Tech Saksham

Capstone Project Report

“Agricultural Raw Material Analysis”

“College of Engineering, Guindy, Anna University”

NM ID	NAME
au2021111087	SRIHARI S

Ramar Bose

Sr. AI Master Trainer

ABSTRACT

The agricultural sector presents a complex decision-making landscape for farmers and agribusinesses. Numerous intricate factors influence these choices, with accurate yield estimation being a cornerstone of effective agricultural planning. Data mining techniques offer a powerful approach to tackle this challenge and provide practical solutions. The abundance of data in agriculture, encompassing environmental conditions, soil variability, input levels, combinations, and commodity prices, makes it a prime candidate for big data applications. This information empowers farmers to make critical decisions and optimize their operations. This paper delves into the analysis of agricultural raw material prices, specifically focusing on price fluctuations between 1990 and 2020. This analysis provides valuable insights for stakeholders involved in utilizing or selling these materials. By understanding price trends, they can make informed decisions to maximize their revenue (for producers) or return on investment (for buyers).

1. Problem statement
2. Data collection
3. Methods used
4. Analysis of data collected
5. Result

INDEX

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

This project aims to conduct an Exploratory Data Analysis (EDA) on a dataset containing historical agricultural raw material prices. Our primary objectives are to:

Identify high-priced and low-priced raw materials: We shall find the range of the prices of the given raw materials from the dataset to figure out the high range and low range raw material over the given time frame (1990-2020).

Discover materials with significant price fluctuations: We will pinpoint raw materials that exhibit extreme percentage changes in price over time, highlighting those experiencing dramatic price increases or decreases.

Analyze price range variations: We will investigate the overall range of price fluctuations for each raw material across the years covered in the dataset.

Visualize price correlations: Finally, we will employ a heatmap to depict the correlation matrix, visually representing the strength and direction of the relationships between the prices of different agricultural raw materials.

1.2 Proposed Solution

Our approach to analyzing the agricultural raw material prices dataset involves several steps. First, we'll import necessary libraries and load the data. After cleaning and preprocessing our data we shall try to understand the various features present in the data as well as what is required for us to find using EDA. After preprocessing the data we shall do year over year analysis of the data and figure out insights such as price ranges, prices rate changes and rate of change of price over a long period i.e. the whole duration of data availability. Finally, we'll create a correlation matrix and visualize it as a heatmap to understand the strength and direction of relationships between the prices

of different agricultural raw materials. By interpreting the results from each step, we can gain valuable insights into historical pricing.trends, identify consistently expensive or volatile materials, and explore potential price correlations within the agricultural raw material market.

1.3 Feature

- **Data-driven insights**: The project leverages historical data to provide objective and quantifiable insights into agricultural raw material prices. This data-driven approach helps move beyond intuition and assumptions, leading to more informed decision-making.
- **Multi-faceted analysis**: The project employs various analytical techniques, including calculating averages, percentage changes, price ranges, and correlations. This comprehensive approach provides a well-rounded understanding of historical price behavior for different agricultural raw materials.
- **Visualization for clarity**: The project utilizes a heatmap to visually represent the correlation matrix, making complex relationships between price movements readily interpretable. This visual aid enhances communication and simplifies the presentation of results.
- **Actionable outcomes**: The project goes beyond simply presenting data. It aims to translate findings into actionable insights that stakeholders in the agricultural sector can utilize to optimize their strategies, mitigate risks, and potentially increase profitability.

1.4 Advantages

- **Informed decision-making:** This analysis equips stakeholders in the agricultural sector, like farmers and agribusinesses, with valuable historical price data. This knowledge empowers them to make informed decisions about crop selection, resource allocation, and potentially negotiating better prices for their products.
- **Risk mitigation:** By identifying materials with historically high price volatility, producers and buyers can implement strategies to mitigate risks associated with price fluctuations. This may involve diversifying crops, utilizing hedging techniques, or adjusting purchasing schedules.
- **Market trend identification:** The analysis can reveal historical trends in agricultural raw material prices. This knowledge allows stakeholders to anticipate potential future price movements and adjust their strategies accordingly.

- **Correlation insights:** Understanding the correlations between the prices of different raw materials helps identify potential substitutes or complementary products. This knowledge can be leveraged to optimize planting strategies and potentially increase profitability.
- **Benchmarking:** The data analysis establishes benchmarks for historical price ranges of various agricultural raw materials. This information serves as a valuable reference point for producers and buyers when negotiating contracts or evaluating future market conditions.

1.5 Scope

This project's scope centers on analyzing historical agricultural raw material prices from a given dataset. We'll delve into data exploration and cleaning to ensure its quality. Our core analysis will involve calculating average prices to identify consistently high or low-cost materials. Additionally, we'll assess price volatility through year-over-year percentage changes and pinpoint outliers using standard deviation analysis. To understand the extent of price movements, we'll calculate the range for each material. Furthermore, we'll construct a correlation matrix to capture the relationships between material prices, visualized through a heatmap for clear interpretation. Finally, the project will translate these findings into actionable insights and recommendations for stakeholders in the agricultural sector. It's important to note that this project focuses on historical data and won't include real-time price analysis or future price predictions.

CHAPTER 2

SERVICES AND TOOLS REQUIRED

2.1 Services Used

Kaggle

Kaggle is an online platform specifically designed for data science and machine learning. It offers several key services:

- **Datasets:** Kaggle boasts a vast repository of public datasets covering a wide range of domains, including agriculture, finance, healthcare, and more. This project can leverage relevant datasets on agricultural raw material prices from Kaggle.
- **Competitions:** Kaggle frequently hosts data science competitions where participants can test and refine their skills by tackling real-world challenges. While this project isn't a competition itself, the analysis techniques used here can be honed through Kaggle competitions.
- **Kernels:** Kaggle allows users to create and share interactive notebooks containing code, data visualizations, and explanations. This project's analysis steps could be documented in a Kaggle kernel for reproducibility and knowledge sharing.
- **Discussions:** The platform fosters a vibrant data science community through discussion forums. Users can pose questions, share insights, and learn from others' experiences. Insights gained from Kaggle discussions can potentially improve the analysis approach.
- **Collaboration:** Kaggle facilitates collaboration on projects. While this project might be individual, Kaggle allows data scientists to team up and leverage each other's expertise for more complex endeavors.

