TAYLOR’S UWE DUAL AWARDS PROGRAMMES

JANUARY 2024 SEMESTER

Data Analytics and Machine Learning

(ITS69304)

Module Head – Mr. Anmol Adhikari

Individual Assignment

DUE DATE: 17th Feb 2024 via MyTIMeS

STUDENT DECLARATION

1. ***I confirm that I am aware of the University’s Regulation Governing Cheating in a University Test and Assignment and of the guidance issued by the School of Computing and IT concerning plagiarism and proper academic practice and that the assessed work now submitted is in accordance with this regulation and guidance.***
2. ***I understand that, unless already agreed with the School of Computing and IT, assessed work may not be submitted that has previously been submitted, either in whole or in part, at this or any other institution.***
3. ***I recognise that should evidence emerge that my work fails to comply with either of the above declarations, then I may be liable to proceedings under Regulation.***

No	Student Name	Student ID	Date	Signature	Score
1	Adarsh Sthapit	0355409	17th Feb		

Contents

Introduction

The livestock production is a major enterprise in Nepal’s agricultural economy and landscape, significantly contributing to the economy, food security and overall rural development. Due to its diverse topography and agro-climate, Nepal has a wide range of environments that suit a large variety of animals: from cattle to buffalo to chicken to goat to pig – providing essential meat, dairy and other animal products – Nepal also produces a wide range of agricultural commodities including the likes of oilseed, lentil, pea and wheat – and, of course, rice, maize and wheat.

This document presents and analyzes eight different datasets, each one focused on a particular aspect of Nepal’s livestock and agricultural production. Together, they cover the population of different livestock species as well as the production of meat, milk, eggs, wool, and other essential commodities. The datasets have been carefully chosen to shed light on the quantity, geographical distribution, and trends of different livestock and agricultural variables.

Improved animal husbandry, increased livestock and commodity productivity, and sustainable agricultural practices have all received more attention in recent years. These initiatives have been aided by technological advancements, international cooperation, and government initiatives.

Through the exploration, cleaning, and analysis of these datasets, this documentation will guide the process and provide insight into the complex dynamics of livestock and commodity production in Nepal. In order to fully comprehend the patterns, correlations, and trends found in the datasets, the documentation will also include statistical insights and visualizations. This will help readers gain a thorough understanding of the roles that these sectors play in Nepal's agricultural and economic landscape.

Problem Statement

Despite the fact that Nepal’s economy is agricultural, the production of both crops and animals faces many challenges. Using various datasets, agricultural commodities like rice, maize, wheat, pulses, oilseeds, cattle, buffalo, poutlry, goats, and pigs were studied to get a snapshot of Nepal’s situation in those sectors. One of the main challenges is the fact that many famers use traditional farming methods, which reduces the their efficiency and productivity. Moreover, poor infrastructure — such as lack of transportation and storage facilities — can limit access to markets and increase post harvest losses. The country’s vulnerability to natural disasters like earthquakes and floods is a huge threat to livestock and agricultural productivity.

Problem Solution

Overcoming these obstacles requires a comprehensive approach — one that includes, among other things, boosting agricultural output in ways that protect the environment; promoting environmentally friendly farming practices and modern farming techniques can help to boost output even as they protect our fragile environment. Reducing post-harvest losses and improving market access is possible, but not without strategic investment in infrastructure, including processing plants, transportation networks and storage facilities. And, of course, we would also be able to make way for stronger disaster preparedness by employing more resilient farming methods and diminished losses if we also made considerable efforts to lessen the impact of natural hazards on crops and livestock.

In order to equip farmers, trainers, and other stakeholders with the information and abilities necessary for efficient management of livestock and commodities, capacity-building initiatives are essential. Additionally, encouraging the use of technology in farming practices—such as tools for data-driven decision-making—can greatly improve productivity and efficiency. In order to ensure the long-term success of these crucial sectors, a more resilient, sustainable, and productive livestock and commodity production system in Nepal is being created through this cooperative effort involving the government, non-governmental organizations, the private sector, and local communities.

Objective

The main aim of this documentation is to give readers serious stats on Nepal's livestock and commodity production environment through the analysis and display of datasets associated with multiple aspects of livestock such as milking cows, egg hens, meat production, horses/asses, wool, etc., with the idea that patterns of production, variations by region, and other insights would be evident.

The documentation tries to serve as a useful tool for researchers, policy makers and stakeholders interested in Nepal's agricultural and economic development through exploratory data analysis, visualization and a discussion of data cleaning and processing. The ultimate goal is to support initiatives that encourage the increase of livestock and cash crop production in Nepal, that encourage responsible resource and land use and that support the making of well informed decisions.

Dataset overview

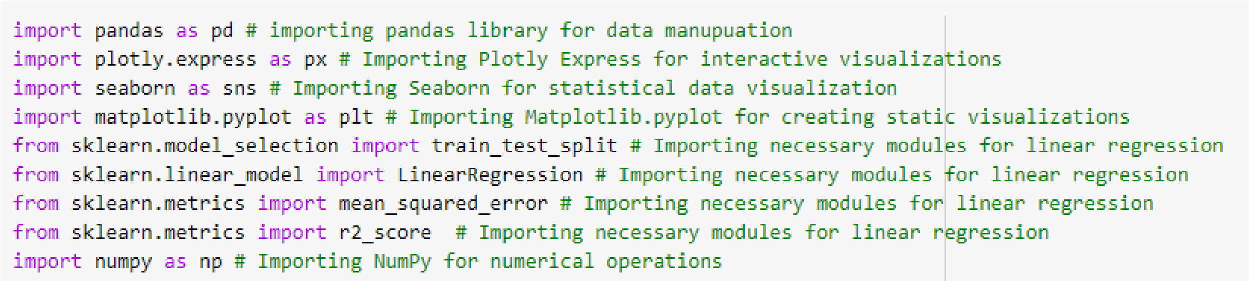
The dataset used in this documentation is a collection of various datasets on Comodity and Livestock Production of Nepal. The datasets has a wide range of topics from the livestock including the population of Horses and asses, the amount of milk, meat, wool, meat (pork, beef, mutton), and eggs produced, the population of yaks, naks, and chauri in various districts of Nepal.

Each dataset provides a quick look at certain fields of livestock and agriculture categories such as the condition of the country's production environment through the number of animals, quantity of production and other relevant metrics. It includes data regarding different districts, thus offering a view on the production of commodities and livestock.

To guarantee the accuracy and dependability of the data, a comprehensive exploration, cleaning, and processing of these datasets will be performed. The data exploration procedure, important metrics visualization, and discussions of the correlation between different variables will all be covered in detail in the ensuing sections of this documentation.

Implementation

Importing Libraries



```
import pandas as pd # importing pandas library for data manupuation
import plotly.express as px # Importing Plotly Express for interactive visualizations
import seaborn as sns # Importing Seaborn for statistical data visualization
import matplotlib.pyplot as plt # Importing Matplotlib.pyplot for creating static visualizations
from sklearn.model_selection import train_test_split # Importing necessary modules for linear regression
from sklearn.linear_model import LinearRegression # Importing necessary modules for linear regression
from sklearn.metrics import mean_squared_error # Importing necessary modules for linear regression
from sklearn.metrics import r2_score  # Importing necessary modules for linear regression
import numpy as np # Importing NumPy for numerical operations
```

i. pandas (pd):

Pandas is a powerful Python data analysis and manipulation library. It provides data structures to make structured data handling and manipulation like, a quick activity. DataFrame helps to perform such tasks.

ii. plotly.express (px):

Plotly Express is a high-level Python visualization library. Using this, we will be able to make highly interactive and publication-quality interactive plots. With this charts can be rendered very quickly.

iii. seaborn (sns):

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics

iv. matplotlib.pyplot (plt):

Python users can create static, interactive, and animated plots with the help of the extensive Matplotlib library. A MATLAB-style interface for making basic plots and charts is offered by the pyplot module.

v. train_test_split:

'train_test_split' is a function in Sklearn model selection module for Split arrays or matrices into random train and test subsets. It is used from model selection for splitting data-set.

vi. LinearRegression:

A simple but effective statistical approach to modeling the relationship between a dependent variable and one or more independent variables is given by Linear Regression. LinearRegression class is used to implement this model.

vii. mean_squared_error:

The average of the square of the difference between the actual and predicted values is known as Mean Squared Error. It's a measure of how well the model is doing and is widely used to measure the performance of regression models.

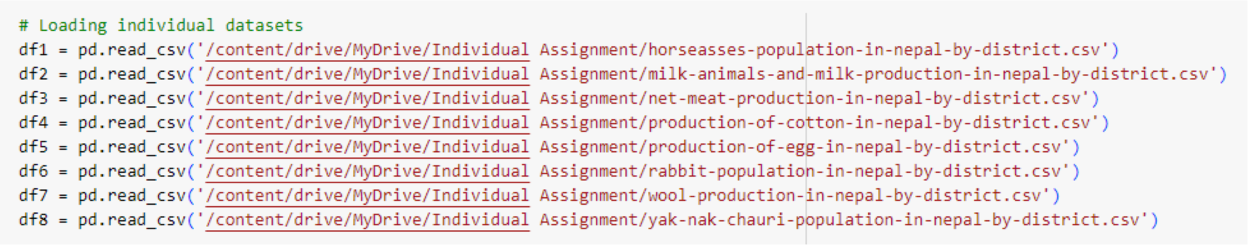
viii. r2_score:

R-squared (R2) score is statistically a measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. It is an additional measure to look for the goodness of fit for a regression model.

ix. numpy (np):

For scientific computing in Python, NumPy is an essential package. Along with the mathematical functions needed to effectively work on these arrays, it supports large, multi-dimensional matrices and arrays.

Dataset Loading



```
# Loading individual datasets
df1 = pd.read_csv('/content/drive/MyDrive/Individual Assignment/horseasses-population-in-nepal-by-district.csv')
df2 = pd.read_csv('/content/drive/MyDrive/Individual Assignment/milk-animals-and-milk-production-in-nepal-by-district.csv')
df3 = pd.read_csv('/content/drive/MyDrive/Individual Assignment/net-meat-production-in-nepal-by-district.csv')
df4 = pd.read_csv('/content/drive/MyDrive/Individual Assignment/production-of-cotton-in-nepal-by-district.csv')
df5 = pd.read_csv('/content/drive/MyDrive/Individual Assignment/production-of-egg-in-nepal-by-district.csv')
df6 = pd.read_csv('/content/drive/MyDrive/Individual Assignment/rabbit-population-in-nepal-by-district.csv')
df7 = pd.read_csv('/content/drive/MyDrive/Individual Assignment/wool-production-in-nepal-by-district.csv')
df8 = pd.read_csv('/content/drive/MyDrive/Individual Assignment/yak-nak-chauri-population-in-nepal-by-district.csv')
```

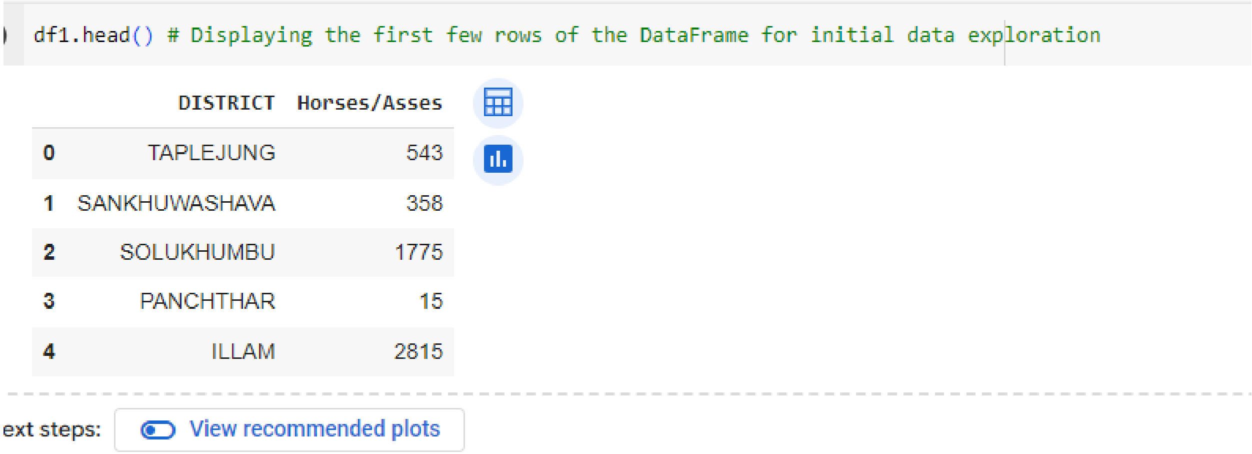
The read_csv() function in Pandas is used to load the dataset into distinct DataFrames (df1 to df8). Different districts in Nepal's livestock and commodity production are represented by distinct DataFrames:

1. Horse and Asses Population (df1):
2. Milk Animals and Milk Production (df2):
3. Net Meat Production (df3):
4. Cotton Production (df4):
5. Egg Production (df5):
6. Rabbit Population (df6):
7. Wool Production (df7):
8. Yak, Nak, Chauri Population (df8):

Data Exploration

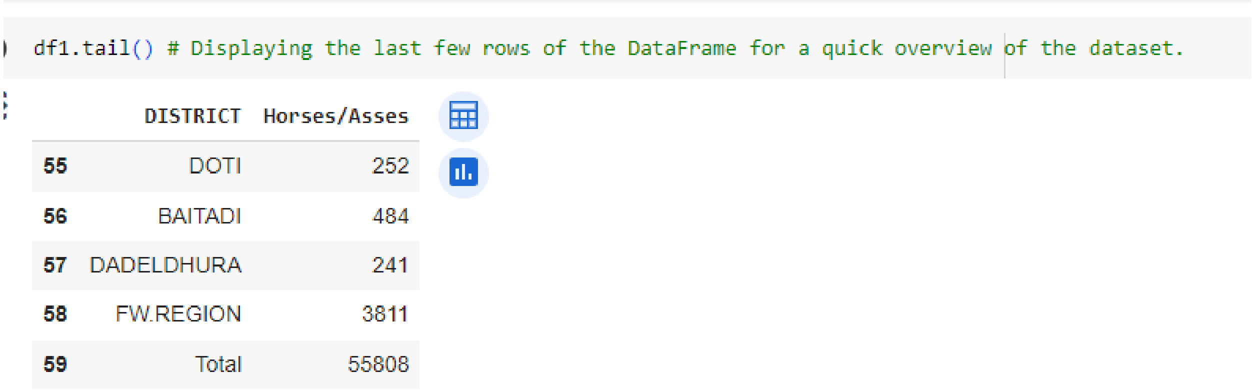
1. Horse and Asses Population

- Head and Tail Exploration of the Dataset:



```
df1.head() # Displaying the first few rows of the DataFrame for initial data exploration
```

	DISTRICT	Horses/Asses
0	TAPLEJUNG	543
1	SANKHUWASHAVA	358
2	SOLUKHUMBU	1775
3	PANCHTHAR	15
4	ILLAM	2815



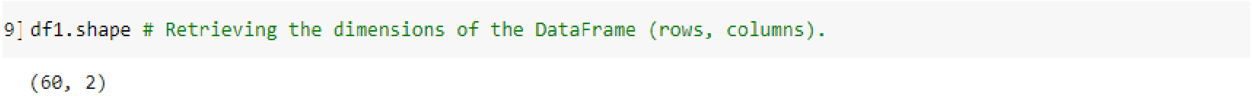
```
df1.tail() # Displaying the last few rows of the DataFrame for a quick overview of the dataset.
```

	DISTRICT	Horses/Asses
55	DOTI	252
56	BAITADI	484
57	DADELDHURA	241
58	FW.REGION	3811
59	Total	55808

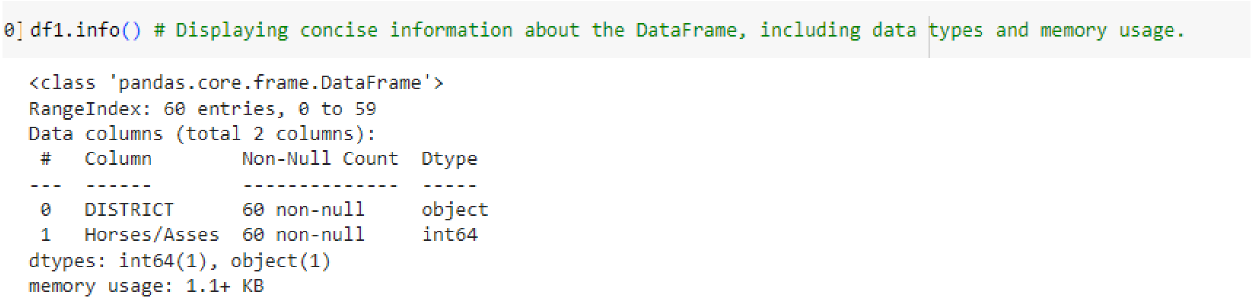
Upon first inspection of the dataframe df1, which features the population of horses and asses in Nepal by district, the head section displayed the first few rows, revealing important columns such as 'DISTRICT' and 'Horses/Asses.' They conveyed the basic structure and content of the dataset, which made it easy to comprehend.

Simultaneously, the last section displayed the last few rows, letting one have a comprehensive look of the dataset's shape all the way through. It was a full round-up that really delivers a first good impression of the dataset's general properties!

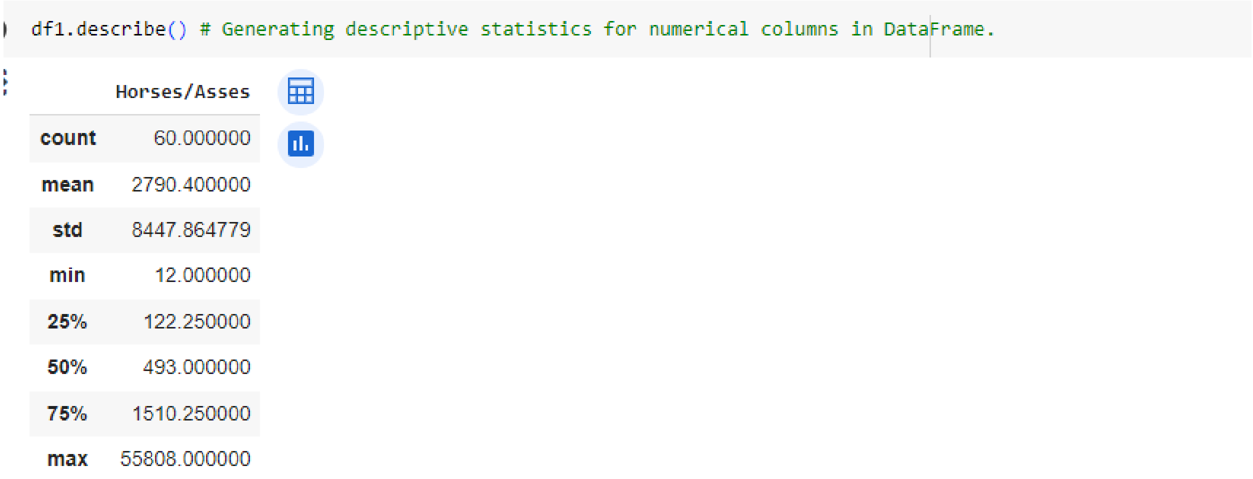
- Dimensions and Descriptive Information:



```
9] df1.shape # Retrieving the dimensions of the DataFrame (rows, columns).

  (60, 2)
```



```
0] df1.info() # Displaying concise information about the DataFrame, including data types and memory usage.

  <class 'pandas.core.frame.DataFrame'>
  RangeIndex: 60 entries, 0 to 59
  Data columns (total 2 columns):
   #   Column        Non-Null Count  Dtype 
  ---  ------        --------------  ----- 
   0   DISTRICT      60 non-null     object
   1   Horses/Asses  60 non-null     int64 
  dtypes: int64(1), object(1)
  memory usage: 1.1+ KB
```



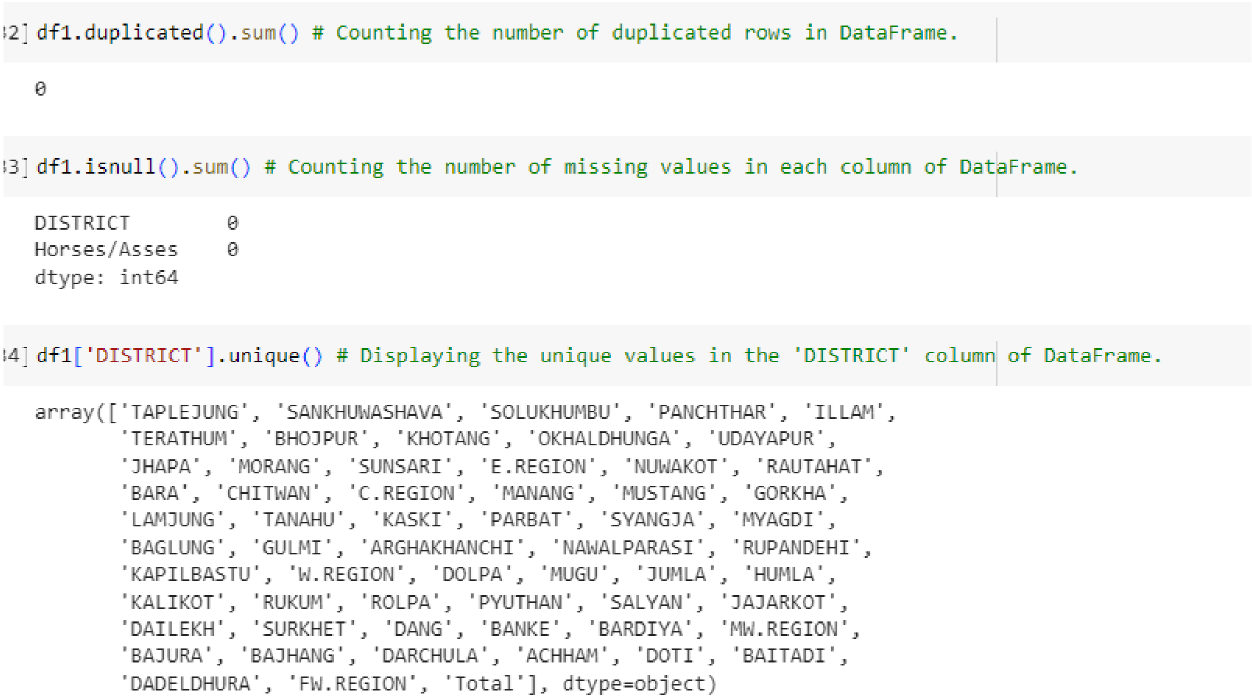
```
) df1.describe() # Generating descriptive statistics for numerical columns in DataFrame.
```

	Horses/Asses
count	60.000000
mean	2790.400000
std	8447.864779
min	12.000000
25%	122.250000
50%	493.000000
75%	1510.250000
max	55808.000000

The shape attribute gives a general idea of the structure of the dataset. The current dataset consists of 60 rows and 2 columns.

This is a good general overview of the dataset, which can also be augmented using describe. The describe function produces a statistical summary of the data. It helps to summarize the numerical columns. The statistical information includes mean, standard deviation, minimum value, maximum value as well as the values for the three quartiles.

- Duplicate Rows, Missing Values, and Unique Districts:



```
2] df1.duplicated().sum() # Counting the number of duplicated rows in DataFrame.

   0
```

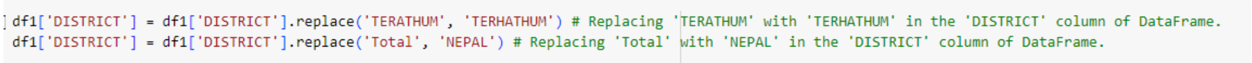
```
3] df1.isnull().sum() # Counting the number of missing values in each column of DataFrame.

   DISTRICT        0
   Horses/Asses    0
   dtype: int64
```

```
4] df1['DISTRICT'].unique() # Displaying the unique values in the 'DISTRICT' column of DataFrame.

   array(['TAPLEJUNG', 'SANKHUWASHAVA', 'SOLUKHUMBU', 'PANCHTHAR', 'ILLAM',
          'TERATHUM', 'BHOJPUR', 'KHOTANG', 'OKHALDHUNGA', 'UDAYAPUR',
          'JHAPA', 'MORANG', 'SUNSARI', 'E.REGION', 'NUWAKOT', 'RAUTAHAT',
          'BARA', 'CHITWAN', 'C.REGION', 'MANANG', 'MUSTANG', 'GORKHA',
          'LAMJUNG', 'TANAHU', 'KASKI', 'PARBAT', 'SYANGJA', 'MYAGDI',
          'BAGLUNG', 'GULMI', 'ARGHAKHANCHI', 'NAWALPARASI', 'RUPANDEHI',
          'KAPILBASTU', 'W.REGION', 'DOLPA', 'MUGU', 'JUMLA', 'HUMLA',
          'KALIKOT', 'RUKUM', 'ROLPA', 'PYUTHAN', 'SALYAN', 'JAJARKOT',
          'DAILEKH', 'SURKHET', 'DANG', 'BANKE', 'BARDIYA', 'MW.REGION',
          'BAJURA', 'BAJHANG', 'DARCHULA', 'ACHHAM', 'DOTI', 'BAITADI',
          'DADELDHURA', 'FW.REGION', 'Total'], dtype=object)
```

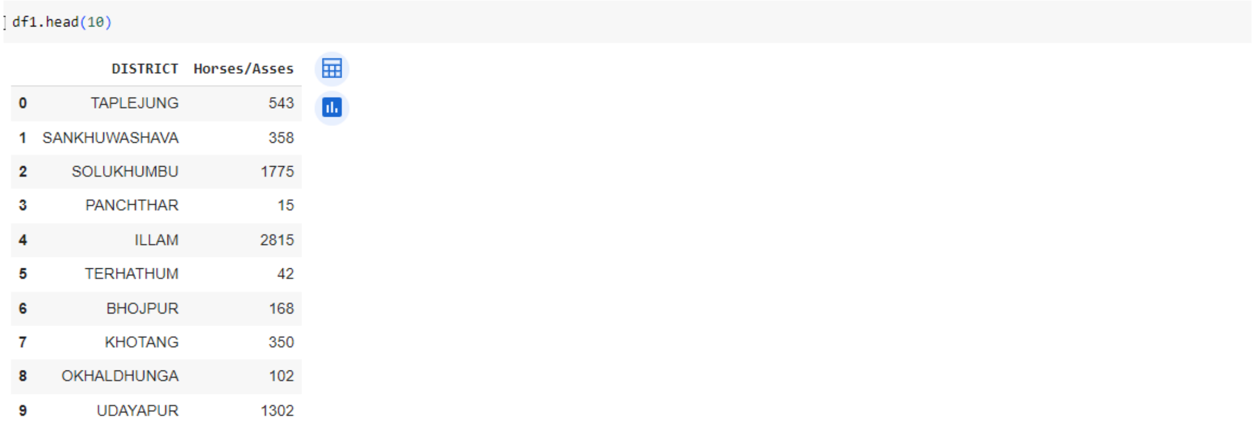
df1.duplicated() is used to search the dataset for duplicate rows.sum() aids in locating and managing any possible data redundancy.

df1.isnull() is used to explore missing values. The extent of data completeness can be understood using sum().
The use of df1['DISTRICT'] to examine unique values in a categorical column, in this case 'DISTRICT'. The distinct entries in that column can be quickly reviewed with the help of unique().

