TAYLOR'S UWE DUAL AWARDS PROGRAMMES
JANUARY 2024 SEMESTER

Data Analytics and Machine Learning

(ITS69304)

Module Head – Mr. Anmol Adhikari

Assignment -Individual

DUE DATE: 17th Feb 2024 via MyTIMeS (11:59pm) - Nepali Time

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	Aayush Ghising	0355747	2024/02/17		

Table of Contents

1. Introduction

1.1. Background Study

Livestock and commodities production in Nepal is the pillars of this country which contributes significantly in the economy and agricultural sector. With a dataset containing 75 districts, covering livestock populations and commodity productions metrics, this study begins on a comprehensive analysis to explain the dynamics of these sectors. Because of the Nepal's diverse topography and climate, it creates suitable environment for a variety of livestock and agricultural commodities. However, despite the crucial role of Nepal's diversity, a challenge such as limited access to modern farming technique and vulnerability to natural disasters continues to exist. With all of these challenges, there have been seen a surge in efforts aimed at fostering sustainable agricultural practices and enhancing the productivity of livestock and commodities in recent years.

1.2. Problem Statement

Even though livestock and commodities contribute significantly, Nepal faces with a challenge that ranges from limited access to modern farming techniques to vulnerability to natural disasters. This brings the need for a deeper understanding of the factors that influences the productions across every district. Moreover, there is also a necessity for identifying strategies that addresses the existing challenges and capitalizes the opportunities for growth and sustainability within these sectors.

1.3. Problem Solution

For solving the challenges faced in livestock and commodities production following approach is required that are:

- Access to modern farming techniques and technologies should be provided through collaborative efforts.
- Improved infrastructure resilience to mitigate the impact of natural disasters is required.
- Data driven insights from data analysis and visualization of collected dataset for building strategies are required.

1.4. Aims & Objectives

- To analyze the distribution and trends of livestock populations and commodity production across the 75 districts of Nepal.
- To identify the key factors that influences livestock and commodities production, including geographical regions and available resources.
- To assess the impact of traditional agricultural practices, infrastructure, and vulnerability to natural disasters on production dynamics.
- To propose strategies for enhancing the productivity, resilience, and sustainability of livestock and commodities production in Nepal.
- To contribute to the body of knowledge regarding the role of livestock and commodities production in Nepal's agricultural economy and food security.

1.5. Dataset Overview

For the livestock and commodities production in Nepal there are total of 7 dataset that has different data. They are given below:

- **Horse/Asses Population in Nepal Dataset:**
  - This dataset provides information on the population of horses and asses (donkeys) in different districts of Nepal.
  - **Columns:**
    - **DISTRICT**: Provides the name of the district in Nepal.
    - **Horses/Asses**: Provides population count of horses and asses in each district.
- **Milking Animal and Milk Production in Nepal Dataset:**
  - This dataset presents data on milking cows, milking buffaloes, milk production from cows, milk production from buffaloes, and total milk production in various districts of Nepal.
  - **Columns:**
    - **DISTRICT**: Provides name of the district in Nepal.
    - **MILKING COWS NO.**: Provides the number of milking cows in the district.
    - **MILKING BUFFALOES NO.**: Number of milking buffaloes in the district.
    - **COW MILK**: Quantity of milk produced by cows in kilograms.

        - **BUFF MILK**: Quantity of milk produced by buffaloes in kilograms.
        - **TOTAL MILK PRODUCED**: Total milk production in kilograms.

- **Meat Production in Nepal Dataset:**
    - This dataset provides information on meat production in different districts of Nepal, including buffalo meat, mutton, chevon (goat meat), pork, chicken, and duck meat.
    - **Columns:**
        - **DISTRICT**: Gives insight on name of the district in Nepal.
        - **BUFF, MUTTON, CHEVON, PORK, CHICKEN, and DUCK MEAT**: Provides the quantities of meat produced from each respective animal in kilograms.
        - **TOTAL MEAT**: Gives the total meat production in kilograms of each district.
- **Cotton Production in Nepal Dataset:**
    - This dataset contains information about cotton production in Nepal, including the area cultivated, production in metric tons, and yield per hectare.
    - **Columns:**
        - **DISTRICT**: Name of the district in Nepal.
        - **AREA (Ha.)**: Area of land (in hectares) used for cotton cultivation.
        - **PROD. (Mt.)**: Production of cotton in metric tons.
        - **YIELD Kg/Ha**: Yield of cotton per hectare in kilograms.
- **Egg Production in Nepal Dataset:**
    - This dataset provides details on egg production in various districts of Nepal, including the number of laying hens and ducks, hen egg production, duck egg production, and total egg production.
    - **Columns:**
        - **DISTRICT**: Name of the district in Nepal.
        - **LAYING HEN, LAYING DUCK**: Number of laying hens and ducks.
        - **HEN EGG, DUCK EGG**: Quantities of eggs produced by hens and ducks.
        - **TOTAL EGG**: Total egg production.

- **Rabbit Population in Nepal Dataset:**
  - This dataset gives information on the population of rabbits in different districts of Nepal.
  - **Columns:**
    - **DISTRICT**: Name of the district in Nepal.
    - **Rabbit**: Population count of rabbits.
- **Wool Production in Nepal Dataset:**
  - This dataset provides data on sheep population and wool production in various districts of Nepal.
  - **Columns:**
    - **DISTRICT**: Name of the district in Nepal.
    - **SHEEPS NO.**: Number of sheep in the district.
    - **SHEEP WOOL PRODUCED**: Quantity of wool produced by sheep.
- **Yak/Nak/Chauri Population in Nepal:**
  - This dataset provides data on yak, nak, chauri population in various districts of Nepal.
  - **Columns:**
    - **DISTRICT:** Provides name of districts of Nepal.
    - **YAK/NAK/CHAURI:** Gives population of yaks in various districts of Nepal.

These datasets offer a comprehensive overview of various aspects of agriculture and animal husbandry in Nepal, including livestock populations, milk and meat production, cotton and wool production, egg production, and rabbit populations across different regions of the Nepal.

2. Importing Necessary Libraries & Dataset

Before starting to manipulate and analyze the dataset "Livestock and Commodities Production in Nepal", I first imported all of the libraries and dataset that would be needed for my model. Each imported libraries and the dataset for my model is described below:

- **Pandas:** It is one of the mostly used python library that is used for manipulation and analysis of the data. It provides data structures like DataFrame for organizing and working with tabular data.
- **Numpy:** It is a package that is imported for numerical computing like: creating arrays, counting null values, etc.
- **Plotly Express:** It is a high level data visualization library for creating interactive plots and charts such as histogram, scatter plot, box plot, etc
- **Train_test_split:** It is a function of Scikit-learn that helps split the dataset into training and testing sets according to out choice like: 80% for training and 20% for testing.
- **LinearRegression:** It is a supervised model that is used to find relationship between dependent and independent variable by fitting a linear equation to observed data.
- **Mean Squared Error:** It is one of the evaluating metric for regression model that measures the average of squared of the errors, i.e. the differences between actual and predicted values.
- **Mean Absolute Error:** It is a metric for evaluating regression models that measures average absolute difference between the predicted values and the actual values.
- **R2_score:** It is also one the evaluating metric used for regression algorithm which measures how well the independent variables explain the variability of the dependent variable.
- **Loading Dataset:** After importing all of the libraries, I loaded the every 8 dataset of livestock and commodities production in Nepal into variable 'df1', df2', 'df3', 'df4', 'df5', 'df6', 'df7', 'df8' that was provided to us using the code "pd.read_csv()" function.

