Assignment 1

Aim:- The aim of this assignment is to explore various operations on a dataset and perform data analysis using machine learning techniques.

Theory:- Data analysis and machine learning are necessary to understand the different types of datasets and their properties. Data preprocessing techniques such as cleaning, transforming, and scaling can be used to prepare the data for analysis. Exploratory data analysis techniques such as data visualization and statistical analysis can then be used to gain insights from the data. Machine learning algorithms such as classification, regression, clustering, and dimensionality reduction can be applied to the dataset to make predictions and identify patterns.

Case study:- The dataset used is named ‘apy\_1’. It contains data regarding the district wise production of various crops across India from the year 1997 to 2015.

The dataset contains 73,827 rows and 8 columns. The columns contain the name of the state, name of the district, year for which the data is available, season, type of crop, area covered by the crop and the produce obtained.

**import** pandas **as** pd

**import** numpy **as** np

temp = pd.read\_csv("apy\_1.csv") print(temp.index)

temp.shape

RangeIndex(start=0, stop=73827, step=1) (73827, 8)

print(temp.columns) print(temp.shape) temp.info()

Index(['Unnamed: 0', 'State\_Name', 'District\_Name', 'Crop\_Year', 'Season',

'Crop', 'Area', 'Production'], dtype='object')

(73827, 8)

<class 'pandas.core.frame.DataFrame'> RangeIndex: 73827 entries, 0 to 73826 Data columns (total 8 columns):

# Column Non-Null Count Dtype

1. Unnamed: 0 73827 non-null int64
2. State\_Name 73827 non-null object
3. District\_Name 73827 non-null object
4. Crop\_Year 73827 non-null int64
5. Season 73827 non-null object
6. Crop 73827 non-null object
7. Area 73827 non-null float64
8. Production 72731 non-null float64 dtypes: float64(2), int64(2), object(4) memory usage: 4.5+ MB

print(temp.head())

Unnamed: 0 State\_Name District\_Name Crop\_Year Season \

1. 0 Bihar NALANDA 1997 Rabi
2. 1 Assam KARBI ANGLONG 2011 Whole Year
3. 2 Gujarat ANAND 2012 Summer
4. 3 Karnataka UTTAR KANNAD 2005 Rabi
5. 4 Uttar Pradesh JAUNPUR 2008 Rabi

Crop Area Production 0 Wheat 81934.0 160425.0

1 Onion 257.0 514.0

2 Maize 100.0 100.0

3 Groundnut 2872.0 4572.0

4 Onion 110.0 1290.0

print(temp.tail())

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Season | Unnamed: 0  \ | State\_Name | | District\_Name | Crop\_Year |  |
| 73822 | 73822 | Bihar | | BUXAR | 2004 | Whole Year |
| 73823 | 73823 | Madhya Pradesh | | RAJGARH | 2013 | Kharif |
| 73824 | 73824 | Madhya Pradesh | | RAJGARH | 1998 | Kharif |
| 73825 | 73825 | Rajasthan | | TONK | 2000 | Kharif |
| 73826 | 73826 | Jharkhand | | KODERMA | 2002 | Winter |
|  | Crop | Area | Production | | | |
| 73822 | Onion | 203.00 | 1918.00 | | | |
| 73823 | Bajra | 12.00 | 11.00 | | | |
| 73824 | Groundnut | 8700.00 | 9100.00 | | | |
| 73825 | Bajra | 34477.00 | 20137.00 | | | |
| 73826 | Rice | 14685.71 | 18552.95 | | | |
| *# to know unique*  temp.nunique() | | *values* | | | | |
| Unnamed: 0 | | 73827 | | | | |
| State\_Name | | 33 | | | | |
| District\_Name | | 646 | | | | |
| Crop\_Year | | 19 | | | | |
| Season | | 6 | | | | |
| Crop | | 122 | | | | |
| Area | | 17798 | | | | |
| Production dtype: int64 | | 22196 | | | | |
| temp.isna().sum() | |  | | | | |
| Unnamed: 0 | | 0 | | | | |
| State\_Name | | 0 | | | | |
| District\_Name | | 0 | | | | |
| Crop\_Year | | 0 | | | | |
| Season | | 0 | | | | |
| Crop | | 0 | | | | |
| Area | | 0 | | | | |
| Production dtype: int64 | | 1096 | | | | |

temp.isnull().sum() Unnamed: 0 0

State\_Name 0

|  |  |
| --- | --- |
| District\_Name | 0 |
| Crop\_Year | 0 |
| Season | 0 |
| Crop | 0 |
| Area | 0 |
| Production | 1096 |
| dtype: int64 |  |

print(temp.describe())