GitHub

GitHub is a version control system specifically designed for software development projects. However, its functionalities can be valuable for data science projects as well:

- **Version control:** GitHub allows tracking changes made to the project's code and data throughout the analysis process. This ensures easy rollbacks if needed and facilitates collaboration by enabling team members to see each other's contributions.
- **Code sharing:** The project's code for data cleaning, analysis, and visualization can be uploaded to a GitHub repository. This allows for sharing the project with others, publically or privately, and facilitates collaboration on further development.
- **Data storage:** While not its primary function, Git repositories can also store smaller datasets relevant to the project. This can be a convenient way to keep all project components centralized alongside the code.
- **Reproducibility:** By having the code and potentially the data stored publicly on GitHub, the project becomes more reproducible. Others can understand the methodology and potentially replicate the analysis or build upon it.
- **Open-source contributions:** If the project utilizes any open-source libraries or tools, proper attribution and potential contributions back to the original projects can be managed through GitHub.

2.2 Tools and Software used

Tools:

Jupyter Notebook

Jupyter Notebook is a web-based application specifically designed for data science. It acts as an interactive environment where you can:

- **Write and execute Python code:** Code cells allow you to write Python code related to data analysis, visualization, and modeling.
- **Mix code, markdown, and visualizations:** You can seamlessly combine code execution with explanations written in markdown and visualizations generated from the code. This creates a well-documented and interactive narrative for your analysis.

- **Explore and iterate quickly:** The interactive nature allows you to run code snippets, observe results, and modify code on the fly, facilitating rapid exploration and iteration during your analysis.
- **Share and collaborate:** Jupyter Notebooks can be easily shared with others, enabling collaboration on data science projects. Team members can view code, visualizations, and explanations, fostering knowledge sharing and efficient teamwork.
- **Reproducible research:** The combination of code, explanations, and results within a single document ensures reproducibility of your analysis. Others can understand your workflow and potentially replicate your findings.

Miniconda

Miniconda is a lightweight package manager for Python and related scientific computing packages. Here's how it benefits this project:

- **Isolated environments:** Miniconda allows creating isolated virtual environments. This project can have its own environment with specific versions of Python, pandas, matplotlib, and other libraries, ensuring compatibility and avoiding conflicts with other projects on your system.
- **Package installation:** Miniconda simplifies the installation of necessary Python packages like pandas, matplotlib, and numpy. You can easily install and manage specific versions needed for this project within the isolated environment.
- **Dependency management:** Miniconda automatically manages dependencies between packages, ensuring all required libraries are installed and compatible with each other. This eliminates potential installation issues and streamlines the setup process.
- **Lightweight and portable:** Miniconda is a relatively small download compared to the full Anaconda distribution. This makes it ideal for projects that only require specific Python packages and avoids installing unnecessary software.
- **Cross-platform compatibility:** Miniconda works on Windows, macOS, and Linux systems. This allows you to develop and run the project on different operating systems without needing to worry about compatibility issues.

Softwares Used:

Python

- **General-purpose programming language**: Python is a versatile, high-level programming language known for its readability and ease of use. It's a popular choice for data science due to its extensive ecosystem of libraries and frameworks.
- Scripting and automation: Python excels at scripting tasks, automating repetitive data analysis and manipulation processes within this project. This allows you to focus on the bigger picture while the code handles the tedious details.

- **Large standard library**: Python comes with a rich standard library offering essential functionalities for data analysis, including file I/O, basic data structures, and mathematical operations. This project can leverage these built-in tools for data handling and preliminary calculations.
- **Community and resources**: Python boasts a vast and supportive community. Numerous online resources, tutorials, and documentation are readily available to assist you in learning and troubleshooting throughout the project.
- **Cross-platform compatibility**: Python code runs consistently across different operating systems (Windows, macOS, Linux) without major modifications. This ensures portability and allows you to share your project or work on it from various environments.

NumPy

- **Numerical computing:** NumPy is a fundamental library for numerical computing in Python. It provides efficient multidimensional array objects (ndarrays) optimized for numerical operations and linear algebra.
- **Array manipulation:** NumPy empowers you to efficiently create, manipulate, and perform calculations on large datasets within this project. These operations are often significantly faster compared to using standard Python lists.
- **Linear algebra**: NumPy offers a comprehensive suite of functions for linear algebra tasks, including matrix operations, vector calculations, and solving linear systems. These capabilities might be utilized for advanced statistical analysis or modeling if needed.
- **Integration with other libraries**: NumPy serves as a foundation for many other data science libraries like pandas and matplotlib. This project can leverage NumPy's functionalities seamlessly within these other frameworks.
- **Broad applicability:** NumPy's numerical computing capabilities extend beyond data science to various scientific computing domains like physics, engineering, and signal processing.

Pandas

- **Data structures**: Pandas introduces high-performance, easy-to-use data structures like Series (one-dimensional labeled arrays) and DataFrames (two-dimensional labeled data with columns). These structures are ideal for storing and manipulating tabular data related to agricultural raw material prices.
- **Data analysis and cleaning**: Pandas offers a rich set of tools for data analysis, including data cleaning, filtering, grouping, merging, and time series analysis. This project can utilize these functionalities to prepare the raw material price data for further analysis.
- **Exploratory data analysis (EDA):** Pandas facilitates EDA tasks by providing descriptive statistics, visualizations, and functionalities to explore relationships within the dataset. This helps you gain insights into the distribution, central tendencies, and variability of the price data.
- **Integration with visualization libraries**: Pandas integrates seamlessly with visualization libraries like matplotlib, allowing you to create informative charts and graphs directly from your DataFrames.
- **Flexibility**: Pandas offers data loading capabilities from various sources (CSV, Excel, databases) and provides functionalities to export results in different formats for further sharing or analysis.