Importing Necessary Libraries

```
] import pandas as pd
  import numpy as np
  import plotly.express as px
  from sklearn.model_selection import train_test_split
  from sklearn.linear_model import LinearRegression
  from sklearn.metrics import mean_squared_error, r2_score
```

```
df1 = pd.read_csv('/content/drive/MyDrive/Data Analytics/Livestock_Nepal/Part 2/horseasses-population-in-nepal-by-district.csv')
df2 = pd.read_csv('/content/drive/MyDrive/Data Analytics/Livestock_Nepal/Part 2/milk-animals-and-milk-production-in-nepal-by-district.csv')
df3 = pd.read_csv('/content/drive/MyDrive/Data Analytics/Livestock_Nepal/Part 2/net-meat-production-in-nepal-by-district.csv')
df4 = pd.read_csv('/content/drive/MyDrive/Data Analytics/Livestock_Nepal/Part 2/production-of-cotton-in-nepal-by-district.csv')
df5 = pd.read_csv('/content/drive/MyDrive/Data Analytics/Livestock_Nepal/Part 2/production-of-egg-in-nepal-by-district.csv')
df6 = pd.read_csv('/content/drive/MyDrive/Data Analytics/Livestock_Nepal/Part 2/rabbit-population-in-nepal-by-district.csv')
df7 = pd.read_csv('/content/drive/MyDrive/Data Analytics/Livestock_Nepal/Part 2/wool-production-in-nepal-by-district.csv')
df8 = pd.read_csv('/content/drive/MyDrive/Data Analytics/Livestock_Nepal/Part 2/yak-nak-chauri-population-in-nepal-by-district.csv')
```

3. Data Exploration

3.1. Merging Different Dataset:

After importing all 8 different dataset of livestock and commodities production in Nepal, I moved towards merging those dataset into a single dataset for getting insight from all dataset using only one dataset and make my work easier to work with it.

For merging, first I kept all the dataset into one list ‘dfs’ and then from what I observed from all the dataset there was DISTRICT column that was common in all but in some of the dataset DISTRICT column values were in lowercase too. So, to not have two rows for the same district while merging the dataset, I used ‘for’ loop and ‘.str.upper() function in all datasets DISTRICT column value.

After converting all values of DISTRICT column to uppercase in every dataset, I then used “.merge()” function putting “DISTRICT” as same column in every dataset and “outer” to join the dataset.

```
[165] pd.set_option('display.max_rows', None)
      # List of DataFrame objects
      dfs = [df1, df2, df3, df4, df5, df6, df7, df8]

      # Convert 'DISTRICT' column to uppercase for all dataframes
      for df in dfs:
          df['DISTRICT'] = df['DISTRICT'].str.upper()

      # Merge dataframes based on the 'DISTRICT' column
      merged_df = df1.merge(df2, on='DISTRICT', how='outer') \
                     .merge(df3, on='DISTRICT', how='outer') \
                     .merge(df4, on='DISTRICT', how='outer') \
                     .merge(df5, on='DISTRICT', how='outer') \
                     .merge(df6, on='DISTRICT', how='outer') \
                     .merge(df7, on='DISTRICT', how='outer') \
                     .merge(df8, on='DISTRICT', how='outer')
```

3.2. Finding Basic Information on Merged Dataset:

For finding information on the dataset that I just merged, first I used function ‘.info()’ which gave me information about the different columns of the merged dataset, data types of each columns, null value count of the columns, memory usage by that dataframe, etc

```
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 105 entries, 0 to 104
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   DISTRICT               105 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    95 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: 22.1+ KB
```

To find the descriptive statistics of the numerical value of data frame like: count, mean, standard deviation (std), minimum, Q1, Q2, Q3, and maximum, I used the function '.describe()'.

```
merged_df.describe()
```

	Horses/Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	PORK
count	60.000000	9.600000e+01	9.600000e+01	96.000000	9.600000e+01	95.000000	96.00000	96.000000	96.000000	96.000000
mean	2790.400000	4.275562e+04	5.647433e+04	26825.260417	5.043505e+04	58555.189474	7291.87500	111.833333	2732.625000	979.541667
std	8447.864779	1.144496e+05	1.508551e+05	71948.998086	1.358044e+05	96696.838132	19484.37418	314.001598	7245.635676	2713.977477
min	12.000000	4.520000e+02	0.000000e+00	259.000000	0.000000e+00	259.000000	0.00000	0.000000	56.000000	1.000000
25%	122.250000	8.074750e+03	1.020550e+04	4630.750000	9.085000e+03	14179.500000	1438.50000	10.000000	575.000000	114.000000
50%	493.000000	1.513050e+04	1.954000e+04	8343.500000	1.710250e+04	28027.000000	2558.00000	31.000000	890.000000	326.000000
75%	1510.250000	2.600800e+04	3.674975e+04	15694.000000	3.110500e+04	43324.000000	4447.00000	90.750000	1689.500000	711.250000
max	55808.000000	1.026135e+06	1.355384e+06	643806.000000	1.210441e+06	536299.000000	175005.00000	2684.000000	65583.000000	23509.000000

8 rows × 25 columns

Now to find the basic overview of a merged dataset, I used the function '.head()' and '.tail()' which gave me the first five column from top and bottom of data frame respectively.

```
merged_df.head()
```

	DISTRICT	Horses/Asses	MILKING COWS NO.	MILKING BUFFALOES NO.	COW MILK	BUFF MILK	TOTAL MILK PRODUCED	BUFF	MUTTON	CHEVON	...	YI Kg/
0	TAPLEJUNG	543.0	8123.0	4987.0	5389.0	4257.0	9645.0	607.0	31.0	491.0	...	N
1	SANKHUWASHAVA	358.0	15342.0	13367.0	6988.0	10589.0	17577.0	NaN	NaN	NaN	...	N
2	SOLUKHUMBU	1775.0	7819.0	13501.0	2948.0	5493.0	8441.0	1123.0	28.0	416.0	...	N
3	PANCHTHAR	15.0	14854.0	11331.0	8511.0	9835.0	18346.0	1496.0	4.0	940.0	...	N
4	ILLAM	2815.0	26821.0	5759.0	19735.0	15261.0	34996.0	1974.0	1.0	870.0	...	N