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Unnamed: 0 | Crop\_Year | Area | Production |
| count | 73827.000000 | 73827.000000 | 7.382700e+04 | 7.273100e+04 |
| mean | 36913.000000 | 2005.657551 | 1.218297e+04 | 6.210320e+05 |
| std | 21312.163499 | 4.954910 | 5.348285e+04 | 1.749756e+07 |
| min | 0.000000 | 1997.000000 | 4.000000e-02 | 0.000000e+00 |
| 25% | 18456.500000 | 2002.000000 | 8.100000e+01 | 8.900000e+01 |
| 50% | 36913.000000 | 2006.000000 | 5.900000e+02 | 7.290000e+02 |
| 75% | 55369.500000 | 2010.000000 | 4.500000e+03 | 7.200000e+03 |
| max | 73826.000000 | 2015.000000 | 8.580100e+06 | 1.125000e+09 |

*#Measure of central tendency* print(temp.mean()) print(temp.median()) print(temp.mode()) print(temp['Area'].mean())

print(temp['Area'].median())

print(temp['Area'].mode())

|  |  |
| --- | --- |
| Unnamed: 0 | 36913.000000 |
| Crop\_Year | 2005.657551 |
| Area | 12182.973146 |
| Production | 621031.953358 |
| dtype: float64 |  |
| Unnamed: 0 | 36913.0 |
| Crop\_Year | 2006.0 |
| Area | 590.0 |
| Production | 729.0 |
| dtype: float64 |  |

Unnamed: 0 State\_Name District\_Name Crop\_Year Season

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Crop | \ |  | | | | | | |
| 0 |  | 0 | Uttar | Pradesh | BIJAPUR | 2003.0 | Kharif |  |
| Rice |  |  |  |  |  |  |  |  |
| 1 |  | 1 |  | NaN | NaN | NaN |  | NaN |
| NaN |  |  |  |  |  |  |  |  |
| 2 |  | 2 |  | NaN | NaN | NaN |  | NaN |
| NaN |  |  |  |  |  |  |  |  |
| 3 |  | 3 |  | NaN | NaN | NaN |  | NaN |
| NaN |  |  |  |  |  |  |  |  |
| 4 |  | 4 |  | NaN | NaN | NaN |  | NaN |
| NaN |  |  |  |  |  |  |  |  |
| ... |  | ... |  | ... | ... | ... |  | ... |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ... 73822 | 73822 | | NaN | NaN | NaN | NaN |
| NaN 73823 | 73823 | | NaN | NaN | NaN | NaN |
| NaN 73824 | 73824 | | NaN | NaN | NaN | NaN |
| NaN 73825 | 73825 | | NaN | NaN | NaN | NaN |
| NaN 73826 | 73826 | | NaN | NaN | NaN | NaN |
| NaN |  | |  |  |  |  |
|  | Area | Production | | | | |
| 0 | 1.0 | 1.0 | | | | |
| 1 | NaN | NaN | | | | |
| 2 | NaN | NaN | | | | |
| 3 | NaN | NaN | | | | |
| 4 | NaN | NaN | | | | |
| ... | ... | ... | | | | |
| 73822 | NaN | NaN | | | | |
| 73823 | NaN | NaN | | | | |
| 73824 | NaN | NaN | | | | |
| 73825 | NaN | NaN | | | | |
| 73826 | NaN | NaN | | | | |

[73827 rows x 8 columns]

