PROJECT FILE

CROP YIELD ANALYSIS

Submitted by – Abhishek Yadav

Submitted to – Ms. Shreya Chittranshi

INDEX

INDEX

Chapter 1: Introduction

1.1 Background of Agricultural Data Analysis

Agriculture plays a crucial role in the economic development of many countries, especially in India, where a large portion of the population depends on farming for their livelihood. Crop yield is influenced by several factors such as rainfall, soil quality, fertilizer usage, pesticide application, seasonality, and cultivated area. With the increasing availability of agricultural data, data analysis has become an essential tool for understanding these factors and improving agricultural productivity.

Agricultural data analysis helps policymakers, researchers, and farmers make informed decisions by identifying patterns, trends, and relationships within historical data. By analyzing crop yield data across different states, seasons, and years, it is possible to gain insights into regional performance, resource utilization, and the impact of environmental factors on production.

1.2 Importance of Crop Yield Analysis

Crop yield analysis is important for ensuring food security, optimizing resource usage, and increasing farmer income. Understanding how factors such as rainfall, fertilizer, and pesticide usage affect yield can help in planning better farming practices. Accurate analysis also supports government initiatives in agricultural planning, subsidy allocation, and risk management.

With climate variability and changing weather patterns, traditional farming methods are no longer sufficient. Data-driven insights allow stakeholders to predict trends, reduce losses, and improve overall productivity. Crop yield analysis also supports sustainable agriculture by identifying efficient practices that minimize environmental impact.

1.3 Objective of the Project

The primary objective of this project is to analyze crop yield data using Python and Power BI to extract meaningful insights that can support agricultural decision-making. The project focuses on cleaning raw agricultural data, performing exploratory data analysis (EDA), and visualizing the results through interactive dashboards.

The specific objectives of this project are:

- To understand the structure and quality of the crop yield dataset
- To perform data cleaning and preprocessing using Python
- To analyze crop yield patterns across different states, seasons, and years

- To study the impact of rainfall, fertilizer, and pesticide usage on crop yield
- To develop an interactive Power BI dashboard for effective data visualization

Chapter 2: Problem Statement

2.1 Challenges in Crop Yield Optimization

Agricultural productivity is affected by multiple interconnected factors such as climatic conditions, seasonal variations, resource availability, and farming practices. Farmers often face challenges in determining which crops perform best in a particular season or region. Irregular rainfall patterns, excessive or insufficient use of fertilizers and pesticides, and lack of data-driven planning further complicate crop yield optimization.

Traditional decision-making in agriculture is largely based on experience and intuition, which may not always lead to optimal results. Without systematic data analysis, it becomes difficult to identify the key factors that influence crop yield and to understand their combined impact on agricultural output.

2.2 Need for Data-Driven Agricultural Analysis

With the increasing availability of historical agricultural data, there is a strong need to adopt data-driven approaches for improving crop productivity. Analyzing past crop yield data can help uncover hidden patterns and trends related to crop performance across different states, seasons, and years.

Data-driven agricultural analysis enables stakeholders such as farmers, researchers, and policymakers to make informed decisions regarding crop selection, resource allocation, and risk management. By leveraging data analytics tools, it becomes possible to assess the impact of rainfall, fertilizer usage, and pesticide application on crop yield, leading to more efficient and sustainable farming practices.

2.3 Problem Definition

The primary problem addressed in this project is the lack of structured analysis of crop yield data to support effective agricultural decision-making. Although large volumes of agricultural data are available, they are often underutilized due to issues such as poor data quality, lack of proper analysis, and absence of intuitive visual representations.

This project aims to address the following key questions:

- How does crop yield vary across different states and seasons?
- Which crops show consistently high or low yield performance?
- What is the relationship between rainfall and crop yield?
- How do fertilizer and pesticide usage impact agricultural production?

Chapter 3: Dataset Description

3.1 Data Source

The dataset used in this project is a crop yield dataset containing historical agricultural data from various regions of India. The data represents crop production statistics collected over multiple years and seasons. Such datasets are commonly compiled from government agricultural records and surveys, making them reliable for analytical and research purposes.

The dataset is provided in CSV (Comma-Separated Values) format, which allows easy handling and processing using data analysis tools such as Python and Power BI.

3.2 Dataset Structure

The crop yield dataset consists of **19,689 records** and **10 attributes**. Each row represents agricultural data for a specific crop cultivated in a particular state, season, and year. The dataset includes both numerical and categorical variables, making it suitable for exploratory data analysis and visualization.

The structure of the dataset enables multi-dimensional analysis such as crop-wise, state-wise, season-wise, and year-wise comparisons.

	A	B	C	D	E	F	G	H	I	J
1	Crop	Crop_Year	Season	State	Area	Production	Annual_Rainfall	Fertilizer	Pesticide	Yield
2	Arecanut	1997	Whole Year	Assam	73814	56708	2051.4	7024878.38	22882.34	0.796086957
3	Arhar/Tur	1997	Kharif	Assam	6637	4685	2051.4	631643.29	2057.47	0.710434783
4	Castor seed	1997	Kharif	Assam	796	22	2051.4	75755.32	246.76	0.238333333
5	Coconut	1997	Whole Year	Assam	19656	126905000	2051.4	1870661.52	6093.36	5238.051739
6	Cotton(lint)	1997	Kharif	Assam	1739	794	2051.4	165500.63	539.09	0.420909091
7	Dry chillies	1997	Whole Year	Assam	13587	9073	2051.4	1293074.79	4211.97	0.643636364
8	Gram	1997	Rabi	Assam	2979	1507	2051.4	283511.43	923.49	0.465454545
9	Jute	1997	Kharif	Assam	94520	904095	2051.4	8995468.4	29301.2	9.919565217
10	Linseed	1997	Rabi	Assam	10098	5158	2051.4	961026.66	3130.38	0.461363636
11	Maize	1997	Kharif	Assam	19216	14721	2051.4	1828786.72	5956.96	0.615652174
12	Mesta	1997	Kharif	Assam	5915	29003	2051.4	562930.55	1833.65	4.568947368
13	Niger seed	1997	Whole Year	Assam	9914	5076	2051.4	943515.38	3073.34	0.482352941

Figure 3.1: Dataset structure

3.3 Feature Description

The dataset contains the following features:

- **Crop:** Name of the crop cultivated
- **Crop_Year:** Year in which the crop was grown
- **Season:** Agricultural season (Kharif, Rabi, Whole Year, etc.)
- **State:** Indian state where the crop was cultivated
- **Area:** Total area under cultivation (in hectares)

- **Production:** Total crop production
- **Annual_Rainfall:** Annual rainfall received (in millimeters)
- **Fertilizer:** Amount of fertilizer used
- **Pesticide:** Amount of pesticide used
- **Yield:** Crop yield per unit area

These features collectively help in understanding the relationship between environmental factors, resource usage, and agricultural output

3.4 Initial Data Observations

Before performing data cleaning and analysis, an initial inspection of the dataset was conducted. This step helped identify data quality issues such as missing values, inconsistent data types, and potential outliers. Some numerical columns showed wide variation in values, indicating the presence of extreme observations.

Categorical variables such as crop type, state, and season contained multiple unique values, making them suitable for comparative analysis. The presence of both environmental and input-related variables makes this dataset valuable for studying factors affecting crop yield.

Chapter 4: Tools & Technologies Used

4.1 Programming Language – Python

Python is used as the primary programming language in this project due to its simplicity, flexibility, and extensive support for data analysis. Python provides powerful libraries that make it easy to handle large datasets, perform data cleaning, and conduct exploratory data analysis efficiently.

