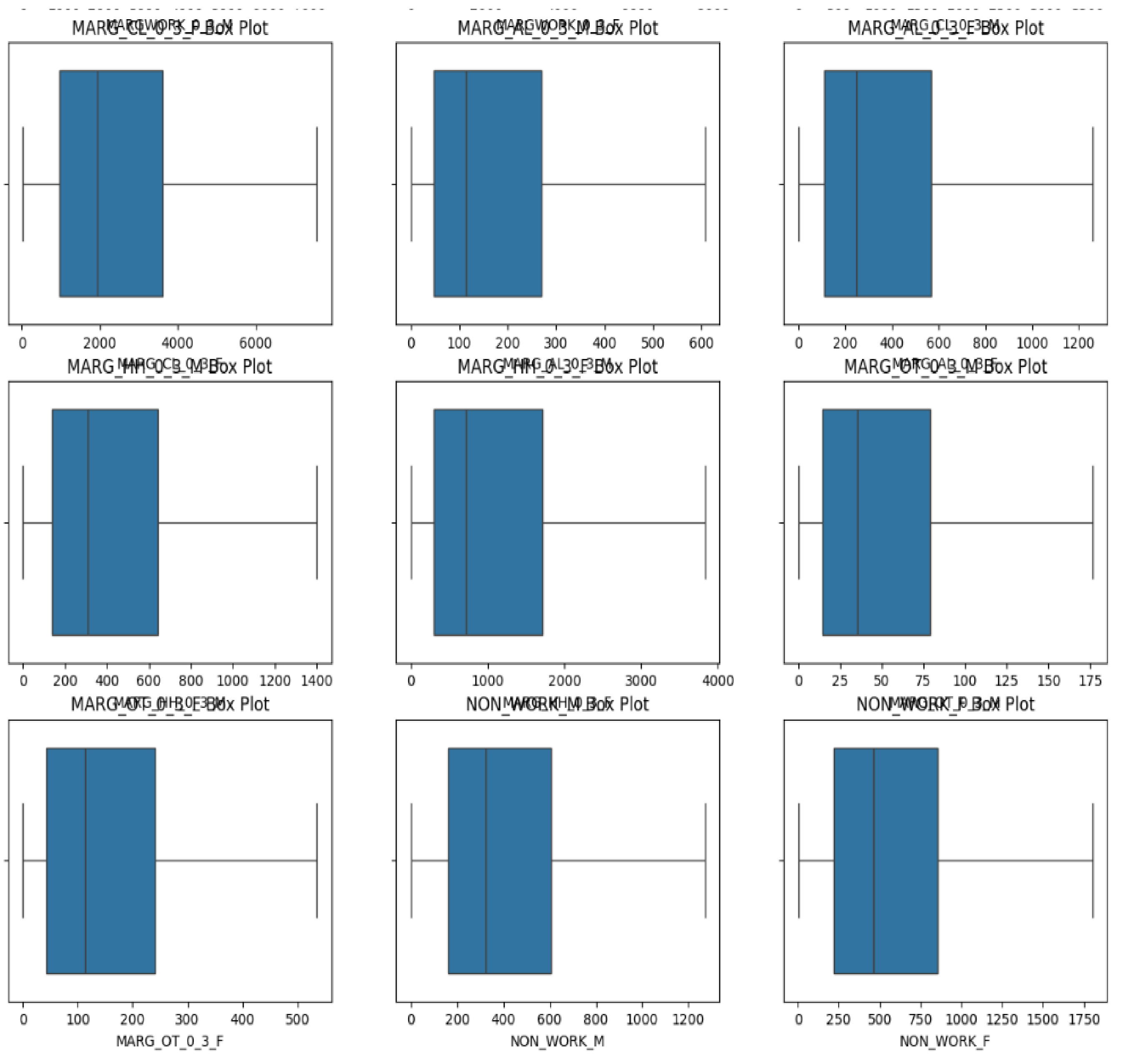

MARG_OT_3_6_M Box Plot
MARG_AL_3_6_F
0
100
200
300
400
500
600


MARG_OT_3_6_F Box Plot
MARG_HH_3_6_M
0
250
500
750
1000
1250
1500


MARGWORK_0_3_M Box Plot
MARG_HH_3_6_F
0
1000
2000
3000
4000
5000
6000
7000


MARGWORK_0_3_F Box Plot
MARG_OT_3_6_M
0
2000
4000
6000
8000


MARG_CL_0_3_M Box Plot
MARG_OT_3_6_F
0
500
1000
1500
2000
2500
3000
3500



All the outliers are eliminated by processing the dataframe through the function.