12182.973146138953

590.0

0 1.0

Name: Area, dtype: float64

C:\Users\Lenovo\AppData\Local\Temp\ipykernel\_4872\317555596.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric\_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.

print(temp.mean()) C:\Users\Lenovo\AppData\Local\Temp\ipykernel\_4872\317555596.py:3: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric\_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.

print(temp.median())

*#Measure of Variability*

print("Variance of Area column: ", temp['Area'].var()) print("Standard Deviation of Area column: ", temp['Area'].std())

*# mean (average) absolute deviation*

print("Mean Absolute Deviation of Area column", temp['Area'].mad())

*#range*

range1 = max(temp['Area'])-min(temp['Area']) print("Range of values in Area column: ",range1)

*#IQR*

Q1 = np.percentile(temp['Area'], 25) Q3 = np.percentile(temp['Area'], 75)

print("First Quartile: ",Q1," ", "Thrid Quartile: ", Q3)

IQR = Q3 - Q1

print("Inter-Quartile Range: ", IQR)

Variance of Area column: 2860415014.457252

Standard Deviation of Area column: 53482.8478529075

Mean Absolute Deviation of Area column 18070.698246088185 Range of values in Area column: 8580099.96

First Quartile: 81.0 Thrid Quartile: 4500.0

Inter-Quartile Range: 4419.0

*#Measure of frequency destribution*

unique, counts = np.unique(temp['Area'], return\_counts=True) print(unique, counts)

[4.0000e-02 5.0000e-02 7.0000e-02 ... 3.1799e+06 3.3047e+06 8.5801e+06] [1 1 1 ... 1 1 1]

matrix = np.corrcoef(temp['Area'], temp['Production']) print(matrix)

print('\n\n')

print(temp[['Area', 'Production']].corr())

[[ 1. nan]

[nan nan]]

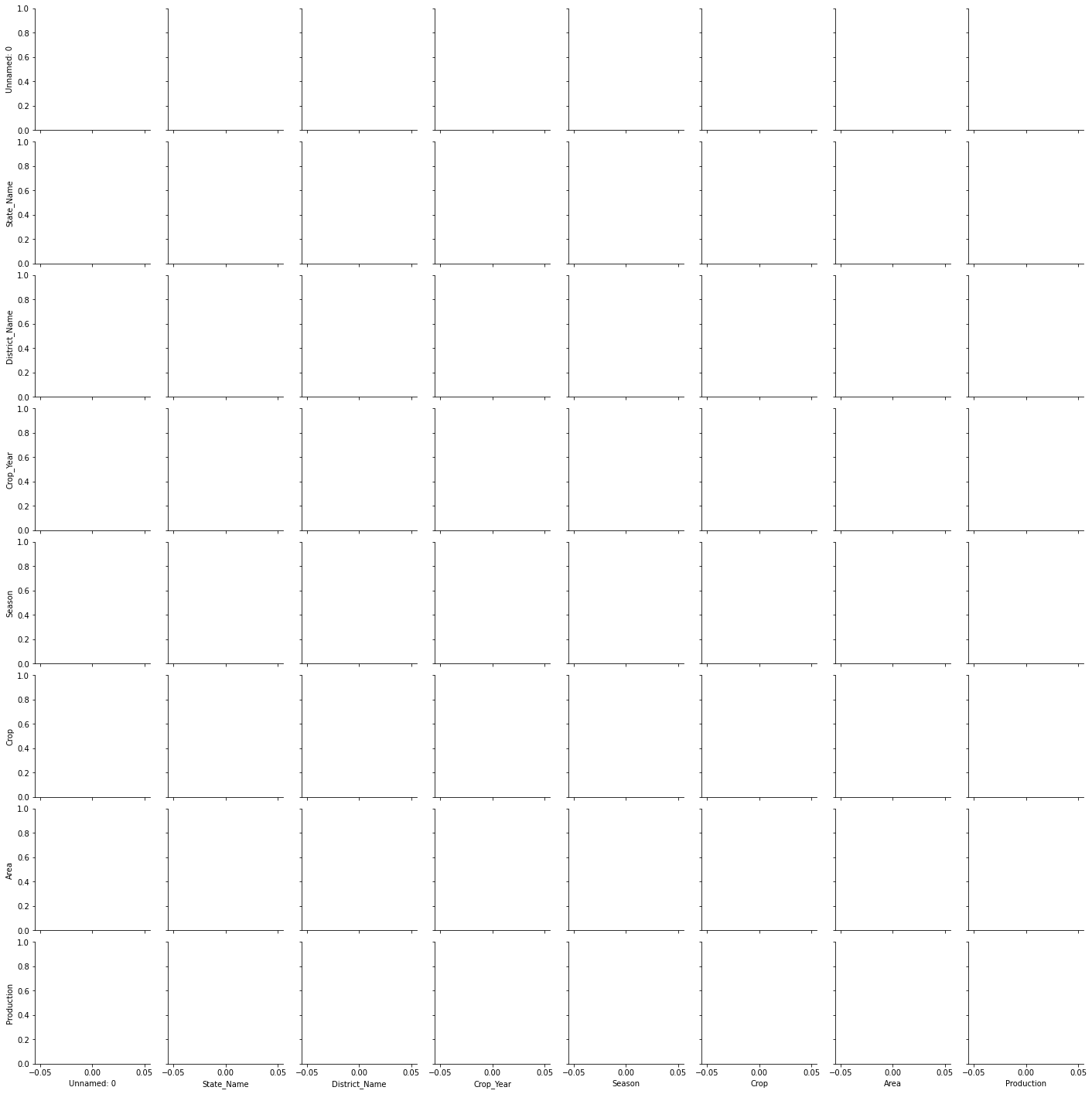
|  |  |  |
| --- | --- | --- |
|  | Area | Production |
| Area | 1.000000 | 0.041043 |
| Production | 0.041043 | 1.000000 |