- District Names Standardization:

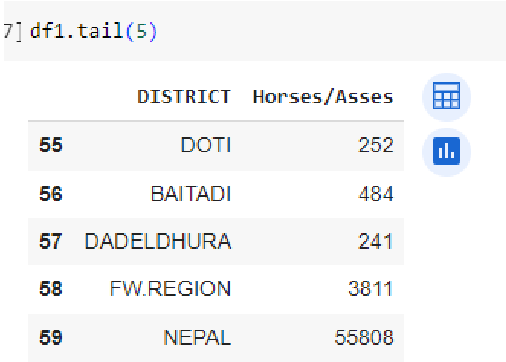


```
] df1['DISTRICT'] = df1['DISTRICT'].replace('TERATHUM', 'TERHATHUM') # Replacing 'TERATHUM' with 'TERHATHUM' in the 'DISTRICT' column of DataFrame.
  df1['DISTRICT'] = df1['DISTRICT'].replace('Total', 'NEPAL') # Replacing 'Total' with 'NEPAL' in the 'DISTRICT' column of DataFrame.
```



```
] df1.head(10)
```

```
        DISTRICT  Horses/Asses
0      TAPLEJUNG           543
1  SANKHUWASHAVA           358
2     SOLUKHUMBU          1775
3     PANCHTHAR             15
4         ILLAM           2815
5      TERHATHUM            42
6        BHOJPUR           168
7       KHOTANG            350
8    OKHALDHUNGA           102
9       UDAYAPUR          1302
```



```
7] df1.tail(5)
```

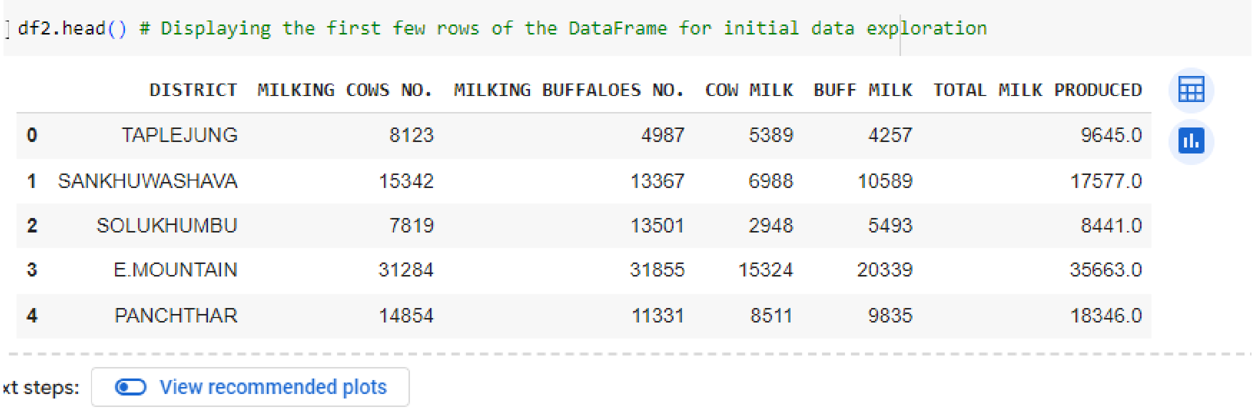
```
       DISTRICT  Horses/Asses
55         DOTI           252
56      BAITADI           484
57   DADELDHURA           241
58    FW.REGION          3811
59        NEPAL         55808
```

This step meant fixing the district names in the 'DISTRICT' column of the dataset. Entries such as "TERATHUM" were replaced by the correct "TERHATHUM" and "Total" by "NEPAL". By avoiding ad hoc representations of data, having standard district names reduces the potential for errors in later analysis, particularly when integrating with other datasets for even more elaborate analyses. The resulting DataFrame is shown, so that the user can quickly verify the changes to the 'DISTRICT' column by way of df1.head(10).

The tail of the DataFrame, df1.tail(5), is presented to verify that the district names have been correctly replaced.

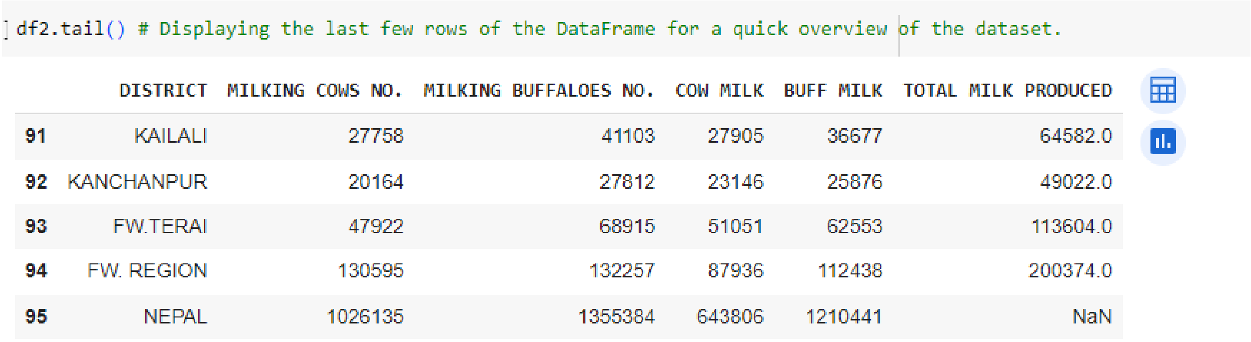
2. Milk Animals and Milk Production (df2):

- Head, Tail and Dimension Exploration of the Dataset:



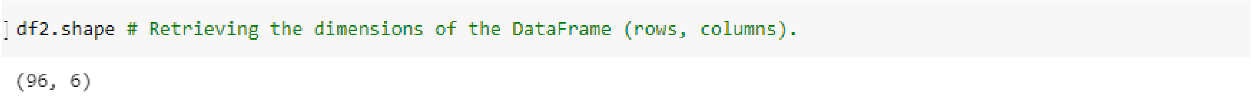
```
] df2.head() # Displaying the first few rows of the DataFrame for initial data exploration
```

	DISTRICT	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED
0	TAPLEJUNG	8123	4987	5389	4257	9645.0
1	SANKHUWASHAVA	15342	13367	6988	10589	17577.0
2	SOLUKHUMBU	7819	13501	2948	5493	8441.0
3	E.MOUNTAIN	31284	31855	15324	20339	35663.0
4	PANCHTHAR	14854	11331	8511	9835	18346.0



```
] df2.tail() # Displaying the last few rows of the DataFrame for a quick overview of the dataset.
```

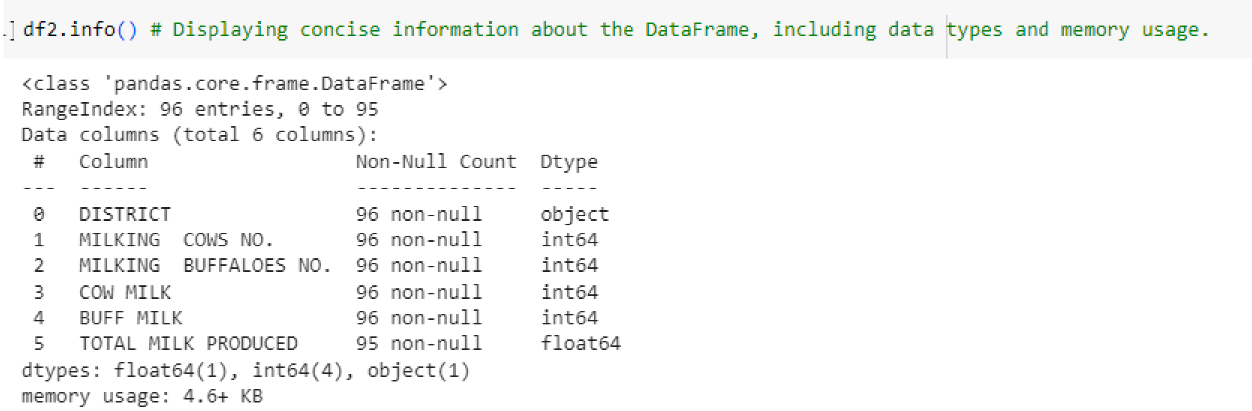
	DISTRICT	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED
91	KAILALI	27758	41103	27905	36677	64582.0
92	KANCHANPUR	20164	27812	23146	25876	49022.0
93	FW.TERAI	47922	68915	51051	62553	113604.0
94	FW. REGION	130595	132257	87936	112438	200374.0
95	NEPAL	1026135	1355384	643806	1210441	NaN



```
] df2.shape # Retrieving the dimensions of the DataFrame (rows, columns).
(96, 6)
```

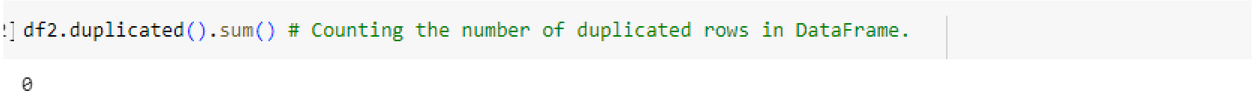
We see that the dataset of milk animals and milk production in Nepal has a head and tail section shown from the DataFrame df2. We also display the shape of the DataFrame, which shows the number of rows and columns in the dataset. This is useful to quickly understand the structure of the data. Here we see that the shape is (96,6) which means that there are 96 rows and 6 columns.

- Descriptive Information:



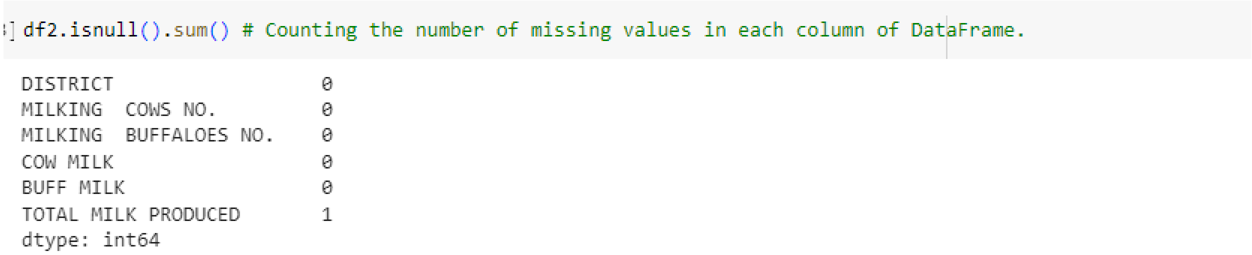
```
df2.info() # Displaying concise information about the DataFrame, including data types and memory usage.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 6 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   DISTRICT                 96 non-null     object 
 1   MILKING  COWS NO.        96 non-null     int64  
 2   MILKING  BUFFALOES NO.   96 non-null     int64  
 3   COW MILK                 96 non-null     int64  
 4   BUFF MILK                96 non-null     int64  
 5   TOTAL MILK PRODUCED      95 non-null     float64
dtypes: float64(1), int64(4), object(1)
memory usage: 4.6+ KB
```



```
df2.duplicated().sum() # Counting the number of duplicated rows in DataFrame.
```

```
0
```

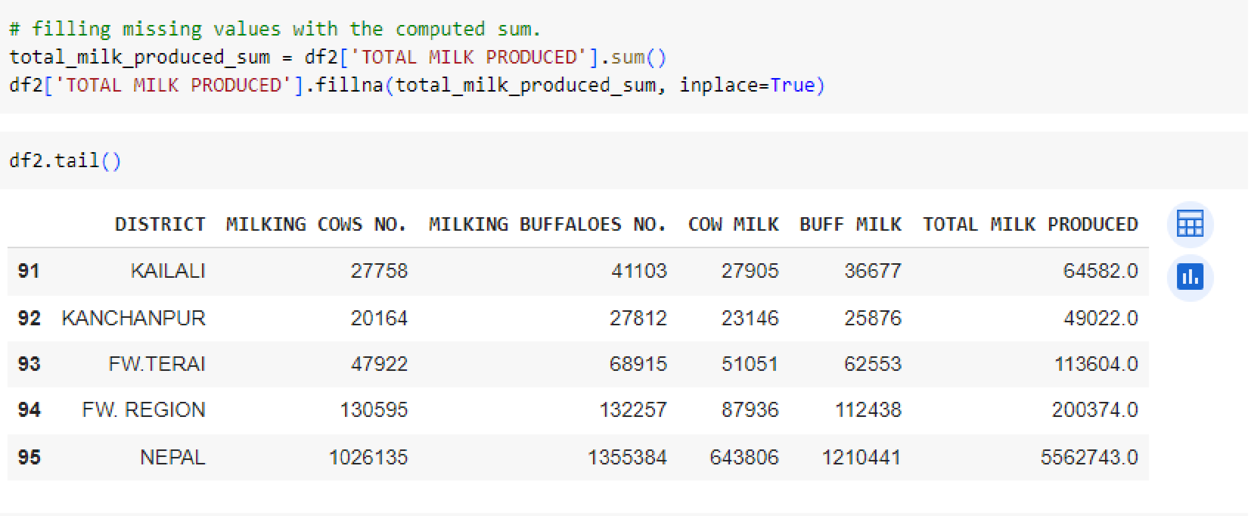


```
df2.isnull().sum() # Counting the number of missing values in each column of DataFrame.
```

```
DISTRICT                 0
MILKING  COWS NO.        0
MILKING  BUFFALOES NO.   0
COW MILK                 0
BUFF MILK                0
TOTAL MILK PRODUCED      1
dtype: int64
```

This section aims to explore what df2 dataset looks like. it describes the fundamental aspects of the data and provide informations. The function info() gives a nice summary of the data showing the type of the dataset. To find duplicate data, it gives the number of duplicate rows which I zero in our case. In addition to that, to see how complete the data is, it further specifies the number of missing values in each column.

- Handling missing values



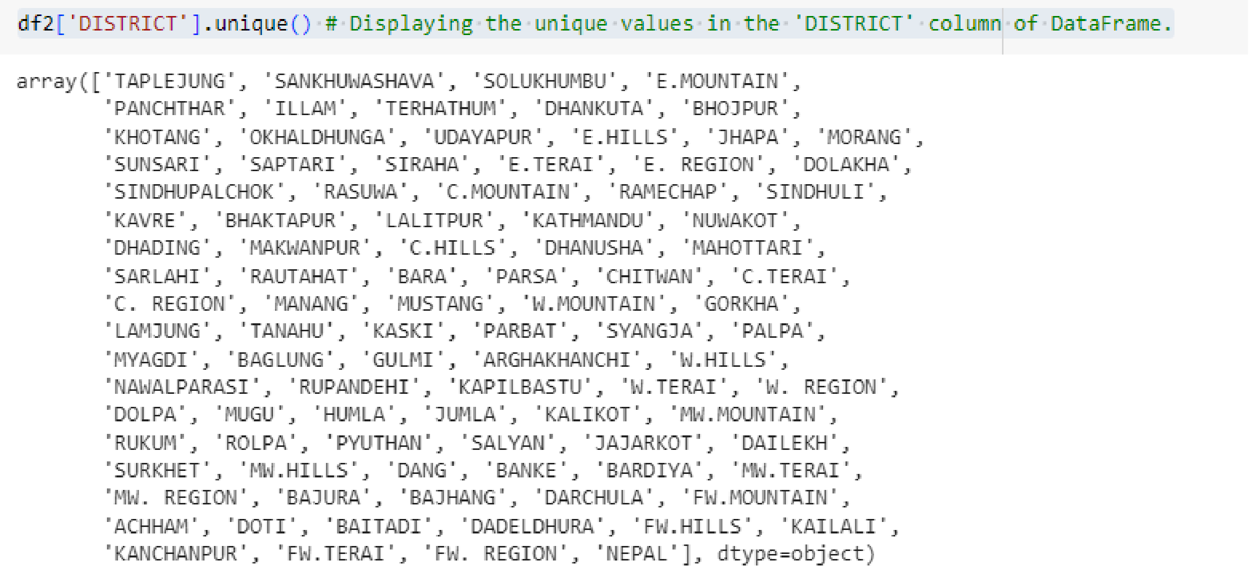
```
# filling missing values with the computed sum.
total_milk_produced_sum = df2['TOTAL MILK PRODUCED'].sum()
df2['TOTAL MILK PRODUCED'].fillna(total_milk_produced_sum, inplace=True)
```

```
df2.tail()
```

	DISTRICT	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED
91	KAILALI	27758	41103	27905	36677	64582.0
92	KANCHANPUR	20164	27812	23146	25876	49022.0
93	FW.TERAI	47922	68915	51051	62553	113604.0
94	FW. REGION	130595	132257	87936	112438	200374.0
95	NEPAL	1026135	1355384	643806	1210441	5562743.0

The missing values in the 'TOTAL MILK PRODUCED' column are taken care of in this phase of data exploration by going for a practical approach. We compute the actual value of the total sum of milk production so as to fill in the gaps in the dataset. This calculated approach ensures that we use meaningful and contextually appropriate estimations for the missing values instead of just dropping the missing values or substituting through mean which however simple is much less accurate. Then, the tail of the dataset is shown to confirm that the missing value imputation was executed appropriately.

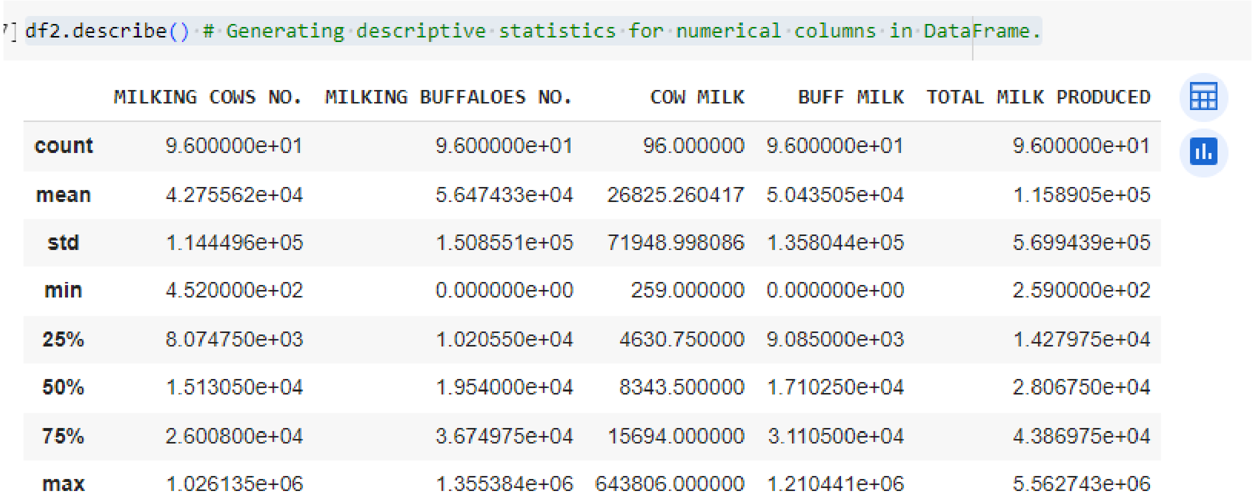
- Unique Districts in Milk Production Dataset (df2):



```
df2['DISTRICT'].unique() # Displaying the unique values in the 'DISTRICT' column of DataFrame.
```

```
array(['TAPLEJUNG', 'SANKHUWASHAVA', 'SOLUKHUMBU', 'E.MOUNTAIN',
       'PANCHTHAR', 'ILLAM', 'TERHATHUM', 'DHANKUTA', 'BHOJPUR',
       'KHOTANG', 'OKHALDHUNGA', 'UDAYAPUR', 'E.HILLS', 'JHAPA', 'MORANG',
       'SUNSARI', 'SAPTARI', 'SIRAHA', 'E.TERAI', 'E. REGION', 'DOLAKHA',
       'SINDHUPALCHOK', 'RASUWA', 'C.MOUNTAIN', 'RAMECHAP', 'SINDHULI',
       'KAVRE', 'BHAKTAPUR', 'LALITPUR', 'KATHMANDU', 'NUWAKOT',
       'DHADING', 'MAKWANPUR', 'C.HILLS', 'DHANUSHA', 'MAHOTTARI',
       'SARLAHI', 'RAUTAHAT', 'BARA', 'PARSA', 'CHITWAN', 'C.TERAI',
       'C. REGION', 'MANANG', 'MUSTANG', 'W.MOUNTAIN', 'GORKHA',
       'LAMJUNG', 'TANAHU', 'KASKI', 'PARBAT', 'SYANGJA', 'PALPA',
       'MYAGDI', 'BAGLUNG', 'GULMI', 'ARGHAKHANCHI', 'W.HILLS',
       'NAWALPARASI', 'RUPANDEHI', 'KAPILBASTU', 'W.TERAI', 'W. REGION',
       'DOLPA', 'MUGU', 'HUMLA', 'JUMLA', 'KALIKOT', 'MW.MOUNTAIN',
       'RUKUM', 'ROLPA', 'PYUTHAN', 'SALYAN', 'JAJARKOT', 'DAILEKH',
       'SURKHET', 'MW.HILLS', 'DANG', 'BANKE', 'BARDIYA', 'MW.TERAI',
       'MW. REGION', 'BAJURA', 'BAJHANG', 'DARCHULA', 'FW.MOUNTAIN',
       'ACHHAM', 'DOTI', 'BAITADI', 'DADELDHURA', 'FW.HILLS', 'KAILALI',
       'KANCHANPUR', 'FW.TERAI', 'FW. REGION', 'NEPAL'], dtype=object)
```

This section explores the distinct values present in the 'DISTRICT' column of the Milk Production dataset (df2). By checking the unique districts, we gain insight into the geographical representation within the dataset.



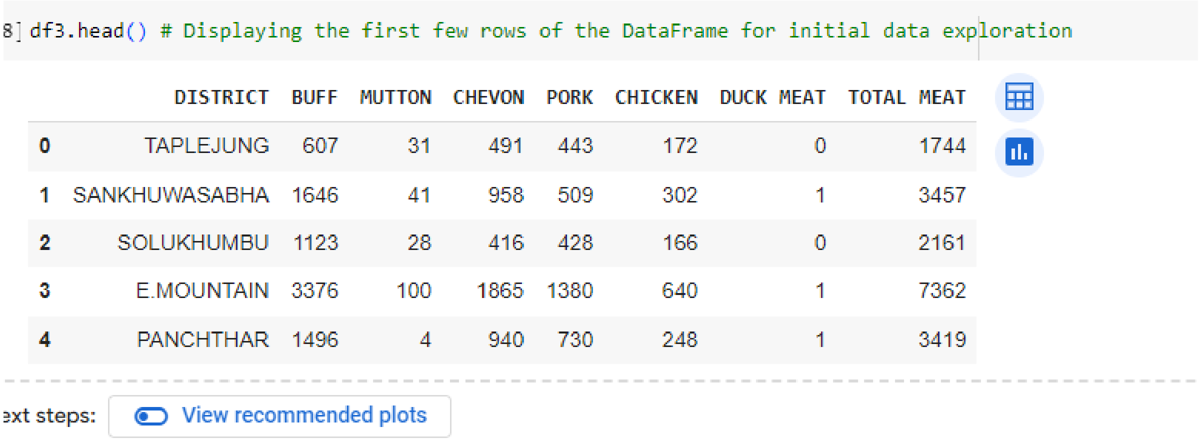
```
7] df2.describe() # Generating descriptive statistics for numerical columns in DataFrame.
```

	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED
count	9.600000e+01	9.600000e+01	96.000000	9.600000e+01	9.600000e+01
mean	4.275562e+04	5.647433e+04	26825.260417	5.043505e+04	1.158905e+05
std	1.144496e+05	1.508551e+05	71948.998086	1.358044e+05	5.699439e+05
min	4.520000e+02	0.000000e+00	259.000000	0.000000e+00	2.590000e+02
25%	8.074750e+03	1.020550e+04	4630.750000	9.085000e+03	1.427975e+04
50%	1.513050e+04	1.954000e+04	8343.500000	1.710250e+04	2.806750e+04
75%	2.600800e+04	3.674975e+04	15694.000000	3.110500e+04	4.386975e+04
max	1.026135e+06	1.355384e+06	643806.000000	1.210441e+06	5.562743e+06

An overview of the statistical summary for the numerical columns in the Milk Production dataset (df2) is given in this section. Important metrics including mean, standard deviation, minimum, 25th percentile, median (50th percentile), 75th percentile, and maximum values are included in the descriptive statistics.

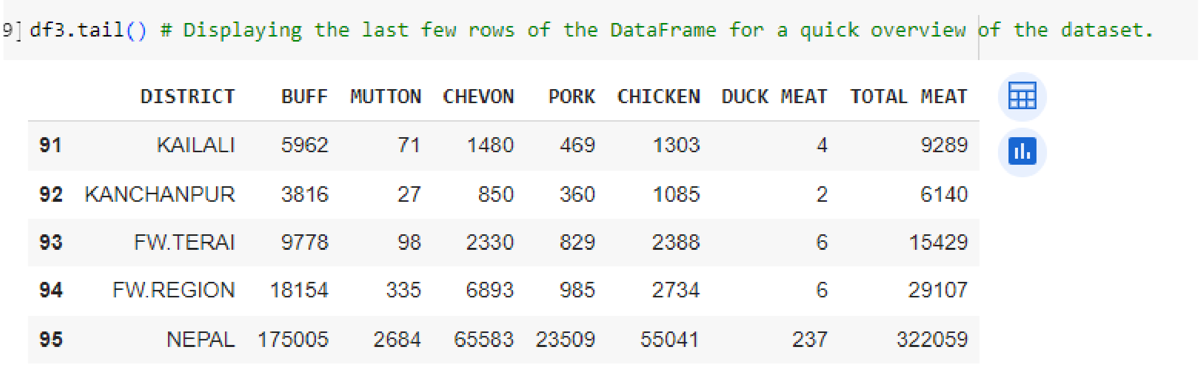
3. Net Meat Production (df3):

- Exploring Meat Production Dataset (df3)



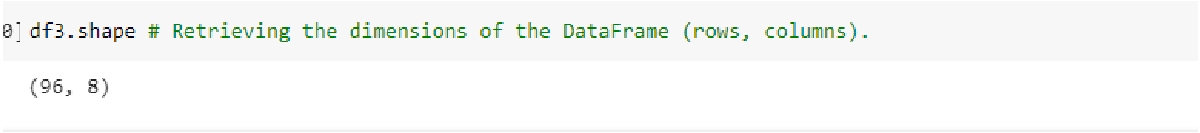
```
8] df3.head() # Displaying the first few rows of the DataFrame for initial data exploration
```

	DISTRICT	BUFF	MUTTON	CHEVON	PORK	CHICKEN	DUCK MEAT	TOTAL MEAT
0	TAPLEJUNG	607	31	491	443	172	0	1744
1	SANKHUWASABHA	1646	41	958	509	302	1	3457
2	SOLUKHUMBU	1123	28	416	428	166	0	2161
3	E.MOUNTAIN	3376	100	1865	1380	640	1	7362
4	PANCHTHAR	1496	4	940	730	248	1	3419



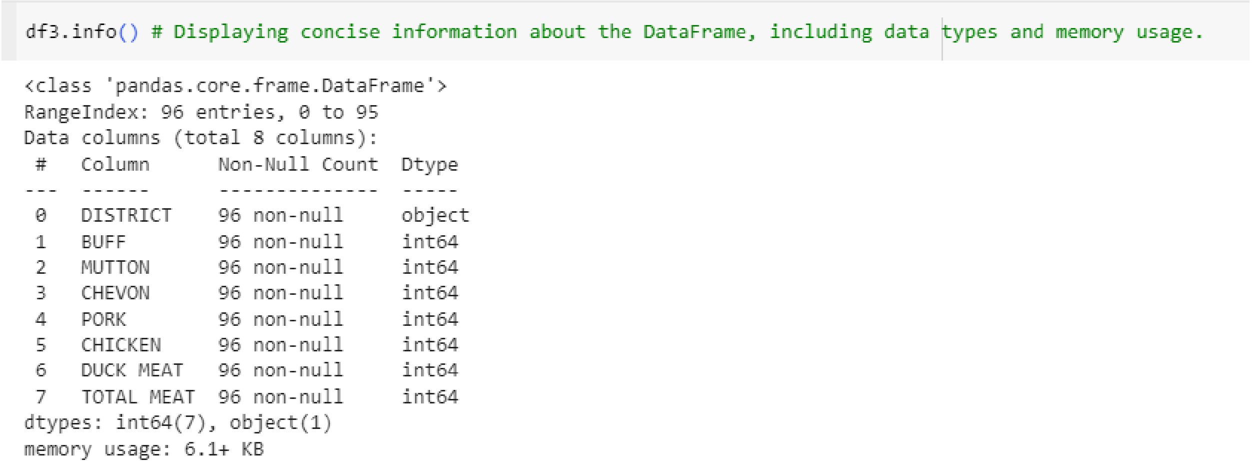
```
9] df3.tail() # Displaying the last few rows of the DataFrame for a quick overview of the dataset.
```

	DISTRICT	BUFF	MUTTON	CHEVON	PORK	CHICKEN	DUCK MEAT	TOTAL MEAT
91	KAILALI	5962	71	1480	469	1303	4	9289
92	KANCHANPUR	3816	27	850	360	1085	2	6140
93	FW.TERAI	9778	98	2330	829	2388	6	15429
94	FW.REGION	18154	335	6893	985	2734	6	29107
95	NEPAL	175005	2684	65583	23509	55041	237	322059



```
0] df3.shape # Retrieving the dimensions of the DataFrame (rows, columns).
```