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	...	Y K
100	FW.TERAI	NaN	47922.0	68915.0	51051.0	62553.0	113604.0	9778.0	98.0	2330.0	...	
101	FW. REGION	NaN	130595.0	132257.0	87936.0	112438.0	200374.0	NaN	NaN	NaN	...	
102	NEPAL	NaN	1026135.0	1355384.0	643806.0	1210441.0	NaN	175005.0	2684.0	65583.0	...	8
103	SANKHUWASABHA	NaN	NaN	NaN	NaN	NaN	NaN	1646.0	41.0	958.0	...	
104	RAMECHHAP	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	

5 rows × 26 columns

Finally, in data exploration part I used functions '.duplicated()', and '.isnull().sum()' to find the duplicated rows and count of null values in each column of the data frame 'merged_df'.

```
duplicate_rows = merged_df.duplicated()

# Printing rows marked as duplicates if any
print("Duplicate rows:")
print(merged_df[duplicate_rows])
```

```
Duplicate rows:
Empty DataFrame
Columns: [DISTRICT, Horses/Asses, MILKING  COWS NO., MILKING  BUFFALOES NO., CC
Index: []

[0 rows x 26 columns]
```

```
null_values = merged_df.isnull().sum() # Checking null values if there is in a dataframe

print("No of missing values in each columns:")
print(null_values)
```

```
No of missing values in each columns:
DISTRICT                    0
Horses/Asses               45
MILKING  COWS NO.           9
MILKING  BUFFALOES NO.      9
COW MILK                    9
BUFF MILK                   9
TOTAL MILK PRODUCED        10
BUFF                        9
MUTTON                      9
CHEVON                      9
PORK                        9
CHICKEN                     9
DUCK MEAT                   9
TOTAL MEAT                  9
AREA (Ha.)                101
PROD. (Mt.)               101
YIELD Kg/Ha               101
LAYING HEN                  9
LAYING DUCK                 9
HEN EGG                     9
DUCK EGG                    9
TOTAL EGG                   9
Rabbit                     50
SHEEPS NO.                  9
SHEEP WOOL PRODUCED         9
YAK/NAK/CHAURI             70
dtype: int64
```

4. Data Preprocessing

4.1. Handling Missing Values:

From the above data exploration part, I had found that there were null values in every column except 'DISTRICT' columns. So for handling it, I decided to fill those null values with '0' as there were null values in almost every row of the dataset and if I were to drop every row that had null value there probably would only be 2, or 3 rows. To fill null values with 0, I used the function '.fillna'.

```
[173] merged_df.fillna(0, inplace=True)
```

4.2. Assigning Appropriate Data Type:

I had seen that every column had the data type float while exploring the data frame except DISTRICT column. From what I explored that, it would be appropriate to assign the data type 'int64' to some columns such as 'Horses/Asses' column, this column represents population of horse and a horse population can never be 1.5 or in any point numbering so it is common practice to assign it data type 'int64' and likewise to other column similar to it.

To assign the columns integer data type, I first listed the columns in variable 'columns_to_conver' and then used function '.astype(int)' to convert them into integer data type.

```
# Columns to convert to integer
columns_to_convert = ['Horses/Asses', 'MILKING  COWS NO.', 'MILKING  BUFFALOES NO.', 'LAYING HEN',
                      'LAYING DUCK', 'HEN EGG', 'DUCK EGG', 'TOTAL EGG', 'Rabbit', 'SHEEPS NO.', 'YAK/NAK/CHAURI']

# Convert selected columns to integer
merged_df[columns_to_convert] = merged_df[columns_to_convert].astype(int)
```

```
[200] merged_df.dtypes
```

```
DISTRICT                 object
HORSES/ASSES              int64
MILKING  COWS NO.         int64
MILKING  BUFFALOES NO.    int64
COW MILK                float64
BUFF MILK               float64
TOTAL MILK PRODUCED     float64
BUFF                    float64
MUTTON                  float64
CHEVON                  float64
PORK                    float64
CHICKEN                 float64
DUCK MEAT               float64
TOTAL MEAT              float64
AREA (Ha.)              float64
PROD. (Mt.)             float64
YIELD Kg/Ha             float64
LAYING HEN                int64
LAYING DUCK               int64
HEN EGG                   int64
DUCK EGG                  int64
TOTAL EGG                 int64
RABBIT                    int64
SHEEPS NO.                int64
SHEEP WOOL PRODUCED     float64
YAK/NAK/CHAURI            int64
dtype: object
```

4.3. Checking unique value of DISTRICT column:

After assigning the appropriate data type to each column, I then moved towards checking the unique value in DISTRICT column. To find unique value I used the function '.unique()' which showed me unique value of district column and there I also observed values like 'NEPAL', 'TOTAL', 'E.REGION', 'C.REGION', 'E.MOUNTAIN', etc which were not related to the DISTRICT column as this column represents 75 district of Nepal.

```
[201] #checking unique value of DISTRICT column of dataframe
      merged_df["DISTRICT"].unique()
```

```
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', 'E.MOUNTAIN', 'TERHATHUM',
       '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', 'NEPAL', 'SANKHUWASABHA',
       'RAMECHHAP'], dtype=object)
```

4.4. Filtering Values in DISTRICT column:

While checking the unique value in 'DISTRICT' column I found some value that were not related or shouldn't be in the 'DISTRICT' column. So, for filtering out the values that are not required in the column "DISTRICT" I used the code "filtered_df = merged_df[~merged_df['DISTRICT'].str.contains('\.|NEPAL|TOTAL')]" which filtered out the values that contained values like '.', 'NEPAL', or 'TOTAL' and then the filtered data is stored in variable 'filtered_df'

```
filtered_df = merged_df[~merged_df['DISTRICT'].str.contains('\.|NEPAL|TOTAL')]
filtered_df['DISTRICT'].unique()
```

```
array(['TAPLEJUNG', 'SANKHUWASHAVA', 'SOLUKHUMBU', 'PANCHTHAR', 'ILLAM',
       'TERATHUM', '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', 'TERHATHUM', 'DHANKUTA', 'SAPTARI',
       'SIRAHA', 'DOLAKHA', 'SINDHUPALCHOK', 'RASUWA', 'RAMECHAP',
       'SINDHULI', 'KAVRE', 'BHAKTAPUR', 'LALITPUR', 'KATHMANDU',
       'DHADING', 'MAKWANPUR', 'DHANUSHA', 'MAHOTTARI', 'SARLAHI',
       'PARSA', 'PALPA', 'KAILALI', 'KANCHANPUR', 'SANKHUWASABHA',
       'RAMECHHAP'], dtype=object)
```

After filtering values that contains '.', 'NEPAL', or 'TOTAL', there again more values that had to be filtered or merged into one row like: "there were values like 'RAMECHAP' and 'RAMECHHAP' which meant the same district but were in different rows. So for combining those types of rows into one row and also put the correct value from both rows, first I made a mapping of those districts that were misspelled into variable 'district_mapping' and then I replaced the districts that were misspelled with correct spelling which can be seen in below snapshot.

```
district_mapping = {
    'TERHATHUM': 'TERATHUM',
    'SANKHUWASHAVA': 'SANKHUWASABHA',
    'RAMECHAP': 'RAMECHHAP'
}

# Replace values in DISTRICT column according to the mapping
filtered_df['DISTRICT'] = filtered_df['DISTRICT'].replace(district_mapping)
```