Now we can proceed to scaling the data before principal component analysis.

Feature engineering - Scaling

We'll be using MinMaxScaler from the sklearn library to get the values of all the variables scaled and transformed

Scaling the data, in the context of data analysis and machine learning, refers to the process of transforming numerical variables to a standard range or distribution. This is done to ensure that all the variables have the same scale, making it easier to compare and analyze them.

MinMaxScaler is a scaling technique used to transform your data so that it falls within a specific range, typically (0, 1), but you can customize the range using the feature_range parameter. Here's a brief theoretical explanation of how it works:

The MinMaxScaler() scales the data using the following formula for each feature (variable) individually:

$X_{scaled} = (X - X_{min})/(X_{max} - X_{min})$
X_{scaled} is the scaled value of the feature.
X is the original value of the feature.
X_{min} is the minimum value of the feature in your dataset.
X_{max} is the maximum value of the feature in your dataset.

```
from sklearn.preprocessing import MinMaxScaler
# Creating a Min-Max scaler
scale = MinMaxScaler()
# Fitting the scaler to data df (compute min and max values for each feature)
scale.fit(df)
# Transform your data using the scaler and name it as sc_df(an array)
sc_df = scale.transform(df)
```

Now we proceed with the array towards principal component analysis function & its applications

Principal component analysis

```
# Creating a PCA model named pca
pca = PCA()
# Fitting the PCA model to your scaled data
pca.fit(sc_df)
```

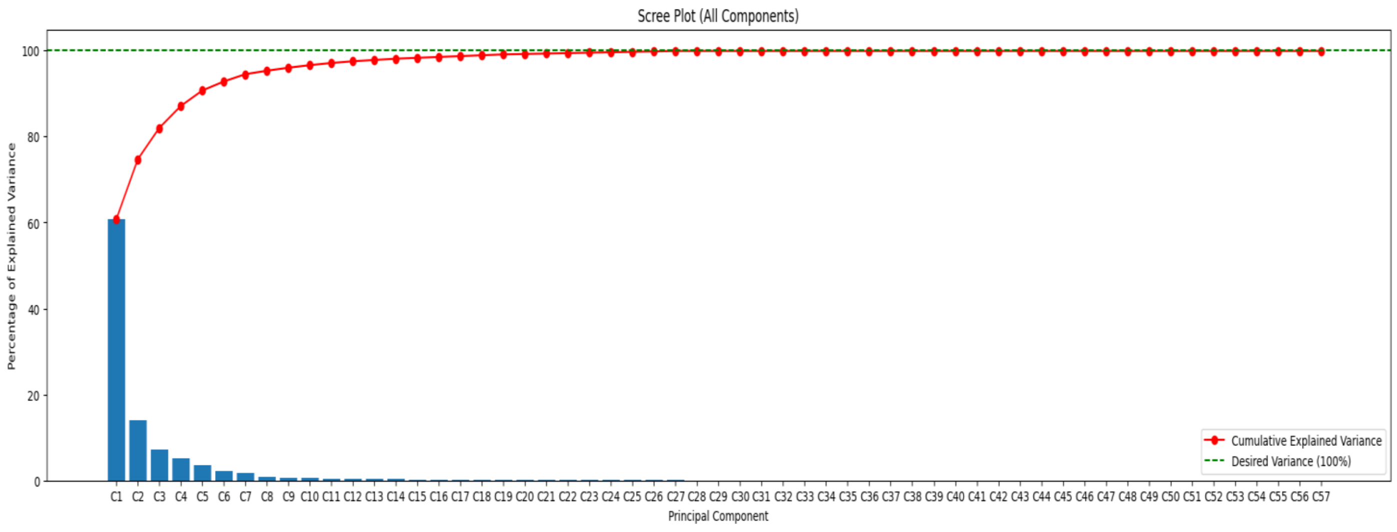
PCA(): This line creates an instance of the PCA (Principal Component Analysis) model from scikit-learn. PCA is a dimensionality reduction technique that helps in transforming the original features into a new set of linearly uncorrelated variables called principal components.

pca.fit(sc_df): Here, we are fitting the PCA model to your scaled data, which is stored in the variable sc_df. The fit method computes the principal components based on the variance-covariance matrix of your data. After this step, the PCA model is trained and ready to be used for dimensionality reduction.