In this project, Python is used to load the crop yield dataset, preprocess the data, handle missing values, and analyze relationships between different variables. The use of Python ensures reproducibility and accuracy throughout the data analysis process.

4.2 Development Environment – Jupyter Notebook

Jupyter Notebook is used as the development environment for performing Python-based data analysis in this project. Jupyter Notebook provides an interactive platform where code execution, output visualization, and documentation can be combined in a single interface.

Using Jupyter Notebook allows step-by-step execution of data cleaning and EDA processes, making the analysis easier to understand and reproduce. It also helps in visualizing plots and statistical summaries immediately after code execution, which is beneficial for exploratory analysis.

Figure 4.1: Jupyter Notebook Interface

4.3 Python Libraries

The following Python libraries are used extensively in this project:

- **Pandas:** For data loading, cleaning, manipulation, and transformation of tabular data.
- **NumPy:** For numerical operations and efficient handling of arrays.
- **Matplotlib:** For creating basic visualizations such as line charts, bar charts, and histograms.
- **Seaborn:** For advanced statistical visualizations and enhanced graphical representation.

These libraries together form a powerful toolkit for performing exploratory data analysis on the crop yield dataset.

```
[1]: import pandas as pd
     import numpy as np
     import matplotlib as plt
     import seaborn as sns

     C:\Users\abhis\AppData\Roaming\Pyth
     'bottleneck' (version '1.3.5' curre
       from pandas.core import (
```

Figure 4.2: Python Libraries Used in the Project

4.4 Visualization Tool – Power BI

Microsoft Power BI is used as the primary data visualization tool in this project to present analyzed crop yield data in an interactive and user-friendly manner. Power BI helps convert complex datasets into meaningful visual insights that can be easily understood by different stakeholders such as farmers, researchers, and policymakers.

After completing data cleaning and exploratory data analysis in Jupyter Notebook using Python, the cleaned dataset is exported and imported into Power BI. Various visualization components such as bar charts, line charts, KPI cards, tables, and slicers are used to analyze crop yield trends across different states, seasons, crops, and years.

One of the major advantages of Power BI is its interactivity. Users can apply filters and slicers to dynamically explore the data and compare crop performance based on multiple parameters. This interactive approach enhances data interpretation and supports data-driven decision-making in the agricultural domain.

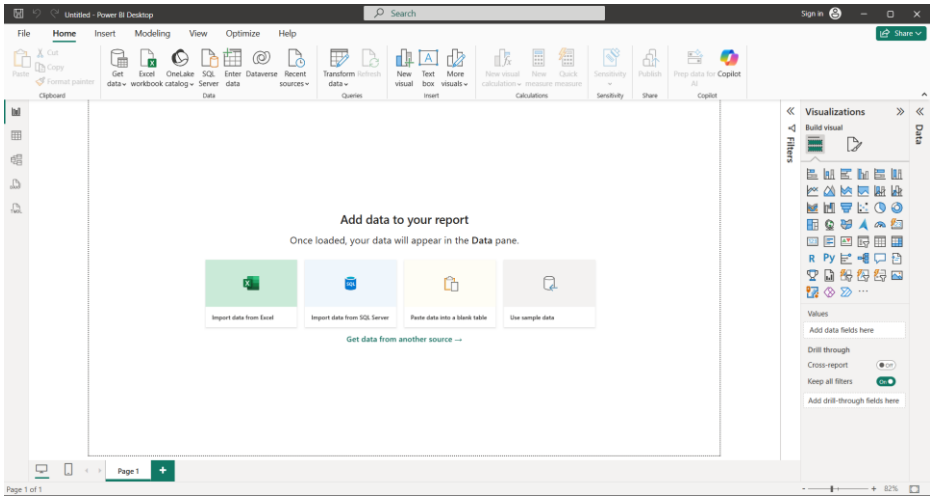

Figure 4.3: Power BI Dashboard and Visualization Interface

Chapter 5: Project Workflow & Architecture

5.1 Overall Workflow

The crop yield analysis project follows a structured and systematic workflow to ensure accurate analysis and meaningful insights. The workflow begins with understanding the dataset and progresses through data cleaning, exploratory data analysis, and visualization. Each stage of the workflow is designed to transform raw data into actionable information.

The major steps involved in the project workflow are:

- Data collection and understanding
- Data cleaning and preprocessing using Python
- Exploratory Data Analysis (EDA)
- Data export for visualization
- Dashboard creation using Power BI
- Insight generation and reporting

This step-by-step workflow ensures data consistency, reliability, and clarity in analysis.

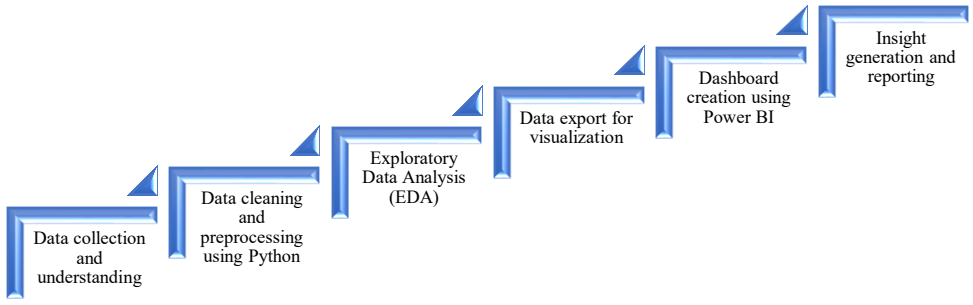

Figure 5.1: Overall Project Workflow Diagram

5.2 Data Processing Pipeline

The data processing pipeline defines how raw data is converted into a cleaned and analysis-ready format. Initially, the dataset is loaded into Python using the Pandas library. Various preprocessing steps such as handling missing values, correcting data types, and removing duplicates are applied.

Once the data is cleaned, it is used for exploratory data analysis to identify trends, patterns, and relationships between variables such as crop yield, rainfall, fertilizer usage, and pesticide application. After EDA, the cleaned dataset is exported and imported into Power BI for visualization and dashboard development.

Chapter 6: Data Cleaning Using Python

Data cleaning is a critical step in the data analysis process, as the accuracy and reliability of insights depend on the quality of the dataset. Raw agricultural data often contains missing values, inconsistent data formats, duplicate records, and outliers. In this project, Python is used in a Jupyter Notebook environment to systematically clean and prepare the crop yield dataset for analysis and visualization.

6.1. Identification of Data Quality

The initial examination of the dataset was performed to understand its structure and identify potential data quality issues. This included checking the dataset size, data types, missing values, and summary statistics. During this process, issues such as missing values in numerical columns, inconsistent data formats, and unusually high or low values were identified.

Early identification of these issues helped in selecting appropriate cleaning techniques and ensured that the dataset was suitable for further exploratory data analysis and visualization.

1. **Checking the dataset shape and size**
    - df.size - property returns the total number of elements in a DataFrame or Series.
    - df.shape - property of a DataFrame (or Series) that returns its dimensions as a tuple

Importing dataset

```
df = pd.read_csv("crop_yield.csv")
```

```
df.head()
```

	Crop	Crop_Year	Season	State	Area	Production	Annual_Rainfall	Fertilizer	Pesticide	Yield
0	Arecanut	1997	Whole Year	Assam	11.209317	10.945688	2051.4	15.764969	10.038164	0.796087
1	Arhar/Tur	1997	Kharif	Assam	8.800566	8.452335	2051.4	13.356082	7.629718	0.710435
2	Castor seed	1997	Kharif	Assam	6.680855	3.135494	2051.4	11.235277	5.512461	0.238333
3	Coconut	1997	Whole Year	Assam	9.886189	18.658949	2051.4	14.441803	8.715119	5.072222
4	Cotton(lint)	1997	Kharif	Assam	7.461640	6.678342	2051.4	12.016736	6.291736	0.420909

Checking shape and zize of dataset

```
print(df.shape)
print(df.size)
```

```
(19689, 10)
196890
```

Figure 6.1: Dataset shape and size

2. **Checking the data types**
    - df.dtypes - property of a DataFrame that returns the data type (dtype) of each column.