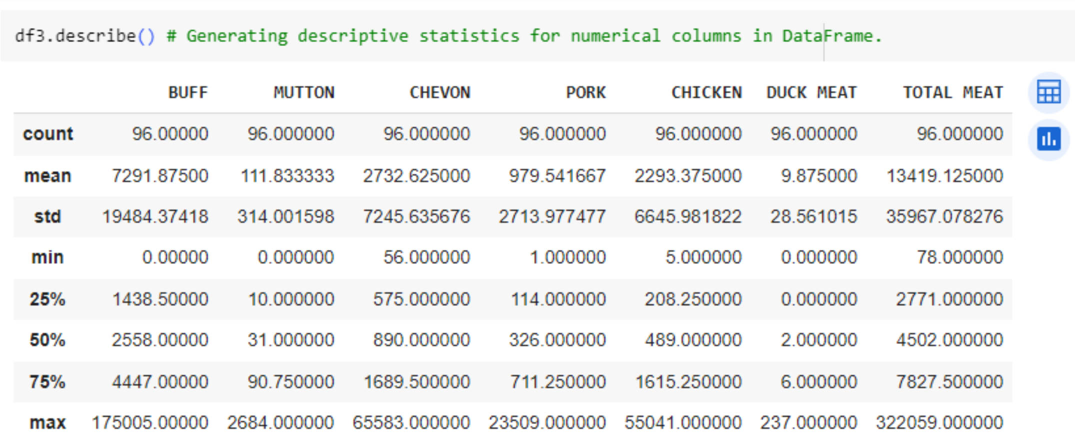
```
(96, 8)
```



```
df3.info() # Displaying concise information about the DataFrame, including data types and memory usage.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   DISTRICT    96 non-null     object
 1   BUFF        96 non-null     int64 
 2   MUTTON      96 non-null     int64 
 3   CHEVON      96 non-null     int64 
 4   PORK        96 non-null     int64 
 5   CHICKEN     96 non-null     int64 
 6   DUCK MEAT   96 non-null     int64 
 7   TOTAL MEAT  96 non-null     int64 
dtypes: int64(7), object(1)
memory usage: 6.1+ KB
```

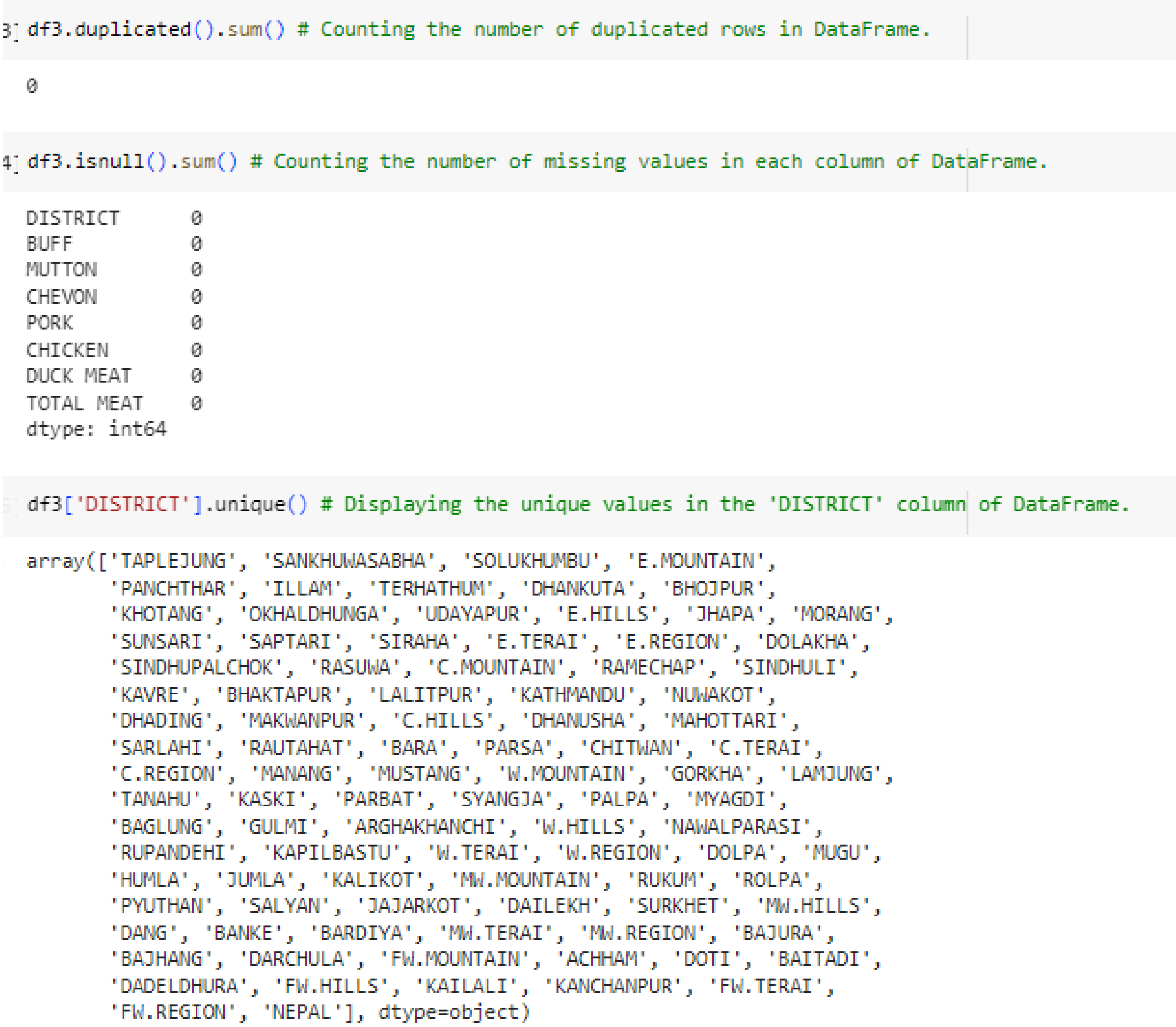
To better understand the structure and content of the Meat Production dataset (df3), we examine it in this section. To give a brief overview of the data's format, it starts by displaying the dataset's first and last few rows. The size of the dataset is then indicated by the shape information, which is the number of rows and columns.



```
df3.describe() # Generating descriptive statistics for numerical columns in DataFrame.
```

```
              BUFF       MUTTON        CHEVON          PORK       CHICKEN   DUCK MEAT      TOTAL MEAT
count     96.00000    96.000000     96.000000     96.000000     96.000000   96.000000       96.000000
mean    7291.87500   111.833333   2732.625000    979.541667   2293.375000    9.875000    13419.125000
std    19484.37418   314.001598   7245.635676   2713.977477   6645.981822   28.561015    35967.078276
min        0.00000     0.000000     56.000000      1.000000      5.000000    0.000000       78.000000
25%     1438.50000    10.000000    575.000000    114.000000    208.250000    0.000000     2771.000000
50%     2558.00000    31.000000    890.000000    326.000000    489.000000    2.000000     4502.000000
75%     4447.00000    90.750000   1689.500000    711.250000   1615.250000    6.000000     7827.500000
max   175005.00000  2684.000000  65583.000000  23509.000000  55041.000000  237.000000   322059.000000
```

The comprehensive explanation of the dataset's numerical features is then given. Included in this are summary statistics like the mean, standard deviation, minimum, 25th, 50th, and 75th percentiles, as well as maximum values.



```
df3.duplicated().sum() # Counting the number of duplicated rows in DataFrame.
```

```
0
```

```
df3.isnull().sum() # Counting the number of missing values in each column of DataFrame.
```

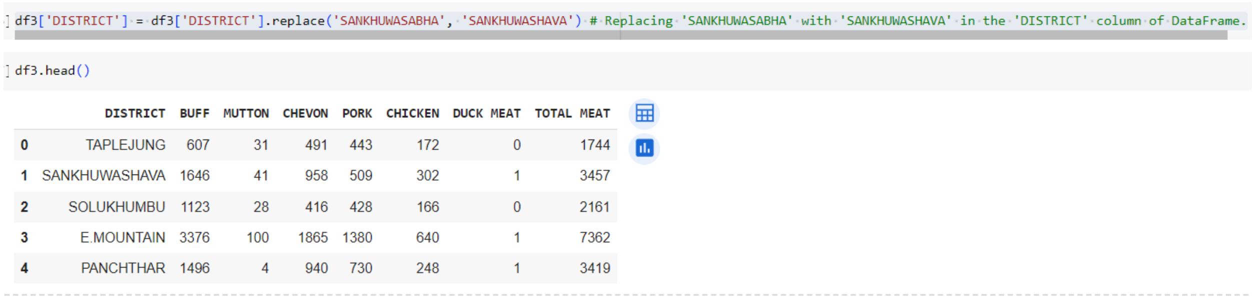
```
DISTRICT      0
BUFF          0
MUTTON        0
CHEVON        0
PORK          0
CHICKEN       0
DUCK MEAT     0
TOTAL MEAT    0
dtype: int64
```

```
df3['DISTRICT'].unique() # Displaying the unique values in the 'DISTRICT' column of DataFrame.
```

```
array(['TAPLEJUNG', 'SANKHUWASABHA', 'SOLUKHUMBU', 'E.MOUNTAIN',
       'PANCHTHAR', 'ILLAM', 'TERHATHUM', 'DHANKUTA', 'BHOJPUR',
       'KHOTANG', 'OKHALDHUNGA', 'UDAYAPUR', 'E.HILLS', 'JHAPA', 'MORANG',
       'SUNSARI', 'SAPTARI', 'SIRAHA', 'E.TERAI', 'E.REGION', 'DOLAKHA',
       'SINDHUPALCHOK', 'RASUWA', 'C.MOUNTAIN', 'RAMECHAP', 'SINDHULI',
       'KAVRE', 'BHAKTAPUR', 'LALITPUR', 'KATHMANDU', 'NUWAKOT',
       'DHADING', 'MAKWANPUR', 'C.HILLS', 'DHANUSHA', 'MAHOTTARI',
       'SARLAHI', 'RAUTAHAT', 'BARA', 'PARSA', 'CHITWAN', 'C.TERAI',
       'C.REGION', 'MANANG', 'MUSTANG', 'W.MOUNTAIN', 'GORKHA', 'LAMJUNG',
       'TANAHU', 'KASKI', 'PARBAT', 'SYANGJA', 'PALPA', 'MYAGDI',
       'BAGLUNG', 'GULMI', 'ARGHAKHANCHI', 'W.HILLS', 'NAWALPARASI',
       'RUPANDEHI', 'KAPILBASTU', 'W.TERAI', 'W.REGION', 'DOLPA', 'MUGU',
       'HUMLA', 'JUMLA', 'KALIKOT', 'MW.MOUNTAIN', 'RUKUM', 'ROLPA',
       'PYUTHAN', 'SALYAN', 'JAJARKOT', 'DAILEKH', 'SURKHET', 'MW.HILLS',
       'DANG', 'BANKE', 'BARDIYA', 'MW.TERAI', 'MW.REGION', 'BAJURA',
       'BAJHANG', 'DARCHULA', 'FW.MOUNTAIN', 'ACHHAM', 'DOTI', 'BAITADI',
       'DADELDHURA', 'FW.HILLS', 'KAILALI', 'KANCHANPUR', 'FW.TERAI',
       'FW.REGION', 'NEPAL'], dtype=object)
```

The investigation continues by looking for and measuring any possible duplicates in the dataset. A review of the missing values in every column is also carried out. Ultimately, the distinct values found in the 'DISTRICT' column are shown, offering insight into the category entries as well as any possible irregularities or differences in naming standards.

- Handling District Names in Meat Production Dataset (df3):



```
df3['DISTRICT'] = df3['DISTRICT'].replace('SANKHUWASABHA', 'SANKHUWASHAVA') # Replacing 'SANKHUWASABHA' with 'SANKHUWASHAVA' in the 'DISTRICT' column of DataFrame.
```

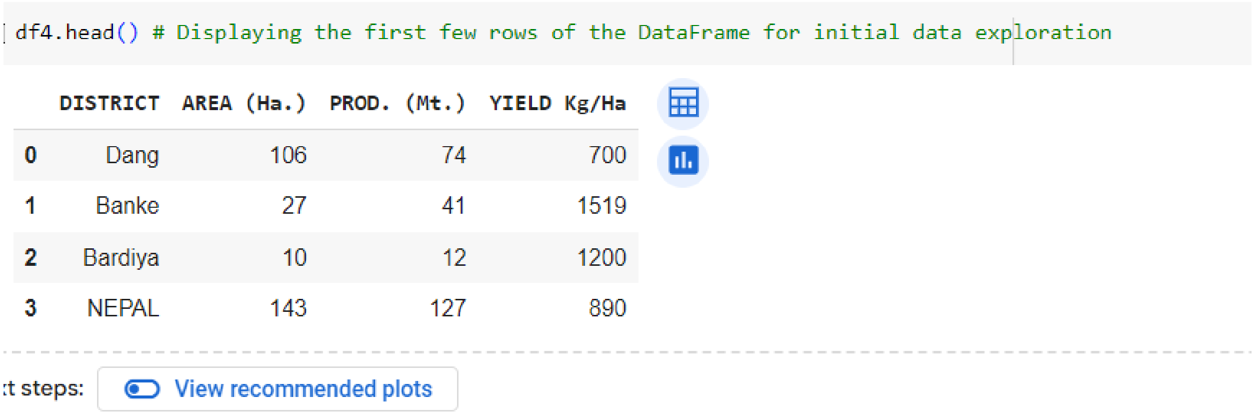
```
df3.head()
```

	DISTRICT	BUFF	MUTTON	CHEVON	PORK	CHICKEN	DUCK MEAT	TOTAL MEAT
0	TAPLEJUNG	607	31	491	443	172	0	1744
1	SANKHUWASHAVA	1646	41	958	509	302	1	3457
2	SOLUKHUMBU	1123	28	416	428	166	0	2161
3	E.MOUNTAIN	3376	100	1865	1380	640	1	7362
4	PANCHTHAR	1496	4	940	730	248	1	3419

Within the Meat Production dataset (df3), which is utilized for further investigation, changes are made to guarantee homogeneity in district names. More specifically, the 'DISTRICT' column is edited to reflect that ‘SANKHUWASHAVA’ was wrongly entered as 'SANKHUWASABHA’. By accounting for possible discrepancies in naming conventions, this standardization enables the uniformity and correctness of representations of districts.

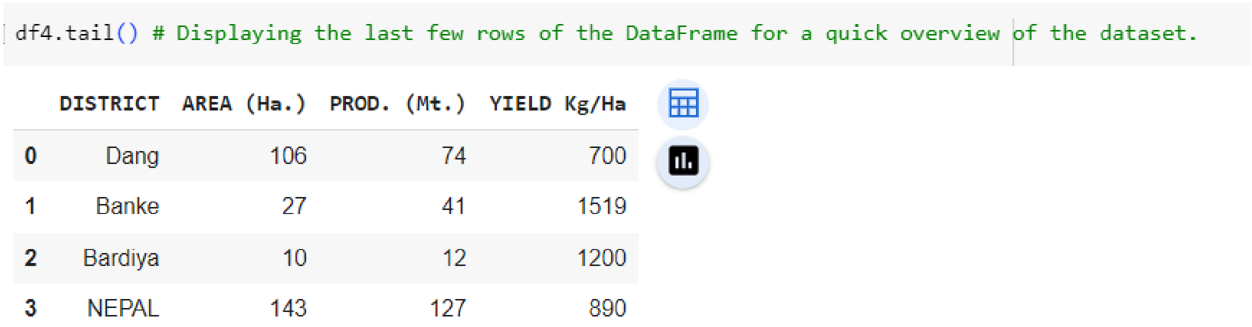
Afterwards, to check the amendments made to the ‘DISTRICT’ column, the head of the dataset is shown, ensuring that the data remains coherent and of high quality throughout the process of exploration.

4. **Cotton Production (df4):**



```
df4.head() # Displaying the first few rows of the DataFrame for initial data exploration
```

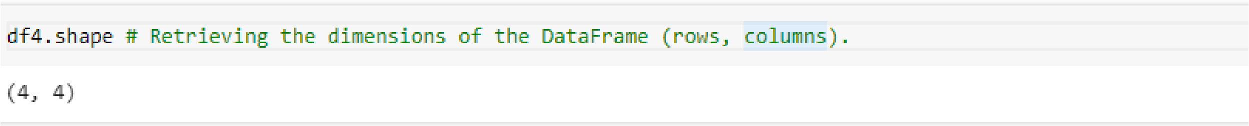
	DISTRICT	AREA (Ha.)	PROD. (Mt.)	YIELD Kg/Ha
0	Dang	106	74	700
1	Banke	27	41	1519
2	Bardiya	10	12	1200
3	NEPAL	143	127	890



```
df4.tail() # Displaying the last few rows of the DataFrame for a quick overview of the dataset.
```

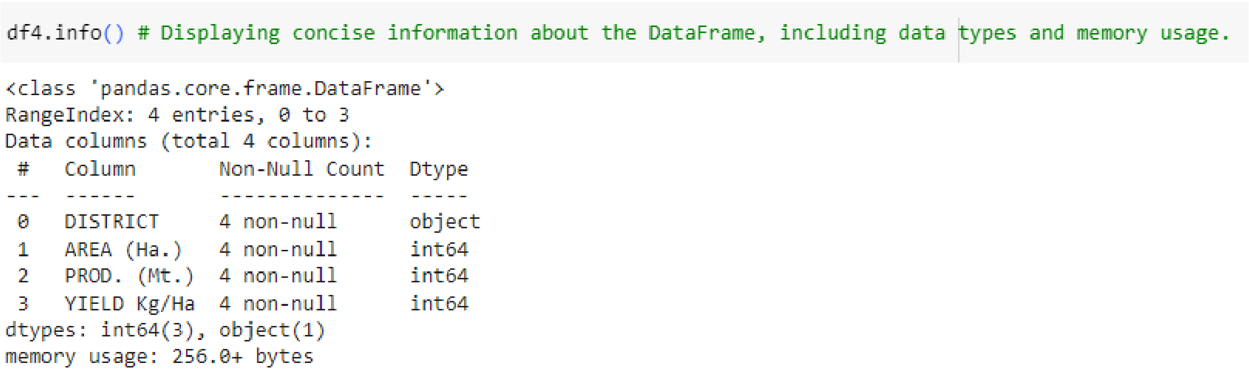
	DISTRICT	AREA (Ha.)	PROD. (Mt.)	YIELD Kg/Ha
0	Dang	106	74	700
1	Banke	27	41	1519
2	Bardiya	10	12	1200
3	NEPAL	143	127	890

An overview of the first few rows of the Cotton Production dataset (df4) is the first step in the exploration process, which offers insights into the data's structure and content. The final few rows of the dataset are also visualized.



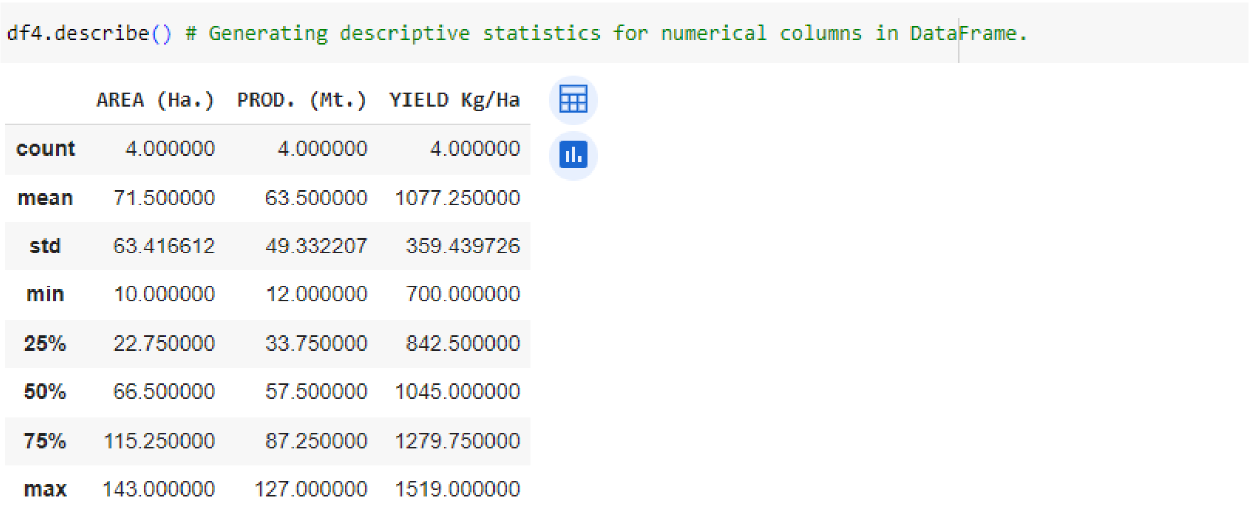
```
df4.shape # Retrieving the dimensions of the DataFrame (rows, columns).
```

```
(4, 4)
```



```
df4.info() # Displaying concise information about the DataFrame, including data types and memory usage.
```

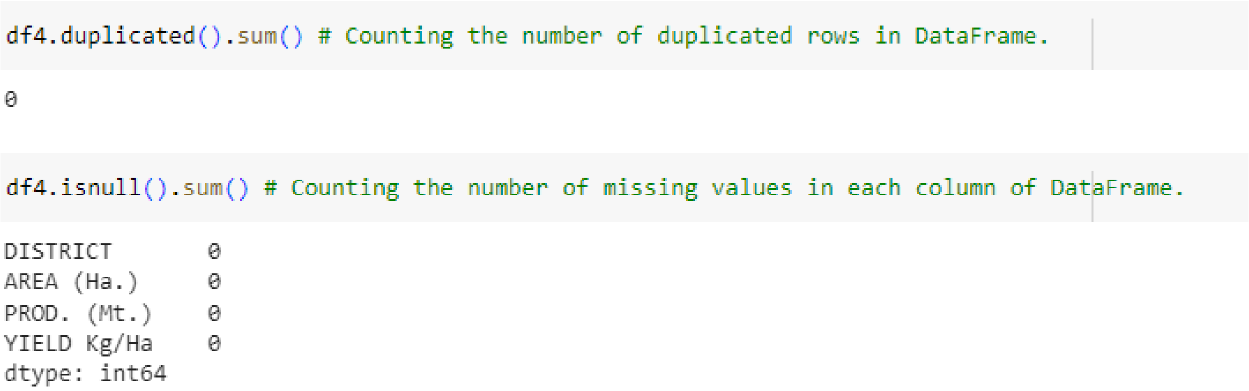
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   DISTRICT     4 non-null      object
 1   AREA (Ha.)   4 non-null      int64 
 2   PROD. (Mt.)  4 non-null      int64 
 3   YIELD Kg/Ha  4 non-null      int64 
dtypes: int64(3), object(1)
memory usage: 256.0+ bytes
```



```
df4.describe() # Generating descriptive statistics for numerical columns in DataFrame.
```

	AREA (Ha.)	PROD. (Mt.)	YIELD Kg/Ha
count	4.000000	4.000000	4.000000
mean	71.500000	63.500000	1077.250000
std	63.416612	49.332207	359.439726
min	10.000000	12.000000	700.000000
25%	22.750000	33.750000	842.500000
50%	66.500000	57.500000	1045.000000
75%	115.250000	87.250000	1279.750000
max	143.000000	127.000000	1519.000000

Info() provides brief details about the DataFrame, such as data types and memory usage, while the shape attribute retrieves the dimensions (rows, columns) of df4. With describe(), descriptive statistics for numerical columns can be produced.



```
df4.duplicated().sum() # Counting the number of duplicated rows in DataFrame.
```

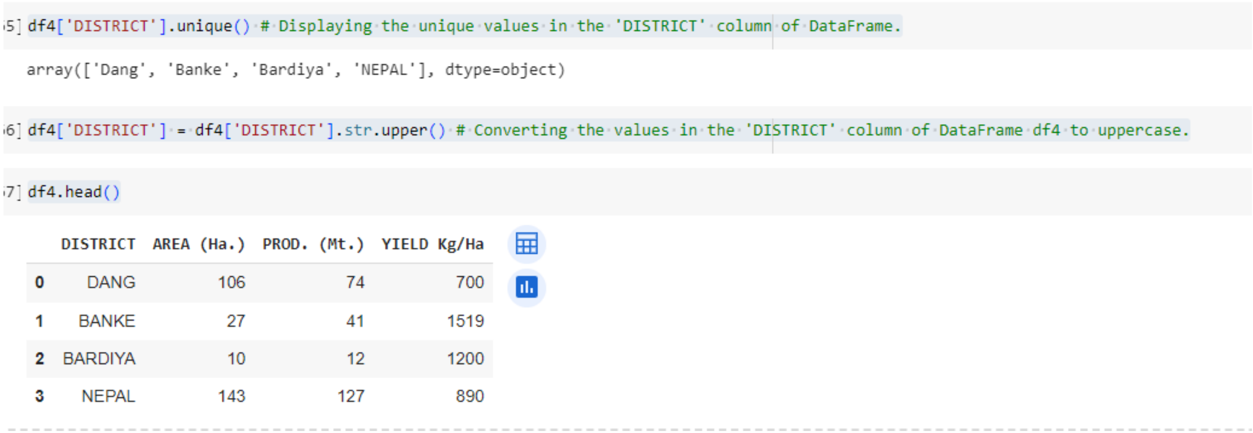
```
0
```

```
df4.isnull().sum() # Counting the number of missing values in each column of DataFrame.
```

```
DISTRICT       0
AREA (Ha.)     0
PROD. (Mt.)    0
YIELD Kg/Ha    0
dtype: int64
```

Furthermore, duplicated() is used to search the dataset for duplicate rows.sum(), and isnull().sum() is used to determine the number of missing values in each column.

- Handling String Case and Cleaning



```
df4['DISTRICT'].unique() # Displaying the unique values in the 'DISTRICT' column of DataFrame.
```

```
array(['Dang', 'Banke', 'Bardiya', 'NEPAL'], dtype=object)
```

```
df4['DISTRICT'] = df4['DISTRICT'].str.upper() # Converting the values in the 'DISTRICT' column of DataFrame df4 to uppercase.
```

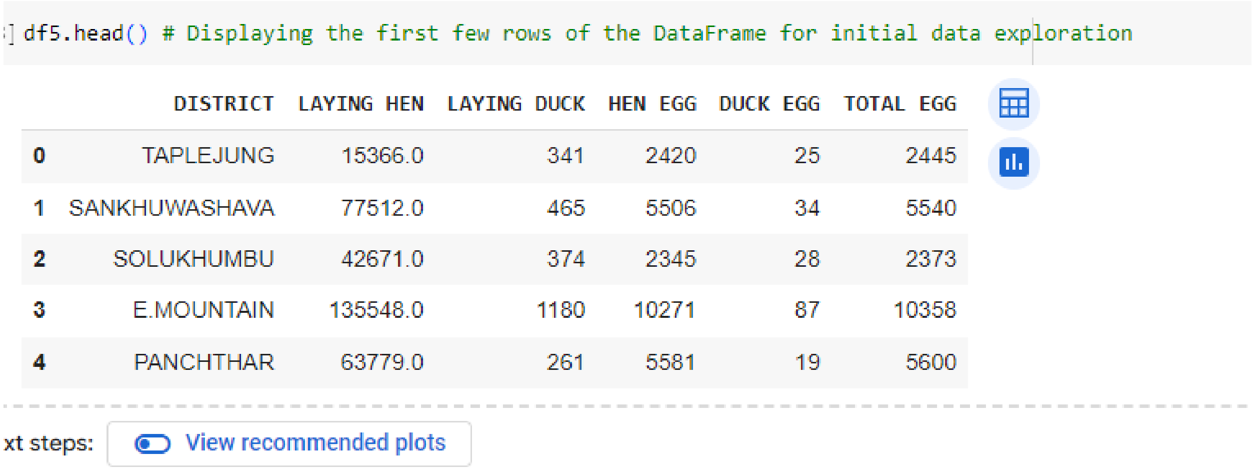
```
df4.head()
```

	DISTRICT	AREA (Ha.)	PROD. (Mt.)	YIELD Kg/Ha
0	DANG	106	74	700
1	BANKE	27	41	1519
2	BARDIYA	10	12	1200
3	NEPAL	143	127	890

To check for any variations in district names, the unique values in the 'DISTRICT' column of the Cotton Production dataset (df4) are shown. To ensure consistency, district names are represented uniformly by converting them to uppercase using str.upper().

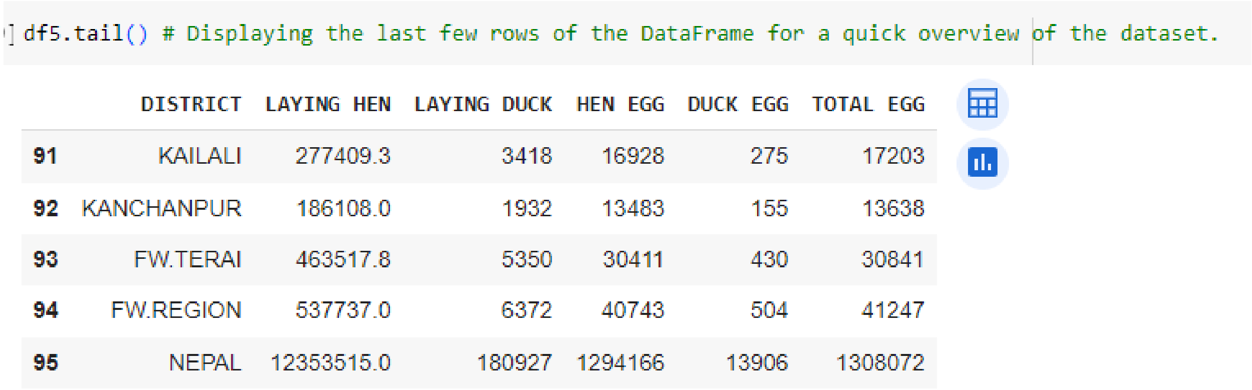
Then the head of the dataset is shown to verify that the district names have been converted to uppercase. This stage is critical to ensure data integrity, enable smooth merging of datasets, and promote consistent approach throughout the project.

5. Egg Production (df5):