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

```
array(['TAPLEJUNG', 'SANKHUWASABHA', 'SOLUKHUMBU', 'PANCHTHAR', 'ILLAM',
       'TERATHUM', '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', 'RAMECHHAP', 'SINDHULI',
       'KAVRE', 'BHAKTAPUR', 'LALITPUR', 'KATHMANDU', 'DHADING',
       'MAKWANPUR', 'DHANUSHA', 'MAHOTTARI', 'SARLAHI', 'PARSA', 'PALPA',
       'KAILALI', 'KANCHANPUR'], dtype=object)
```

Now that the district spelling that were wrong has been corrected, I grouped the rows of dataframe ‘filtered_df’ by the ‘DISTRICT’ column and then summed up the values of other columns for each group and finally reset the index of ‘filtered_df’ dataframe which also sorted my dataframe alphabetically based on ‘DISTRICT’ column. Finally my dataframe ‘filtered_df’ only contains 75 district and has been cleaned.

```
# Merging rows based on the DISTRICT column
filtered_df = filtered_df.groupby('DISTRICT').sum().reset_index()
```

```
[370] filtered_df
```

	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
0	ACHHAM	95	5796	10381	3321.0	9010.0	12331.0	1329.0	10.0	710.0	...	0.0	12096	143	1905	9
1	ARGHAKHANCHI	17	6219	27698	3805.0	25232.0	29037.0	3246.0	2.0	638.0	...	0.0	77924	118	7289	7
2	BAGLUNG	1250	8950	22929	5128.0	18093.0	23221.0	2124.0	19.0	578.0	...	0.0	57523	1370	2199	104
3	BAITADI	484	9845	12699	4641.0	10184.0	14825.0	1727.0	1.0	730.0	...	0.0	3509	107	594	6
4	BAJHANG	724	15936	9679	4600.0	4149.0	8749.0	1208.0	89.0	572.0	...	0.0	8917	188	985	14
5	BAJURA	1262	12019	5534	4887.0	4801.0	9688.0	708.0	66.0	451.0	...	0.0	9844	198	852	15
6	BANKE	3963	14060	36201	8956.0	31062.0	40018.0	3256.0	42.0	1652.0	...	1519.0	194508	858	13063	65
7	BARA	305	18771	39650	11952.0	22738.0	34690.0	4076.0	1.0	1205.0	...	0.0	242429	8244	9955	627
8	BARDIYA	559	15932	27931	10792.0	27784.0	38576.0	3405.0	35.0	1758.0	...	1200.0	123536	1214	15457	92
9	BHAKTAPUR	0	3402	2164	3402.0	4494.0	7896.0	1013.0	9.0	175.0	...	0.0	385908	2722	40781	214
10	BHOJPUR	168	14103	16342	7324.0	14184.0	21508.0	1251.0	51.0	313.0	...	0.0	53957	1136	4037	88
11	CHITWAN	15	14934	18166	14947.0	30230.0	45177.0	5550.0	13.0	1769.0	...	0.0	2751238	2006	411901	274
12	DADELDHURA	241	13963	6108	7045.0	5301.0	12346.0	1011.0	1.0	823.0	...	0.0	10131	205	1596	17
13	DAILEKH	154	9438	24351	5408.0	13942.0	19350.0	2145.0	50.0	578.0	...	0.0	43020	313	6776	26

5. Data Visualization

5.1. Scatter Plot of Relation Between Cow Milk & Milking Cow No.:

For the first visualization, I wanted to see that how was the relation between cow milk and milling cow, so for that I used scatter plot to plot every district cow milk production and number of milking cow and also used color as 'DISTRICT' column to distinct every district plot which can be seen in below figure.

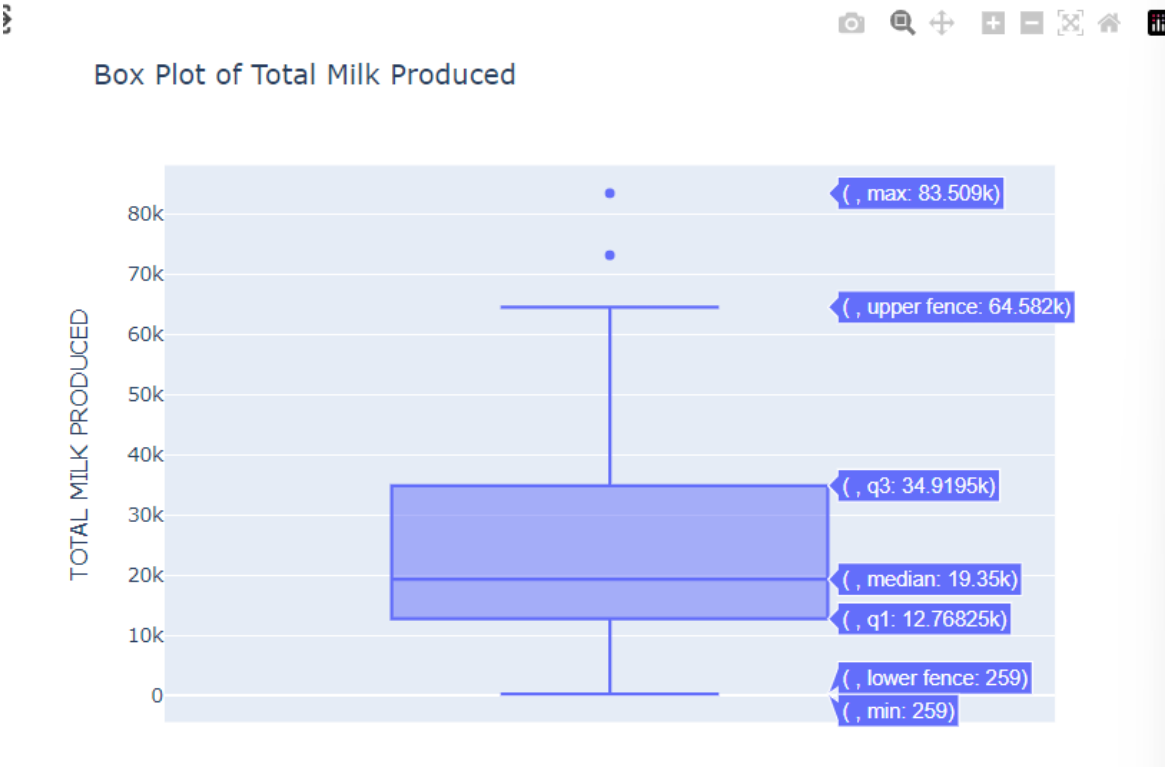
```
fig = px.scatter(filtered_df, x='MILKING  COWS NO.', y='COW MILK', color = 'DISTRICT', trendline='ols', title = "Relation Between Cow Milk and Milking Cow No."
fig.show()
```

So, from the above scatter plot we can see that the relation between 'COW MILK' and 'MILKING COWS NO.' are in linear relation as we can see that as the milking cow number goes on increasing the cow milk also goes on increasing showing linear relationship.