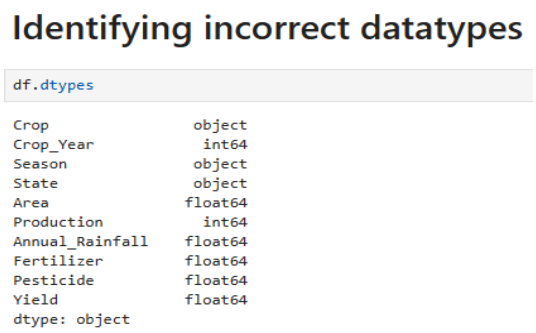

Figure 6.2: Checking data types

3. **Identifying missing values**
    - df.isnull() - Returns a DataFrame of the same shape as df with True where values are NaN and False otherwise.
    - sum() - When applied to a boolean DataFrame, True is treated as 1 and False as 0, so summing counts the number of NaN values per column by default.

Figure 6.3: Identifying missing values

4. **Identifying duplicate data**
    - df.duplicated() - Returns a boolean Series where:
        - True → the row is a duplicate of a previous row.
        - False → the row is the first occurrence of that combination of values.
        - By default, it checks all columns and keeps the first occurrence as False.
    - .sum() –
        - Since True is treated as 1 and False as 0 in Pandas,
        - .sum() counts how many True values there are — i.e., the number of duplicate rows.

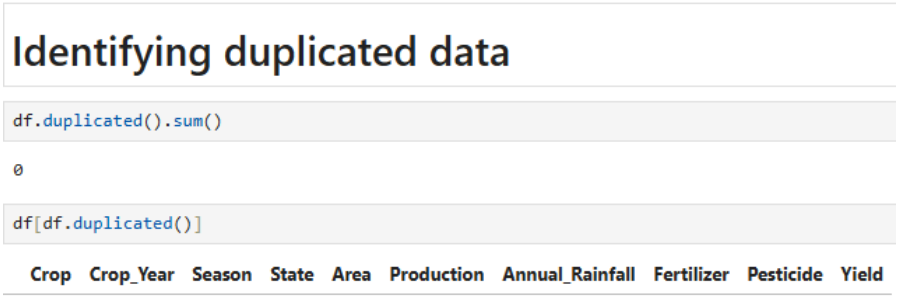

Figure 6.4: Identifying duplicate data

5. **Identifying outliers**
   - **Boxplot method** – The **boxplot method** is a statistical visualization technique used to display the distribution, spread, and potential outliers in a dataset. It is also called a **box-and-whisker plot**.

Key Components of a Boxplot

1. **Median (Q2)** – The middle value of the dataset.
2. **First Quartile (Q1)** – The median of the lower half of the data (25th percentile).
3. **Third Quartile (Q3)** – The median of the upper half of the data (75th percentile).
4. **Interquartile Range (IQR)** – $IQR = Q3 - Q1$ (measures spread of the middle 50% of data).
5. **Whiskers** – Extend from the box to the smallest and largest values within $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$.
6. **Outliers** – Points outside the whiskers.

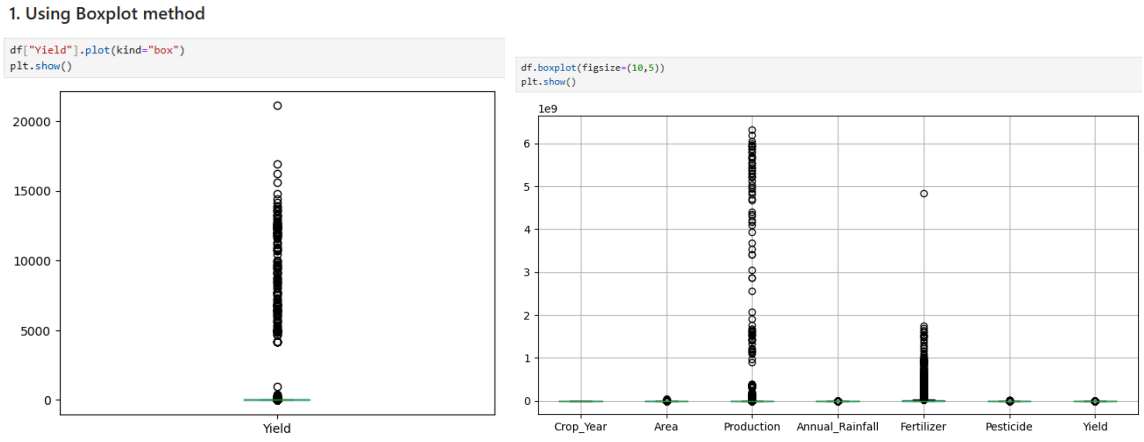

Figure 6.5: Identifying outliers using boxplot method

- **Z-score method –**
  The Z-Score Method is a statistical technique used to measure how many standard deviations a data point is from the mean of a dataset.

2. Using z-score method

```
from scipy import stats
z_scores = np.abs(stats.zscore(df.select_dtypes(include="number")))
outliers = df[(z_scores > 3).any(axis=1)]
outliers
```

	Crop	Crop_Year	Season	State	Area	Production	Annual_Rainfall	Fertilizer	Pesticide	Yield
3	Coconut	1997	Whole Year	Assam	19656.0	126905000	2051.4	1.870662e+06	6093.36	5238.051739
60	Coconut	1997	Whole Year	Kerala	884344.0	5210000000	3252.4	8.416302e+07	274146.64	5376.054286
94	Coconut	1997	Whole Year	West Bengal	24273.0	306202300	1852.9	2.310061e+06	7524.63	13003.787780
110	Niger seed	1997	Rabi	West Bengal	50808100.0	38657300	1852.9	4.835407e+09	15750511.00	0.698889
118	Rice	1997	Winter	West Bengal	4270328.0	8915100	1852.9	4.064071e+08	1323801.68	1.985556
...	...	...	...	...	...	...	...	...	...	...
19557	Rice	2014	Winter	Odisha	3282000.0	7802000	1536.9	4.954507e+08	1083060.00	2.379333
19585	Rice	2015	Winter	Odisha	3167000.0	4702000	1210.1	5.001010e+08	1045110.00	1.417667
19614	Rice	2016	Winter	Odisha	3226450.0	8343100	1460.5	4.944535e+08	1129257.50	2.543000
19643	Rice	2017	Winter	Odisha	3014130.0	5279130	1344.4	4.745446e+08	1145369.40	1.747000
19672	Rice	2018	Winter	Odisha	3130820.0	6280410	1635.9	5.078190e+08	1095787.00	1.982333

856 rows × 10 columns

Figure 6.6: Identifying outliers using z-score method

- **IQR method –**
  IQR is used to measure variability by dividing a data set into quartiles. The data is sorted in ascending order and then we split it into 4 equal parts. The values Q1 (25th percentile), Q2 (50th percentile or median) and Q3 (75th percentile) separate dataset in 4 equal parts.

  If a dataset has 2n or 2n+1 data points, then

  Q2 = median of the dataset.
  Q1 = median of n smallest data points.
  Q3 = median of n highest data points.
  The IQR is calculated as: IQR=Q3−Q1 = Q3 - Q1

  Data points that fall below Q1−1.5×IQR or above Q3+1.5×IQR are considered outliers.