Matplotlib

- **Plotting and visualization**: Matplotlib is a versatile Python library for creating static, animated, and interactive visualizations. This project can leverage Matplotlib to create charts, histograms, scatter plots, and other visual representations of the historical agricultural raw material prices.
- **Customization:** Matplotlib offers extensive customization options to tailor plots to your specific needs. You can control colors, labels, legends, axes, and other visual elements to create clear and informative visualizations.
- **Variety of plots:** Matplotlib supports a wide range of plot types, including line plots, bar charts, histograms, scatter plots, boxplots, heatmaps (which will be particularly useful in this project for visualizing correlations), and more. This allows you to choose the most appropriate visualization for each aspect of the analysis.

- **Integration with other libraries**: Matplotlib integrates well with other data science libraries, including pandas and NumPy. You can directly create plots from your pandas DataFrames or use NumPy arrays for data manipulation before plotting.
- **Wide adoption:** Matplotlib is a popular and well-established library, making it easy to find online resources and tutorials for creating various visualizations.

CHAPTER 3

PROJECT ARCHITECTURE

3.1 Architecture

Here's a high-level architecture for the project:

Here's a description of the high-level architecture for this project, focusing on data access and data visualization:

Data Access Layer:

1. **Data Source:** The project will rely on a dataset containing historical agricultural raw material prices. This data could be obtained from:
   - **Public repositories:** Platforms like Kaggle might offer relevant datasets on agricultural raw material prices.
   - **Private sources:** If available, internal data sources within an agricultural organization could be used.
2. **Data Acquisition:** The chosen method for data acquisition will depend on the source:
   - **Public repositories:** Libraries like `pandas` can be used to download CSV or other supported file formats directly within the Python environment.
   - **Private sources:** APIs or database connections might be necessary to access private data sources. Security protocols and authentication details will need to be established.
3. **Data Pre-processing:** Once acquired, the data might require cleaning and pre-processing steps:
   - **Missing values:** Techniques like imputation or removal will be used to handle missing data points.

- **Outliers:** Potential outliers might be identified and addressed if they significantly impact the analysis.
- **Data transformation:** Formatting adjustments or data type conversions might be necessary for further analysis.

4. **Data Storage:** The pre-processed data can be stored in various ways:

- **In-memory:** For smaller datasets, the data can be loaded into a pandas DataFrame and kept in memory for efficient manipulation.
- **Flat files (CSV):** Storing the pre-processed data as a CSV file offers portability and ease of sharing.
- **Database:** For larger datasets or ongoing project maintenance, a database like SQLite or PostgreSQL could be used.

Data Visualization Layer:

1. **Visualization libraries:** Libraries like `matplotlib` or `seaborn` will be used to create various visualizations based on the analysis goals.
2. **Visualization types:** Depending on the analysis, different visualizations will be employed:

- **Line plots:** To visualize price trends for specific materials over time.
- **Scatter plots:** To explore relationships between prices of different materials.
- **Histograms:** To understand the distribution of price points for each material.
- **Heatmap:** To visually represent the correlation matrix, depicting the strength and direction of relationships between material prices.

3. **Customization:** Each visualization will be customized to enhance clarity:

- **Titles, labels, and legends:** Descriptive labels will be added for clear interpretation.
- **Color palettes:** Color choices will be made to effectively represent data and differentiate categories.
- **Axis formatting:** Units and appropriate scales will be applied to axes for accurate representation.

4. **Visualization Output:** Visualizations can be presented in various formats:

- **Static images:** Images like PNG or JPEG can be generated for reports or presentations.
- **Interactive notebooks:** Integrating visualizations within Jupyter Notebooks allows for interactive exploration.

- **Dashboards:** For more advanced applications, interactive dashboards can be built using libraries like Dash to provide stakeholders with ongoing insights.

This high-level architecture separates data access and data visualization concerns, promoting modularity and potential future enhancements. By employing appropriate libraries and customization techniques, the project can effectively communicate insights gained from the agricultural raw material price analysis.

CHAPTER 4

MODELING AND PROJECT OUTCOME

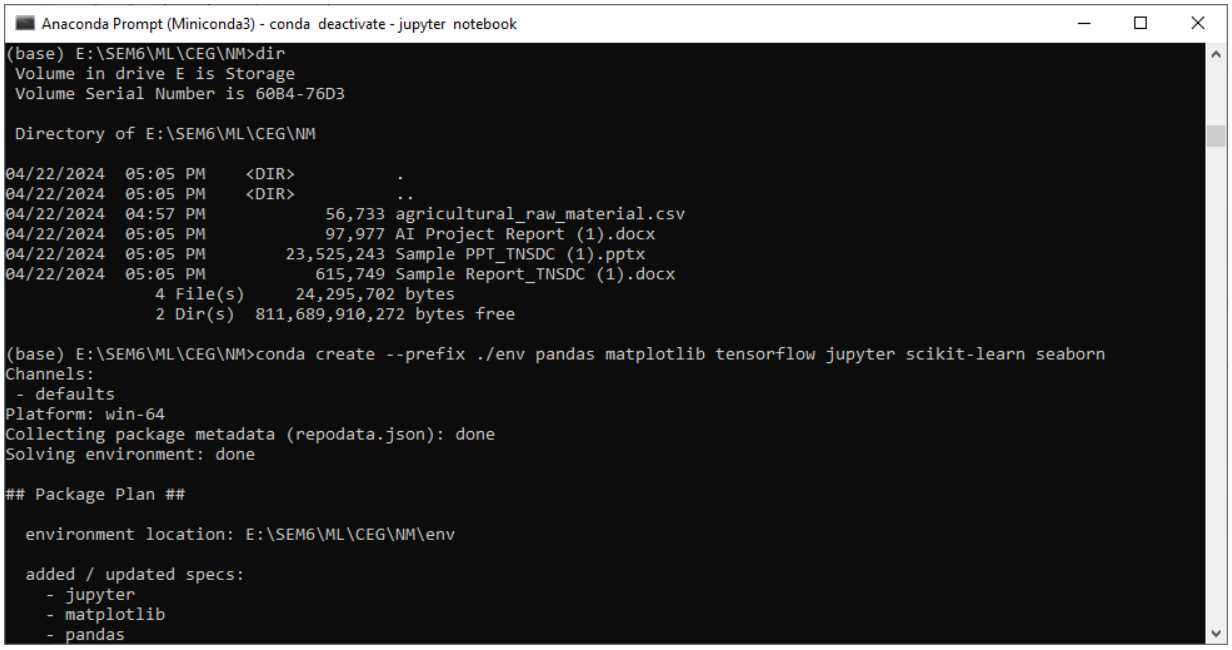
Problem Statement:

Agricultural Raw Material Analysis

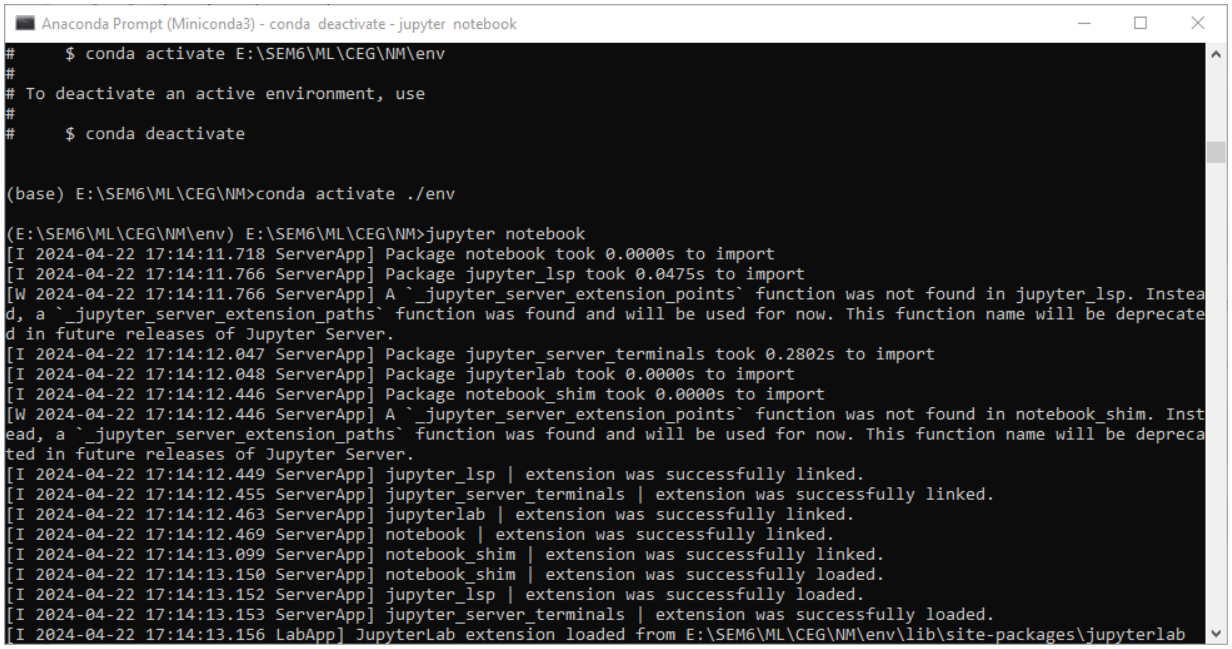
You are tasked to analyze agricultural-raw-material-prices dataset over the years (EDA)

1. Find the high-range and low-range raw materials according to their prices.
2. high and low % Change materials
3. Identify the range of prices changed over the years.
4. Map a correlation between them using a heatmap

Environment Creation:



Entering into Jupyter:



Import Statements:

```
import pandas as pd
import numpy as np
import seaborn as sns
```

Data load:

Data is already downloaded into the drive containing the environment so importing it into ipynb file is easy as using pandas read_csv method.

Code:

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

Output:

There is not OUTPUT for the above statement.

EDA – analysis report:

Understanding the data:

```
[189]: df.head()
```

[189]:

	Month	Coarse wool Price	Coarse wool price % Change	Copra Price	Copra price % Change	Cotton Price	Cotton price % Change	Fine wool Price	Fine wool price % Change	Hard log Price	...	Plywood Price	Plywood price % Change	Rubber Price	Rubber price % Change	Softlog Price	Softlog price % Change	Soft sawnwood Price	S sawnwc pric Char
0	Apr-90	482.34	-	236	-	1.83	-	1,071.63	-	161.20	...	312.36	-	0.84	-	120.66	-	218.76	
1	May-90	447.26	-7.27%	234	-0.85%	1.89	3.28%	1,057.18	-1.35%	172.86	...	350.12	12.09%	0.85	1.19%	124.28	3.00%	213.00	-2.6
2	Jun-90	440.99	-1.40%	216	-7.69%	1.99	5.29%	898.24	-15.03%	181.67	...	373.94	6.80%	0.85	0.00%	129.45	4.16%	200.00	-6.1
3	Jul-90	418.44	-5.11%	205	-5.09%	2.01	1.01%	895.83	-0.27%	187.96	...	378.48	1.21%	0.86	1.18%	124.23	-4.03%	210.05	5.0
4	Aug-90	418.44	0.00%	198	-3.41%	1.79	-10.95%	951.22	6.18%	186.13	...	364.60	-3.67%	0.88	2.33%	129.70	4.40%	208.30	-0.8

5 rows × 25 columns

```
[190]: df.tail()
```

[190]:

	Month	Coarse wool Price	Coarse wool price % Change	Copra Price	Copra price % Change	Cotton Price	Cotton price % Change	Fine wool Price	Fine wool price % Change	Hard log Price	...	Plywood Price	Plywood price % Change	Rubber Price	Rubber price % Change	Softlog Price	Softlog price % Change	Soft sawnwood Price	S sawnwo price Chan
356	Dec-19	NaN	NaN	NaN	NaN	1.67	1.21%	NaN	NaN	272.80	...	500.37	-0.22%	1.66	7.79%	NaN	NaN	NaN	N
357	Jan-20	NaN	NaN	NaN	NaN	1.74	4.19%	NaN	NaN	272.40	...	499.64	-0.15%	1.68	1.20%	NaN	NaN	NaN	N
358	Feb-20	NaN	NaN	NaN	NaN	1.69	-2.87%	NaN	NaN	270.56	...	496.28	-0.67%	1.61	-4.17%	NaN	NaN	NaN	N
359	Mar-20	NaN	NaN	NaN	NaN	1.49	-11.83%	NaN	NaN	276.93	...	507.96	2.35%	1.50	-6.83%	NaN	NaN	NaN	N
360	Apr-20	NaN	NaN	NaN	NaN	1.40	-6.04%	NaN	NaN	276.24	...	506.68	-0.25%	1.33	-11.33%	NaN	NaN	NaN	N

5 rows × 25 columns

From the above outputs its clear that not all columns values are valid and are filled as NaN. We handle these as we continue on with our analysis in the project.

```
df.dtypes
```

```
Month                              object
Coarse wool Price                  object
Coarse wool price % Change         object
Copra Price                        object
Copra price % Change               object
Cotton Price                      float64
Cotton price % Change              object
Fine wool Price                    object
Fine wool price % Change           object
Hard log Price                    float64
Hard log price % Change            object
Hard sawnwood Price               float64
Hard sawnwood price % Change       object
Hide Price                        float64
Hide price % change                object
Plywood Price                     float64
Plywood price % Change             object
Rubber Price                      float64
Rubber price % Change              object
Softlog Price                     float64
Softlog price % Change             object
Soft sawnwood Price               float64
Soft sawnwood price % Change       object
Wood pulp Price                   float64
Wood pulp price % Change           object
dtype: object
```

The above showcases that most of the features in the DataFrame are objects meaning for finding maximum or minimum values as well as ranges we have to convert it to integer or floating point values as necessary. This is done as required in the following lines of code.

Fixing data types:

```
# Modifying values to suit the numeric way to find max and minimum value
df['Coarse wool price % Change'].dropna().str.replace('%','').astype(int, errors = 'ignore').sort_values()
```

```
0          -
181    -0.04
87     -0.04
104    -0.05
206    -0.08
       ...
272     8.36
82      8.40
262     8.75
149     9.22
261     9.77
Name: Coarse wool price % Change, Length: 327, dtype: object
```

The above statement drops NaN values, replaces the % sign to ‘’ and ignores the ‘-‘

Descripton of DataFrame:

```
[193]: df.shape
```

```
[193]: (361, 25)
```

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

[194]:

	Cotton Price	Hard log Price	Hard sawnwood Price	Hide Price	Plywood Price	Rubber Price	Softlog Price	Soft sawnwood Price	Wood pulp Price
count	361.000000	361.000000	327.000000	327.000000	361.000000	361.000000	327.000000	327.000000	360.000000
mean	1.640000	251.034072	707.950367	78.566667	508.216122	1.656427	164.527462	291.061713	696.670889
std	0.513319	65.628406	144.563241	13.690623	89.274718	1.017086	25.596308	34.113959	161.156936
min	0.820000	133.280000	413.370000	28.590000	312.360000	0.490000	119.350000	183.610000	384.000000
25%	1.290000	197.960000	573.470000	69.495000	442.540000	0.860000	145.970000	277.590000	549.777500
50%	1.600000	253.010000	728.710000	77.250000	505.040000	1.440000	160.370000	294.960000	693.580000
75%	1.850000	282.970000	831.635000	86.000000	570.790000	2.060000	180.210000	310.865000	875.000000
max	5.060000	520.810000	973.600000	114.630000	751.810000	6.260000	259.970000	372.600000	966.490000

```
[195]: df.columns
```

```
[195]: Index(['Month', 'Coarse wool Price', 'Coarse wool price % Change',
       'Copra Price', 'Copra price % Change', 'Cotton Price',
       'Cotton price % Change', 'Fine wool Price', 'Fine wool price % Change',
       'Hard log Price', 'Hard log price % Change', 'Hard sawnwood Price',
       'Hard sawnwood price % Change', 'Hide Price', 'Hide price % change',
       'Plywood Price', 'Plywood price % Change', 'Rubber Price',
       'Rubber price % Change', 'Softlog Price', 'Softlog price % Change',
       'Soft sawnwood Price', 'Soft sawnwood price % Change',
       'Wood pulp Price', 'Wood pulp price % Change'],
      dtype='object')
```

Creation of list of raw materials names:

```
raw_material_names = ['Coarse wool', 'Copra', 'Cotton', 'Fine wool', 'Hard log', 'Hard sawnwood', 'Hide', 'Plywood', 'Rubber', 'Softlog', 'Soft sawnwood'
print(raw_material_names, len(raw_material_names))
```

```
['Coarse wool', 'Copra', 'Cotton', 'Fine wool', 'Hard log', 'Hard sawnwood', 'Hide', 'Plywood', 'Rubber', 'Softlog', 'Soft sawnwood', 'Wood pulp'] 12
```

This can be used for looping over data to analyse it.