```
] df5.head() # Displaying the first few rows of the DataFrame for initial data exploration
```

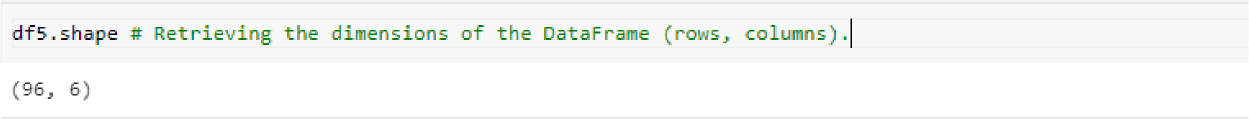
	DISTRICT	LAYING HEN	LAYING DUCK	HEN EGG	DUCK EGG	TOTAL EGG
0	TAPLEJUNG	15366.0	341	2420	25	2445
1	SANKHUWASHAVA	77512.0	465	5506	34	5540
2	SOLUKHUMBU	42671.0	374	2345	28	2373
3	E.MOUNTAIN	135548.0	1180	10271	87	10358
4	PANCHTHAR	63779.0	261	5581	19	5600



```
] df5.tail() # Displaying the last few rows of the DataFrame for a quick overview of the dataset.
```

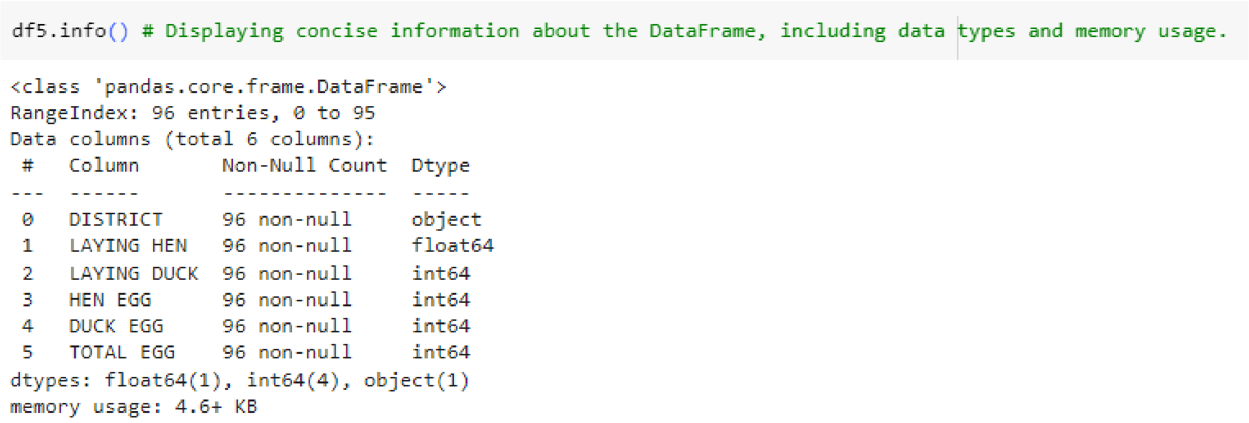
	DISTRICT	LAYING HEN	LAYING DUCK	HEN EGG	DUCK EGG	TOTAL EGG
91	KAILALI	277409.3	3418	16928	275	17203
92	KANCHANPUR	186108.0	1932	13483	155	13638
93	FW.TERAI	463517.8	5350	30411	430	30841
94	FW.REGION	537737.0	6372	40743	504	41247
95	NEPAL	12353515.0	180927	1294166	13906	1308072

In order to obtain an understanding of the data's structure, the first and last few rows of the Egg Production dataset (df5) are examined.



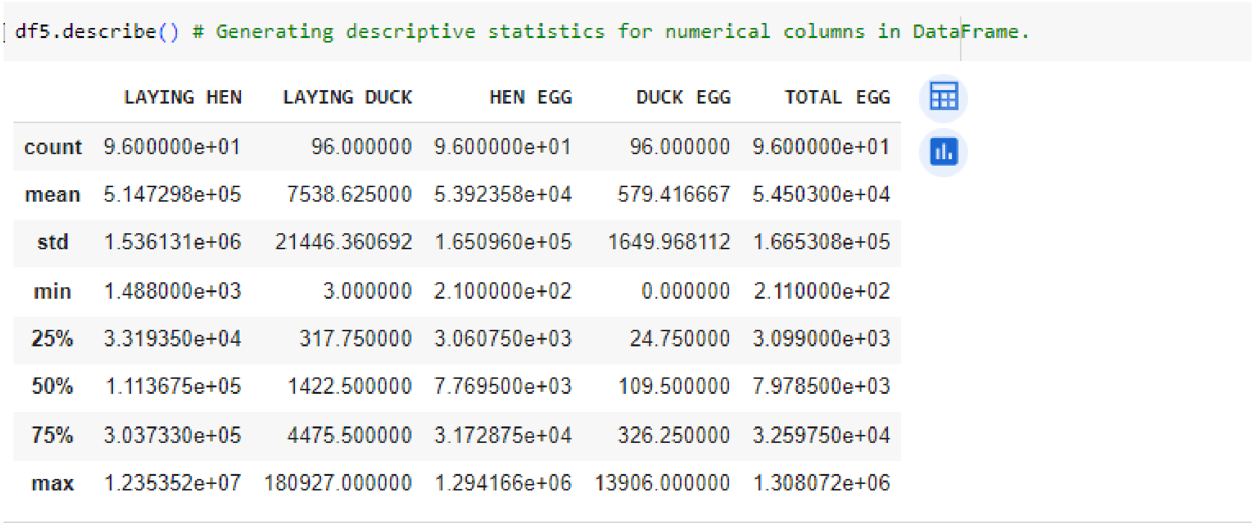
```
df5.shape # Retrieving the dimensions of the DataFrame (rows, columns).
```

```
(96, 6)
```



```
df5.info() # Displaying concise information about the DataFrame, including data types and memory usage.
```

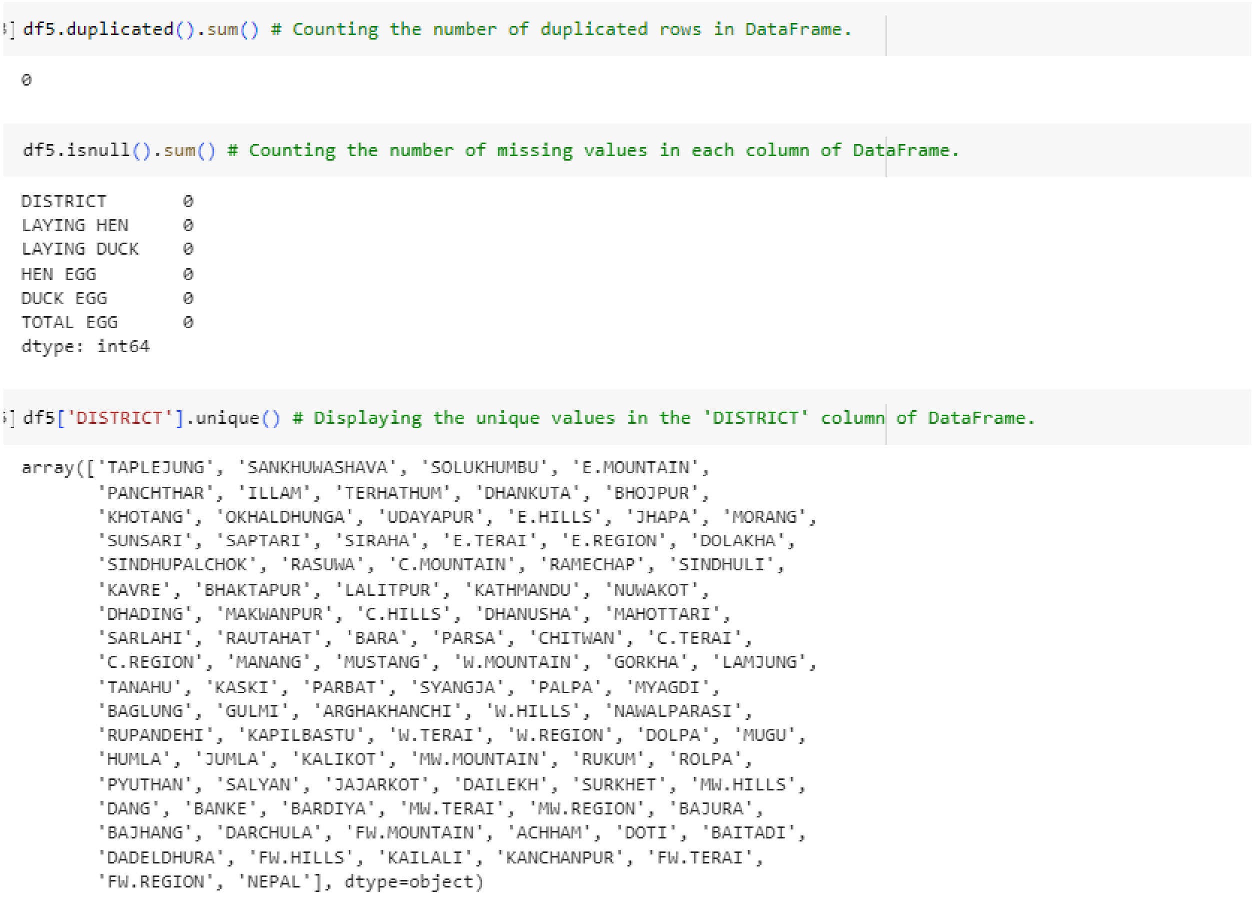
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   DISTRICT     96 non-null     object 
 1   LAYING HEN   96 non-null     float64
 2   LAYING DUCK  96 non-null     int64  
 3   HEN EGG      96 non-null     int64  
 4   DUCK EGG     96 non-null     int64  
 5   TOTAL EGG    96 non-null     int64  
dtypes: float64(1), int64(4), object(1)
memory usage: 4.6+ KB
```



```
df5.describe() # Generating descriptive statistics for numerical columns in DataFrame.
```

	LAYING HEN	LAYING DUCK	HEN EGG	DUCK EGG	TOTAL EGG
count	9.600000e+01	96.000000	9.600000e+01	96.000000	9.600000e+01
mean	5.147298e+05	7538.625000	5.392358e+04	579.416667	5.450300e+04
std	1.536131e+06	21446.360692	1.650960e+05	1649.968112	1.665308e+05
min	1.488000e+03	3.000000	2.100000e+02	0.000000	2.110000e+02
25%	3.319350e+04	317.750000	3.060750e+03	24.750000	3.099000e+03
50%	1.113675e+05	1422.500000	7.769500e+03	109.500000	7.978500e+03
75%	3.037330e+05	4475.500000	3.172875e+04	326.250000	3.259750e+04
max	1.235352e+07	180927.000000	1.294166e+06	13906.000000	1.308072e+06

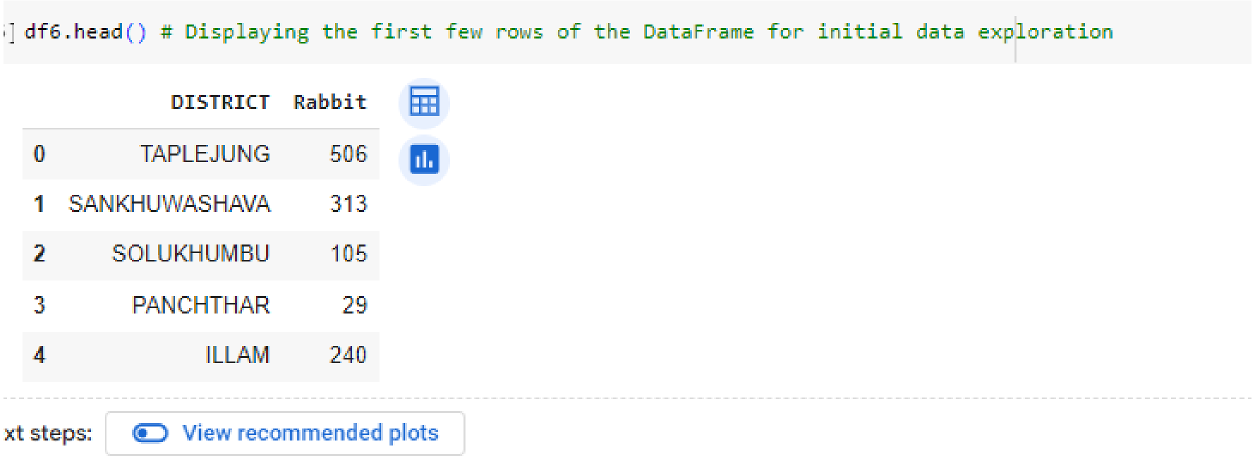
The basic information and the dataset's shape (rows and columns) are retrieved, and numerical columns, descriptive statistics are generated.



```
] df5.duplicated().sum() # Counting the number of duplicated rows in DataFrame.

0

df5.isnull().sum() # Counting the number of missing values in each column of DataFrame.

DISTRICT       0
LAYING HEN     0
LAYING DUCK    0
HEN EGG        0
DUCK EGG       0
TOTAL EGG      0
dtype: int64

] df5['DISTRICT'].unique() # Displaying the unique values in the 'DISTRICT' column of DataFrame.

array(['TAPLEJUNG', 'SANKHUWASHAVA', 'SOLUKHUMBU', 'E.MOUNTAIN',
       'PANCHTHAR', 'ILLAM', 'TERHATHUM', 'DHANKUTA', 'BHOJPUR',
       'KHOTANG', 'OKHALDHUNGA', 'UDAYAPUR', 'E.HILLS', 'JHAPA', 'MORANG',
       'SUNSARI', 'SAPTARI', 'SIRAHA', 'E.TERAI', 'E.REGION', 'DOLAKHA',
       'SINDHUPALCHOK', 'RASUWA', 'C.MOUNTAIN', 'RAMECHAP', 'SINDHULI',
       'KAVRE', 'BHAKTAPUR', 'LALITPUR', 'KATHMANDU', 'NUWAKOT',
       'DHADING', 'MAKWANPUR', 'C.HILLS', 'DHANUSHA', 'MAHOTTARI',
       'SARLAHI', 'RAUTAHAT', 'BARA', 'PARSA', 'CHITWAN', 'C.TERAI',
       'C.REGION', 'MANANG', 'MUSTANG', 'W.MOUNTAIN', 'GORKHA', 'LAMJUNG',
       'TANAHU', 'KASKI', 'PARBAT', 'SYANGJA', 'PALPA', 'MYAGDI',
       'BAGLUNG', 'GULMI', 'ARGHAKHANCHI', 'W.HILLS', 'NAWALPARASI',
       'RUPANDEHI', 'KAPILBASTU', 'W.TERAI', 'W.REGION', 'DOLPA', 'MUGU',
       'HUMLA', 'JUMLA', 'KALIKOT', 'MW.MOUNTAIN', 'RUKUM', 'ROLPA',
       'PYUTHAN', 'SALYAN', 'JAJARKOT', 'DAILEKH', 'SURKHET', 'MW.HILLS',
       'DANG', 'BANKE', 'BARDIYA', 'MW.TERAI', 'MW.REGION', 'BAJURA',
       'BAJHANG', 'DARCHULA', 'FW.MOUNTAIN', 'ACHHAM', 'DOTI', 'BAITADI',
       'DADELDHURA', 'FW.HILLS', 'KAILALI', 'KANCHANPUR', 'FW.TERAI',
       'FW.REGION', 'NEPAL'], dtype=object)
```

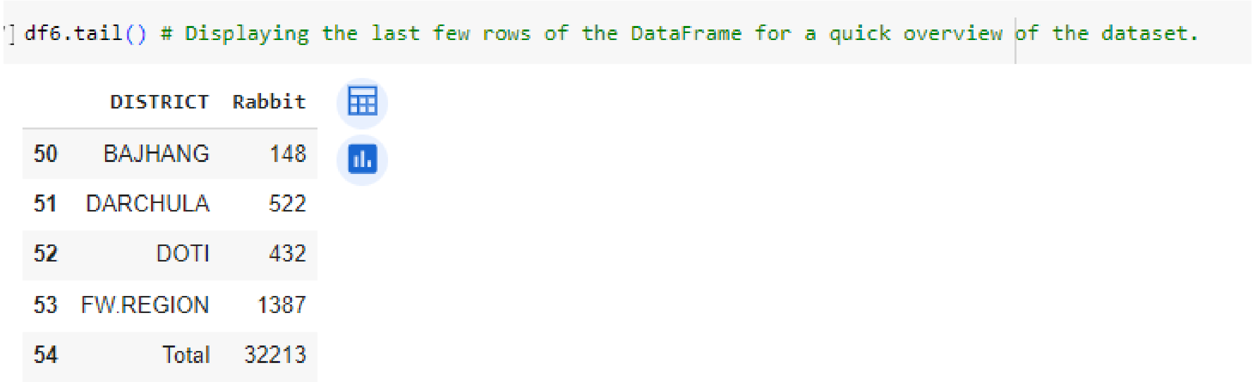
Further analysis entails using duplicated() to count the number of missing values in each column and looking for duplicate rows.isnull() and sum().sum() correspondingly. To identify any variations, the 'DISTRICT' column's unique values are shown.

6. Rabbit Population (df6):



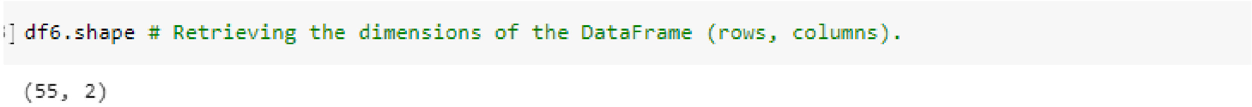
```
] df6.head() # Displaying the first few rows of the DataFrame for initial data exploration
```

	DISTRICT	Rabbit
0	TAPLEJUNG	506
1	SANKHUWASHAVA	313
2	SOLUKHUMBU	105
3	PANCHTHAR	29
4	ILLAM	240



```
] df6.tail() # Displaying the last few rows of the DataFrame for a quick overview of the dataset.
```

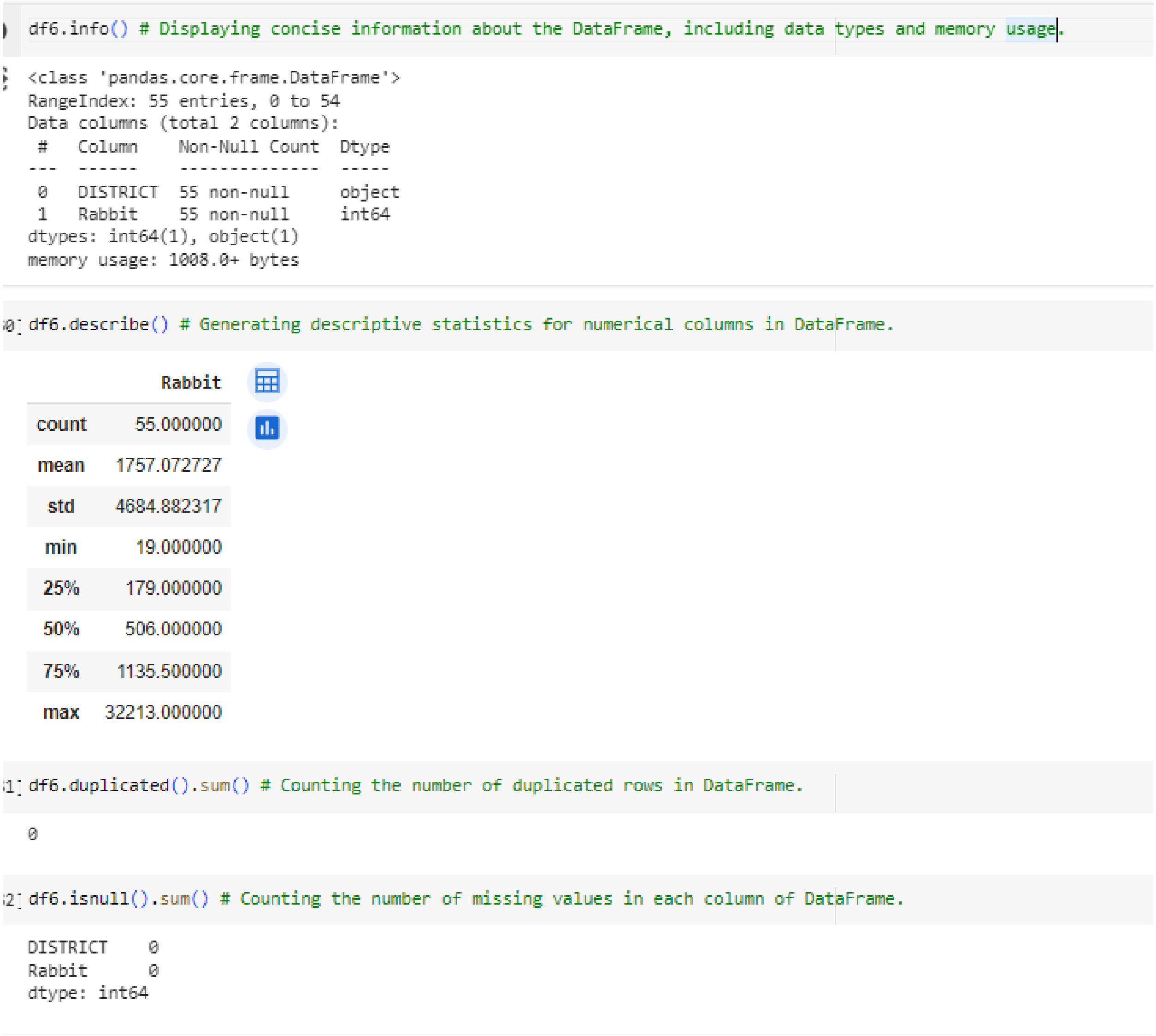
	DISTRICT	Rabbit
50	BAJHANG	148
51	DARCHULA	522
52	DOTI	432
53	FW.REGION	1387
54	Total	32213



```
] df6.shape # Retrieving the dimensions of the DataFrame (rows, columns).
```

```
(55, 2)
```

We first explore the Rabbit Population dataset (df6) in order to gain an understanding of its characteristics. To understand the structure of the dataset, this involves looking at the first and last few rows. Using shape to retrieve the dataset's dimensions (rows, columns).



```
df6.info() # Displaying concise information about the DataFrame, including data types and memory usage.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55 entries, 0 to 54
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   DISTRICT  55 non-null     object
 1   Rabbit    55 non-null     int64 
dtypes: int64(1), object(1)
memory usage: 1008.0+ bytes
```

```
df6.describe() # Generating descriptive statistics for numerical columns in DataFrame.
```

	Rabbit
count	55.000000
mean	1757.072727
std	4684.882317
min	19.000000
25%	179.000000
50%	506.000000
75%	1135.500000
max	32213.000000

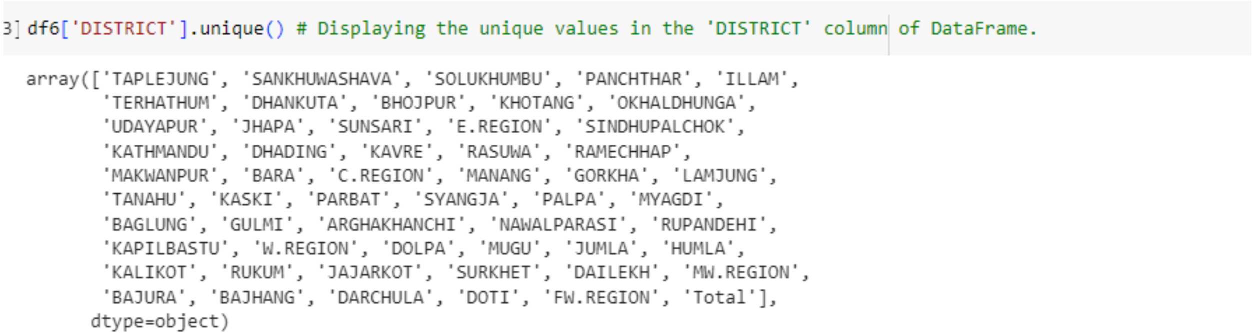
```
df6.duplicated().sum() # Counting the number of duplicated rows in DataFrame.
```

```
0
```

```
df6.isnull().sum() # Counting the number of missing values in each column of DataFrame.
```

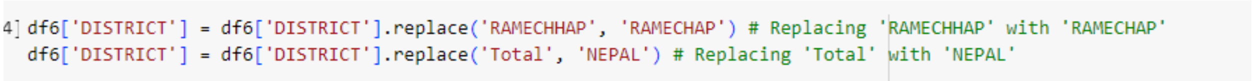
```
DISTRICT    0
Rabbit      0
dtype: int64
```

To get basic information info() and the distribution of numerical columns, describe() is used to obtain statistical information. Duplicate().sum() is used to find duplicate rows, and isnull().sum() counts the number of missing values in each column.



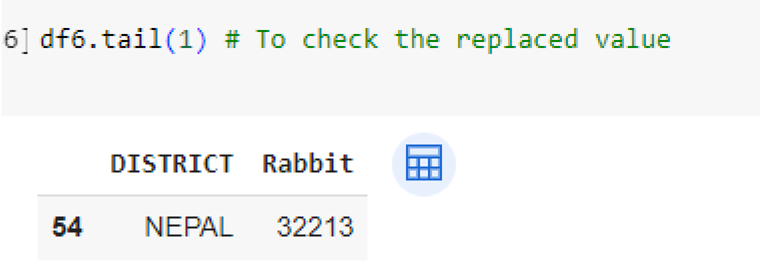
```
3] df6['DISTRICT'].unique() # Displaying the unique values in the 'DISTRICT' column of DataFrame.

  array(['TAPLEJUNG', 'SANKHUWASHAVA', 'SOLUKHUMBU', 'PANCHTHAR', 'ILLAM',
         'TERHATHUM', 'DHANKUTA', 'BHOJPUR', 'KHOTANG', 'OKHALDHUNGA',
         'UDAYAPUR', 'JHAPA', 'SUNSARI', 'E.REGION', 'SINDHUPALCHOK',
         'KATHMANDU', 'DHADING', 'KAVRE', 'RASUWA', 'RAMECHHAP',
         'MAKWANPUR', 'BARA', 'C.REGION', 'MANANG', 'GORKHA', 'LAMJUNG',
         'TANAHU', 'KASKI', 'PARBAT', 'SYANGJA', 'PALPA', 'MYAGDI',
         'BAGLUNG', 'GULMI', 'ARGHAKHANCHI', 'NAWALPARASI', 'RUPANDEHI',
         'KAPILBASTU', 'W.REGION', 'DOLPA', 'MUGU', 'JUMLA', 'HUMLA',
         'KALIKOT', 'RUKUM', 'JAJARKOT', 'SURKHET', 'DAILEKH', 'MW.REGION',
         'BAJURA', 'BAJHANG', 'DARCHULA', 'DOTI', 'FW.REGION', 'Total'],
        dtype=object)
```

Renaming and handling missing values



```
4] df6['DISTRICT'] = df6['DISTRICT'].replace('RAMECHHAP', 'RAMECHAP') # Replacing 'RAMECHHAP' with 'RAMECHAP'
  df6['DISTRICT'] = df6['DISTRICT'].replace('Total', 'NEPAL') # Replacing 'Total' with 'NEPAL'
```

Particular focus is placed on handling missing values and renaming in the 'DISTRICT' column. To be more precise, "RAMECHHAP" is changed to "RAMECHAP" and "Total" to "NEPAL." This guarantees that the district names are clear and consistent.

