Agricultural Raw Material Analysis (ML - Classification & regression)

15s

```
!pip install jovian opendatasets --upgrade --quiet
```

output

```
68.6/68.6 kB 1.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
  Building wheel for uuid (setup.py) ... done
```

---

[2]

0s

```
# Downloading Data from Kaggle
dataset_url = 'https://www.kaggle.com/kianwee/agricultural-raw-material-prices-19902020?select=agricultural_raw_material.csv'
```

---

[3]

1m

```
# We need to enter our API generated credentials upon prompt.
import opendatasets as od
od.download(dataset_url)
```

output

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: danala26
Your Kaggle Key: ··········
Downloading agricultural-raw-material-prices-19902020.zip to ./agricultural-raw-material-prices-19902020
100%|■■■■■■■■■■| 22.8k/22.8k [00:00<00:00, 9.90MB/s]
```

---

[4]

0s

```
# Importing csv file
data_dir = './agricultural-raw-material-prices-19902020'
```

---

[5]

0s

```
import os
os.listdir(data_dir)
```

output

```
['agricultural_raw_material.csv']
```

---

[7]

0s

```
project_name = "analysis-agriculture-raw-mateial-prices" # Give project a name and use lowercase letters and hyphens only
```

---

[8]

6s

```
!pip install jovian --upgrade -q
```

[9]

0s

```
import jovian
```

[10]

0s

```
jovian.commit(project=project_name)
```

output

```
[jovian] Detected Colab notebook...
[jovian] jovian.commit() is no longer required on Google Colab. If you ran this notebook from Jovian,
then just save this file in Colab using Ctrl+S/Cmd+S and it will be updated on Jovian.
Also, you can also delete this cell, it's no longer necessary.
```

[11]

2s

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
sns.set(rc={'figure.figsize':(11, 4)})
# reading the csv data
agri_price_df = pd.read_csv('./agricultural-raw-material-prices-
19902020/agricultural_raw_material.csv')
```

[12]

0s

```
# columns list
agri_price_df.info()
```

output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 361 entries, 0 to 360
Data columns (total 25 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Month                     361 non-null    object
 1   Coarse wool Price         327 non-null    object
 2   Coarse wool price % Change  327 non-null    object
 3   Copra Price               339 non-null    object
 4   Copra price % Change      339 non-null    object
 5   Cotton Price              361 non-null    float64
 6   Cotton price % Change     361 non-null    object
 7   Fine wool Price           327 non-null    object
 8   Fine wool price % Change  327 non-null    object
 9   Hard log Price            361 non-null    float64
 10  Hard log price % Change   361 non-null    object
 11  Hard sawnwood Price       327 non-null    float64
 12  Hard sawnwood price % Change  327 non-null    object
 13  Hide Price                327 non-null    float64
 14  Hide price % change       327 non-null    object
 15  Plywood Price             361 non-null    float64
 16  Plywood price % Change    361 non-null    object
 17  Rubber Price              361 non-null    float64
 18  Rubber price % Change     361 non-null    object
 19  Softlog Price             327 non-null    float64
```

```
 20  Softlog price % Change       327 non-null    object
 21  Soft sawnwood Price          327 non-null    float64
 22  Soft sawnwood price % Change  327 non-null    object
 23  Wood pulp Price              360 non-null    float64
 24  Wood pulp price % Change     360 non-null    object
dtypes: float64(9), object(16)
memory usage: 70.6+ KB
```

---

[13]

0s

```
# It is advisable to make a copy of your dataset, so that we can return to the original data in case we mad
e some wrong computation in our data.
agri_price_df_copy = agri_price_df.copy()
```

---

[14]

0s

```
# Replacing %, "," and "-"
agri_price_df = agri_price_df.replace('%', '', regex=True)
agri_price_df = agri_price_df.replace(',', '', regex=True)
agri_price_df = agri_price_df.replace('-', '', regex=True)
agri_price_df = agri_price_df.replace('', np.nan)
agri_price_df = agri_price_df.replace('MAY90', np.nan)
```

---

[15]

1s

```
# Dropping rows with NaN values
agri_price_df = agri_price_df.dropna()
```

---

[16]

0s