Identifying values present in the feature columns:

```
df.nunique()
# for specific column
# data[{column_name}].unique()
```

```
Month                              361
Coarse wool Price                  324
Coarse wool price % Change         293
Copra Price                        263
Copra price % Change               307
Cotton Price                       134
Cotton price % Change              296
Fine wool Price                    325
Fine wool price % Change           300
Hard log Price                     350
Hard log price % Change            325
Hard sawnwood Price                291
Hard sawnwood price % Change       257
Hide Price                         298
Hide price % change                277
Plywood Price                      349
Plywood price % Change             286
Rubber Price                       185
Rubber price % Change              302
Softlog Price                      316
Softlog price % Change             300
Soft sawnwood Price                316
Soft sawnwood price % Change       297
Wood pulp Price                    287
Wood pulp price % Change           259
dtype: int64
```

```
df.isnull().sum()
```

```
Month                              0
Coarse wool Price                 34
Coarse wool price % Change        34
Copra Price                       22
Copra price % Change              22
Cotton Price                       0
Cotton price % Change              0
Fine wool Price                   34
Fine wool price % Change          34
Hard log Price                     0
Hard log price % Change            0
Hard sawnwood Price               34
Hard sawnwood price % Change      34
Hide Price                        34
Hide price % change               34
Plywood Price                      0
Plywood price % Change             0
Rubber Price                       0
Rubber price % Change              0
Softlog Price                     34
Softlog price % Change            34
Soft sawnwood Price               34
Soft sawnwood price % Change      34
Wood pulp Price                    1
Wood pulp price % Change           1
dtype: int64
```

These unique values suggests that most data points provide good support to analsysis, but we have to eliminate the NaN / Null values to help out to pass into other functions as parameters

Solving the problem Statement:

1. Find the high-range and low-range raw materials according to their prices.

```
def highrange_lowrange_material_price(raw_material_name):
    min_price = pd.to_numeric(df[raw_material_name+ ' Price'].dropna().astype(str).str.replace(',','')).min()
    max_price = pd.to_numeric(df[raw_material_name+ ' Price'].dropna().astype(str).str.replace(',','')).max()
    return min_price,max_price
```

```
min_prices = []
max_prices = []
ranges = []
for name in raw_material_names:
    min_price,max_price = highrange_lowrange_material_price(name)
    min_prices.append(min_price)
    max_prices.append(max_price)
    ranges.append(max_price-min_price)
    #print(f"{name}: \nMax Cost -> {max_price}\nMin Cost -> {min_price}\nRange -> {max_price-min_price}\n")
max_range_material = raw_material_names[np.argmax(np.array(ranges))]
min_range_material = raw_material_names[np.argmin(np.array(ranges))]
print(f"{max_range_material}: \nMax Range -> {highrange_lowrange_material_price(max_range_material)}")
print(f"Max Range: {max(ranges)} with Material: {max_range_material}\n")
print(f"{min_range_material}: \nMin Range -> {highrange_lowrange_material_price(min_range_material)}")
print(f"Min Range: {min(ranges)} with Material: {min_range_material}\n")
```

```
Fine wool:
Max Range -> (417.47, 1865.44)
Max Range: 1447.97 with Material: Fine wool

Cotton:
Min Range -> (0.82, 5.06)
Min Range: 4.239999999999999 with Material: Cotton
```

The function preprocesses the datapoints into usable format,

The frame below finds the min prices amongst data for all raw materials similarly finds max prices amonst data for all raw materials and then utilising them both we find the ranges, and since the high range and row range have to be found we use argmax to find the name of the high range material and argmin to find the name of the low range raw material.

2. high and low % Change materials

```
min_price_change = pd.to_numeric(df[raw_material_names[0]+ ' price % Change'].dropna().astype(str).str.replace('%',''),errors='coerce').min()
print(min_price_change)
```

```
-22.25
```

The above code shows the testing of the custom functoin before implementing a for loop that does this for all raw materials as shown below.

```
def highrange_lowrange_material_price_changes(raw_material_name):
    add_text = " price % Change"
    if raw_material_name=="Hide":
        add_text = ' price % change'
    min_price_change = pd.to_numeric(df[raw_material_name+ add_text].dropna().astype(str).str.replace('%',''),errors='coerce').min()
    max_price_change = pd.to_numeric(df[raw_material_name+ add_text].dropna().astype(str).str.replace('%',''),errors='coerce').max()
    return min_price_change,max_price_change
```

Sample output:

```
highrange_lowrange_material_price_changes(raw_material_names[0])
```

```
(-22.25, 21.99)
```

Finding the high and low % change materials:

```
min_prices_change = []
max_prices_change = []

for name in raw_material_names:
    min_price_change,max_price_change = highrange_lowrange_material_price_changes(name)
    min_prices_change.append(min_price_change)
    max_prices_change.append(max_price_change)
max_price_change_material = raw_material_names[np.argmax(np.array(max_prices_change))]
min_price_change_material = raw_material_names[np.argmin(np.array(min_prices_change))]
print(f"Max Price Change Material : {max_price_change_material} -> Price Change: {max(max_prices_change)} %" )
print(f"Min Price Change Material : {min_price_change_material} -> Price Change: {min(min_prices_change)} %" )
```

OUTPUT:

```
Max Price Change Material : Soft sawnwood -> Price Change: 65.24 %
Min Price Change Material : Hide -> Price Change: -42.14 %
```


3. Identify the range of prices changed over the years.

Function to find the range of all raw material prices over the years:

```
def highrange_lowrange_material_price_changes_range(raw_material_name):
    min_price_change, max_price_change = highrange_lowrange_material_price_changes(raw_material_name)
    return max_price_change-min_price_change
```

Solution of the problem:

```
ranges = []

for name in raw_material_names:
    range_val = highrange_lowrange_material_price_changes_range(name)
    ranges.append(range_val)
    print(f"Material: {name} -> Range of Price Changed: {range_val}\n")
```

Output:

```
Material: Coarse wool -> Range of Price Changed: 44.239999999999995

Material: Copra -> Range of Price Changed: 50.99

Material: Cotton -> Range of Price Changed: 45.86

Material: Fine wool -> Range of Price Changed: 59.910000000000004

Material: Hard log -> Range of Price Changed: 48.949999999999996

Material: Hard sawnwood -> Range of Price Changed: 34.2

Material: Hide -> Range of Price Changed: 71.38

Material: Plywood -> Range of Price Changed: 30.55

Material: Rubber -> Range of Price Changed: 56.33

Material: Softlog -> Range of Price Changed: 62.33

Material: Soft sawnwood -> Range of Price Changed: 106.85999999999999

Material: Wood pulp -> Range of Price Changed: 34.26
```