PCA is often used for reducing the dimensionality of your data while retaining most of its important information. After fitting the PCA model, one can analyse the explained variance to understand how much of the original data's variance is captured by each principal component(s).

Plotting the Principal components

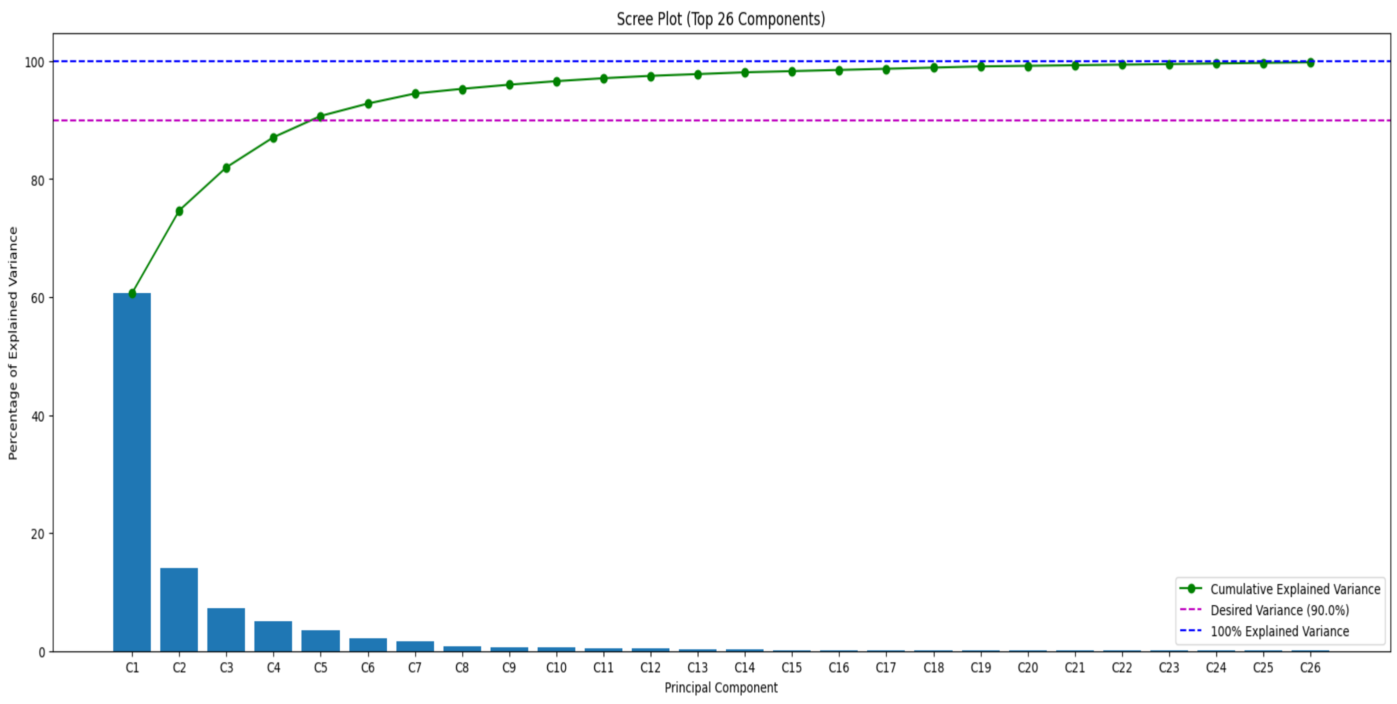
Now we'll plot the amount of variance each of the principal components show using bar plots



To reach the plot we've performed the following steps

1. Setting the desired variance threshold to 100%.
2. Calculating the percentage of explained variance for each component.
3. Determining the actual number of components
4. Creating labels for the principal components (e.g., C1, C2, ...)
5. Adding a Cumulative Explained Variance Line(combined variance till Cn th component) and a desired variance line(100%).

Simultaneously we've created by another plot using the same steps as above and by adjusting number of principal components actually needed to show 100% variance.