3. Using IQR method

```
numeric_cols = df.select_dtypes(include="number")

outliers = {}
for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outliers[col] = df[(df[col] < lower) | (df[col] > upper)]

print(outliers[col])
```

```
            Crop  Crop_Year       Season      State      Area  Production  \
3       Coconut        1997  Whole Year       Assam   19656.0   126905000
7           Jute       1997  Kharif           Assam   94520.0      904095
14        Potato       1997  Whole Year       Assam   75259.0      671871
21     Sugarcane       1997  Kharif           Assam   31318.0     1287451
54     Sugarcane       1997  Whole Year   Karnataka  308857.0    28999269
...          ...        ...         ...         ...       ...         ...
19618  Sugarcane       2016  Winter           Odisha    5493.0      344294
19636     Potato       2017  Winter           Odisha    3966.0       41812
19647  Sugarcane       2017  Winter           Odisha    3713.0      240245
19665     Potato       2018  Winter           Odisha    4900.0       54455
19676  Sugarcane       2018  Winter           Odisha    6778.0      417672

       Annual_Rainfall   Fertilizer  Pesticide        Yield
3               2051.4   1870661.52    6093.36  5238.051739
7               2051.4   8995468.40   29301.20     9.919565
14              2051.4   7162399.03   23330.29     7.561304
21              2051.4   2980534.06    9708.58    41.896957
54              1266.7  29393920.69   95745.67    91.747368
...                ...         ...        ...          ...
19618           1460.5    841802.25    1922.55    56.160400
19636           1344.4    624407.04    1507.08    11.955455
19647           1344.4    584574.72    1410.94    56.588000
19665           1635.9    794780.00    1715.00    13.017308
19676           1635.9   1099391.60    2372.30    57.584545

[3065 rows x 10 columns]
```

Figure 6.7: Identifying outliers using IQR method

6.2. Handling outliers

1. Using log transformation – a **log transformation** means applying a logarithmic function to one or more columns of a DataFrame — usually to reduce skewness, stabilize variance, or make data more normally distributed for statistical modeling or machine learning.

Handling Outliers

1. Using log transformation

```
df["Production"] = np.log1p(df["Production"])
df["Fertilizer"] = np.log1p(df["Fertilizer"])
df["Area"] = np.log1p(df["Area"])
```

Figure 6.8: handing outliers using log transformation

2. Using IQR capping –

IQR capping in Python Pandas is a **data preprocessing technique** used to handle **outliers** by **limiting (capping) extreme values** to a certain range based on the **Interquartile Range (IQR)**.

Instead of **removing** outliers, we **cap** them:

- **Lower Cap** = $Q1 - 1.5 \times IQR$
- **Upper Cap** = $Q3 + 1.5 \times IQR$

Any value **below** the lower cap is replaced with the lower cap. Any value **above** the upper cap is replaced with the upper cap.

2. Using IQR capping

```
Q1 = df["Pesticide"].quantile(0.25)
Q3 = df["Pesticide"].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

df["Pesticide"] = df["Pesticide"].clip(lower_bound, upper_bound)
```

```
Q1 = df["Yield"].quantile(0.25)
Q3 = df["Yield"].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5*IQR
upper_bound = Q3 + 1.5*IQR

df["Yield"] = df["Yield"].clip(lower_bound,upper_bound)
```

Figure 6.9: handing outliers using IQR capping

Chapter 7: Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is the process of understanding and summarizing a dataset to identify patterns, trends, and data quality issues. In this project, EDA was performed on crop yield data by handling missing values, detecting and treating outliers, and analyzing data skewness. Crop-wise, state-wise, and year-wise analyses were carried out to study variations in yield and production. Correlation analysis was used to understand the impact of rainfall, fertilizer, and pesticide usage on crop yield. Overall, EDA helped in gaining meaningful insights and preparing the data for further analysis.

7.1. Correlation insights

Correlation insights

```
num_col = df.select_dtypes(include = "number")
num_col.corr()
```

	Crop_Year	Area	Production	Annual_Rainfall	Fertilizer	Pesticide	Yield
Crop_Year	1.000000	-0.106848	-0.066105	-0.011187	-0.045523	-0.058810	0.063976
Area	-0.106848	1.000000	0.894875	-0.207886	0.997521	0.993572	0.052752
Production	-0.066105	0.894875	1.000000	-0.148167	0.894139	0.891134	0.427950
Annual_Rainfall	-0.011187	-0.207886	-0.148167	1.000000	-0.208513	-0.206394	0.055073
Fertilizer	-0.045523	0.997521	0.894139	-0.208513	1.000000	0.992411	0.056149
Pesticide	-0.058810	0.993572	0.891134	-0.206394	0.992411	1.000000	0.056956
Yield	0.063976	0.052752	0.427950	0.055073	0.056149	0.056956	1.000000

```
larger area - more production
larger area - more fertilizers and pesticides used
more fertilizer and pesticides used - greater production
more fertilizer used - more pesticides used
more production - more yield
```

Figure 7.1: Correlation insights

Insights:

1. Larger cultivated area leads to higher crop production.
2. Bigger farm areas require more fertilizer and pesticide usage.
3. Increased use of fertilizers and pesticides results in greater production.
4. Fertilizer and pesticide usage are positively related to each other.
5. Higher production is associated with increased crop yield.

7.2. Crop wise yield analysis

Average Yield per Crop

```
crop_yield_avg = df.groupby("Crop")["Yield"].mean().sort_values(ascending = False)
crop_yield_avg
```

```
Crop
Banana                  5.030816
Potato                  4.999328
Tapioca                 4.954342
Coconut                 4.851392
Sugarcane               4.790234
Sweet potato            4.648649
Onion                   4.527079
Jute                    3.942178
Mesta                   3.788942
Ginger                  3.528965
Garlic                  3.347397
Turmeric                2.618879
Maize                   2.247685
Rice                    2.215173
Wheat                   2.005086
Tobacco                 1.812110
Oilseeds total          1.788548
Dry chillies            1.755611
Arecanut                1.724488
Cotton(lint)            1.604690
Barley                  1.595540
Bajra                   1.366532
Groundnut               1.360983
Sannhamp                1.245473
Peas & beans (Pulses)   1.224927
Ragi                    1.215408
Soyabean                1.083194
Jowar                   1.072498
other oilseeds          1.044610
Sunflower               0.934086
Gram                    0.866723
Guar seed               0.863315
Other Summer Pulses     0.859925
Other Cereals           0.849546
Arhar/Tur               0.843021
Black pepper            0.829605
Cowpea(Lobia)           0.813224
Khesari                 0.788941
Rapeseed &Mustard       0.785097
Other  Rabi pulses      0.780380
Small millets           0.768722
Masoor                  0.703153
Other Kharif pulses     0.699513
Castor seed             0.693609
Coriander               0.646088
Cashewnut               0.631806
Urad                    0.584048
Safflower               0.565482
Moong(Green Gram)       0.530940
Sesamum                 0.477719
Linseed                 0.473930
Horse-gram              0.462822
Moth                    0.445995
Niger seed              0.423523
Cardamom                0.168355
Name: Yield, dtype: float64
```

Figure 7.2: Average yield per crop

Crop-wise Yield Statistics

```
crop_yield_stats = df.groupby("Crop")["Yield"].agg(
    Avg_Yield="mean",
    Max_Yield="max",
    Min_Yield="min",
    Std_Deviation="std"
).sort_values("Std_Deviation", ascending=False)

crop_yield_stats
```

