DATASOFTIXS

Data Science Project

Submitted by:

J. REINA CHRISTICA

I Year (CSE)

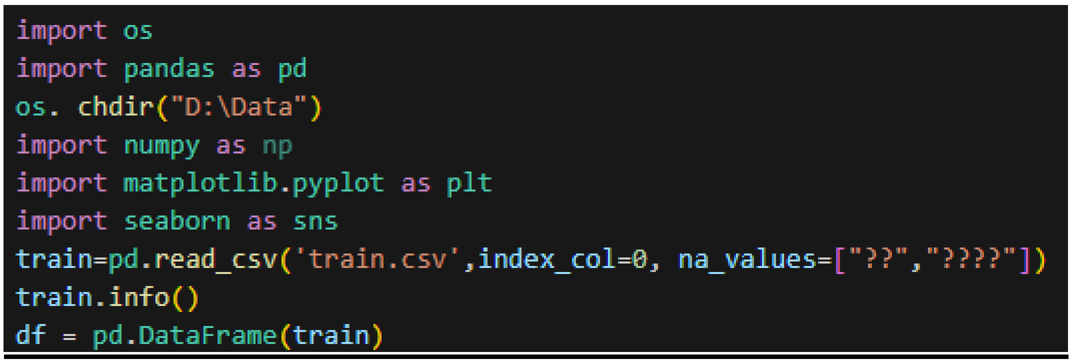
DMI COLLEGE OF ENGINEERING

Week-1

Exploring data basics

Project 1: Sales Data Analysis

1) Load data from a CSV file and clean it for analysis.



```
import os
import pandas as pd
os. chdir("D:\Data")
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
train=pd.read_csv('train.csv',index_col=0, na_values=["??","????"])
train.info()
df = pd.DataFrame(train)
```

- **Importing Necessary Libraries:**

- os: Handles file system operations.
- pandas (pd): Used for data manipulation and analysis.
- numpy (np): Provides numerical computing capabilities.
- matplotlib.pyplot (plt): Enables data visualization.
- seaborn (sns): Used for statistical data visualization.

- **Setting the Working Directory:**

- The os.chdir("D:\Data") command changes the working directory to "D:\Data", where the dataset is stored.

- **Loading the CSV File:**

- pd.read_csv('train.csv', index_col=0, na_values=["??", "????"])
- Reads the **'train.csv'** file.
- **index_col=0**: Uses the first column as the index.
- **na_values=["??", "????"]**: Treats "??" and "????" as missing values (NaN).

- **Exploring the Dataset:**

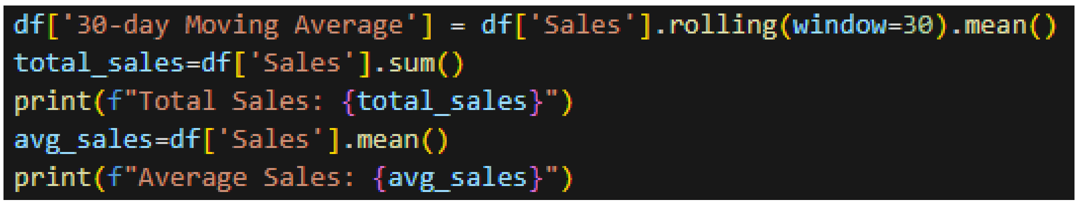
- train.info() displays an overview of the dataset, including column names, data types, and missing values. If any missing values are found they are modified accordingly and variables with different data types are changed to appropriate data types.

- **Creating a Data Frame:**

- df = pd.DataFrame(train) stores the loaded dataset into a Pandas DataFrame (df) for further processing and analysis.

2) Calculate metrics like total sales, average sales, and sales trends over time.

Metrics like total sales, average sales, and sales trends over time are calculated using the specific function.

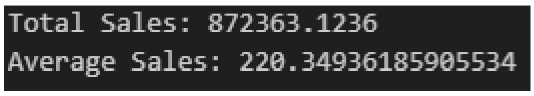


```
df['30-day Moving Average'] = df['Sales'].rolling(window=30).mean()
total_sales=df['Sales'].sum()
print(f"Total Sales: {total_sales}")
avg_sales=df['Sales'].mean()
print(f"Average Sales: {avg_sales}")
```

1. **Total Sales:**
    - The sum of all sales transactions within a given period.
    - Helps determine overall revenue generation.
2. **Average Sales:**
    - Calculated by dividing total sales by the number of transactions.
    - Provides insights into typical sales performance.
3. **Sales Trends Over Time:**
    - Analyses sales patterns across different time intervals (e.g., daily, monthly, yearly).
    - Helps identify seasonal trends, growth patterns, and potential declines.