As evident from the plots above

- 100% of the variance is shown by first 26 principle components.
- While more than 90% of the variance is shown by first 5 principle components alone.

Every component contains all our earlier variables in some contributing proportion.

We'll assign the data to another dataframe named loading_df

Getting the loadings for the first 5 components

```
n_components = 5
loadings = pca.components_[:n_components]
# Creating a DataFrame to store the loadings for the first 5 components
loading_df = pd.DataFrame(loadings, columns=df.columns).T
loading_df.columns = [f'Loading_Component_{i + 1}' for i in range(n_components)]
print(loading_df)
```

n_components = 5: We are setting the number of principal components to retain as 5. This defines how many of the most important dimensions you want to keep after performing PCA.

loadings = pca.components_[:n_components]: We are extracting the loadings of the first 5 principal components from the PCA model. Loadings represent the direction and strength of each original feature's influence on the principal components.

loading_df = pd.DataFrame(loadings, columns=df.columns).T: We are creating a DataFrame called loading_df to store the loadings of the first 5 principal components. Each row corresponds to a feature, and each column corresponds to one of the 5 principal components.

loading_df.columns = [f'Loading_Component_{i + 1}' for i in range(n_components)]: We are assigning meaningful column names to loading_df to indicate that each column represents the loadings for a specific principal component.

This provides insight into how each original feature contributes to these components, which is useful for feature selection and interpretation in PCA analysis.
Here all the columns are the principle components and the rows are our variables of the original dataset df

Feature Selection:

Now we'll proceeding to dimensionality reduction using by selecting top 10 features or variables from the 5 principle components which show more than 90% of the variance.

```
# Getting the loadings from PCA
loadings = pca.components_
# Calculating the importance of each original feature for each component
feature_importance = abs(loadings.T)
# Assuming you want to select the top 'k' features for each component
k = 9
# Replacing with the desired number of features to select
selected_features = []
for i in range(n_components):
  component_importance = feature_importance[i]
  top_k_indices = component_importance.argsort()[-k:][::-1]
  selected_features.extend(top_k_indices)
# Removing duplicate feature indices
selected_features = list(set(selected_features))
# Creating a new DataFrame 'df_new' with the selected features
df_new = df.iloc[:, selected_features]
df_new.head()
```

loadings = pca.components_: You're extracting the loadings from the PCA model, which represent the relationship between the original features and the principal components. These loadings indicate how much each original feature contributes to each principal component.

feature_importance = abs(loadings.T): You're calculating the importance of each original feature for each component by taking the absolute values of the loadings and transposing the matrix to have components as rows and features as columns.

k = 10: You're defining the number of top features to select for each component. In this case, you've set k to 10, meaning you want to select the top 10 most important features for each principal component.

selected_features = []: You're initializing an empty list to store the indices of the selected features.

Inside the loop:

component_importance = feature_importance[i]: You're extracting the importance scores for the current principal component.

top_k_indices = component_importance.argsort()[-k:][::-1]: You're finding the indices of the top-k features for the current component based on their importance scores.

selected_features.extend(top_k_indices): You're adding these indices to the list of selected features.

selected_features = list(set(selected_features)): You're removing duplicate feature indices by converting the list to a set and back to a list.

df_new = df.iloc[:, selected_features]: You're creating a new DataFrame called df_new by selecting only the columns (features) from the original DataFrame df that correspond to the selected feature indices.

The new dataframe contains only 5 features out of the previous 57 numerical features reducing the number of dimensions by more than half.

Finally we export the dataframe by concatenating the identifiers named as id (state code, district code, state name, area name) which were earlier dropped as they only served as identifiers. The new file name is 'data_post_PCA.csv'.

Conclusion:

Thus, the Principal Component analysis has been performed on the census data set and dimensionality reduction has been achieved by eliminating more than 50% of the data based on variance shown by 90% of the data.

The data which has 5 principal components labelled as 'loading_df' shows 90% of variance.

All in all there were 26 components in total required to show 100% variance.

Top 10 features of the all 5 principal components which show more than 90% of variance in the data have been exported as a new dataset at the end of the code.

References:

1. Masterclasses by Netali Agarwal & Kunaal Naik as uploaded on LMS.
2. chat.openai.com
3. statquest.org