Crop	Avg_Yield	Max_Yield	Min_Yield	Std_Deviation
Jute	3.942178	5.072222	0.000000	1.836908
Turmeric	2.618879	5.072222	0.000000	1.745857
Garlic	3.347397	5.072222	0.000000	1.737711
Ginger	3.528965	5.072222	0.000000	1.707779
Mesta	3.788942	5.072222	0.000000	1.676861
Sannhamp	1.245473	5.072222	0.000000	1.665130
Tobacco	1.812110	5.072222	0.000000	1.460467
Dry chillies	1.755611	5.072222	0.205000	1.420660
Oilseeds total	1.788548	5.072222	0.932500	1.402073
Arecanut	1.724488	5.072222	0.000000	1.285900
other oilseeds	1.044610	5.072222	0.000000	1.258291
Onion	4.527079	5.072222	0.000000	1.233787
Maize	2.247685	5.072222	0.000000	1.106283
Cotton(lint)	1.604690	5.072222	0.000000	1.087738
Wheat	2.005086	5.068636	0.000000	1.080466
Peas & beans (Pulses)	1.224927	5.072222	0.211667	1.076422
Coconut	4.851392	5.072222	0.000000	1.019807
Bajra	1.366532	5.072222	0.208000	0.990309
Sweet potato	4.648649	5.072222	0.000000	0.960486
Barley	1.595540	3.988235	0.260000	0.915257
Sugarcane	4.790234	5.072222	0.290000	0.841675
Black pepper	0.829605	4.133571	0.141667	0.835882
Rice	2.215173	5.072222	0.016667	0.783205
Coriander	0.646088	5.072222	0.000000	0.716066
Other Summer Pulses	0.859925	2.800000	0.334667	0.709807
Ragi	1.215408	4.336538	0.077500	0.688306
Guar seed	0.863315	5.072222	0.000000	0.655742
Groundnut	1.360983	3.673448	0.204167	0.623429
Cashewnut	0.631806	5.072222	0.000000	0.569377
Other Cereals	0.849546	4.394000	0.095455	0.532084
Rapeseed &Mustard	0.785097	5.072222	0.080000	0.526542
Sunflower	0.934086	5.020000	0.000000	0.520398
Jowar	1.072498	3.579091	0.003571	0.518530
Other Rabi pulses	0.780380	4.000000	0.000000	0.511245
Tapioca	4.954342	5.072222	0.000000	0.508792
Cowpea(Lobia)	0.813224	2.523333	0.000000	0.491204
Castor seed	0.693609	2.466333	0.070000	0.463937
Banana	5.030816	5.072222	0.000000	0.457339
Soyabean	1.083194	2.845000	0.000000	0.418491
Gram	0.866723	5.072222	0.000000	0.418283
Potato	4.999328	5.072222	0.000000	0.409284
Niger seed	0.423523	4.987647	0.000000	0.400237

Arhar/Tur	0.843021	5.072222	0.136667	0.391025
Cardamom	0.168355	2.870000	0.000000	0.376737
Other Kharif pulses	0.699513	2.800000	0.000000	0.370286
Small millets	0.768722	2.877500	0.000000	0.354805
Sesamum	0.477719	5.072222	0.000000	0.339642
Urad	0.584048	4.187500	0.000000	0.301862
Moth	0.445995	1.000000	0.000000	0.257807
Moong(Green Gram)	0.530940	1.773333	0.000000	0.253417
Masoor	0.703153	2.123750	0.258889	0.239129
Safflower	0.565482	1.256667	0.000000	0.237030
Khesari	0.788941	1.545652	0.281250	0.228552
Horse-gram	0.462822	1.300000	0.000000	0.225087
Linseed	0.473930	1.213333	0.000000	0.223668

Figure 7.3: Crop wise yield statistics

7.3. State wise production analysis

Total Production per State

```
state_production = df.groupby("State")["Production"].sum().sort_values(ascending = False)
state_production
```

```
State
Karnataka             15546.224089
Andhra Pradesh        13376.794363
West Bengal           10576.539099
Gujarat                9686.034721
Uttar Pradesh          9371.074454
Madhya Pradesh         9366.677046
Bihar                  9265.201717
Tamil Nadu             8898.884456
Maharashtra            8889.863477
Assam                  7599.154305
Chhattisgarh           7586.220615
Odisha                 7262.055338
Haryana                6079.256363
Uttarakhand            5921.235260
Nagaland               5705.329582
Meghalaya              5515.772293
Kerala                 4940.325354
Himachal Pradesh       4507.566984
Punjab                 4078.203964
Jammu and Kashmir      3915.815303
Manipur                3581.841530
Telangana              3561.320171
Puducherry             3383.358597
Tripura                3308.472354
Jharkhand              2783.160047
Mizoram                2761.156648
Arunachal Pradesh      2716.431186
Goa                    2220.202646
Sikkim                 1876.841732
Delhi                  1426.255322
Name: Production, dtype: float64
```

Figure 7.4: Total production per state

Average Production per State

```
state_production_avg = df.groupby("State")["Production"].mean().sort_values(ascending = False)
state_production_avg
```

```
State
Gujarat              11.855612
Maharashtra          11.545277
Uttar Pradesh        11.358878
Madhya Pradesh       11.084825
Karnataka            10.856302
Tamil Nadu           10.825893
Andhra Pradesh       10.566188
Assam                10.395560
Bihar                10.340627
Jharkhand            10.308000
Punjab               10.272554
West Bengal           9.667769
Haryana               9.634321
Odisha                9.555336
Arunachal Pradesh     9.302847
Kerala                9.251546
Goa                   9.025214
Telangana             8.970580
Meghalaya             8.498879
Sikkim                8.304609
Chhattisgarh          8.290951
Nagaland              8.280594
Manipur               8.067211
Uttarakhand           7.750308
Tripura               7.502205
Himachal Pradesh      7.317479
Delhi                 7.025888
Mizoram               6.637396
Jammu and Kashmir     6.205729
Puducherry            5.049789
Name: Production, dtype: float64
```

Figure 7.5: Average production per state

Top 10 States by Total Production

```
top_10_states=state_production.head(10)
top_10_states
```

```
State
Karnataka         15546.224089
Andhra Pradesh    13376.794363
West Bengal       10576.539099
Gujarat            9686.034721
Uttar Pradesh      9371.074454
Madhya Pradesh     9366.677046
Bihar              9265.201717
Tamil Nadu         8898.884456
Maharashtra        8889.863477
Assam              7599.154305
Name: Production, dtype: float64
```

Figure 7.6: Top 10 states by total production

7.4. Year wise analysis

Year-wise Total Production

```
year_production = df.groupby("Crop_Year")["Production"].sum().reset_index().sort_values("Crop_Year")
year_production
```

	Crop_Year	Production
0	1997	4219.553935
1	1998	6321.590454
2	1999	6542.245112
3	2000	7428.045727
4	2001	7218.632454
5	2002	7766.204461
6	2003	7773.500882
7	2004	7633.480839
8	2005	7716.283973
9	2006	7854.674941
10	2007	7842.542817
11	2008	8013.359743
12	2009	8067.349843
13	2010	7965.065814
14	2011	8441.741033
15	2012	8204.546076
16	2013	9110.548206
17	2014	9019.047299
18	2015	9281.484118
19	2016	9547.266593
20	2017	9857.825096
21	2018	9694.264810
22	2019	9905.282644
23	2020	282.732147

Figure 7.7: year wise total production

Year-wise Average Yield Trend

```
year_yield_avg = df.groupby("Crop_Year")["Yield"].mean().reset_index().sort_values("Crop_Year")
year_yield_avg
```