By computing these metrics, businesses can make data-driven decisions, optimize strategies, and improve overall sales performance

Output:

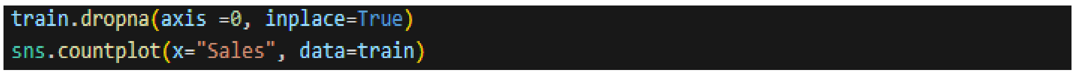


```
Total Sales: 872363.1236
Average Sales: 220.34936185905534
```

3) Visualize sales performance using bar charts or line plots.

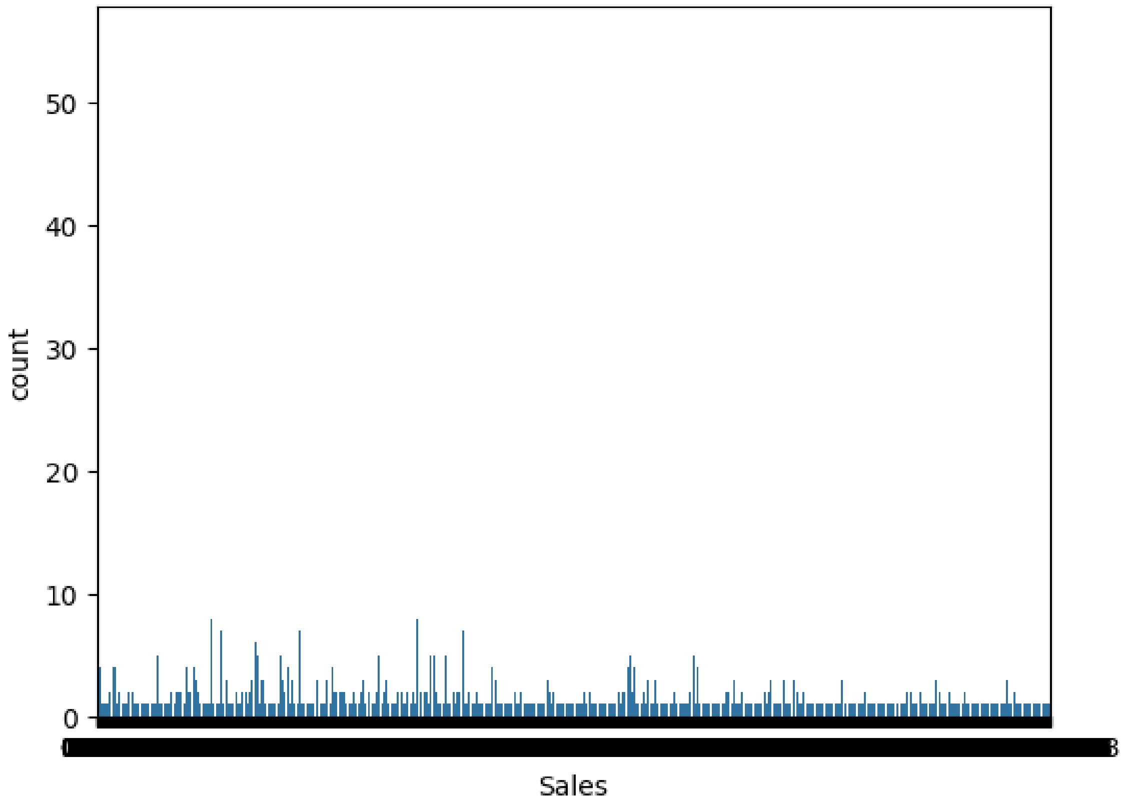
Sales performance is visualized using bar plots or line plots. Visualization helps in understanding sales performance trends effectively. Two common methods are bar charts and line plots.

- **A bar chart for sales performance:**
  A bar chart helps compare sales across different categories, such as months, regions, or products.



```
train.dropna(axis =0, inplace=True)
sns.countplot(x="Sales", data=train)
```

Output:

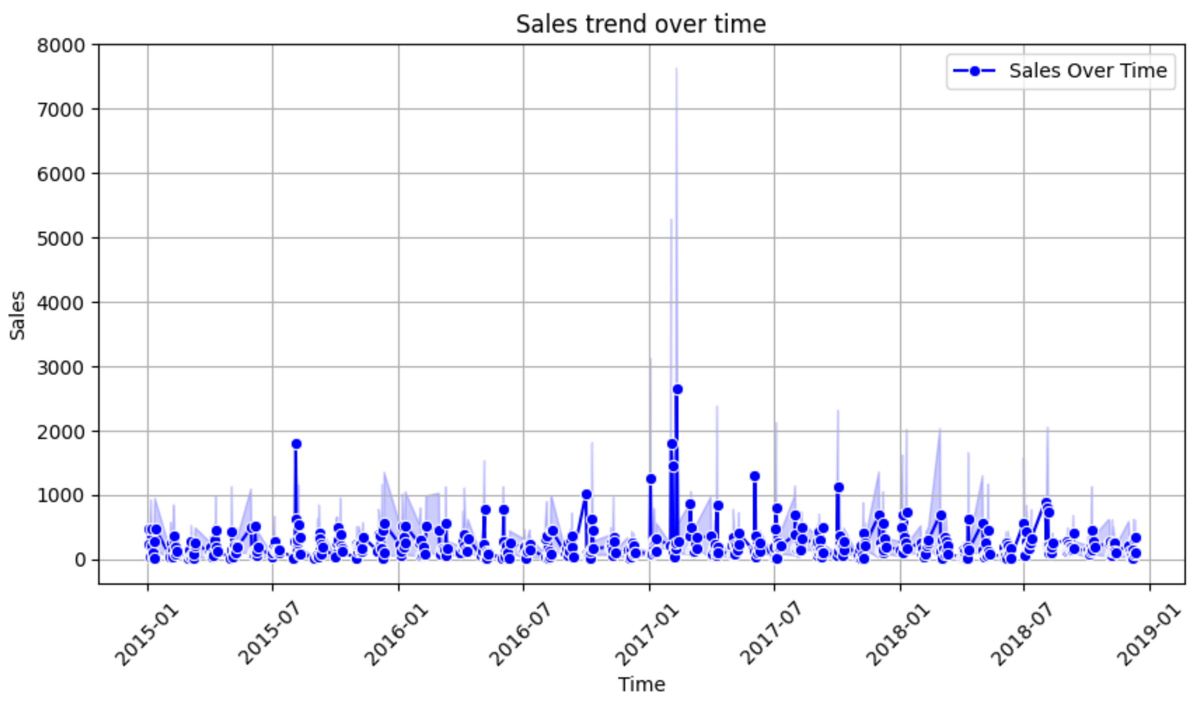


- **Line plots for sales trends over time:**

  A line plot is useful for showing trends in sales over a continuous period, such as daily or monthly sales

```
plt.figure(figsize=(10,5))
sns.lineplot(x=df['Order Date'], y=df['Sales'],marker='o',color='b',label='Sales Over Time')
plt.xlabel('Time')
plt.ylabel('Sales')
plt.title('Sales trend over time')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.show()
```

Output:



Conclusion:

Sales data analysis provides valuable insights that help businesses optimize performance, improve strategies, and drive growth. Businesses can identify seasonal patterns and long-term growth trajectories by examining sales trends, enabling more accurate forecasting and demand planning. Evaluating key metrics such as total revenue, average sales, and profit margins allows companies to assess their overall performance and the effectiveness of marketing campaigns. Additionally, analysing sales data helps identify high-performing and underperforming products.

Week-2

Data Manipulation & Visualization

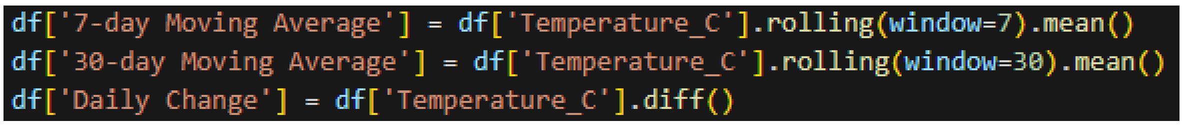
Project 3: Weather data visualization

1) Analyze temperature trends and patterns over time.

Firstly, the necessary libraries and algorithms are imported some of them are pandas, numpy, seaborn, matplotlib and IsolationForest algorithm. Then the dataset is cleaned and loaded.

Analyzing temperature trends over time helps in understanding climate patterns, predicting weather changes, and studying the impact of environmental factors. By examining historical temperature data, scientists and analysts can identify long-term warming or cooling trends, seasonal variations, and extreme weather events.

- **Identifying trends using moving averages and calculation of temperature variability:**



```
df['7-day Moving Average'] = df['Temperature_C'].rolling(window=7).mean()
df['30-day Moving Average'] = df['Temperature_C'].rolling(window=30).mean()
df['Daily Change'] = df['Temperature_C'].diff()
```

7-day Moving Average:

- This calculates the average temperature over a rolling window of 7 days, smoothing short-term fluctuations and highlighting weekly trends.