5.2. Box Plot of Total Milk Produced

Another visualization that I wanted to see is how much is the max, min, quartile range and outlier of column 'TOTAL MILK PRODUCED'. For that I used box plot which easily showed me those values as we can see in the figure below:

```
# Box Plot
fig = px.box(filtered_df, y='TOTAL MILK PRODUCED', title='Box Plot of Total Milk Produced')
fig.show()
```

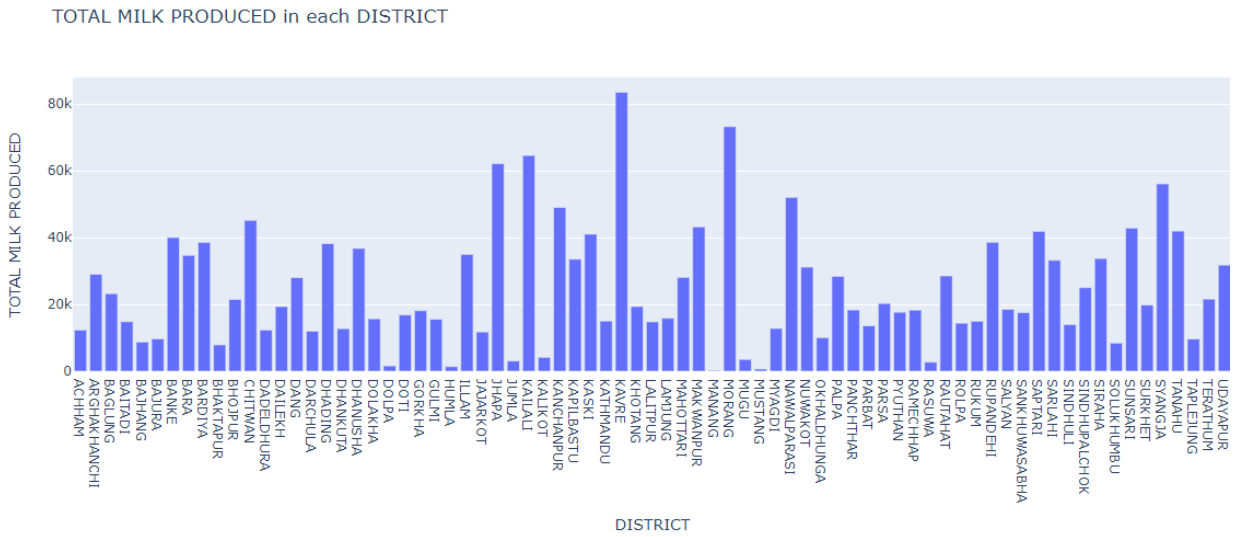


Form the above box plot I analyzed that the minimum production of milk in one of the district of Nepal is '259' liters, median is '19.35k', maximum is '83.509k'. It also shows us q1, q2, and upper fence value that are: '12.76825k', '34.9195k' and '64.582k' respectively. In above box there is also two outliers that means that the total milk produced in those district is very high or abnormally high.

5.3. Bar Graph of Total Milk Produced in each District:

After visualizing the min, max, median, etc of total milk produced in box plot, I wanted to visualize how much a district produces milk for which I used bar graph which can be seen in below figure.

```
# Histogram
fig = px.bar(filtered_df, x='DISTRICT', y='TOTAL MILK PRODUCED', title='TOTAL MILK PRODUCED in each DISTRICT')
fig.show()
```



So, in above bar graph I gained insight on every district total milk production with their name on it just by hovering on each bar of bar graph. From the figure we can see the tallest bar is of ‘DISTRICT’ ‘KAVRE’ in which the amount of ‘TOTAL MILK PRODUCED’ is ’83.509k’ and this represents the district that produces milk the highest amount. Likewise I can gain every district milk production data and compare them.

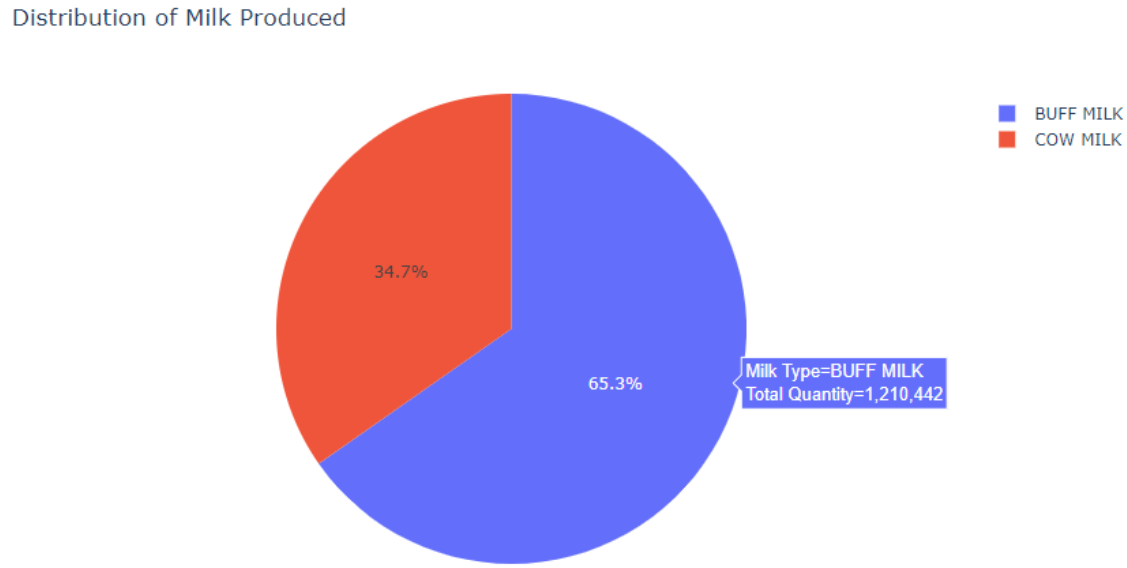
5.4. Pie Chart for Distribution of Milk Produced:

Here in this visualization, as there were columns that had ‘BUFF MILK’ & ‘COW MILK’ in dataframe, I wanted to visualize that what percentage each milk covered. So for that I used pie chart that is show below:

```
# Selecting only the columns related to meat
milk_columns = ['BUFF MILK', 'COW MILK']
milk_data = filtered_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 the pie chart
fig = px.pie(milk_df, values='Total Quantity', names='Milk Type', title='Distribution of Milk Produced')
fig.show()
```