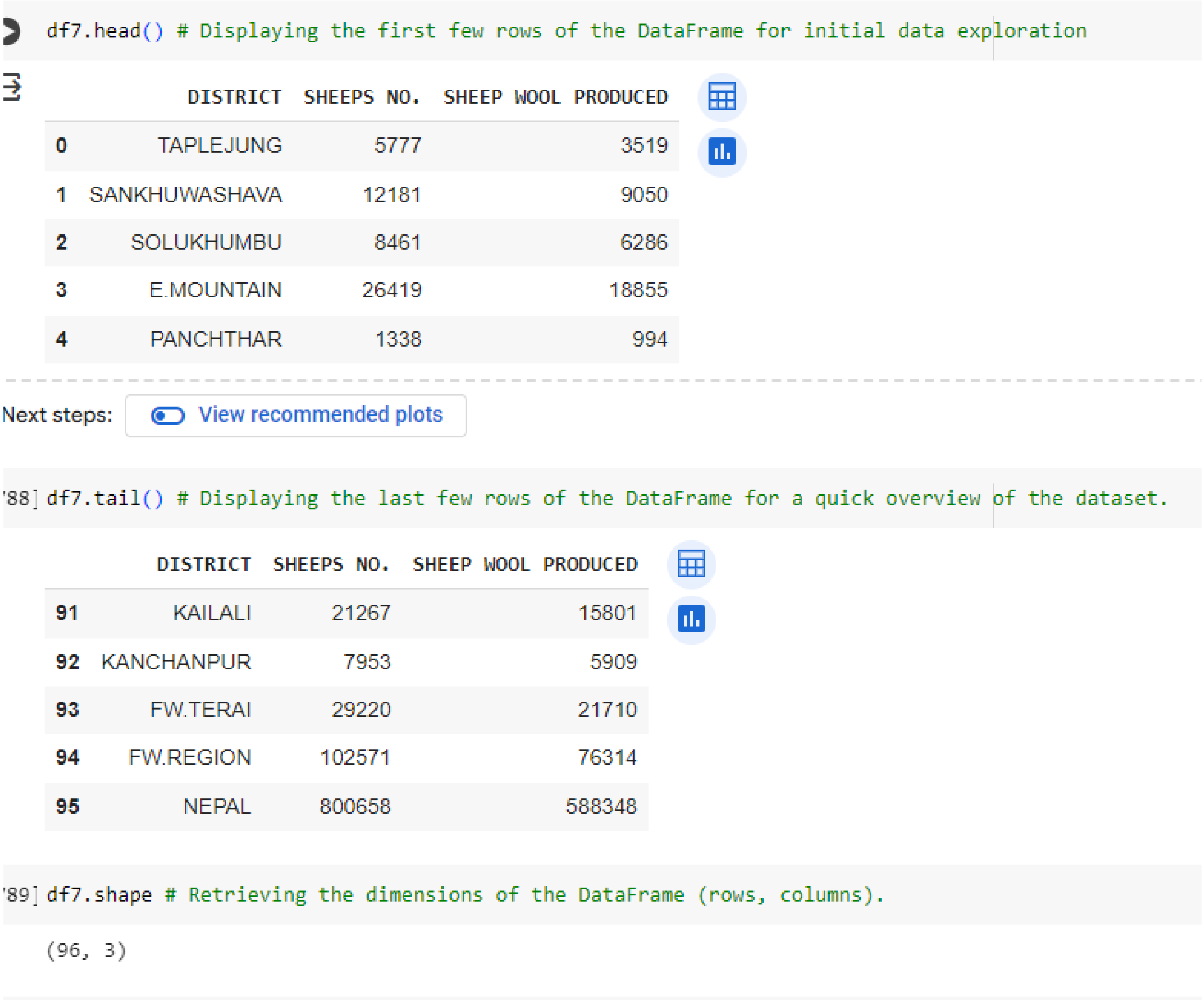


```
6] df6.tail(1) # To check the replaced value
```

	DISTRICT	Rabbit
54	NEPAL	32213

After inspecting the tail of the dataset, a particular row (index 19) is chosen using iloc for additional review.

7. Wool Production (df7):



```
df7.head() # Displaying the first few rows of the DataFrame for initial data exploration
```

	DISTRICT	SHEEPS NO.	SHEEP WOOL PRODUCED
0	TAPLEJUNG	5777	3519
1	SANKHUWASHAVA	12181	9050
2	SOLUKHUMBU	8461	6286
3	E.MOUNTAIN	26419	18855
4	PANCHTHAR	1338	994

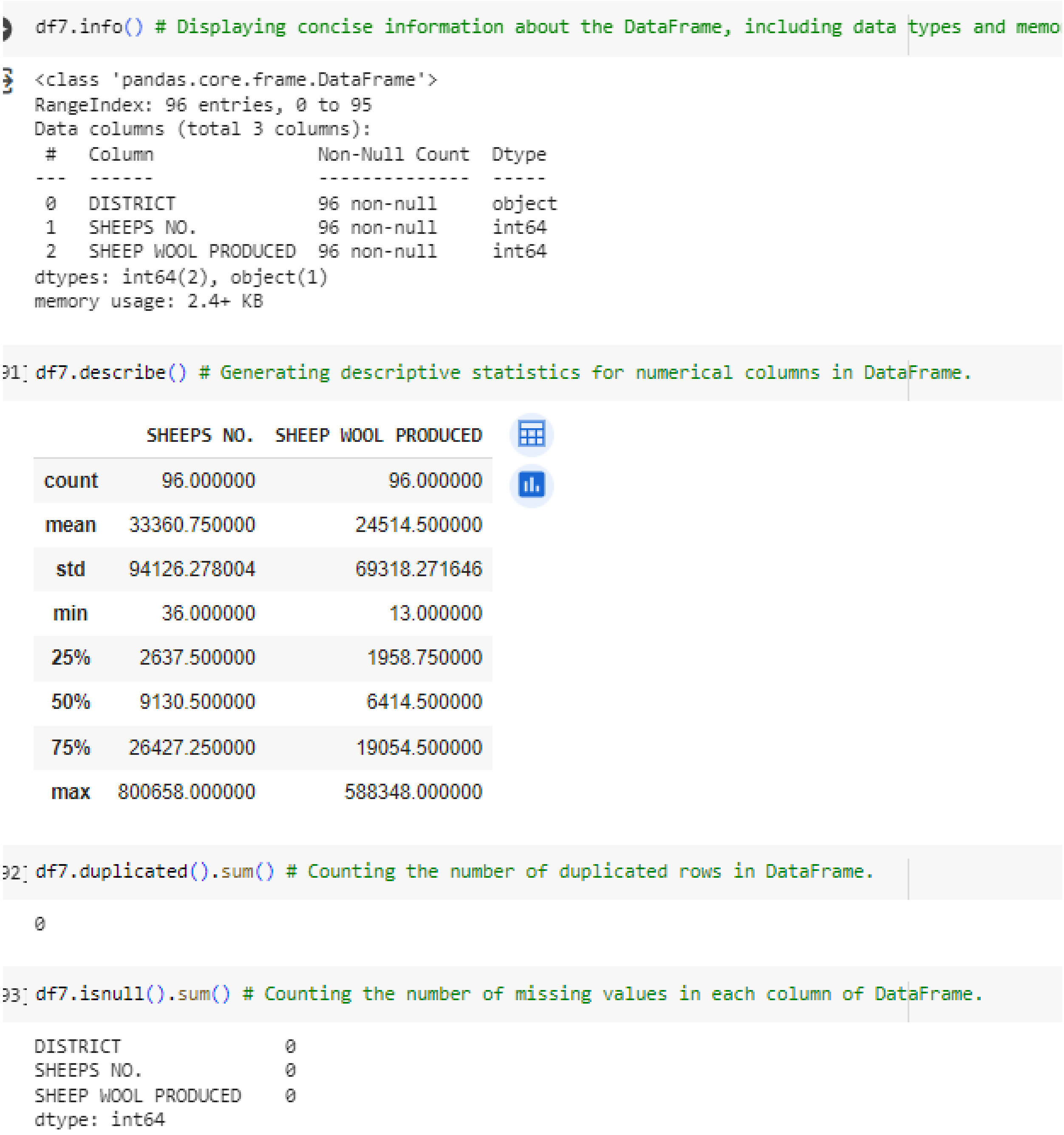
```
df7.tail() # Displaying the last few rows of the DataFrame for a quick overview of the dataset.
```

	DISTRICT	SHEEPS NO.	SHEEP WOOL PRODUCED
91	KAILALI	21267	15801
92	KANCHANPUR	7953	5909
93	FW.TERAI	29220	21710
94	FW.REGION	102571	76314
95	NEPAL	800658	588348

```
df7.shape # Retrieving the dimensions of the DataFrame (rows, columns).
```

```
(96, 3)
```

The exploration begins by simply looking at the first and last few rows of the dataset, to get a sense of the structure of the dataset. The dimensions (rows, columns) of the dataset are checked with the shape.



```
df7.info() # Displaying concise information about the DataFrame, including data types and memo
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96 entries, 0 to 95
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   DISTRICT             96 non-null     object
 1   SHEEPS NO.           96 non-null     int64 
 2   SHEEP WOOL PRODUCED  96 non-null     int64 
dtypes: int64(2), object(1)
memory usage: 2.4+ KB
```

```
91] df7.describe() # Generating descriptive statistics for numerical columns in DataFrame.
```

	SHEEPS NO.	SHEEP WOOL PRODUCED
count	96.000000	96.000000
mean	33360.750000	24514.500000
std	94126.278004	69318.271646
min	36.000000	13.000000
25%	2637.500000	1958.750000
50%	9130.500000	6414.500000
75%	26427.250000	19054.500000
max	800658.000000	588348.000000

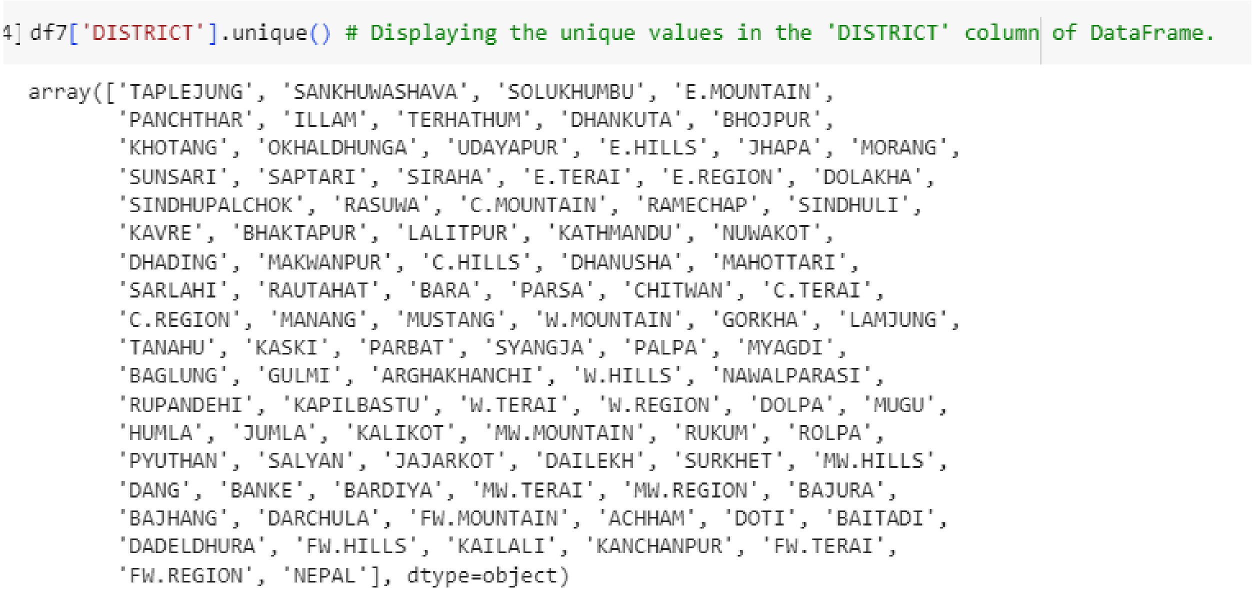
```
92] df7.duplicated().sum() # Counting the number of duplicated rows in DataFrame.
```

```
0
```

```
93] df7.isnull().sum() # Counting the number of missing values in each column of DataFrame.
```

```
DISTRICT               0
SHEEPS NO.             0
SHEEP WOOL PRODUCED    0
dtype: int64
```

To get basic information info() and comprehend the distribution of numerical columns, describe() is used to obtain statistical information. Duplicate().sum() is used to find duplicate rows, and isnull().sum() counts the number of missing values in each column.

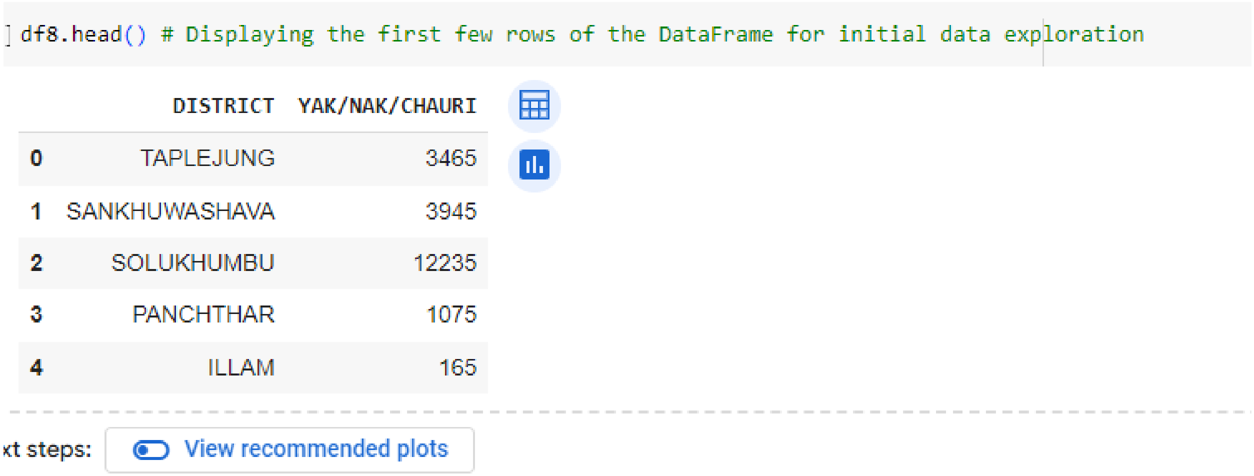


```
4] df7['DISTRICT'].unique() # Displaying the unique values in the 'DISTRICT' column of DataFrame.
```

```
array(['TAPLEJUNG', 'SANKHUWASHAVA', 'SOLUKHUMBU', 'E.MOUNTAIN',
       'PANCHTHAR', 'ILLAM', 'TERHATHUM', 'DHANKUTA', 'BHOJPUR',
       'KHOTANG', 'OKHALDHUNGA', 'UDAYAPUR', 'E.HILLS', 'JHAPA', 'MORANG',
       'SUNSARI', 'SAPTARI', 'SIRAHA', 'E.TERAI', 'E.REGION', 'DOLAKHA',
       'SINDHUPALCHOK', 'RASUWA', 'C.MOUNTAIN', 'RAMECHAP', 'SINDHULI',
       'KAVRE', 'BHAKTAPUR', 'LALITPUR', 'KATHMANDU', 'NUWAKOT',
       'DHADING', 'MAKWANPUR', 'C.HILLS', 'DHANUSHA', 'MAHOTTARI',
       'SARLAHI', 'RAUTAHAT', 'BARA', 'PARSA', 'CHITWAN', 'C.TERAI',
       'C.REGION', 'MANANG', 'MUSTANG', 'W.MOUNTAIN', 'GORKHA', 'LAMJUNG',
       'TANAHU', 'KASKI', 'PARBAT', 'SYANGJA', 'PALPA', 'MYAGDI',
       'BAGLUNG', 'GULMI', 'ARGHAKHANCHI', 'W.HILLS', 'NAWALPARASI',
       'RUPANDEHI', 'KAPILBASTU', 'W.TERAI', 'W.REGION', 'DOLPA', 'MUGU',
       'HUMLA', 'JUMLA', 'KALIKOT', 'MW.MOUNTAIN', 'RUKUM', 'ROLPA',
       'PYUTHAN', 'SALYAN', 'JAJARKOT', 'DAILEKH', 'SURKHET', 'MW.HILLS',
       'DANG', 'BANKE', 'BARDIYA', 'MW.TERAI', 'MW.REGION', 'BAJURA',
       'BAJHANG', 'DARCHULA', 'FW.MOUNTAIN', 'ACHHAM', 'DOTI', 'BAITADI',
       'DADELDHURA', 'FW.HILLS', 'KAILALI', 'KANCHANPUR', 'FW.TERAI',
       'FW.REGION', 'NEPAL'], dtype=object)
```

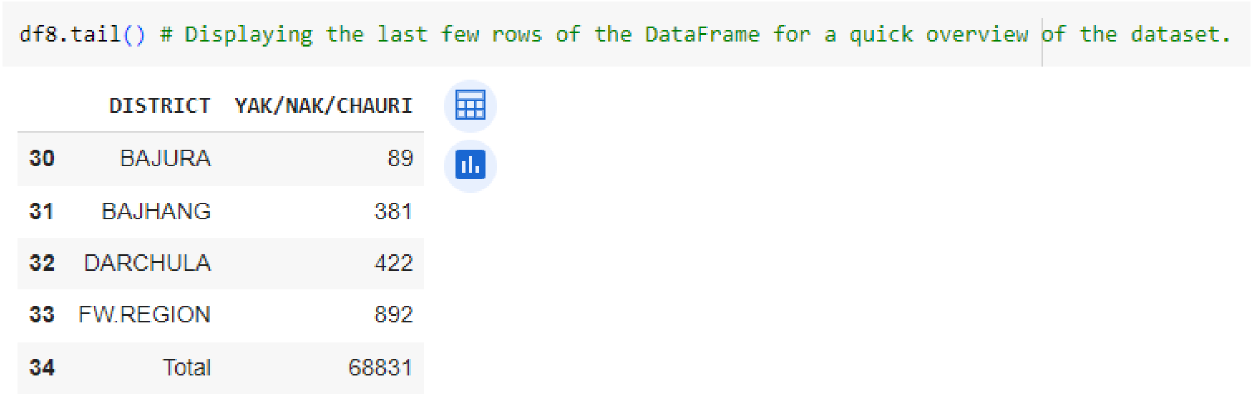
Using unique(), the unique values in this column are shown. No additional cleaning was required.

8. Yak, Nak, Chauri Population (df8):



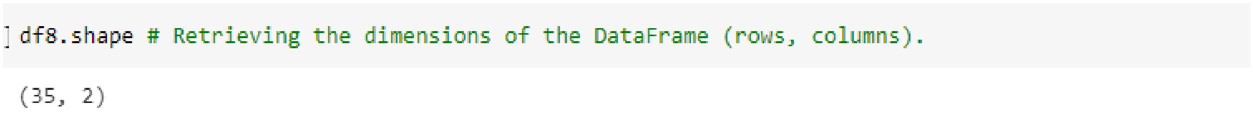
```
] df8.head() # Displaying the first few rows of the DataFrame for initial data exploration
```

	DISTRICT	YAK/NAK/CHAURI
0	TAPLEJUNG	3465
1	SANKHUWASHAVA	3945
2	SOLUKHUMBU	12235
3	PANCHTHAR	1075
4	ILLAM	165



```
df8.tail() # Displaying the last few rows of the DataFrame for a quick overview of the dataset.
```

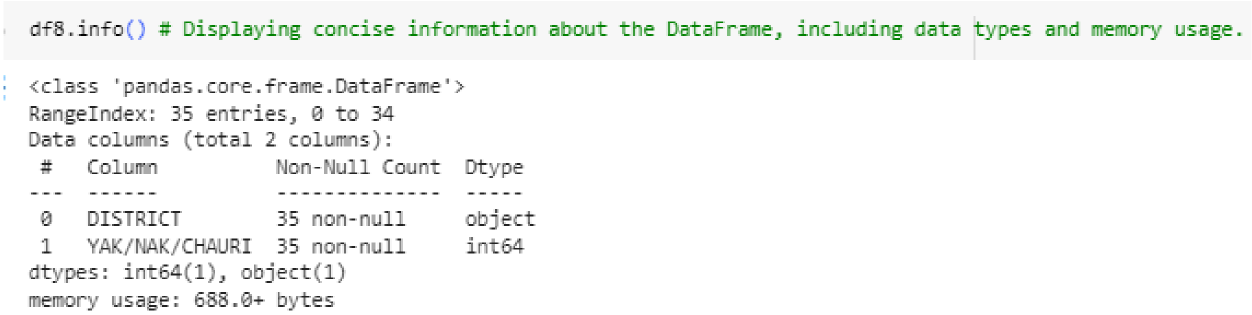
	DISTRICT	YAK/NAK/CHAURI
30	BAJURA	89
31	BAJHANG	381
32	DARCHULA	422
33	FW.REGION	892
34	Total	68831



```
] df8.shape # Retrieving the dimensions of the DataFrame (rows, columns).
```

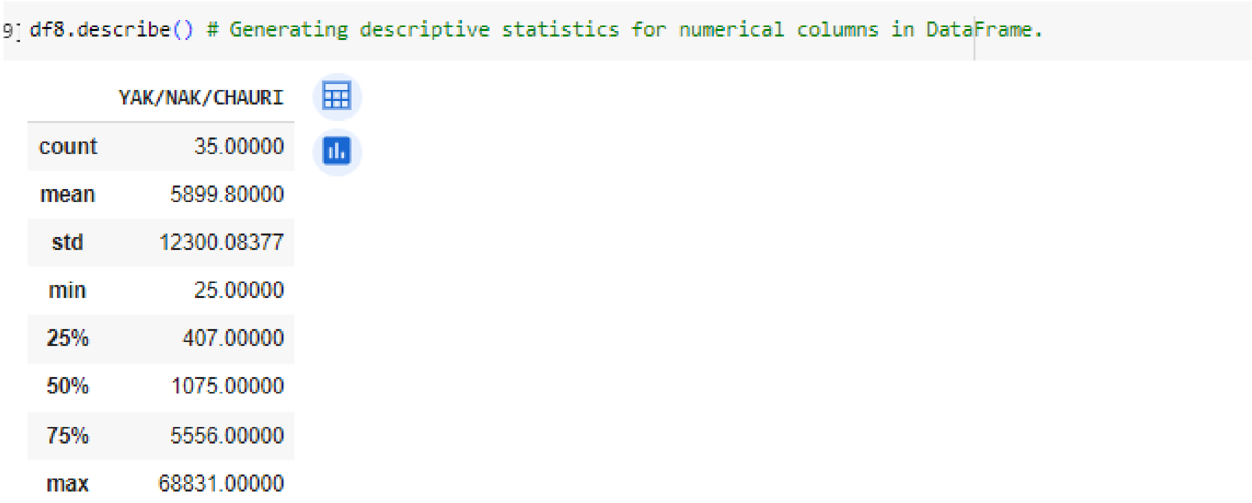
```
(35, 2)
```

To get a sense of the Yak, Nak, Chauri Population dataset's content, the first few rows of the dataset are examined during initial exploration. By revealing the number of rows and columns, the shape attribute helps to determine the size of the dataset.



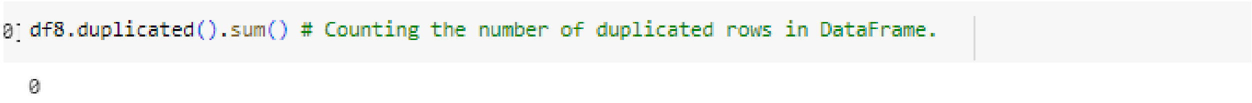
```
df8.info() # Displaying concise information about the DataFrame, including data types and memory usage.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35 entries, 0 to 34
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   DISTRICT        35 non-null     object
 1   YAK/NAK/CHAURI  35 non-null     int64 
dtypes: int64(1), object(1)
memory usage: 688.0+ bytes
```



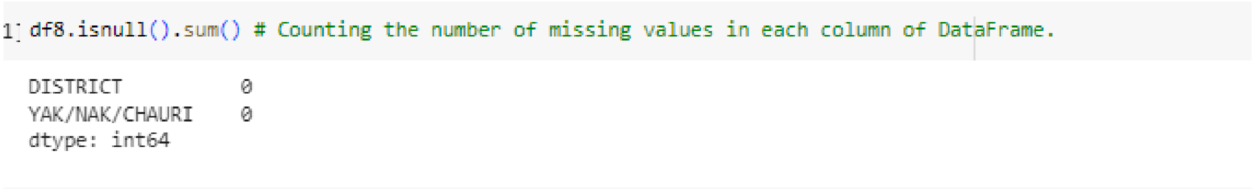
```
9] df8.describe() # Generating descriptive statistics for numerical columns in DataFrame.
```

	YAK/NAK/CHAURI
count	35.00000
mean	5899.80000
std	12300.08377
min	25.00000
25%	407.00000
50%	1075.00000
75%	5556.00000
max	68831.00000



```
0] df8.duplicated().sum() # Counting the number of duplicated rows in DataFrame.
```

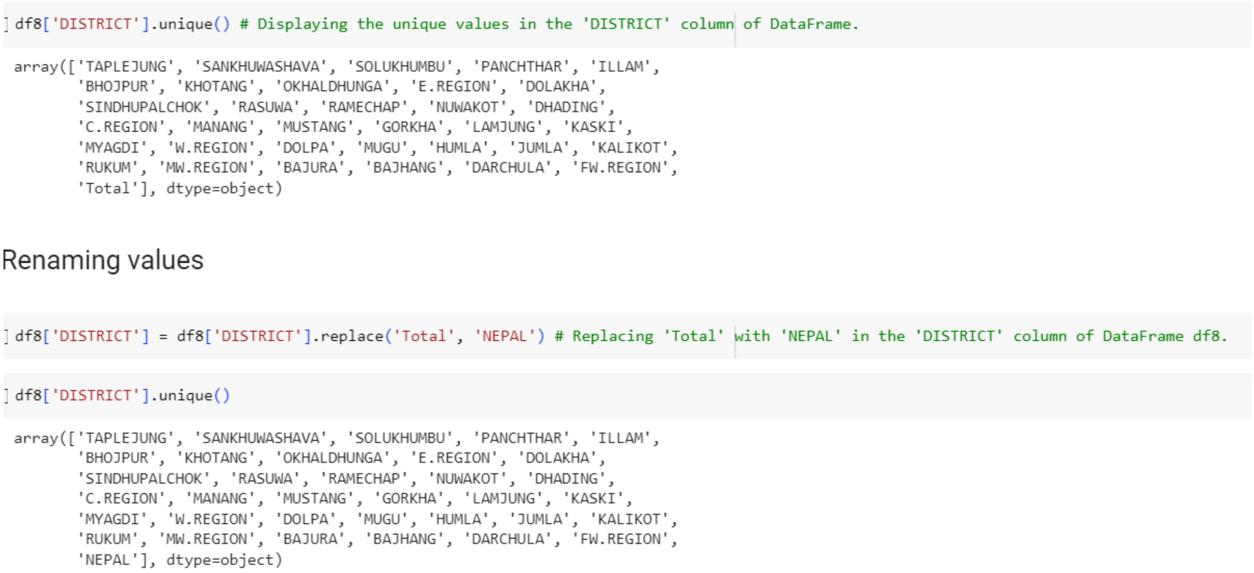
```
0
```



```
1] df8.isnull().sum() # Counting the number of missing values in each column of DataFrame.
```

```
DISTRICT          0
YAK/NAK/CHAURI    0
dtype: int64
```

Basic information was obtained using info() and numerical columns was summarized statistically using describe(), and possible problems with the quality of the data can be found by using duplicated() to check for duplicate .sum(). With isnull(), the existence of missing values is evaluated.sum() for every column.



```
df8['DISTRICT'].unique() # Displaying the unique values in the 'DISTRICT' column of DataFrame.
```

```
array(['TAPLEJUNG', 'SANKHUWASHAVA', 'SOLUKHUMBU', 'PANCHTHAR', 'ILLAM',
       'BHOJPUR', 'KHOTANG', 'OKHALDHUNGA', 'E.REGION', 'DOLAKHA',
       'SINDHUPALCHOK', 'RASUWA', 'RAMECHAP', 'NUWAKOT', 'DHADING',
       'C.REGION', 'MANANG', 'MUSTANG', 'GORKHA', 'LAMJUNG', 'KASKI',
       'MYAGDI', 'W.REGION', 'DOLPA', 'MUGU', 'HUMLA', 'JUMLA', 'KALIKOT',
       'RUKUM', 'MW.REGION', 'BAJURA', 'BAJHANG', 'DARCHULA', 'FW.REGION',
       'Total'], dtype=object)
```

Renaming values

```
df8['DISTRICT'] = df8['DISTRICT'].replace('Total', 'NEPAL') # Replacing 'Total' with 'NEPAL' in the 'DISTRICT' column of DataFrame df8.
```

```
df8['DISTRICT'].unique()
```

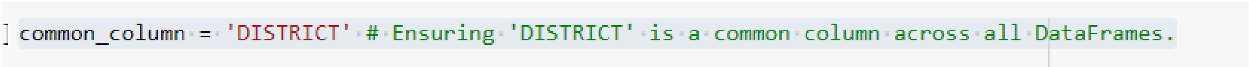
```
array(['TAPLEJUNG', 'SANKHUWASHAVA', 'SOLUKHUMBU', 'PANCHTHAR', 'ILLAM',
       'BHOJPUR', 'KHOTANG', 'OKHALDHUNGA', 'E.REGION', 'DOLAKHA',
       'SINDHUPALCHOK', 'RASUWA', 'RAMECHAP', 'NUWAKOT', 'DHADING',
       'C.REGION', 'MANANG', 'MUSTANG', 'GORKHA', 'LAMJUNG', 'KASKI',
       'MYAGDI', 'W.REGION', 'DOLPA', 'MUGU', 'HUMLA', 'JUMLA', 'KALIKOT',
       'RUKUM', 'MW.REGION', 'BAJURA', 'BAJHANG', 'DARCHULA', 'FW.REGION',
       'NEPAL'], dtype=object)
```

Using unique(), the key identifier 'DISTRICT' column is examined for unique values. To improve consistency, the 'DISTRICT' column's 'Total' is changed to 'NEPAL' in the data cleaning process.