```
# Check to see if all NaN values are resolved
agri_price_df.isnull().sum()
```

output

```
Month                        0
Coarse wool Price            0
Coarse wool price % Change   0
Copra Price                  0
Copra price % Change         0
Cotton Price                 0
Cotton price % Change        0
Fine wool Price              0
Fine wool price % Change     0
Hard log Price               0
Hard log price % Change      0
Hard sawnwood Price          0
Hard sawnwood price % Change 0
Hide Price                   0
Hide price % change          0
Plywood Price                0
Plywood price % Change       0
Rubber Price                 0
Rubber price % Change        0
Softlog Price                0
Softlog price % Change       0
Soft sawnwood Price          0
Soft sawnwood price % Change 0
```

```
Wood pulp Price              0
Wood pulp price % Change     0
dtype: int64
```

---

[17]

0s

```
# Converting data type to float
agri = ["Coarse wool Price", "Coarse wool price % Change", "Copra Price", "Copra price % Change", "Cott
on price % Change","Fine wool Price", "Fine wool price % Change", "Hard log price % Change", "Hard sa
wnwood price % Change", "Hide price % change", "Plywood price % Change", "Rubber price % Change",
"Softlog price % Change", "Soft sawnwood price % Change", "Wood pulp price % Change"]
```

---

[18]

0s

```
agri_price_df[agri] = agri_price_df[agri].astype("float")
# check the data type now..It is changed.
agri_price_df.dtypes
```

output

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

---

[27]

3s

```
axes = agri_price_df[["Plywood Price","Hard log Price","Plywood price % Change","Hard log price % Cha
nge"]].plot(figsize=(11, 9), subplots=True, linewidth=1)
axes = agri_price_df[["Coarse wool Price","Fine wool Price","Coarse wool price % Change","Fine wool pri
ce % Change"]].plot(figsize=(11, 9),subplots=True,linewidth=1)
```

output

---

[28]

1s

```
# Filtering the selected raw materials columns from the main data.

select_df=agri_price_df.filter(['Month','Plywood Price','Hard log Price','Plywood price % Change','Hard log price % Change'])
# Filtering data for the year 2009 and 2010

select_df_year= select_df.loc['Jan09':'Dec10']
axes = select_df_year[["Plywood Price","Hard log Price",]].plot(figsize=(11, 9),subplots=True,linewidth=1)
```

output

---

[29]

2s

```
ax = select_df_year[['Plywood Price','Hard log Price']].plot(kind='bar', title ="Price Comparison", figsize=(18, 12), legend=True, fontsize=12)
ax.set_xlabel("Month", fontsize=12)
ax.set_ylabel("Price", fontsize=12)
ax.set_xticklabels(select_df_year.index.format(),rotation=45)

def add_value_labels(ax, spacing=5):
    # For each bar: Place a label
    for rect in ax.patches:
        # Get X and Y placement of label from rect.
        y_value = rect.get_height()
        x_value = rect.get_x() + rect.get_width() / 2

        # Number of points between bar and label. Change to your liking.
        space = spacing
        # Vertical alignment for positive values
        va = 'bottom'

        # If value of bar is negative: Place label below bar
        if y_value < 0:
            # Invert space to place label below
            space *= -1
            # Vertically align label at top
            va = 'top'

        # Use Y value as label and format number with one decimal place
        label = "{:.1f}".format(y_value)

        # Create annotation
        ax.annotate(
            label,                 # Use `label` as label
            (x_value, y_value),       # Place label at end of the bar
            xytext=(5, space),        # Vertically shift label by `space`
            textcoords="offset points", # Interpret `xytext` as offset in points
            ha='center',              # Horizontally center label
            va=va)                  # Vertically align label differently for
```

```
                    # positive and negative values.


 # Call the function above.
 add_value_labels(ax)
 plt.show()

 # This code is in response to a user question on Stackoverflow. Please refer section Refrences for the link
  to the actual question and code.
```
output

---

[30]

1s

```
 axes = select_df_year[["Plywood price % Change","Hard log price % Change",]].plot(figsize=(11, 9),subpl
 ots=True,linewidth=1)
```
output
addCode
addText