	Crop_Year	Yield
0	1997	1.562372
1	1998	1.725275
2	1999	1.707434
3	2000	1.614543
4	2001	1.613396
5	2002	1.623080
6	2003	1.695892
7	2004	1.678658
8	2005	1.713691
9	2006	1.695833
10	2007	1.717748
11	2008	1.779345
12	2009	1.704620
13	2010	1.722471
14	2011	1.799630
15	2012	1.829022
16	2013	1.959028
17	2014	1.925583
18	2015	1.877824
19	2016	1.881114
20	2017	1.934463
21	2018	1.908319
22	2019	1.951521
23	2020	1.751475

Figure 7.8: Year wise average yield trend

Year wise cultivated area ¶

```
year_area = (
    df.groupby("Crop_Year")["Area"]
    .sum()
    .reset_index()
    .sort_values("Area", ascending =False)
)
year_area
```

	Crop_Year	Area
22	2019	9366.448348
20	2017	9347.845733
21	2018	9194.776106
19	2016	9080.477219
18	2015	8840.649627
16	2013	8627.784675
17	2014	8545.888288
14	2011	8099.794556
15	2012	7854.155533
12	2009	7792.517034
11	2008	7692.924226
13	2010	7682.520354
9	2006	7613.594533
5	2002	7611.529606
10	2007	7560.721441
6	2003	7544.408668
8	2005	7478.141778
7	2004	7419.642870
3	2000	7260.702174
4	2001	7030.222540
2	1999	6335.422805
1	1998	6110.349048
0	1997	4143.306355
23	2020	266.139951

Figure 7.9: Year wise cultivated area

Year wise rainfall

```
year_rainfall = (
    df.groupby("Crop_Year")["Annual_Rainfall"]
    .mean()
    .reset_index()
    .sort_values("Crop_Year")
)
year_rainfall
```

	Crop_Year	Annual_Rainfall
0	1997	1595.778293
1	1998	1762.342879
2	1999	1590.549135
3	2000	1450.290042
4	2001	1406.769550
5	2002	1276.836457
6	2003	1417.740044
7	2004	1450.049253
8	2005	1462.180610
9	2006	1362.007989
10	2007	1549.051027
11	2008	1370.262810
12	2009	1181.512471
13	2010	1467.874636
14	2011	1381.697549
15	2012	1261.990716
16	2013	1434.729713
17	2014	1292.137098
18	2015	1356.429259
19	2016	1500.625193
20	2017	1491.620901
21	2018	1631.231985
22	2019	1520.573031
23	2020	1313.947826

Figure 7.10: Year wise rainfall

7.5. Season wise analysis

season wise rainfall

```
season_rainfall = df.groupby("Season")["Annual_Rainfall"].sum().reset_index().sort_values("Annual_Rainfall", ascending = False)
season_rainfall
```

	Season	Annual_Rainfall
1	Kharif	1.131835e+07
2	Rabi	7.970757e+06
4	Whole Year	6.078787e+06
3	Summer	1.565900e+06
0	Autumn	7.031759e+05
5	Winter	6.709910e+05

Figure 7.11: Season wise rainfall

season wise production

```
season_production = df.groupby(["Season"])["Production"].sum().reset_index().sort_values("Production",ascending = False)
season_production
```

	Season	Production
1	Kharif	75225.306923
2	Rabi	52335.579256
4	Whole Year	38410.472189
3	Summer	11072.370994
5	Winter	4382.849251
0	Autumn	4280.690405

Figure 7.12: Season wise production

Season wise yield

```
season_yield = df.groupby("Season")["Yield"].sum()
season_yield
```

```
Season
Autumn          571.098584
Kharif        11750.923076
Rabi           8151.279404
Summer         2301.865497
Whole Year    11607.658538
Winter          739.301955
Name: Yield, dtype: float64
```

Figure 7.13: Season wise yield

Chapter 8: Data Visualization Using Power BI

Data visualization plays a vital role in transforming analyzed data into meaningful insights that support effective decision-making. In this project, Microsoft Power BI is used to create interactive dashboards that visually represent crop yield patterns and trends across different states, seasons, and years. Power BI enables users to explore data dynamically and gain a clear understanding of complex agricultural datasets.

8.1. Data Import into Power BI

After completing data cleaning and exploratory data analysis using Python in Jupyter Notebook, the cleaned dataset was exported in CSV format and imported into Power BI Desktop. Power BI provides a simple and efficient data loading mechanism that allows seamless integration of external datasets.

During the import process, the dataset was reviewed to ensure correct data types and column names. Any minor inconsistencies were verified to maintain alignment with the cleaned dataset used in Python. Successful data import ensured that the dataset was ready for modeling and visualization.

Steps to import data in Power BI:

1. Open Power BI application stored on the pc.
2. You will see the window shown below, then click on “Excel Workbook”.

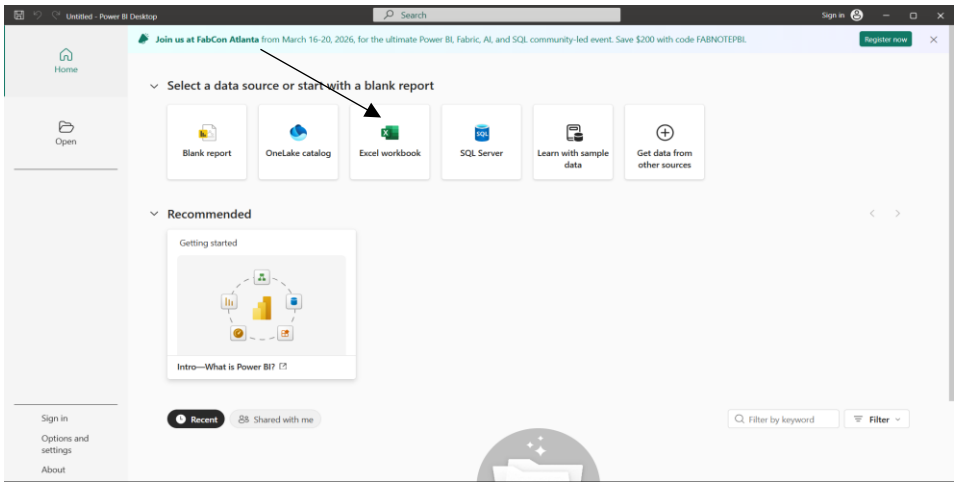

Figure 8.1: Click excel workbook

3. You will a new window as shown below, here you navigate to the file or data you want to import, and then click “Open”.