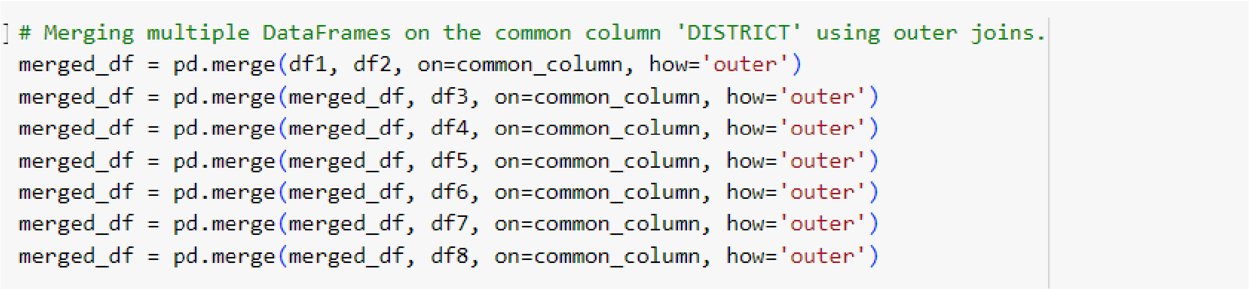
Data Cleaning and Processing of Merged Datasets

Merging All Datasets

To enable a thorough analysis of livestock and commodity production across different districts in Nepal, it is essential to merge all datasets in order to combine information from diverse sources.



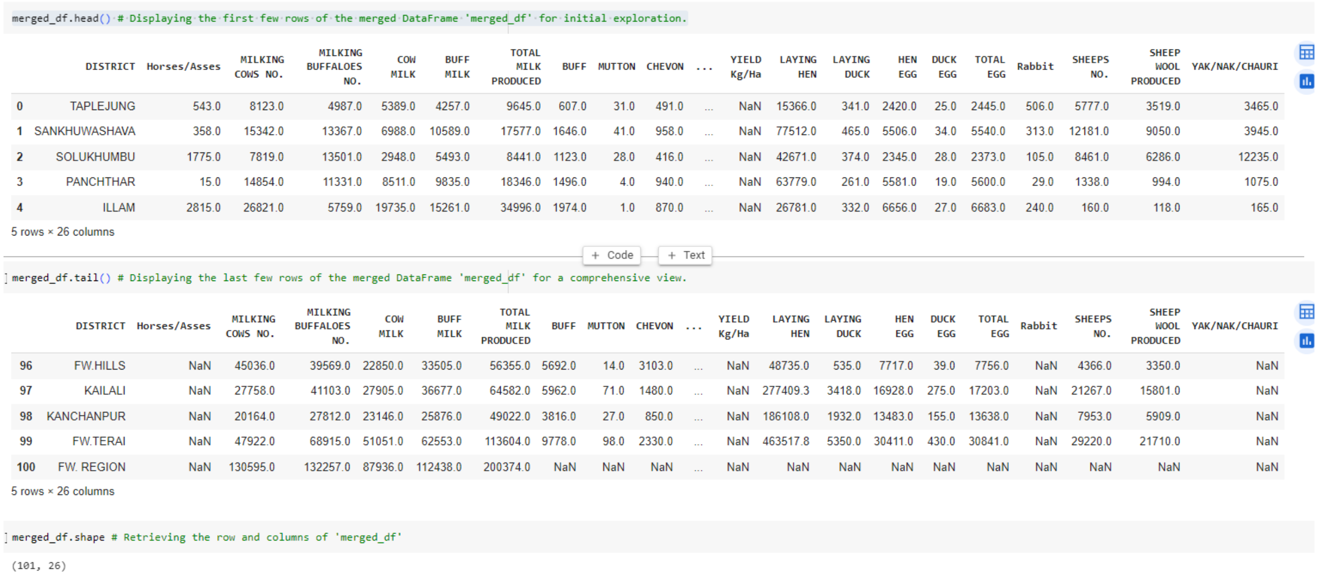
```
common_column = 'DISTRICT' # Ensuring 'DISTRICT' is a common column across all DataFrames.
```



```
# Merging multiple DataFrames on the common column 'DISTRICT' using outer joins.
merged_df = pd.merge(df1, df2, on=common_column, how='outer')
merged_df = pd.merge(merged_df, df3, on=common_column, how='outer')
merged_df = pd.merge(merged_df, df4, on=common_column, how='outer')
merged_df = pd.merge(merged_df, df5, on=common_column, how='outer')
merged_df = pd.merge(merged_df, df6, on=common_column, how='outer')
merged_df = pd.merge(merged_df, df7, on=common_column, how='outer')
merged_df = pd.merge(merged_df, df8, on=common_column, how='outer')
```

Finally, we can merge all of the datasets together into one. To do this, we will use pandas' 'merge' method, and we will merge the datasets together one by one. 'DISTRICT' will be the common column that stitches the datasets together. Since we want to make sure that any unique values of 'DISTRICT' are included in the merged dataset (so that we don't unintentionally leave any data out), we will employ an 'outer' join type. Information from every individual dataset (df1 to df8) is combined during the merging process until we finally get to one DataFrame called 'merged_df.' This 'merged_df' DataFrame now includes information on various facets of agricultural production in the many different districts of Nepal.

Exploring the Merged Dataset



```
merged_df.head() # Displaying the first few rows of the merged DataFrame 'merged_df' for initial exploration.
```

	DISTRICT	Horses/Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	...	YIELD Kg/Ha	LAYING HEN	LAYING DUCK	HEN EGG	DUCK EGG	TOTAL EGG	Rabbit	SHEEPS NO.	SHEEP WOOL PRODUCED	YAK/NAK/CHAURI
0	TAPLEJUNG	543.0	8123.0	4987.0	5389.0	4257.0	9645.0	607.0	31.0	491.0	...	NaN	15366.0	341.0	2420.0	25.0	2445.0	506.0	5777.0	3519.0	3465.0
1	SANKHUWASHAVA	358.0	15342.0	13367.0	6988.0	10589.0	17577.0	1646.0	41.0	958.0	...	NaN	77512.0	465.0	5506.0	34.0	5540.0	313.0	12181.0	9050.0	3945.0
2	SOLUKHUMBU	1775.0	7819.0	13501.0	2948.0	5493.0	8441.0	1123.0	28.0	416.0	...	NaN	42671.0	374.0	2345.0	28.0	2373.0	105.0	8461.0	6286.0	12235.0
3	PANCHTHAR	15.0	14854.0	11331.0	8511.0	9835.0	18346.0	1496.0	4.0	940.0	...	NaN	63779.0	261.0	5581.0	19.0	5600.0	29.0	1338.0	994.0	1075.0
4	ILLAM	2815.0	26821.0	5759.0	19735.0	15261.0	34996.0	1974.0	1.0	870.0	...	NaN	26781.0	332.0	6656.0	27.0	6683.0	240.0	160.0	118.0	165.0

5 rows × 26 columns

```
] merged_df.tail() # Displaying the last few rows of the merged DataFrame 'merged_df' for a comprehensive view.
```

	DISTRICT	Horses/Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	...	YIELD Kg/Ha	LAYING HEN	LAYING DUCK	HEN EGG	DUCK EGG	TOTAL EGG	Rabbit	SHEEPS NO.	SHEEP WOOL PRODUCED	YAK/NAK/CHAURI
96	FW.HILLS	NaN	45036.0	39569.0	22850.0	33505.0	56355.0	5692.0	14.0	3103.0	...	NaN	48735.0	535.0	7717.0	39.0	7756.0	NaN	4366.0	3350.0	NaN
97	KAILALI	NaN	27758.0	41103.0	27905.0	36677.0	64582.0	5962.0	71.0	1480.0	...	NaN	277409.3	3418.0	16928.0	275.0	17203.0	NaN	21267.0	15801.0	NaN
98	KANCHANPUR	NaN	20164.0	27812.0	23146.0	25876.0	49022.0	3816.0	27.0	850.0	...	NaN	186108.0	1932.0	13483.0	155.0	13638.0	NaN	7953.0	5909.0	NaN
99	FW.TERAI	NaN	47922.0	68915.0	51051.0	62553.0	113604.0	9778.0	98.0	2330.0	...	NaN	463517.8	5350.0	30411.0	430.0	30841.0	NaN	29220.0	21710.0	NaN
100	FW. REGION	NaN	130595.0	132257.0	87936.0	112438.0	200374.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

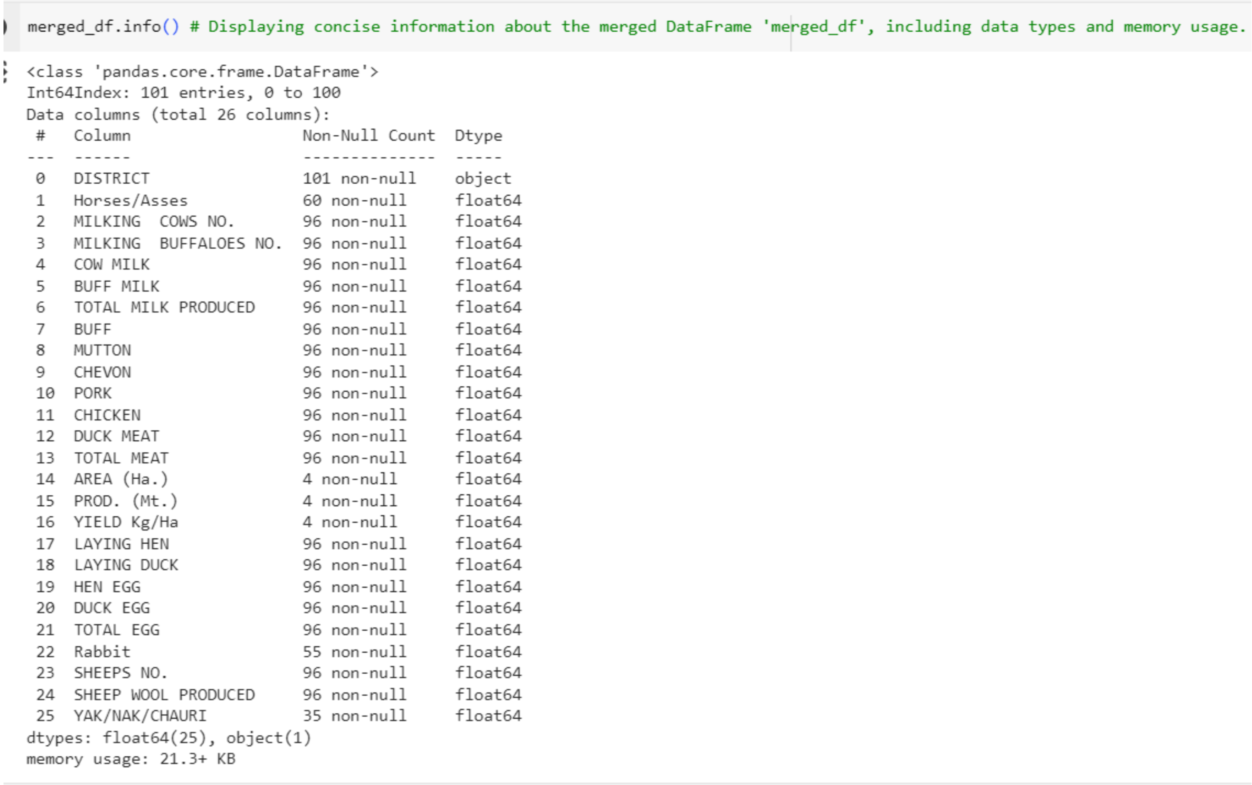
5 rows × 26 columns

```
] merged_df.shape # Retrieving the row and columns of 'merged_df'
```

```
(101, 26)
```

In this section, the merged dataset'merged_df' is examined. The first few rows are shown using the 'head()' method, giving the user a preview of the data. In a similar vein, the final few rows are displayed using the 'tail()' method, providing a full view of the dataset. Next, the dimensions of the'merged_df' DataFrame—that is, its number of rows and columns—are obtained using the'shape' attribute.

Overview of Merged Dataset (merged_df)

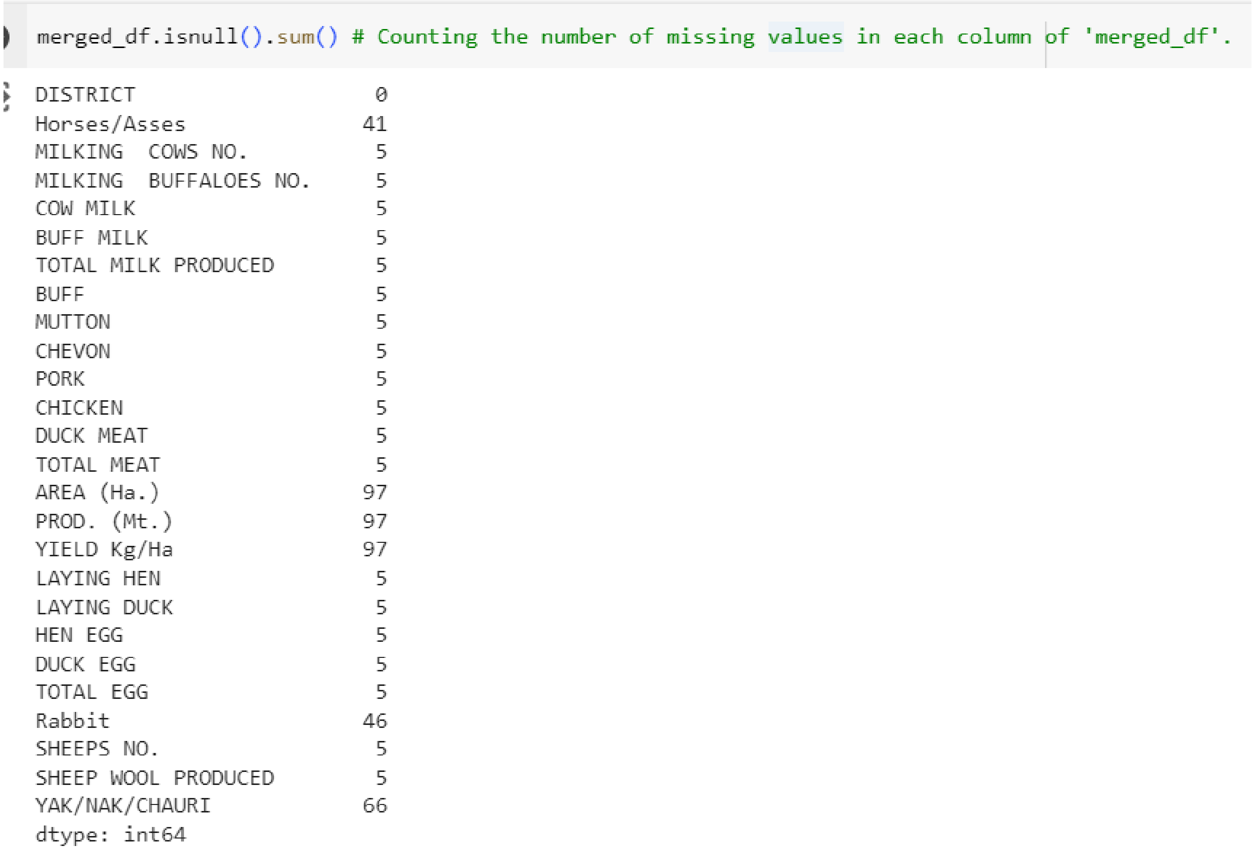


```
merged_df.info() # Displaying concise information about the merged DataFrame 'merged_df', including data types and memory usage.
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 101 entries, 0 to 100
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   DISTRICT               101 non-null    object
 1   Horses/Asses           60 non-null     float64
 2   MILKING  COWS NO.      96 non-null     float64
 3   MILKING  BUFFALOES NO. 96 non-null     float64
 4   COW MILK               96 non-null     float64
 5   BUFF MILK              96 non-null     float64
 6   TOTAL MILK PRODUCED    96 non-null     float64
 7   BUFF                   96 non-null     float64
 8   MUTTON                 96 non-null     float64
 9   CHEVON                 96 non-null     float64
 10  PORK                   96 non-null     float64
 11  CHICKEN                96 non-null     float64
 12  DUCK MEAT              96 non-null     float64
 13  TOTAL MEAT             96 non-null     float64
 14  AREA (Ha.)             4 non-null      float64
 15  PROD. (Mt.)            4 non-null      float64
 16  YIELD Kg/Ha            4 non-null      float64
 17  LAYING HEN             96 non-null     float64
 18  LAYING DUCK            96 non-null     float64
 19  HEN EGG                96 non-null     float64
 20  DUCK EGG               96 non-null     float64
 21  TOTAL EGG              96 non-null     float64
 22  Rabbit                 55 non-null     float64
 23  SHEEPS NO.             96 non-null     float64
 24  SHEEP WOOL PRODUCED    96 non-null     float64
 25  YAK/NAK/CHAURI         35 non-null     float64
dtypes: float64(25), object(1)
memory usage: 21.3+ KB
```

The output of merged_df.info() offers a thorough summary of the combined dataset'merged_df.' The dataset comprises 26 columns and 101 entries, or rows. The district names are represented by object data types in the 'DISTRICT' column. The float64 data type is used for numerical columns with names like "Horses/Asses," "MILKING COWS NO.," and others. Missing values are present in multiple columns, as shown by non-null counts < 101. Limited non-null entries are displayed in certain columns, such as 'AREA (Ha.),' 'PROD. (Mt.),' 'YIELD Kg/Ha,' and 'YAK/NAK/CHAURI'. Understanding the structure of the dataset, identifying areas where data is missing, and organizing the next steps for data cleaning and processing all depend on this information.

Missing Values in Merged Dataset (merged_df)

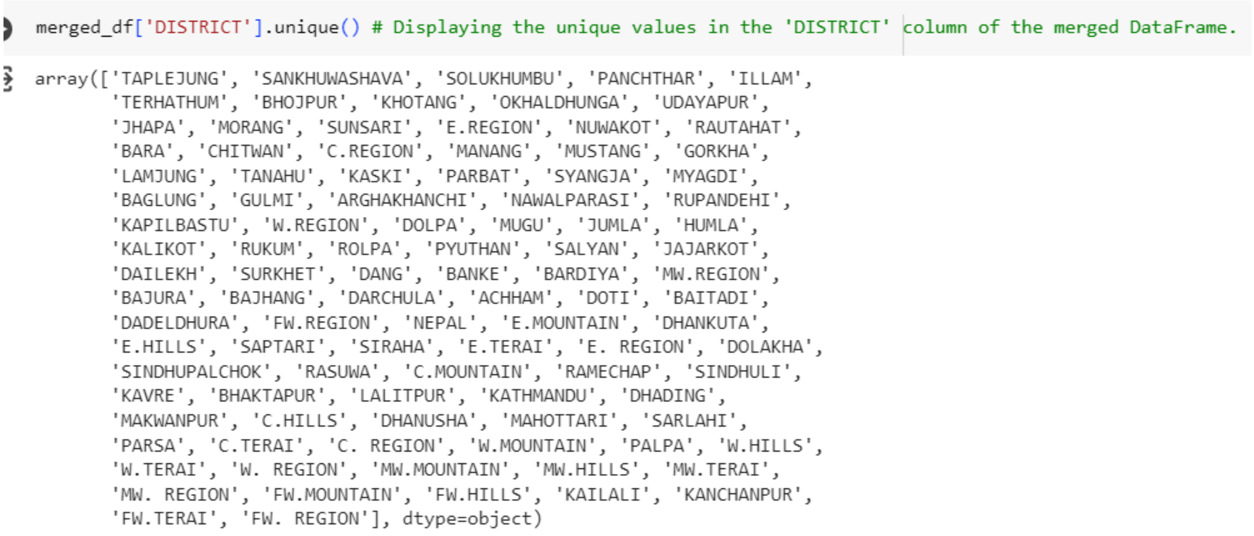


```
merged_df.isnull().sum() # Counting the number of missing values in each column of 'merged_df'.
```

```
DISTRICT                   0
Horses/Asses              41
MILKING  COWS NO.          5
MILKING  BUFFALOES NO.     5
COW MILK                   5
BUFF MILK                  5
TOTAL MILK PRODUCED        5
BUFF                       5
MUTTON                     5
CHEVON                     5
PORK                       5
CHICKEN                    5
DUCK MEAT                  5
TOTAL MEAT                 5
AREA (Ha.)                97
PROD. (Mt.)               97
YIELD Kg/Ha               97
LAYING HEN                 5
LAYING DUCK                5
HEN EGG                    5
DUCK EGG                   5
TOTAL EGG                  5
Rabbit                    46
SHEEPS NO.                 5
SHEEP WOOL PRODUCED        5
YAK/NAK/CHAURI            66
dtype: int64
```

The combined dataset'merged_df' has multiple columns with missing values. Notably, there are no missing values in the 'DISTRICT' column, suggesting that all entries contain the names of the districts. On the other hand, there is varied degrees of missing data in other columns. Interestingly, there are comparatively few missing values in the columns pertaining to agriculture and meat production, including 'Horses/Asses,' 'MILKING COWS NO.,' 'MILKING BUFFALOES NO.,' and others. Conversely, with 97 entries each, columns such as 'AREA (Ha.),' 'PROD. (Mt.),' and 'YIELD Kg/Ha' exhibit a greater frequency of missing values. The 'YAK/NAK/CHAURI' column also has 66 missing values.

Districts in Merged Dataset (merged_df)

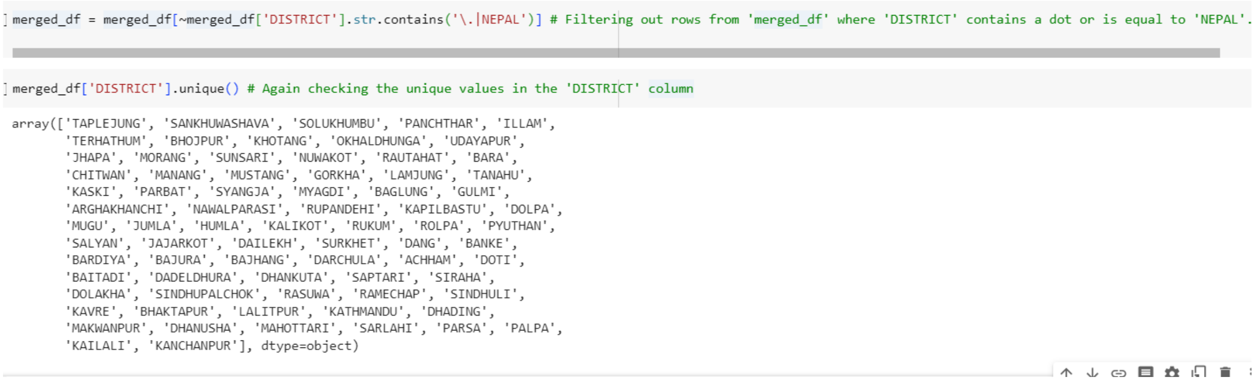


```
merged_df['DISTRICT'].unique() # Displaying the unique values in the 'DISTRICT' column of the merged DataFrame.
```

```
array(['TAPLEJUNG', 'SANKHUWASHAVA', 'SOLUKHUMBU', 'PANCHTHAR', 'ILLAM',
       'TERHATHUM', 'BHOJPUR', 'KHOTANG', 'OKHALDHUNGA', 'UDAYAPUR',
       'JHAPA', 'MORANG', 'SUNSARI', 'E.REGION', 'NUWAKOT', 'RAUTAHAT',
       'BARA', 'CHITWAN', 'C.REGION', 'MANANG', 'MUSTANG', 'GORKHA',
       'LAMJUNG', 'TANAHU', 'KASKI', 'PARBAT', 'SYANGJA', 'MYAGDI',
       'BAGLUNG', 'GULMI', 'ARGHAKHANCHI', 'NAWALPARASI', 'RUPANDEHI',
       'KAPILBASTU', 'W.REGION', 'DOLPA', 'MUGU', 'JUMLA', 'HUMLA',
       'KALIKOT', 'RUKUM', 'ROLPA', 'PYUTHAN', 'SALYAN', 'JAJARKOT',
       'DAILEKH', 'SURKHET', 'DANG', 'BANKE', 'BARDIYA', 'MW.REGION',
       'BAJURA', 'BAJHANG', 'DARCHULA', 'ACHHAM', 'DOTI', 'BAITADI',
       'DADELDHURA', 'FW.REGION', 'NEPAL', 'E.MOUNTAIN', 'DHANKUTA',
       'E.HILLS', 'SAPTARI', 'SIRAHA', 'E.TERAI', 'E. REGION', 'DOLAKHA',
       'SINDHUPALCHOK', 'RASUWA', 'C.MOUNTAIN', 'RAMECHAP', 'SINDHULI',
       'KAVRE', 'BHAKTAPUR', 'LALITPUR', 'KATHMANDU', 'DHADING',
       'MAKWANPUR', 'C.HILLS', 'DHANUSHA', 'MAHOTTARI', 'SARLAHI',
       'PARSA', 'C.TERAI', 'C. REGION', 'W.MOUNTAIN', 'PALPA', 'W.HILLS',
       'W.TERAI', 'W. REGION', 'MW.MOUNTAIN', 'MW.HILLS', 'MW.TERAI',
       'MW. REGION', 'FW.MOUNTAIN', 'FW.HILLS', 'KAILALI', 'KANCHANPUR',
       'FW.TERAI', 'FW. REGION'], dtype=object)
```

A wide variety of districts from various parts of Nepal are included in the 'DISTRICT' column of the combined dataset'merged_df'. Districts classified under specific regions such as 'E.REGION,' 'C.REGION,' 'W.REGION,' 'MW.REGION,' 'FW.REGION,' and others are among the unique values in this column, along with districts from the eastern, central, and western regions.

Data Cleaning



```
merged_df = merged_df[~merged_df['DISTRICT'].str.contains('\.|NEPAL')] # Filtering out rows from 'merged_df' where 'DISTRICT' contains a dot or is equal to 'NEPAL'.
```

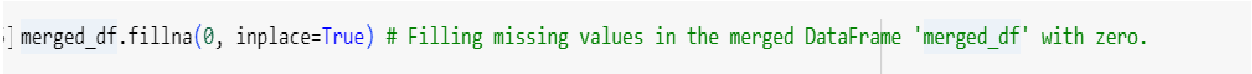
```
merged_df['DISTRICT'].unique() # Again checking the unique values in the 'DISTRICT' column
```

```
array(['TAPLEJUNG', 'SANKHUWASHAVA', 'SOLUKHUMBU', 'PANCHTHAR', 'ILLAM',
       'TERHATHUM', 'BHOJPUR', 'KHOTANG', 'OKHALDHUNGA', 'UDAYAPUR',
       'JHAPA', 'MORANG', 'SUNSARI', 'NUWAKOT', 'RAUTAHAT', 'BARA',
       'CHITWAN', 'MANANG', 'MUSTANG', 'GORKHA', 'LAMJUNG', 'TANAHU',
       'KASKI', 'PARBAT', 'SYANGJA', 'MYAGDI', 'BAGLUNG', 'GULMI',
       'ARGHAKHANCHI', 'NAWALPARASI', 'RUPANDEHI', 'KAPILBASTU', 'DOLPA',
       'MUGU', 'JUMLA', 'HUMLA', 'KALIKOT', 'RUKUM', 'ROLPA', 'PYUTHAN',
       'SALYAN', 'JAJARKOT', 'DAILEKH', 'SURKHET', 'DANG', 'BANKE',
       'BARDIYA', 'BAJURA', 'BAJHANG', 'DARCHULA', 'ACHHAM', 'DOTI',
       'BAITADI', 'DADELDHURA', 'DHANKUTA', 'SAPTARI', 'SIRAHA',
       'DOLAKHA', 'SINDHUPALCHOK', 'RASUWA', 'RAMECHAP', 'SINDHULI',
       'KAVRE', 'BHAKTAPUR', 'LALITPUR', 'KATHMANDU', 'DHADING',
       'MAKWANPUR', 'DHANUSHA', 'MAHOTTARI', 'SARLAHI', 'PARSA', 'PALPA',
       'KAILALI', 'KANCHANPUR'], dtype=object)
```

Some rows have been filtered out in order to guarantee a targeted analysis on individual districts and to eliminate aggregated or non-specific entries. Rows that have a dot ('.') in the 'DISTRICT' column or that equal 'NEPAL' have been removed. The purpose of this filtering step is to protect district-level data integrity from entries that represent larger categories or the entire nation.