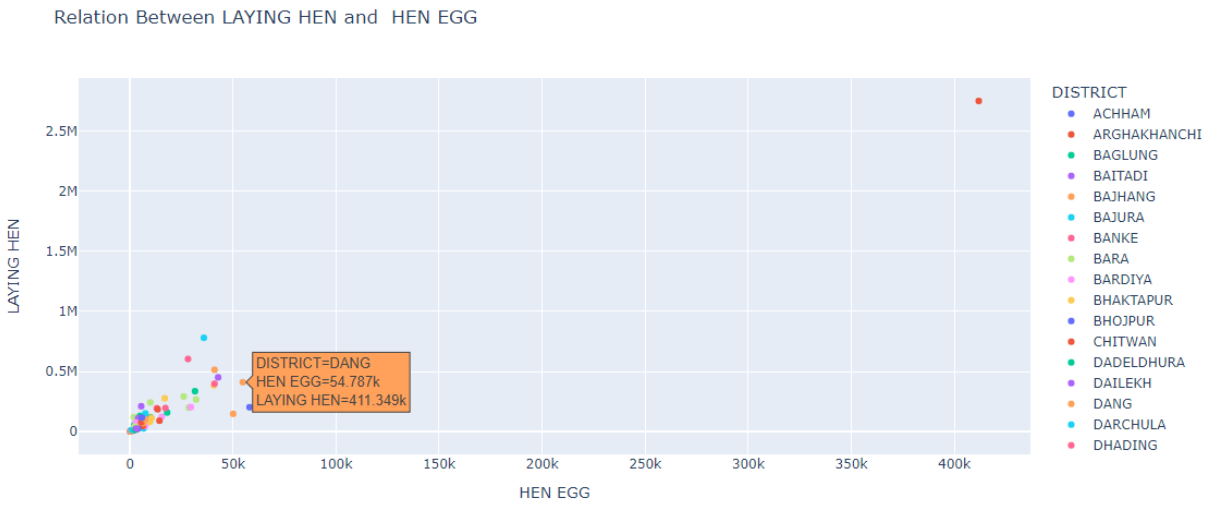


In above pie chart I gained insight about the distribution of buff milk and cow milk. As buff milk is represented by blue color and cow milk is represented by red color I gained insight that 65.3% of total milk produced in Nepal is covered by 'BUFF MILK' and 34.7% of total milk produced in Nepal is covered by 'COW MILK' giving me insight that in Nepal the production of BUFF MILK is greater than production of COW MILK.

5.5. Scatter Plot of Relation Between Laying Hen and Hen Egg:

Finally in this visualization, I wanted to see that how was the relation between laying hen and hen egg, so for that I used scatter plot to plot every district laying hen populaiton and number of hen egg produced and also used color as 'DISTRICT' column to distinct every district plot which can be seen in below figure.

```
fig = px.scatter(filtered_df, x='HEN EGG', y='LAYING HEN', color = 'DISTRICT', trendline='ols', title = "Relation Between LAYING HEN and  HEN EGG")
fig.show()
```



So, from the above scatter plot we can see that the relation between ‘LAYING HEN’ and ‘HEN EGG’ are in linear relation as we can see that as the number of hen egg goes on increasing the number of laying hen also goes on increasing showing linear relationship.

6. Correlation Matrix

A correlation matrix is a tabular representation of the correlation coefficient between variables in a dataset which shows how closely two variables or columns are related to each other that ranges from -1 to 1.

For the finding the correlation between each column of ‘filtered_df’ dataframe, I used the function ‘.corr()’ which helped be find the correlation. Now for the prediction I have decided to build a model that predicts ‘TOTAL MILK PRODUCED’ & ‘TOTAL EGG’. So, to find out which columns are the most correlated columns to both of the target variable column, I used the code ‘correlation_matrix['TOTAL MILK PRODUCED'].sort_values(ascending=False)’ which gave the correlation with ‘TOTAL MILK PRODUCED’ only in decreasing order and also the code ‘correlation_matrix[‘TOTAL EGG’].sort_values(ascending=False)’ which gave the correlation with ‘TOTAL EGG’ only in decreasing order.

```
correlation_matrix = filtered_df.corr()

total_milk_produced_correlation = correlation_matrix['TOTAL MILK PRODUCED'].sort_values(ascending=False)
# Printing the correlation values
print("Correlation with TOTAL MILK PRODUCED: ")
print(total_milk_produced_correlation)
```

```
Correlation with TOTAL MILK PRODUCED:
TOTAL MILK PRODUCED       1.000000
BUFF MILK                 0.956100
BUFF                      0.881438
COW MILK                  0.854584
TOTAL MEAT                0.790589
MILKING  BUFFALOES NO.    0.762105
MILKING  COWS NO.         0.729867
CHEVON                    0.648134
DUCK MEAT                 0.566292
DUCK EGG                  0.545841
LAYING DUCK               0.535832
CHICKEN                   0.424755
PORK                      0.419302
LAYING HEN                0.401847
TOTAL EGG                 0.291946
HEN EGG                   0.288688
YIELD Kg/Ha               0.139145
PROD. (Mt.)               0.082186
AREA (Ha.)                0.055353
RABBIT                    0.003312
MUTTON                   -0.292865
HORSES/ASSES             -0.333573
SHEEPS NO.               -0.338699
SHEEP WOOL PRODUCED      -0.345448
YAK/NAK/CHAURI           -0.365772
Name: TOTAL MILK PRODUCED, dtype: float64
<ipython-input-389-c07cf80c8e72>:1: FutureWarning:
```

```
total_egg_correlation = correlation_matrix['TOTAL EGG'].sort_values(ascending=False)
# Printing the correlation values
print("Correlation with TOTAL EGG: ")
print(total_egg_correlation)
```

```
Correlation with TOTAL EGG:
TOTAL EGG                 1.000000
HEN EGG                   0.999980
LAYING HEN                0.958335
CHICKEN                   0.938770
TOTAL MEAT                0.691281
BUFF                      0.402634
DUCK MEAT                 0.337007
CHEVON                    0.330610
TOTAL MILK PRODUCED       0.291946
BUFF MILK                 0.287442
COW MILK                  0.234761
PORK                      0.222309
MILKING  BUFFALOES NO.    0.138717
DUCK EGG                  0.138678
MILKING  COWS NO.         0.137634
LAYING DUCK               0.096142
AREA (Ha.)                0.085230
PROD. (Mt.)               0.073758
YIELD Kg/Ha               0.020822
RABBIT                   -0.025380
MUTTON                   -0.101925
SHEEPS NO.               -0.117284
SHEEP WOOL PRODUCED      -0.117358
YAK/NAK/CHAURI           -0.123451
HORSES/ASSES             -0.132193
Name: TOTAL EGG, dtype: float64
```

From the above correlation with TOTAL MILK PRODUCED, we can see that BUFF MILK is correlated with it mostly with the correlation value ‘0.956100’as it is the one that is close to 1 showing strong positive correlation and YAK/NAK/CHAURI is correlated with it the least with the correlation value ‘0.003312’ as it is the one that is close to 0 showing weak correlation. In case of correlation with TOTAL EGG, we can see that HEN EGG is correlated with it mostly with correlation value ‘0.999980’ as it is the one that is close to 1 showing strong positive correlation and YIELD Kg/Ha is correlated with it the least with the correlation value ‘0.020822’ as it is the one that is close to 0 showing weak correlation.