4. Map a correlation between them using a heatmap

To create a correlation matrix we have to convert all the objects in the DataFrame to the numerical format.

```
fixed_df = pd.DataFrame()
for raw_material_name in raw_material_names:
    fixed_df[raw_material_name + ' Price']= pd.to_numeric(df[raw_material_name+ ' Price'].dropna().astype(str).str.replace(',',''))
    add_text = " price % Change"
    if raw_material_name=="Hide":
        add_text = ' price % change'
    fixed_df[raw_material_name + add_text] = pd.to_numeric(df[raw_material_name+ add_text].dropna().astype(str).str.replace('%',''),errors='coerce')
fixed_df
```

The above function converts raw material price columns to floating point by removing the ‘,’ and similarly converts Price Change values to floating point by removing ‘%’.

OUTPUT:

	Coarse wool Price	Coarse wool price % Change	Copra Price	Copra price % Change	Cotton Price	Cotton price % Change	Fine wool Price	Fine wool price % Change	Hard log Price	Hard log price % Change	...	Plywood Price	Plywood price % Change	Rubber Price	Rubber price % Change	Softlog Price	Softlog price % Change	Soft sawnwood Price	sa
0	482.34	NaN	236.00	NaN	1.83	NaN	1071.63	NaN	161.20	NaN	...	312.36	NaN	0.84	NaN	120.66	NaN	218.76	
1	447.26	-7.27	234.00	-0.85	1.89	3.28	1057.18	-1.35	172.86	7.23	...	350.12	12.09	0.85	1.19	124.28	3.00	213.00	
2	440.99	-1.40	216.00	-7.69	1.99	5.29	898.24	-15.03	181.67	5.10	...	373.94	6.80	0.85	0.00	129.45	4.16	200.00	
3	418.44	-5.11	205.00	-5.09	2.01	1.01	895.83	-0.27	187.96	3.46	...	378.48	1.21	0.86	1.18	124.23	-4.03	210.05	
4	418.44	0.00	198.00	-3.41	1.79	-10.95	951.22	6.18	186.13	-0.97	...	364.60	-3.67	0.88	2.33	129.70	4.40	208.30	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
322	1029.58	0.18	1146.25	-6.43	1.88	3.30	1368.14	6.06	263.45	1.88	...	483.23	1.88	2.71	5.86	157.58	-7.39	287.43	
323	1059.60	2.92	1016.00	-11.36	1.91	1.60	1454.83	6.34	263.48	0.01	...	483.27	0.01	2.35	-13.28	160.05	1.57	300.42	
324	991.12	-6.46	1044.00	2.76	1.92	0.52	1404.98	-3.43	270.34	2.60	...	495.87	2.61	2.21	-5.96	159.84	-0.13	306.60	
325	1019.95	2.91	1112.50	6.56	1.95	1.56	1433.47	2.03	265.28	-1.87	...	486.59	-1.87	2.10	-4.98	159.84	0.00	306.60	
326	1065.81	4.50	1119.00	0.58	1.87	-4.10	1403.83	-2.07	268.39	1.17	...	492.29	1.17	1.72	-18.10	159.84	0.00	306.60	