Following this filtering process, a revised set of unique values can be seen in the 'DISTRICT' column in'merged_df'. In the absence of combined or national entries, these values reflect specific districts. The foundation for a more accurate analysis of livestock and commodity production in Nepal at the district level is laid by this careful curation of the dataset.

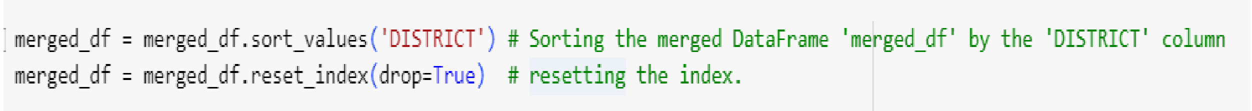
Handling Missing Values in Merged Dataset



```
merged_df.fillna(0, inplace=True) # Filling missing values in the merged DataFrame 'merged_df' with zero.
```

A methodical approach has been used to address missing values in the merged dataset 'merged_df'. The dataset has been made more comprehensive and consistent for further analysis by replacing all NaN (Not a Number) entries with zero. By strategically managing the missing values, the impact of incomplete data on the overall analysis is minimized and the dataset is prepared for exploration and modeling.

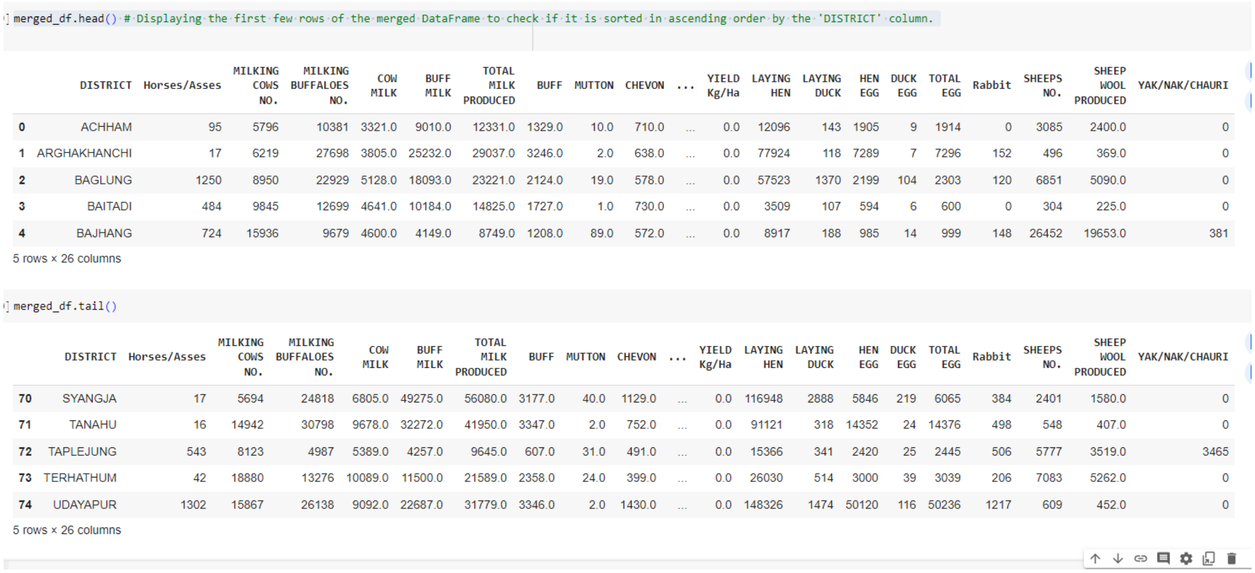
Sorting and Resetting Index



```
merged_df = merged_df.sort_values('DISTRICT') # Sorting the merged DataFrame 'merged_df' by the 'DISTRICT' column
merged_df = merged_df.reset_index(drop=True)  # resetting the index.
```

Based on the 'DISTRICT' column, the combined dataset'merged_df' has been sorted alphabetically. This configuration makes the text easier to read and makes comparing districts simpler. In order to preserve a sequential order the dataset's index has also been reset.

Head and Tail of Sorted Merged Dataset



```
] merged_df.head() # Displaying the first few rows of the merged DataFrame to check if it is sorted in ascending order by the 'DISTRICT' column.
```

	DISTRICT	Horses/Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	...	YIELD Kg/Ha	LAYING HEN	LAYING DUCK	HEN EGG	DUCK EGG	TOTAL EGG	Rabbit	SHEEPS NO.	SHEEP WOOL PRODUCED	YAK/NAK/CHAURI
0	ACHHAM	95	5796	10381	3321.0	9010.0	12331.0	1329.0	10.0	710.0	...	0.0	12096	143	1905	9	1914	0	3085	2400.0	0
1	ARGHAKHANCHI	17	6219	27698	3805.0	25232.0	29037.0	3246.0	2.0	638.0	...	0.0	77924	118	7289	7	7296	152	496	369.0	0
2	BAGLUNG	1250	8950	22929	5128.0	18093.0	23221.0	2124.0	19.0	578.0	...	0.0	57523	1370	2199	104	2303	120	6851	5090.0	0
3	BAITADI	484	9845	12699	4641.0	10184.0	14825.0	1727.0	1.0	730.0	...	0.0	3509	107	594	6	600	0	304	225.0	0
4	BAJHANG	724	15936	9679	4600.0	4149.0	8749.0	1208.0	89.0	572.0	...	0.0	8917	188	985	14	999	148	26452	19653.0	381

5 rows × 26 columns

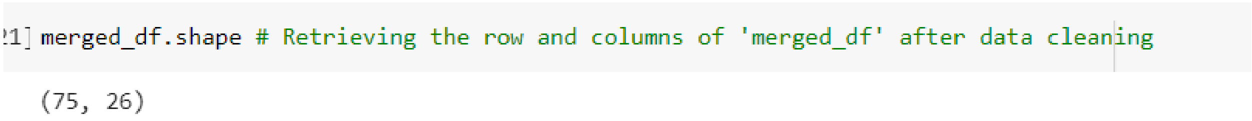
```
] merged_df.tail()
```

	DISTRICT	Horses/Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	...	YIELD Kg/Ha	LAYING HEN	LAYING DUCK	HEN EGG	DUCK EGG	TOTAL EGG	Rabbit	SHEEPS NO.	SHEEP WOOL PRODUCED	YAK/NAK/CHAURI
70	SYANGJA	17	5694	24818	6805.0	49275.0	56080.0	3177.0	40.0	1129.0	...	0.0	116948	2888	5846	219	6065	384	2401	1580.0	0
71	TANAHU	16	14942	30798	9678.0	32272.0	41950.0	3347.0	2.0	752.0	...	0.0	91121	318	14352	24	14376	498	548	407.0	0
72	TAPLEJUNG	543	8123	4987	5389.0	4257.0	9645.0	607.0	31.0	491.0	...	0.0	15366	341	2420	25	2445	506	5777	3519.0	3465
73	TERHATHUM	42	18880	13276	10089.0	11500.0	21589.0	2358.0	24.0	399.0	...	0.0	26030	514	3000	39	3039	206	7083	5262.0	0
74	UDAYAPUR	1302	15867	26138	9092.0	22687.0	31779.0	3346.0	2.0	1430.0	...	0.0	148326	1474	50120	116	50236	1217	609	452.0	0

5 rows × 26 columns

To verify that the merged dataset'merged_df' was sorted alphabetically using the 'DISTRICT' column, the first and last few rows of the sorted dataset are shown.

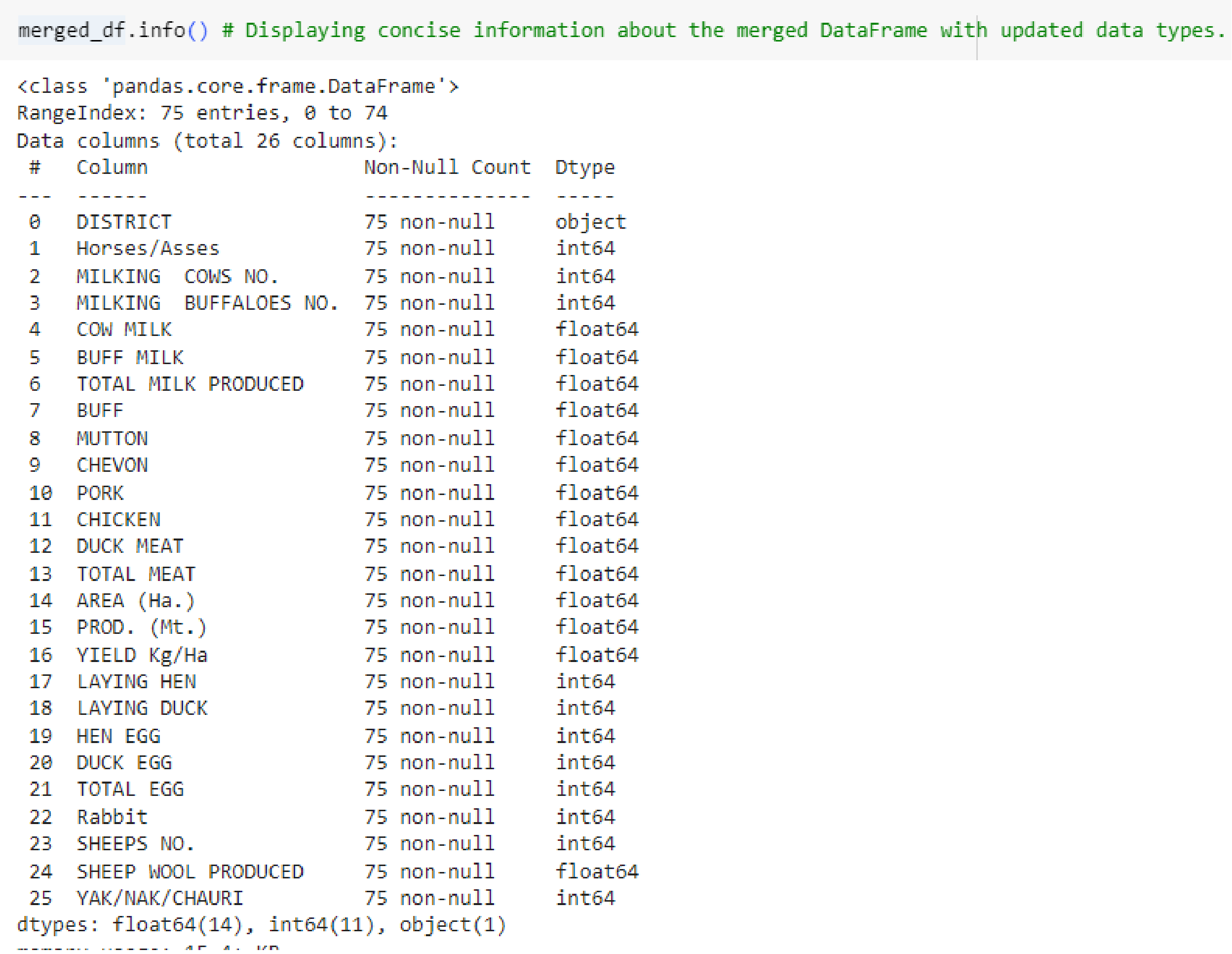
Final Dataset Dimensions



```
21] merged_df.shape # Retrieving the row and columns of 'merged_df' after data cleaning

(75, 26)
```

The combined dataset'merged_df' now has 26 columns and 75 rows following the data cleaning procedure. This is the refined dataset, which has been sorted and processed so that it is prepared for additional analysis. The elimination of unnecessary or insufficient data during the cleaning stage is the cause of the decrease in the number of rows.

Updated Dataset Information

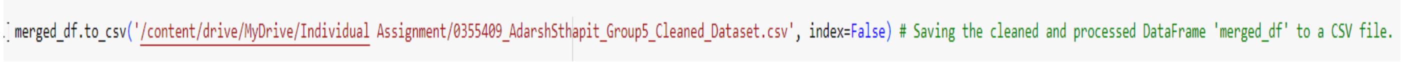


```
merged_df.info() # Displaying concise information about the merged DataFrame with updated data types.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 75 entries, 0 to 74
Data columns (total 26 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   DISTRICT                75 non-null     object 
 1   Horses/Asses            75 non-null     int64  
 2   MILKING  COWS NO.       75 non-null     int64  
 3   MILKING  BUFFALOES NO.  75 non-null     int64  
 4   COW MILK                75 non-null     float64
 5   BUFF MILK               75 non-null     float64
 6   TOTAL MILK PRODUCED     75 non-null     float64
 7   BUFF                    75 non-null     float64
 8   MUTTON                  75 non-null     float64
 9   CHEVON                  75 non-null     float64
 10  PORK                    75 non-null     float64
 11  CHICKEN                 75 non-null     float64
 12  DUCK MEAT               75 non-null     float64
 13  TOTAL MEAT              75 non-null     float64
 14  AREA (Ha.)              75 non-null     float64
 15  PROD. (Mt.)             75 non-null     float64
 16  YIELD Kg/Ha             75 non-null     float64
 17  LAYING HEN              75 non-null     int64  
 18  LAYING DUCK             75 non-null     int64  
 19  HEN EGG                 75 non-null     int64  
 20  DUCK EGG                75 non-null     int64  
 21  TOTAL EGG               75 non-null     int64  
 22  Rabbit                  75 non-null     int64  
 23  SHEEPS NO.              75 non-null     int64  
 24  SHEEP WOOL PRODUCED     75 non-null     float64
 25  YAK/NAK/CHAURI          75 non-null     int64  
dtypes: float64(14), int64(11), object(1)
```

The combined dataset'merged_df' has been improved with updated data types after the data cleaning and processing stages. 'DISTRICT' is still an object type in the dataset, which now has 75 entries and 26 columns. The numerical columns have been converted to integers values.

Saving the Cleaned Dataset



```
merged_df.to_csv('/content/drive/MyDrive/Individual Assignment/0355409_AdarshSthapit_Group5_Cleaned_Dataset.csv', index=False) # Saving the cleaned and processed DataFrame 'merged_df' to a CSV file.
```

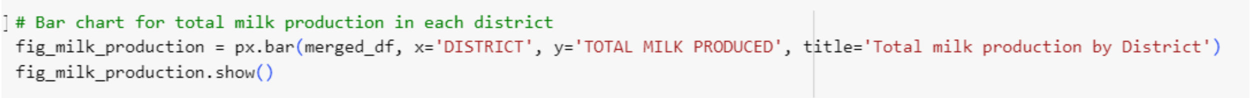
The'merged_df' dataset has been successfully cleaned and processed, and saved as '0355409_AdarshSthapit_Group5_Cleaned_Dataset.csv,' a CSV file. The refined data will be preserved for use in future research and analysis. The CSV file is kept in the directory '/content/drive/MyDrive/Individual Assignment/.'

Visualization

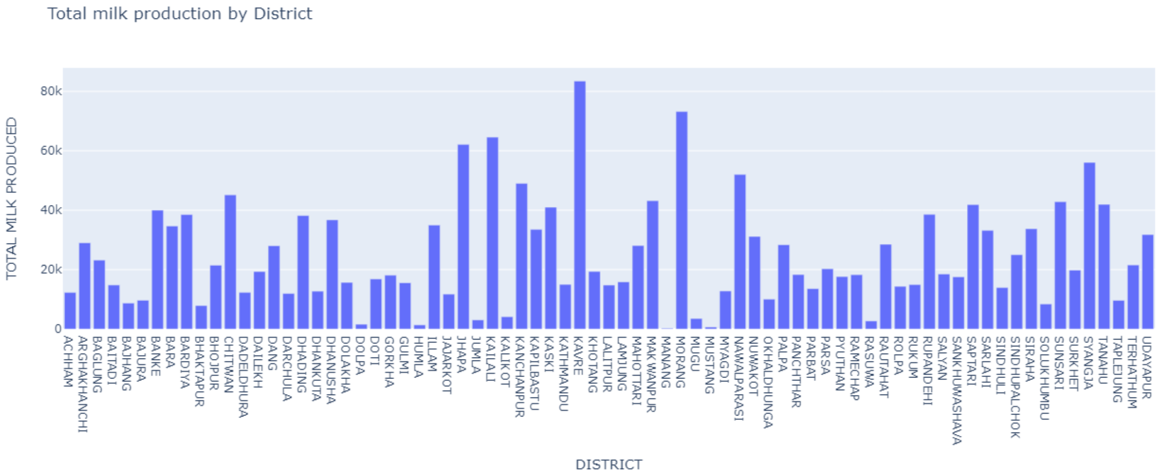
An important tool for understanding the properties of the dataset is its visualization. Several visualizations have been used in this section to give readers a thorough understanding of the data. Interactive visualizations have been produced using libraries like Matplotlib.pyplot, Seaborn, and Plotly Express.

Bar Chart

Total Milk Production by District

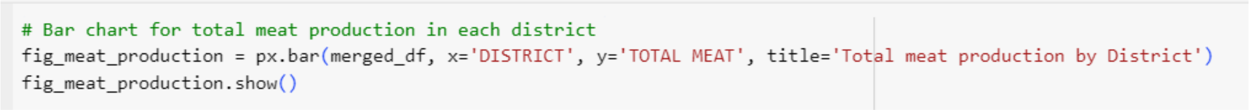


```
# Bar chart for total milk production in each district
fig_milk_production = px.bar(merged_df, x='DISTRICT', y='TOTAL MILK PRODUCED', title='Total milk production by District')
fig_milk_production.show()
```

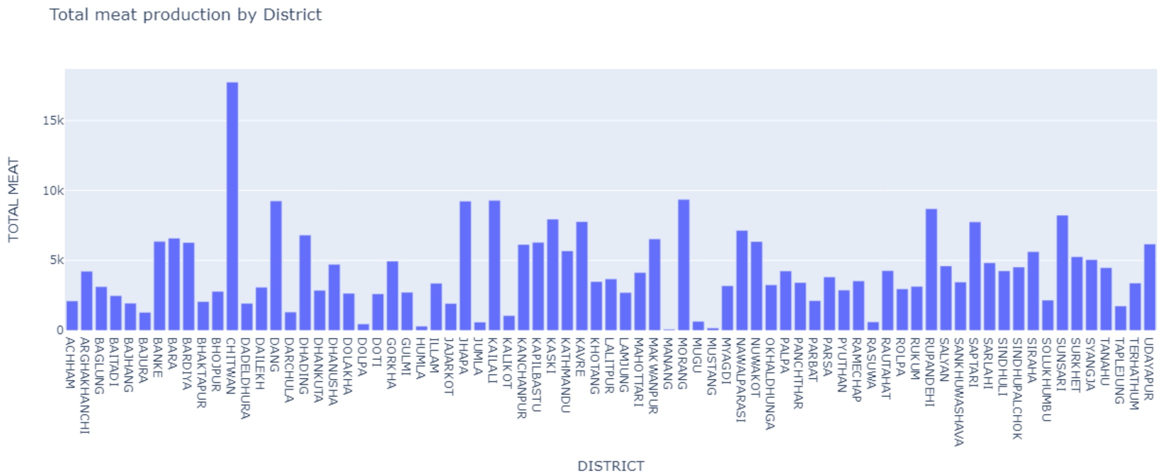


The total amount of milk produced in every district is visualized in the bar chart found above. Each bar represents a district and the height of each bar shows the total milk production of the respective district. As a result of this visual, it's easy to compare the amounts of milk produced in different regions. Districts with higher bars denote higher total milk production, and therefore give a short summary of the distribution of milk production in the dataset.

Total Meat Production by District

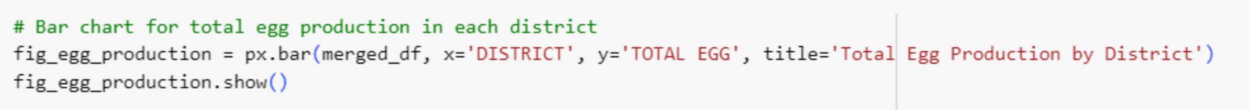


```
# Bar chart for total meat production in each district
fig_meat_production = px.bar(merged_df, x='DISTRICT', y='TOTAL MEAT', title='Total meat production by District')
fig_meat_production.show()
```

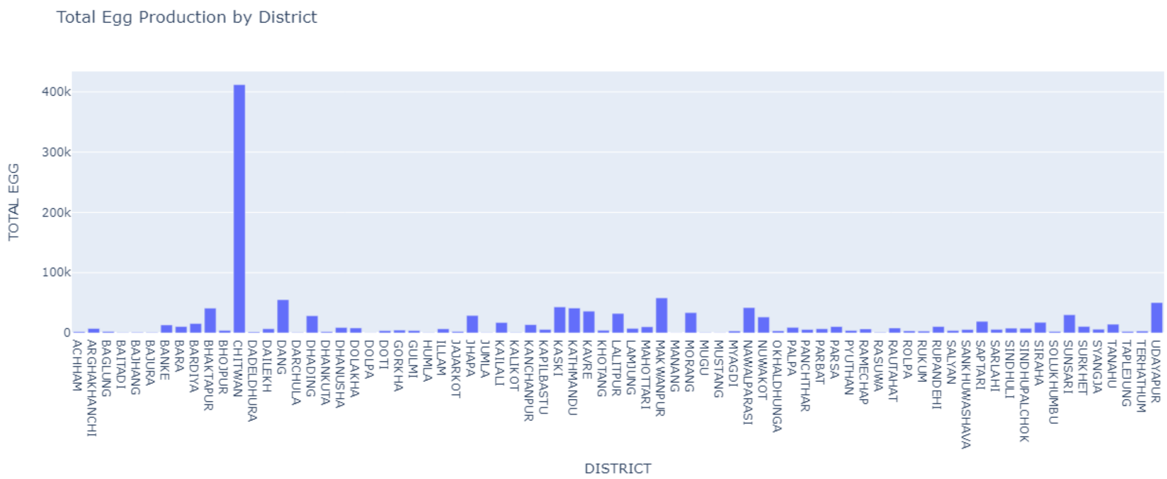


The total amount of meat produced in each district is shown in the above bar chart. Every bar signifies a district, and the height of the bar denotes the total amount of meat produced in that particular area. Greater total meat production is indicated by higher bars, which provide information about the dataset's distribution of meat production.

Total Egg Production by District



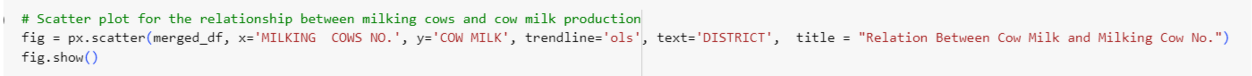
```
# Bar chart for total egg production in each district
fig_egg_production = px.bar(merged_df, x='DISTRICT', y='TOTAL EGG', title='Total Egg Production by District')
fig_egg_production.show()
```



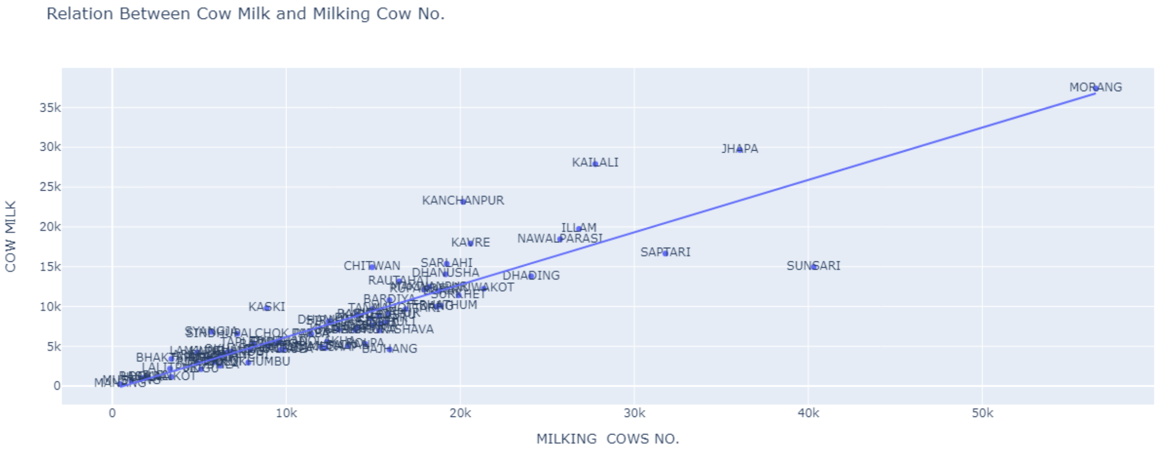
An overview of each district's total egg production is given by the bar chart. Every bar represents a district, and the height of the bar shows the total amount of eggs produced in that particular area. It is possible to compare the levels of egg production in various districts due to this visual representation. Greater total egg production is indicated by higher bars, which provide information about the distribution of egg production within the dataset.

Scatter plot

Relation Between Cow Milk and Milking Cow No.

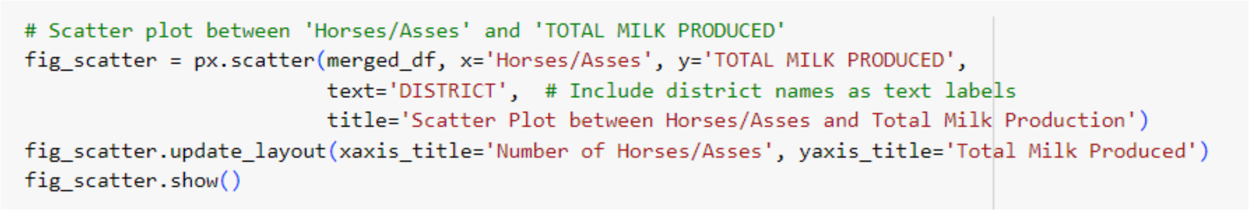


```
# Scatter plot for the relationship between milking cows and cow milk production
fig = px.scatter(merged_df, x='MILKING  COWS NO.', y='COW MILK', trendline='ols', text='DISTRICT',  title = "Relation Between Cow Milk and Milking Cow No.")
fig.show()
```

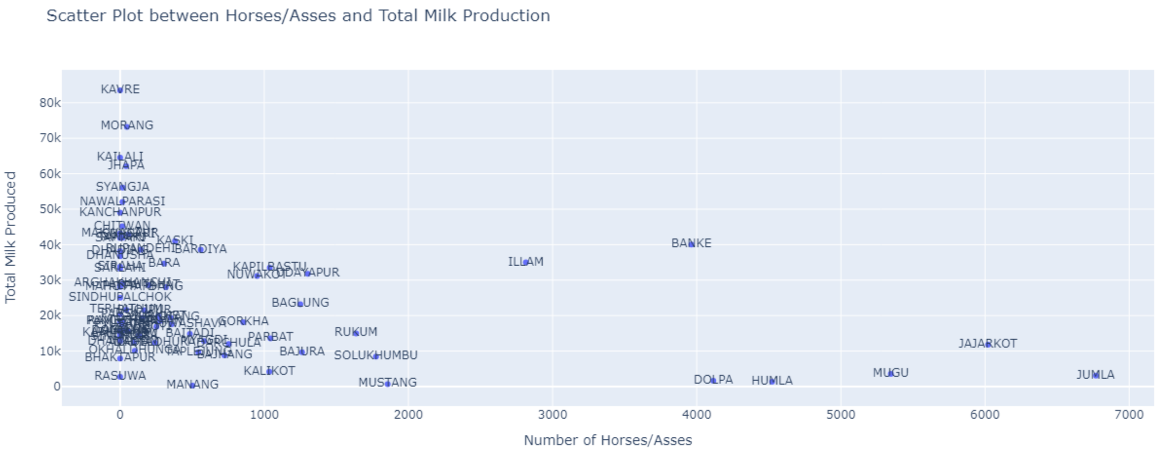


The relationship between the quantity of milking cows and the amount of milk produced in each district is depicted in this scatter plot. The x-axis shows the number of cows being milked, and the y-axis shows the corresponding amount of milk produced by the cows. Each point on the plot represents a district. The possible linear relationship between these two variables is represented visually by the trendline.