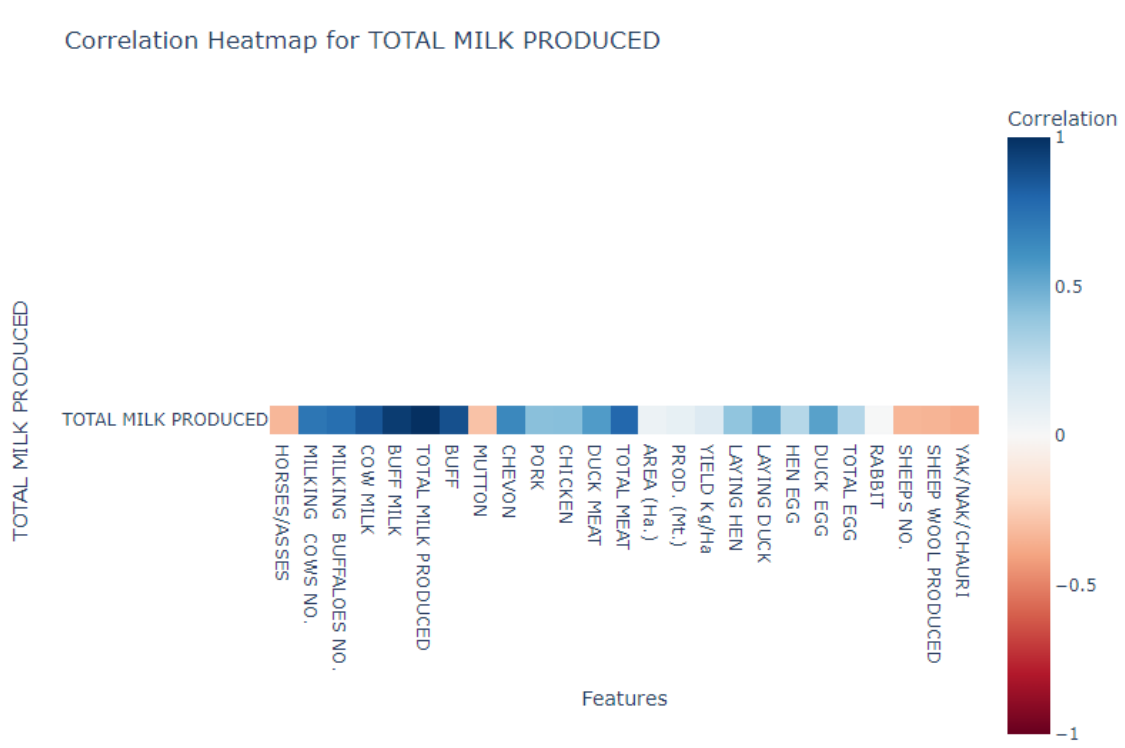
For more visualization of correlation with TOTAL MILK PRODUCED, and TOTAL EGG, I used heat map which showed correlation using the color range from red to blue in which red representing strong negative correlation, white representing weak correlation and blue representing strong positive correlation.

Heat map of Correlation with TOTAL MILK PRODUCED

```
fig = px.imshow(correlation_matrix['TOTAL MILK PRODUCED'].values.reshape(1, -1),
                labels=dict(color="Correlation"),
                x=correlation_matrix.columns,
                y=['TOTAL MILK PRODUCED'],
                color_continuous_scale='RdBu',
                color_continuous_midpoint=0)

# Update layout for better visualization
fig.update_layout(title='Correlation Heatmap for TOTAL MILK PRODUCED',
                  xaxis_title='Features',
                  yaxis_title='TOTAL MILK PRODUCED',
                  width=800,
                  height=600)

# Show the plot
fig.show()
```

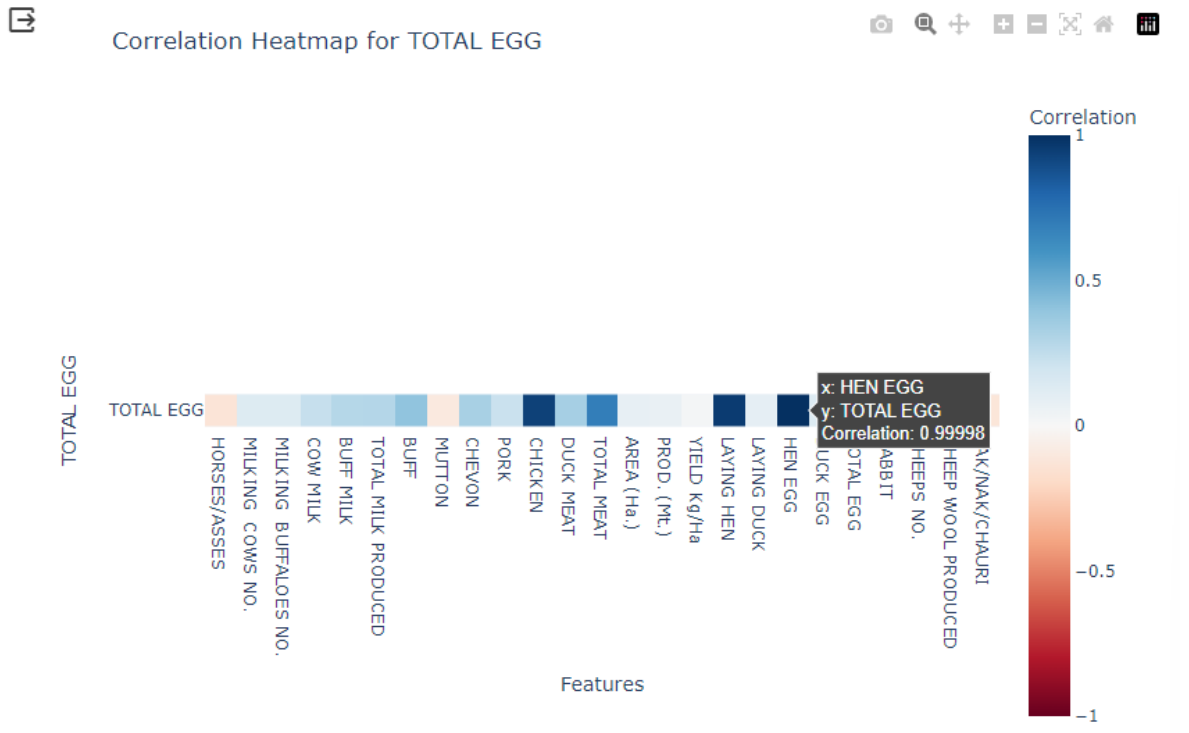


Heat map of Correlation with TOTAL EGG

```
fig = px.imshow(correlation_matrix['TOTAL EGG'].values.reshape(1, -1),
                labels=dict(color="Correlation"),
                x=correlation_matrix.columns,
                y=['TOTAL EGG'],
                color_continuous_scale='RdBu',
                color_continuous_midpoint=0)

# Update layout for better visualization
fig.update_layout(title='Correlation Heatmap for TOTAL EGG',
                  xaxis_title='Features',
                  yaxis_title='TOTAL EGG',
                  width=800,
                  height=600)

# Show the plot
fig.show()
```