matrix = temp.corr() print(matrix)

Unnamed: 0 Crop\_Year Area Production

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Unnamed: 0 | 1.000000 | -0.002016 | 0.000864 | 0.001313 |
| Crop\_Year | -0.002016 | 1.000000 | -0.023720 | 0.007422 |
| Area | 0.000864 | -0.023720 | 1.000000 | 0.041043 |
| Production | 0.001313 | 0.007422 | 0.041043 | 1.000000 |

**import** seaborn **as** sns print(temp[temp['State\_Name']=='Gujrat'].corr()) sns.pairplot(temp[temp['State\_Name']=='Gujrat'])



**import** matplotlib.pyplot **as** plt

bxplt = sns.boxplot(temp['Production']) plt.show()

Shape, rectangle

Description automatically generated

Q1 = np.percentile(temp['Production'], 25.) *# 25th percentile of the data of the given feature*

Q3 = np.percentile(temp['Production'], 75.) *# 75th percentile of the data of the given feature*

IQR = Q3-Q1 *#Interquartile Range*

*#outlier\_step = IQR \* 1.5 #That's we were talking about above #outliers = feature\_data[~((feature\_data >= Q1 - outlier\_step) & (feature\_data <= Q3 + outlier\_step))].index.tolist()*

ll = Q1 - (1.5\*IQR) ul = Q3 + (1.5\*IQR)

upper\_outliers = temp[temp['Production'] > ul].index.tolist() lower\_outliers = temp[temp['Production'] < ll].index.tolist() bad\_indices = list(set(upper\_outliers + lower\_outliers))

drop = True

*#if not drop:*

*#print('For the feature {}, No of Outliers is*

*{}'.format(temp['sepal.width'], len(bad\_indices)))*

**if** drop:

temp.drop(bad\_indices, inplace = True, errors = 'ignore') print('Outliers from {} feature

removed'.format(temp['Production']))

bxplt = sns.boxplot(temp['Production']) plt.show()

Shape, rectangle

Description automatically generated

Conclusion:- In conclusion, this assignment demonstrates the importance of data analysis and machine learning in solving real-world problems. By exploring various operations on a dataset, we can gain insights from the data and make predictions using machine learning algorithms. The strengths and limitations of these techniques must be carefully considered when applying them to a real-world problem.