Scatter Plot between Horses/Asses and Total Milk Production



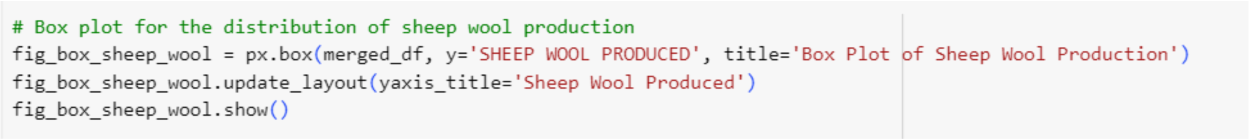
```
# Scatter plot between 'Horses/Asses' and 'TOTAL MILK PRODUCED'
fig_scatter = px.scatter(merged_df, x='Horses/Asses', y='TOTAL MILK PRODUCED',
                         text='DISTRICT',  # Include district names as text labels
                         title='Scatter Plot between Horses/Asses and Total Milk Production')
fig_scatter.update_layout(xaxis_title='Number of Horses/Asses', yaxis_title='Total Milk Produced')
fig_scatter.show()
```



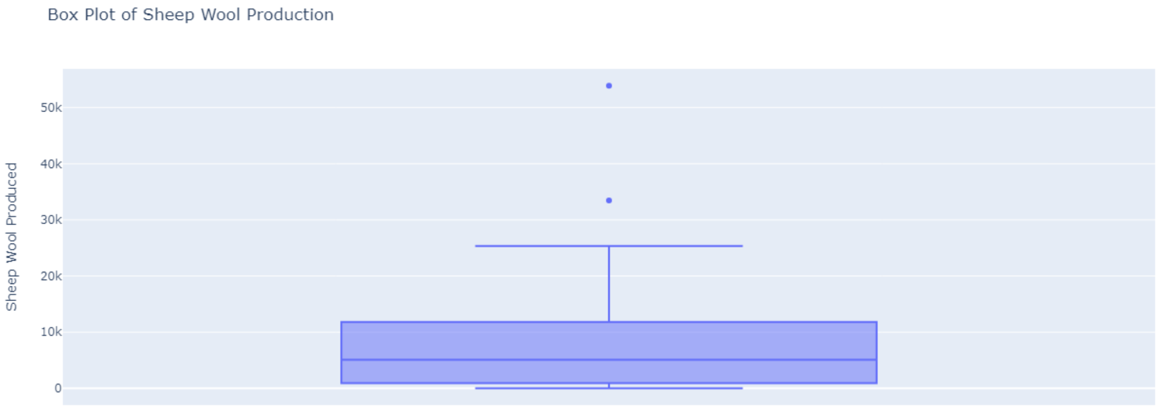
The relationship between the total amount of milk produced in each district and the number of horses or asses is represented visually in this scatter plot. Every point on the plot represents a district, and the y-axis shows the total amount of milk produced, while the x-axis shows the number of horses or asses. The plot facilitates comprehension of the possible trend or correlation between these two variables in various districts. This scatter plot can be used to analyze the relationship between the total amount of milk produced in different districts and the presence of horses or asses.

Box Plot

Box Plot of Sheep Wool Production



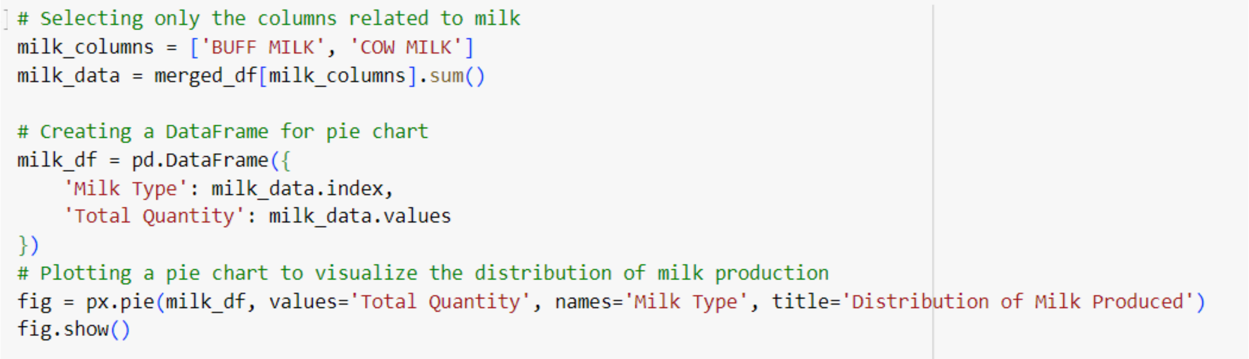
```
# Box plot for the distribution of sheep wool production
fig_box_sheep_wool = px.box(merged_df, y='SHEEP WOOL PRODUCED', title='Box Plot of Sheep Wool Production')
fig_box_sheep_wool.update_layout(yaxis_title='Sheep Wool Produced')
fig_box_sheep_wool.show()
```



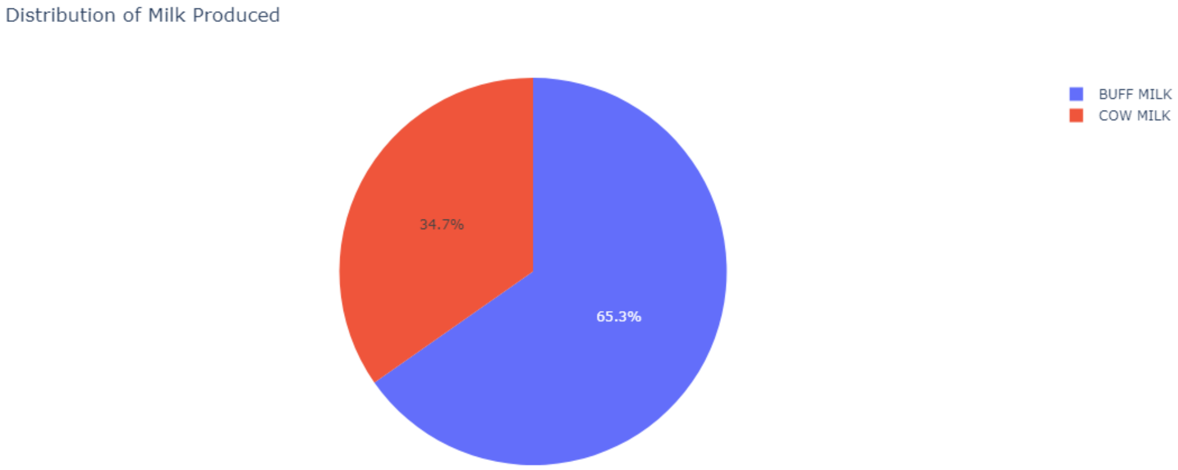
The distribution of sheep wool production among various districts is depicted in this box plot. Each box represents the y-axis that shows the amount of sheep wool produced. Each box's central line shows the median value, and the box itself shows the interquartile range (IQR. If there are any outliers, they are shown outside the whiskers. The unit of measurement for sheep wool produced is shown in the y-axis title.

Pie Chart

Distribution of Milk Produced

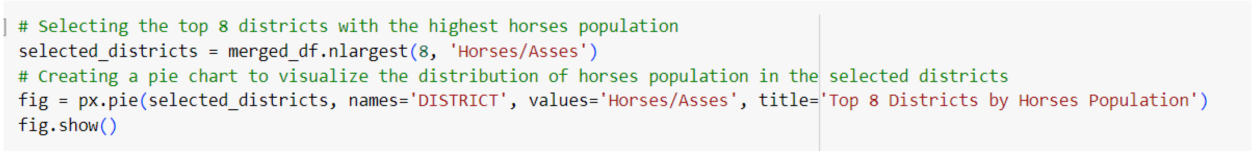


```
# Selecting only the columns related to milk
milk_columns = ['BUFF MILK', 'COW MILK']
milk_data = merged_df[milk_columns].sum()

# Creating a DataFrame for pie chart
milk_df = pd.DataFrame({
    'Milk Type': milk_data.index,
    'Total Quantity': milk_data.values
})
# Plotting a pie chart to visualize the distribution of milk production
fig = px.pie(milk_df, values='Total Quantity', names='Milk Type', title='Distribution of Milk Produced')
fig.show()
```

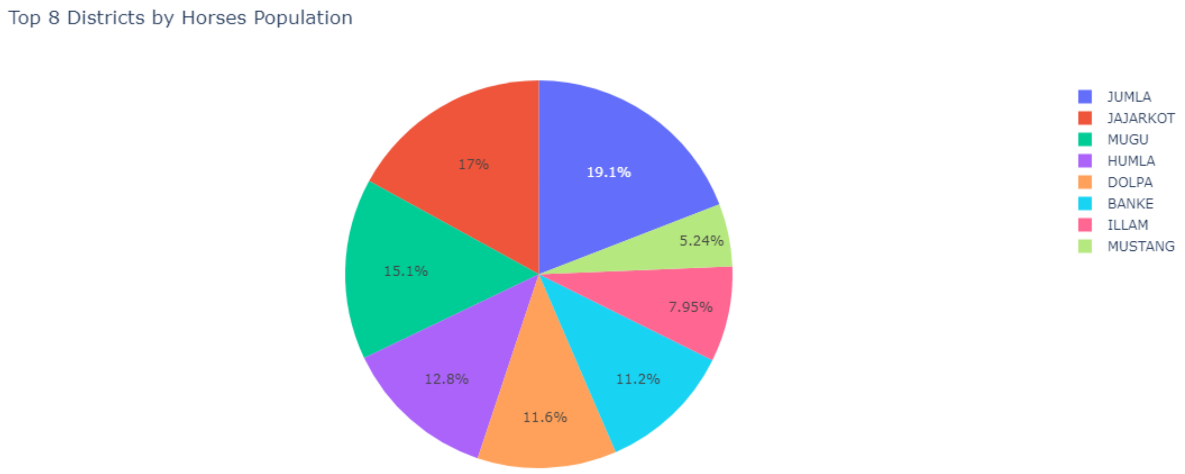


The distribution of milk production is shown graphically in this pie chart, which is divided into two categories: cow milk (shown in red) and buffalo milk (shown in blue). The percentage of each type of milk in total milk production is shown in the chart. Every slice is a different type of milk, and its size indicates what proportion of the total amount of milk was produced. The two types of milk are more visually appealing as a result of the color scheme.

Top 8 Districts by Horses Population



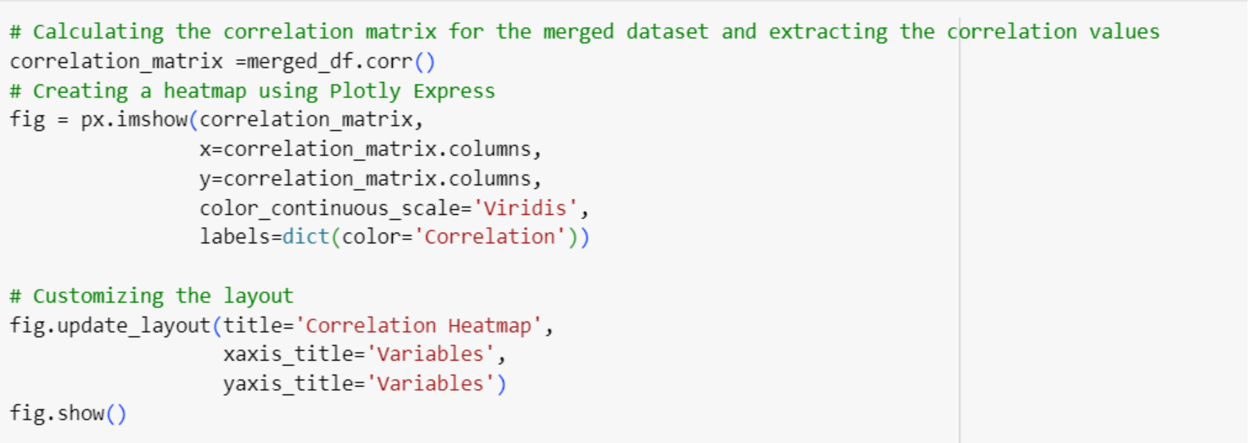
```
# Selecting the top 8 districts with the highest horses population
selected_districts = merged_df.nlargest(8, 'Horses/Asses')
# Creating a pie chart to visualize the distribution of horses population in the selected districts
fig = px.pie(selected_districts, names='DISTRICT', values='Horses/Asses', title='Top 8 Districts by Horses Population')
fig.show()
```



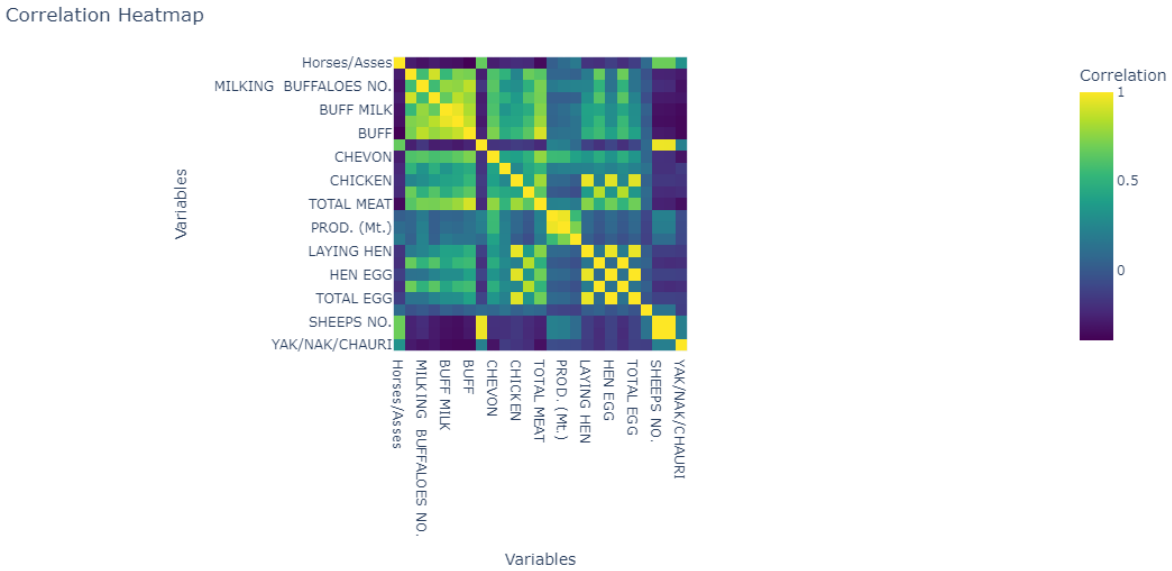
The population distribution of horses in the top 8 districts is shown in this pie chart. Every slice symbolizes a district, and its size reflects the percentage of all horses residing in that district. This chart helps us to determine which districts have the highest number of horses. The color labeling of the slices denotes the corresponding districts. In this case, using a pie chart highlights the relative numbers of horses in the various districts.

Correlation Matrix

In this analysis, the correlation matrix is used to determine the relationships between various variables in the combined dataset. Indicators of livestock and agricultural productivity can be arranged to reveal patterns and dependencies by using this numerical measure of the strength and direction of linear associations.



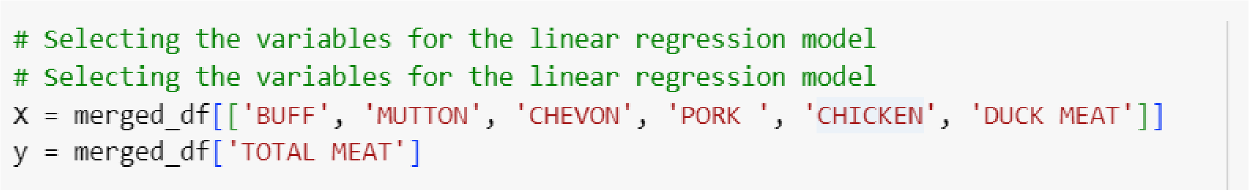
```
# Calculating the correlation matrix for the merged dataset and extracting the correlation values
correlation_matrix =merged_df.corr()
# Creating a heatmap using Plotly Express
fig = px.imshow(correlation_matrix,
                x=correlation_matrix.columns,
                y=correlation_matrix.columns,
                color_continuous_scale='Viridis',
                labels=dict(color='Correlation'))

# Customizing the layout
fig.update_layout(title='Correlation Heatmap',
                  xaxis_title='Variables',
                  yaxis_title='Variables')
fig.show()
```



The correlation matrix of the combined dataset is represented visually in this heatmap, highlighting the connections between various variables. Warmer colors indicate positive correlations and cooler colors indicate negative correlations.

Model Training

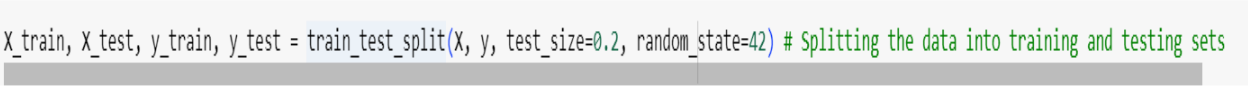
Linear Regression



```
# Selecting the variables for the linear regression model
# Selecting the variables for the linear regression model
X = merged_df[['BUFF', 'MUTTON', 'CHEVON', 'PORK ', 'CHICKEN', 'DUCK MEAT']]
y = merged_df['TOTAL MEAT']
```

In this section, we develop a linear regression model to predict 'TOTAL MEAT' production from a selection of variables. We select quantities of buffalo meat ('BUFF'), mutton, chevon, pork, chicken and duck meat, as independent variables (X) that will capture the effect of different meat sources on total meat production as a function of these variables, 'TOTAL MEAT' is our dependent variable (y).

Data Splitting for Model Training

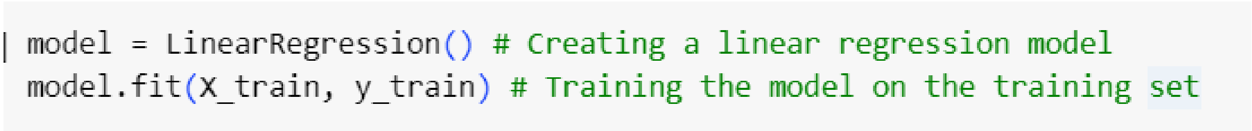


```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Splitting the data into training and testing sets
```

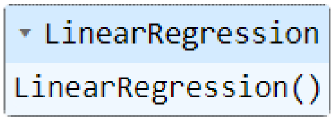
To evaluate the performance of the linear regression model, the dataset is split into training and testing sets. This can be done using the train_test_split function from scikit-learn.

Generally, the size of the training set is set to 80% of the data, and 20% is left for the testing set. It's common practice to set the random_state=42 because this allows us to train the model on one subset and test on another subset of the data, which gives information on how well our model might generalize to new, unseen data.

Linear Regression Model Training



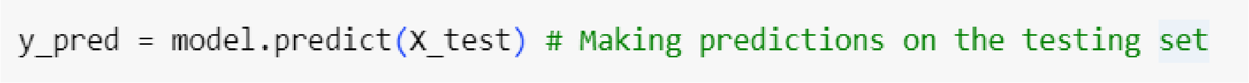
```
model = LinearRegression() # Creating a linear regression model
model.fit(X_train, y_train) # Training the model on the training set
```



```
▾ LinearRegression
LinearRegression()
```

To train our algorithm, we take two: X_train and y_train of train datasets, the X_train will have the independent dataset, so it will have many columns, so as to train the data of the X_train we use fit() method, and in training data, it needs to fit the two parameters of the train data, which are X_train and Y_train.

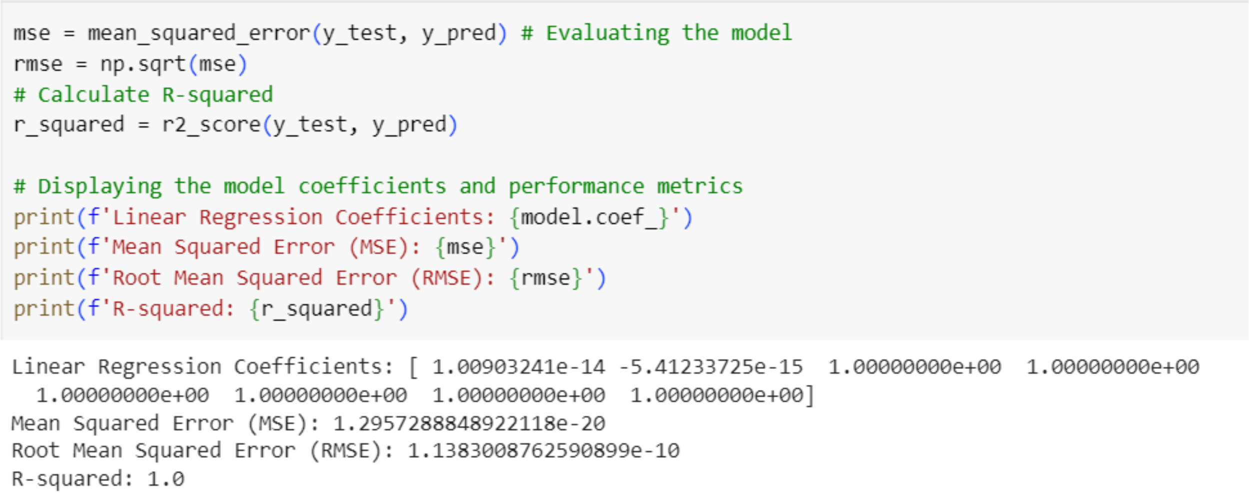
Making Predictions



```
y_pred = model.predict(X_test) # Making predictions on the testing set
```

Once the model has been trained, we can use it to generate predictions on a testing set (X_test) using the predict method.

Model Evaluation and Coefficients



```
mse = mean_squared_error(y_test, y_pred) # Evaluating the model
rmse = np.sqrt(mse)
# Calculate R-squared
r_squared = r2_score(y_test, y_pred)

# Displaying the model coefficients and performance metrics
print(f'Linear Regression Coefficients: {model.coef_}')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'R-squared: {r_squared}')
```

```
Linear Regression Coefficients: [ 1.00903241e-14 -5.41233725e-15  1.00000000e+00  1.00000000e+00
  1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00]
Mean Squared Error (MSE): 1.2957288848922118e-20
Root Mean Squared Error (RMSE): 1.1383008762590899e-10
R-squared: 1.0
```

We have evaluated the linear regression model using various metrics as:

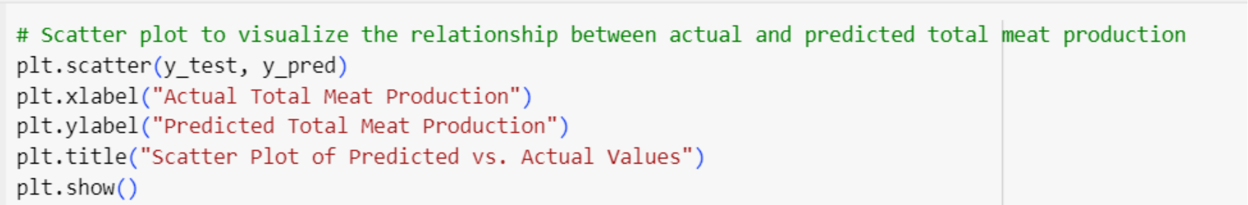
Model Coefficients: The linear regression model's coefficients show the weights given to each input variable. [1.00903241e-14, -5.41233725e-15, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00] are the coefficients in this instance MSE, or mean squared error.

Mean Squared Error (MSE): The average squared difference between the expected and actual values is measured by MSE. This model has an MSE of 1.2957288848922118e-20.

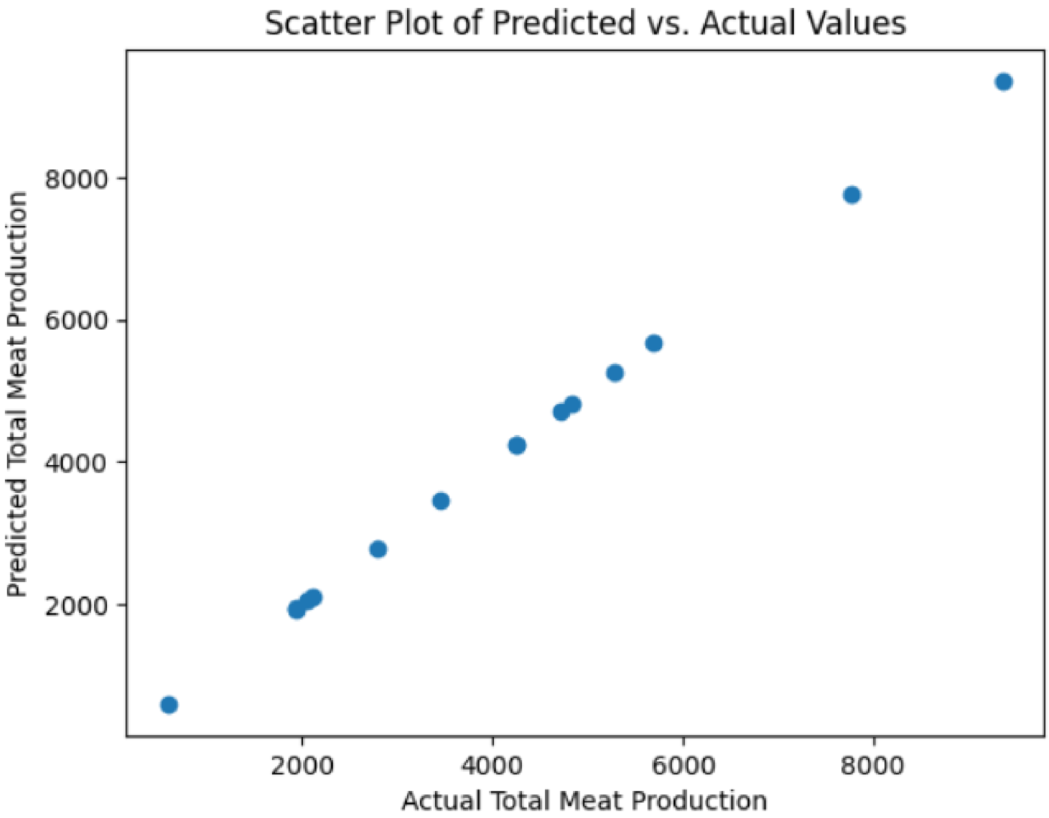
Root Mean Squared Error (RMSE): The average magnitude of errors is measured by RMSE, which is the square root of the MSE. The present instance has an RMSE of 1.1383008762590899e-10.

R-squared (R^2): Total meat production is the dependent variable, and R-squared shows how much of the variance in this variable is explained by the independent variables. A perfect fit is indicated by an R-squared of 1.0, and in this instance, the R-squared of 1.0 indicates that the model and the data are perfectly fitted.

Scatter Plot of Predicted vs. Actual Values



```
# Scatter plot to visualize the relationship between actual and predicted total meat production
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Total Meat Production")
plt.ylabel("Predicted Total Meat Production")
plt.title("Scatter Plot of Predicted vs. Actual Values")
plt.show()
```



The scatter plot visually represents the relationship between the actual total meat production values (y_test) and the predicted values by the linear regression model (y_pred). Each point on the plot represents an observation in the testing set. The x-axis represents the actual values and the y-axis represent the predicted values.

The plot shows that the predicted values are well aligned the actual values, forming an almost perfect diagonal line. This further validates the high R-squared value obtained during model evaluation, indicating the model has done an excellent job of explaining the variability in total meat production. The tight clustering around the diagonal line also implies minimal prediction errors, thus providing further evidence that the linear regression model works very effectively.

Conclusion

In summary, this paper provides a comprehensive overview of livestock and other commodities production in Nepal, with particular focus on the relative importance of factors such as geographic location, traditional practices, change over time, and the influence of these various factors on rural economies and food security. The datasets cover a variety of livestock and agricultural commodities, providing the ability to explore the production patterns of the various commodities.

The features of the separate datasets are revealed in the data exploration section, and the merged dataset is then ready for analysis. Visualization tools like pie charts, bar charts, and scatter plots provide informative depictions of different facets of the production of commodities and livestock.

In order to guarantee the consistency and quality of the dataset, the documentation also thoroughly examines the data cleaning and processing procedures. The combined dataset's statistical relationships and predictive power are demonstrated by the correlation matrix and linear regression model. This documentation provides a deeper understanding of Nepal's agricultural landscape through the presentation of visualizations and analyses.