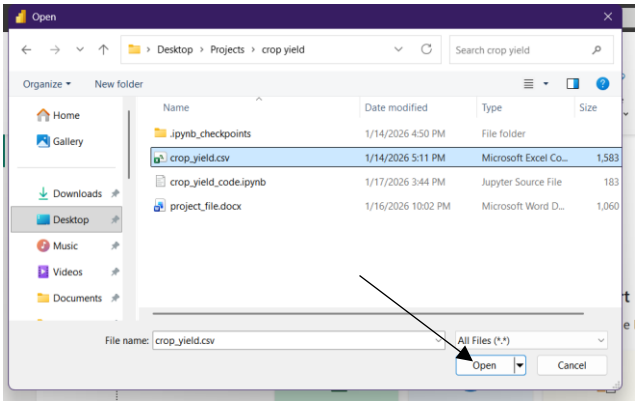

Figure 8.2: Open data file

4. Then new window will appear showing the data, and you cleaned your data previously, then click on “Load”.

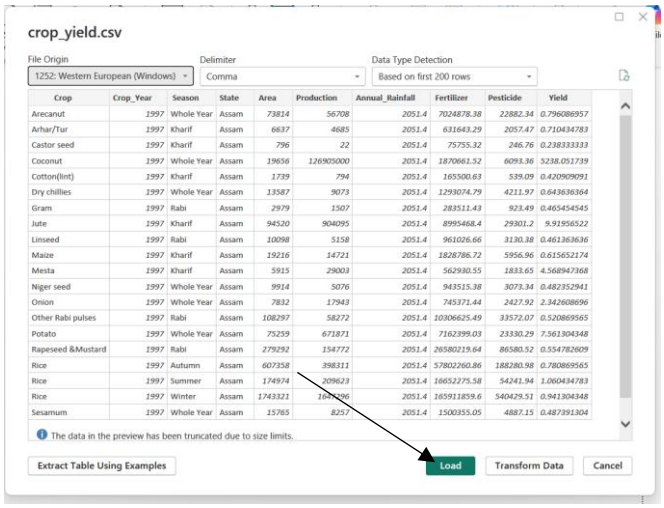

Figure 8.3: Load data

5. Then you will see the window as shown below, now you can visualize the data as per your requirements.

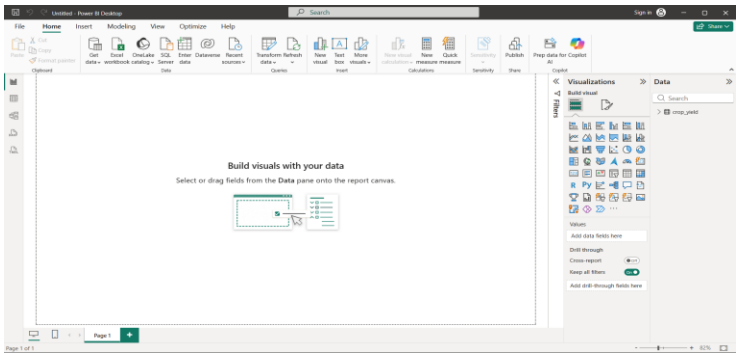

Figure 8.4: Data loaded

8.2. Data Modeling

Data modeling is an essential step in Power BI that defines relationships between different fields and ensures accurate analysis. Since the project uses a single cleaned dataset, the data model was designed to support efficient filtering, aggregation, and slicing of data.

Appropriate data types were assigned to numerical and categorical columns to enable correct calculations and visual interactions. Measures and calculated fields were created where necessary to support key performance indicators such as total production, average yield, and cultivated area. A well-structured data model helped improve dashboard performance and accuracy.

8.3. Visual Selection Strategy

Selecting the right type of visualization is crucial for effective data interpretation. In this project, different visuals were chosen based on the nature of the data and the analytical questions being addressed.

Bar charts and column charts were used to compare crop yield and production across states and crops. Line charts were used to analyze trends over years. KPI cards were used to display key metrics such as total production and average yield, while slicers were added to allow users to filter data by year, state, crop, and season.

This visual selection strategy ensured clarity, usability, and meaningful insight generation from the Power BI dashboard.

Chapter 9: Dashboard Explanation

9.1. KPI Metrics

- Total Production

Figure 9.1: Total Production

- Average Yield

Figure 9.2: Average Yield

- Cultivated Area

Figure 9.3: Cultivate Area

- Average Rainfall

Figure 9.4: Average Rainfall

9.2. Visual Field Mapping

- Year-Wise Trend Analysis

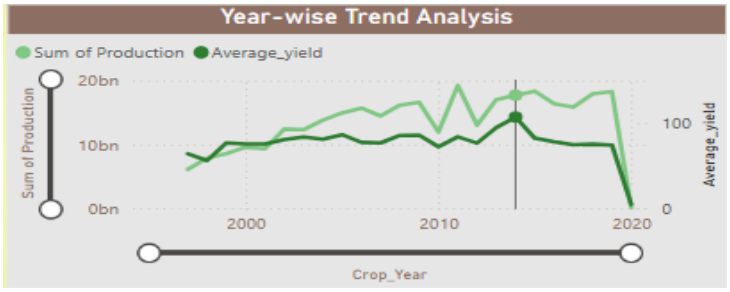

Figure 9.5: Year-wise trend analysis

Insights:

- Total production shows a gradual increasing trend over the years, indicating improvements in agricultural practices.
- Yield has improved over time, suggesting better seeds, fertilizers, and farming techniques.
- Sudden drops or peaks in certain years are closely linked to rainfall variability.

- State-Wise Production Analysis

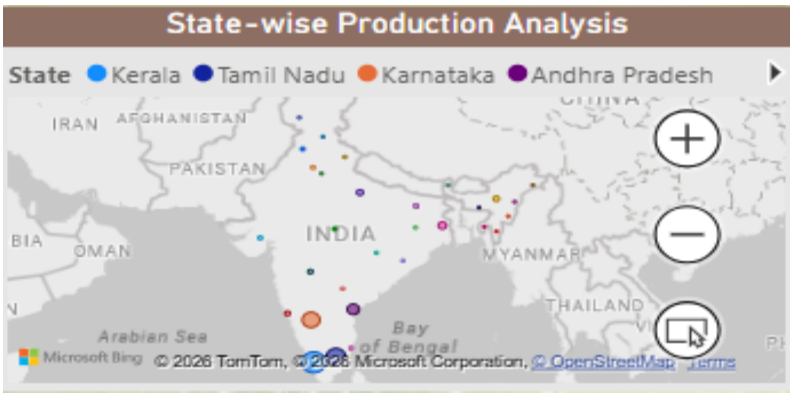

Figure 9.6: State-wise production analysis

Insights:

- A small number of states dominate total agricultural production, highlighting regional concentration.
- States with high production often combine large area, better rainfall, and higher input usage**.**

- Crop-Wise Yield Analysis

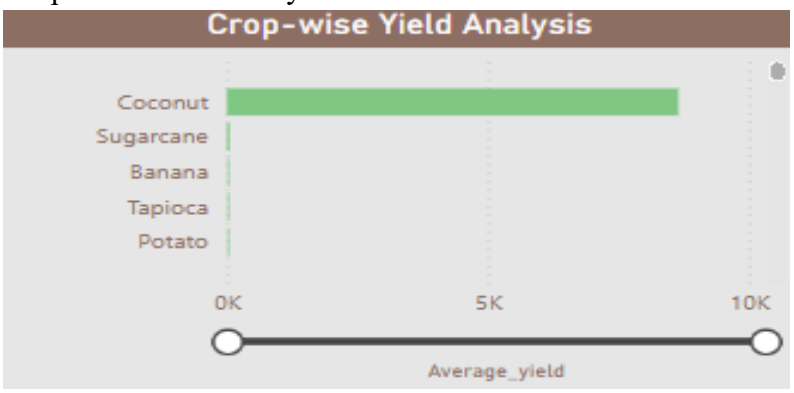

Figure 9.7: Crop-wise yield analysis

Insights:

- Crop-wise analysis shows significant variation in yield across different crops.
- Some crops consistently deliver high yield with low variability, while others show unstable yield patterns, possibly due to climate or input dependency.