7. Data Modeling & Development

7.1. 'TOTAL MILK PRODUCED' Predicting Model:

For predicting the 'TOTAL MILK PRODUCED' first I separated the data into two variable i.e. independent and dependent variable. For independent variable I used top four features that was correlated the most with 'TOTAL MILK PRODUCED' column like 'TOTAL MEAT', 'BUFF', 'COW MILK', & 'BUFF MILK' and for the dependent variable I used the column to be predicted i.e. 'TOTAL MILK PRODUCED'. After defining the independent and dependent variable in 'X' and 'y' then that data is split into train and test set where 80% of data is used for training and 20% for testing.

```
[44] # Split the data into training and testing sets
     X= filtered_df[['TOTAL MEAT', 'BUFF', 'COW MILK', 'BUFF MILK']]
     y=filtered_df['TOTAL MILK PRODUCED']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

After splitting data into training and testing set, the Linear Regression model is created into a variable 'model' and then the model is trained using function '.fit()' with data 'X_train' and 'y_train'.

```
[39] # Creating a linear regression model
     model = LinearRegression()
```

```
     # Training the model
     model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Finally, the model predicts on the basis of 'X_test' as features using the function '.predict()'.

```
# Making predictions
y_pred = model.predict(X_test)
y_pred
```

```
array([ 8748.96629658, 33198.97037974, 21507.9570943 , 12330.97569007,
       83509.01865616, 17576.96582624,  3060.94283754, 12345.95184397,
       19823.96575554, 13978.97031216, 15032.98255442, 73206.96174744,
       28401.99647635, 36765.96926532,  7895.96297917])
```

7.2. ‘TOTAL EGG’ Predicting Model:

For predicting the TOTAL EGG’ first I separated the data into two variable i.e. independent and dependent variable. For independent variable I used features that was correlated the most with TOTAL EGG’ column like ‘HEN EGG’, ‘LAYING HEN’, ‘LAYING DUCK’, & ‘DUCK EGG’ and for the dependent variable I used the column to be predicted i.e. ‘TOTAL EGG’. After defining the independent and dependent variable in ‘X’ and ‘y’ then that data is split into train and test set where 80% of data is used for training and 20% for testing.

```
[102] # Split the data into training and testing sets
      X= filtered_df[['HEN EGG', 'LAYING HEN', 'LAYING DUCK', 'DUCK EGG']]
      y=filtered_df['TOTAL EGG']
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

After splitting data into training and testing set, the Linear Regression model is created into a variable ‘model’ and then the model is trained using function ‘.fit()’ with data ‘X_train’ and ‘y_train’.

```
[51] # Creating a linear regression model
     model = LinearRegression()

     # Training the model
     model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Finally, the model predicts on the basis of ‘X_test’ as features using the function ‘.predict()’.

```
# Making predictions
y_pred = model.predict(X_test)
y_pred
```

```
array([ 1013.08545299,  5757.35183202,  4122.99537877,  1939.88455698,
       36324.73519648,  5632.87647169,   396.11675901,  1623.16575419,
       10778.564037  ,  7884.99217863, 41360.6554617 , 32084.76931911,
        9171.95145212,  8858.35171806, 40733.00976756])
```

8. Evaluation of Developed Model

8.1. Evaluating 'TOTAL MILK PRODUCED' Predicting Model:

Evaluation is one the most important process which measures the built model performance whether the model is performing good or not. There are different metric that evaluates a model and for our model which uses regression algorithm metric like: MSE, MAE, R2-score, etc are used. So, for evaluating the 'TOTAL MILK PRODUCED' Predicting Model, I used metric such as:

- **Mean Square Error (MSE):** It is a measure of the average squared difference between the actual and predicted values. For the model that predicts 'TOTAL MILK PRODUCED', MSE is approximately 0.00118 which indicates that the model prediction is close to the actual values on average.
- **Mean Absolute Error (MAE):** It is a measure of the average absolute difference between the actual and predicted values. For the model that predicts 'TOTAL MILK PRODUCED', MAE is approximately 0.03197 which indicates that the model is performing better.
- **R^2 Score:** It represents the proportion of variance in the dependent variable that is predictable from the independent variables. For the model that predicts 'TOTAL MILK PRODUCED', R^2 score is approximately 0.99999 which indicates an extremely high value, suggesting that the model fits the data extremely well.

```
92] # Importing necessary libraries
    from sklearn.metrics import mean_squared_error, r2_score

    # Evaluate the model
    print("Mean Squared Error:", mean_squared_error(y_test, y_pred))  # Using y instead of y_test
    print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
    print("Coefficient of Determination (R^2 Score):", r2_score(y_test, y_pred))  # Using y instead of y_test

    Mean Squared Error: 0.0011797891032862286
    Mean Absolute Error: 0.03197315450882646
    Coefficient of Determination (R^2 Score): 0.99999999999768
```

8.2. Evaluating 'TOTAL Egg' Predicting Model:

Evaluation is one the most important process which measures the built model performance whether the model is performing good or not. There are different metric that evaluates a

model and for our model which uses regression algorithm metric like: MSE, MAE, R2-score, etc are used. So, for evaluating the 'TOTAL Egg' Predicting Model, I used metric such as:

- **Mean Square Error (MSE):** It is a measure of the average squared difference between the actual and predicted values. For the model that predicts 'TOTAL EGG', MSE is approximately 2.52246 which indicates that the model prediction is close to the actual values on average.
- **Mean Absolute Error (MAE):** It is a measure of the average absolute difference between the actual and predicted values. For the model that predicts 'TOTAL EGG', MAE is approximately 6.88942 which indicates that the model is performing better.
- **R^2 Score:** It represents the proportion of variance in the dependent variable that is predictable from the independent variables. For the model that predicts 'TOTAL EGG, R^2 score is approximately 1.0 which indicates an perfect fit, suggesting that the model fits the data perfectly well.

```
# Importing necessary libraries
from sklearn.metrics import mean_squared_error, r2_score

# Evaluate the model
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))  # Using y instead of y_test
print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
print("Coefficient of Determination (R^2 Score):", r2_score(y_test, y_pred))  # Using y instead of y_test
```

```
Mean Squared Error: 2.5224614285863154e-22
Mean Absolute Error: 6.8894223659299316e-12
Coefficient of Determination (R^2 Score): 1.0
```

9. Conclusion

In conclusion, the analysis of Nepal's livestock and commodities production reveals key insights despite challenges like limited access to modern techniques and natural disasters. Through thorough data exploration and visualization, we identified trends, correlations, and factors influencing production across 75 districts.

Our predictive models for 'TOTAL MILK PRODUCED' and 'TOTAL EGG' showed strong performance, indicating potential for practical application. Evaluation metrics underscored the accuracy of these models.

Overall, this study contributes valuable knowledge to Nepal's agricultural sector, highlighting opportunities for sustainable practices and productivity enhancement. By leveraging data-driven strategies, Nepal can address challenges and promote economic growth and food security.