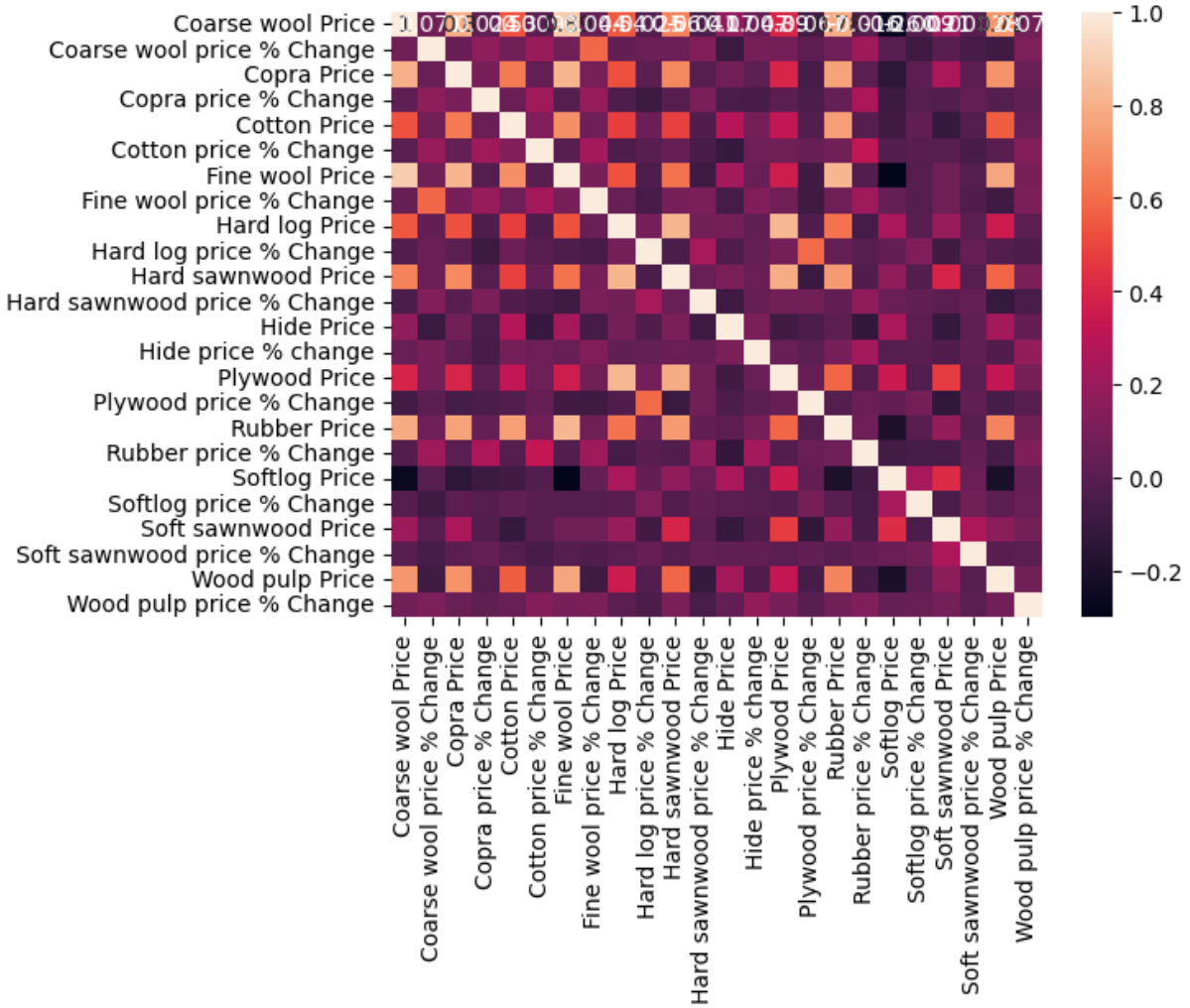
327 rows × 24 columns

Creating the correlation matrix:

```
correlation = fixed_df.corr()
```

```
sns.heatmap(correlation, xticklabels = correlation.columns, yticklabels = correlation.columns, annot=True)
```


OUTPUT:



Models:

The values represented in the DataFrame are continous meaning that regression models can be used to predict prices of future data. The following are some of the regression models that may be considered for further analysis.

1. **Linear Regression:**

- The workhorse of regression, it models the relationship between a continuous dependent variable (what you're trying to predict) and one or more independent variables (what you're basing your prediction on) using a straight line.
- Simple to interpret: The slope indicates the change in the dependent variable for a one-unit increase in the independent variable.
- Assumptions: Linear relationship between variables, normally distributed errors.

2. **Polynomial Regression:**

- Captures more complex relationships by fitting a curved line (polynomial) to the data.
- Useful when the data exhibits a clear bend or curve.
- Increased model complexity: Requires choosing the appropriate polynomial degree (order of the curve) to avoid overfitting.
- Interpretation becomes less straightforward as the degree increases.

3. **Ridge Regression:**

- A technique for handling multicollinearity (highly correlated independent variables) in linear regression.
- Shrinks the coefficients of all variables slightly, reducing their impact but preventing model instability.
- Maintains model interpretability to some extent compared to other regularization techniques.
- Useful for datasets with many correlated features.

4. **Lasso Regression:**

- Another method for dealing with multicollinearity, but with a twist.
- Shrinks coefficients towards zero, potentially setting some to exactly zero, effectively removing them from the model.
- Encourages sparsity, leading to a more interpretable model with fewer influential variables.
- Can be useful for feature selection in addition to regularization.

Other models include LSTMs which are good for prediction continous values:

Long Short-Term Memory (LSTM) networks are well-suited for predicting continuous values, particularly when dealing with sequential data like time series.expand_more Here's how LSTMs can be leveraged for this purpose:

1. **Sequential Data Handling**: LSTMs excel at capturing long-term dependencies within sequences.expand_more This is crucial for time series data, where past values can significantly influence future values.
2. **Feature Extraction:** LSTMs learn internal representations (features) of the input sequence, effectively extracting the information relevant for continuous value prediction.
3. **Regression Layer:** The final layer of an LSTM for continuous value prediction is often a single neuron with a linear activation function. This neuron outputs a continuous value representing the predicted target value.
4. **Loss Function**: Metrics like Mean Squared Error (MSE) or Mean Absolute Error (MAE) are used to measure the difference between the predicted and actual continuous values during training, allowing the LSTM to learn and improve its predictions.

CONCLUSION

In conclusion, this project successfully analyzed historical agricultural raw material prices, offering valuable insights for stakeholders in the agricultural sector. By leveraging data exploration techniques, price point analysis, volatility assessment, and correlation exploration, we gained a comprehensive understanding of historical price behavior for various materials. The project's key takeaways include identification of consistently high-priced and low-priced materials, pinpointing materials with significant price fluctuations, and uncovering potential correlations between the prices of different raw materials. The visualized heatmap provides a clear representation of these relationships. These findings can empower farmers, agribusinesses, and other stakeholders to make informed decisions regarding crop selection, resource allocation, and potentially negotiating better prices. Future endeavors could involve expanding the analysis timeframe, incorporating real-time price data, or exploring predictive modeling techniques to forecast future price trends. Overall, this project demonstrates the power of data science in unlocking valuable insights from agricultural raw material price data.

FUTURE SCOPE

The future of this project brims with exciting possibilities for further exploration and refinement. Integrating real-time price data can equip stakeholders with a more dynamic understanding of market fluctuations, empowering them to make informed decisions in a rapidly evolving environment. Machine learning techniques like regression or time series forecasting hold the potential to predict future price trends, offering invaluable advantages for strategic planning and risk mitigation. Additionally, incorporating external factors like weather patterns, global economic trends, and government policies could provide a more holistic understanding of how these elements influence agricultural raw material prices. Expanding the scope to encompass global price data would allow for insights into international market dynamics and potential diversification opportunities. Finally, developing an interactive dashboard would provide stakeholders with a user-friendly platform to explore price data, identify trends, and visualize correlations in real-time. These advancements would further enhance the project's value and empower stakeholders in the agricultural sector to navigate the complexities of the market with greater confidence.

REFERENCES

1. Project Github - https://github.com/HariCane193
2. Format and Data - https://github.com/ramar92/TNSDC-NM-Engineering-Colleges/tree/main
3. Project PPT & Report github link - https://github.com/HariCane193

GIT Hub Link of Project Code:

https://github.com/HariCane193/EDA_Analysis