- Fertilizer Vs Yield

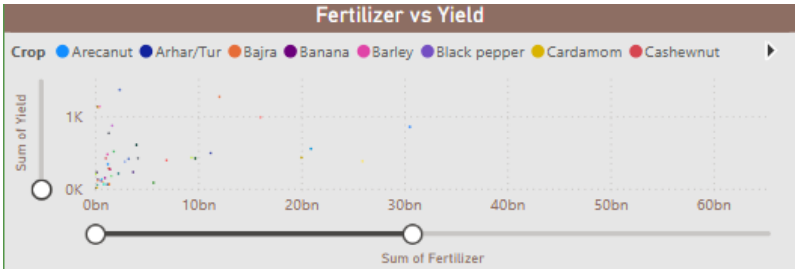

Figure 9.8: Fertilizer vs yield

- Pesticide Vs Yield

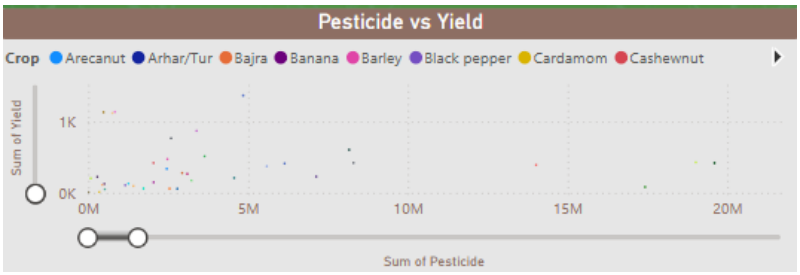

Figure 9.9: Pesticide vs yield

Insights:

- Increased fertilizer usage is positively correlated with higher yield and production.
- Pesticide usage supports yield improvement but shows a weaker impact compared to fertilizers.
- Fertilizer and pesticide usage are strongly related, indicating combined application practices.

9.3. Interactive Filters

- Season

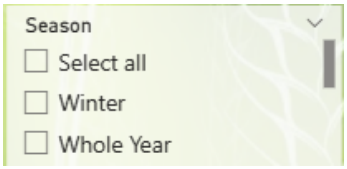

Figure 9.10: Season filter

- State

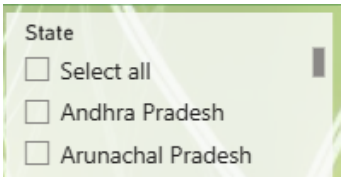

Figure 9.11: State filter

- Crop

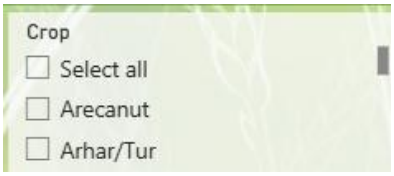

Figure 9.12: Crop filter

- Crop Year

Figure 9.13: Year filter

9.4. Dashboard

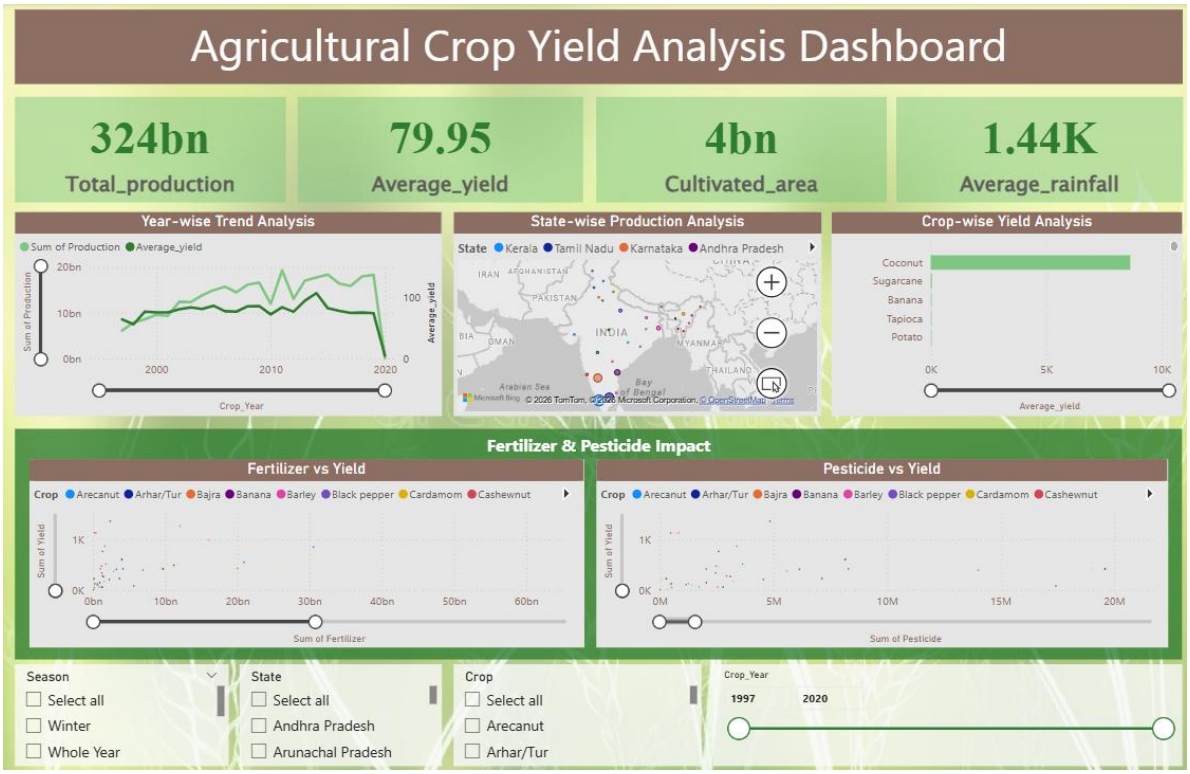

Figure 9.14: Dashboard

Chapter 10: Conclusion

10.1. Summary of Analysis

This project performed an in-depth exploratory data analysis on agricultural crop data to understand patterns in production, yield, and input usage across different states, crops, and years. The analysis involved data cleaning, outlier treatment, skewness analysis, and correlation study to ensure data reliability. Crop-wise, state-wise, and year-wise analyses were conducted to identify productivity trends. Power BI was used to visualize key metrics through interactive dashboards, enabling effective comparison and trend identification. Overall, the analysis provided a comprehensive understanding of factors influencing agricultural performance.

10.2. Agricultural Insights & Learnings

- Larger cultivated areas contribute significantly to higher agricultural production.
- Crop yield varies widely across different crops, indicating the need for crop-specific farming strategies.
- Fertilizer usage has a strong positive impact on both yield and total production.
- Pesticide usage supports crop protection and yield improvement but has a comparatively weaker impact than fertilizers.
- Fertilizer and pesticide usage are closely related, suggesting combined application practices in farming.
- Rainfall plays a moderate role in influencing yield and causes year-to-year production fluctuations.
- Agricultural production has shown an increasing trend over the years, reflecting improvements in farming practices and technology.
- Efficient use of agricultural inputs and climate-aware planning can further enhance productivity.

Chapter 11: Future Scope

11.1. Yield Prediction Using Machine Learning

In the future, machine learning models can be developed to predict crop yield based on historical data such as cultivated area, rainfall, fertilizer usage, pesticide usage, and crop type. Algorithms like Linear Regression, Random Forest, and XGBoost can be used to improve prediction accuracy. Such predictive models can assist farmers and policymakers in planning crop selection and resource allocation more effectively.

11.2. Weather API Integration

The analysis can be further enhanced by integrating real-time and forecasted weather data using weather APIs. Parameters such as rainfall, temperature, humidity, and wind speed can be included to study their impact on crop yield. Weather API integration would allow dynamic analysis and improve the accuracy of yield prediction and risk assessment related to climate variability.

11.3. Real-Time Dashboard

A real-time dashboard can be developed using Power BI or web-based visualization tools to monitor agricultural performance continuously. By connecting live data sources such as weather APIs, satellite data, and market inputs, stakeholders can track production trends, input usage, and yield performance in real time. This would support timely decision-making and enable proactive agricultural management.

Chapter 12: References

- Dataset Source
- Python Documentation
- Power BI Documentation