30-day Moving Average:

- Similar to the 7-day moving average, but over 30 days, helping to identify longer-term temperature trends.

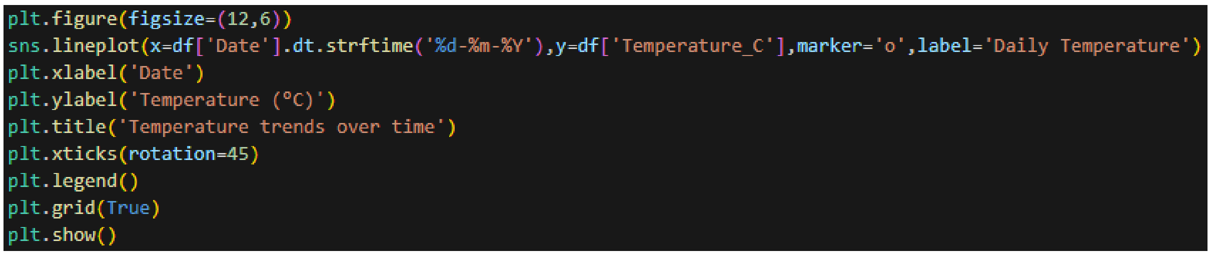
Daily Change in Temperature:

- This computes the difference between consecutive temperature values, allowing the identification of day-to-day temperature fluctuations.

2) Creating line plots, scatter plots, and bar charts to visualize findings.

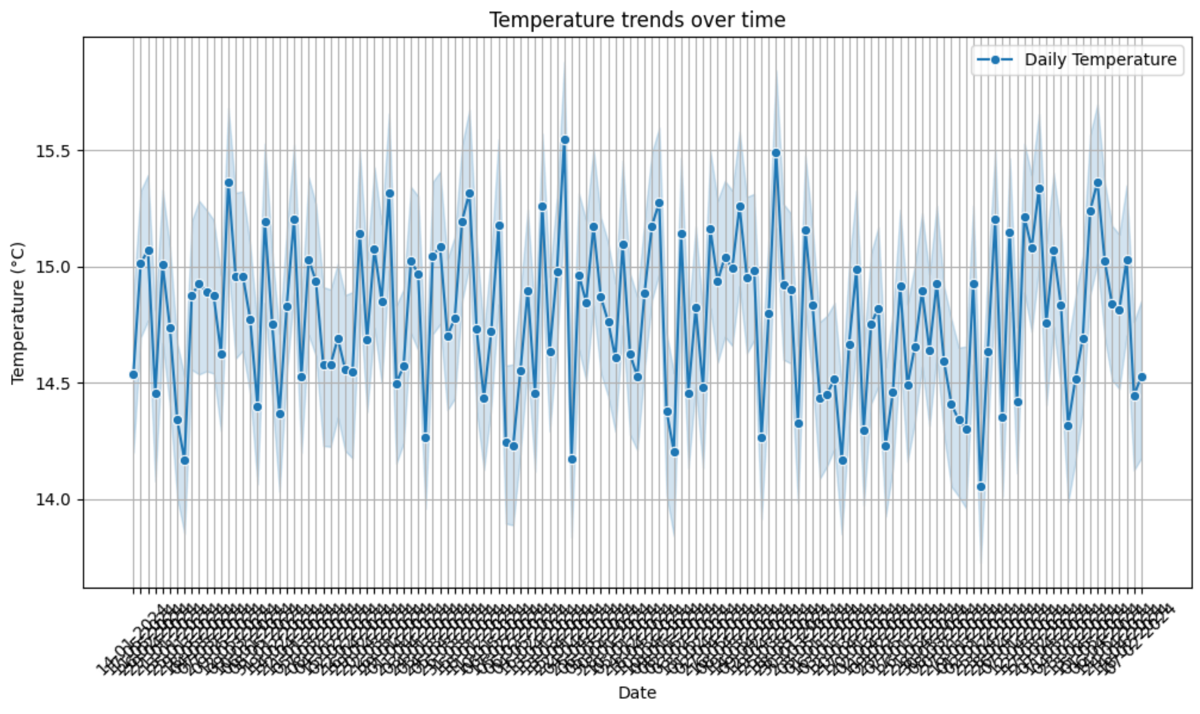
Data visualization is essential for understanding patterns, trends, and relationships within a dataset. Three common types of visualizations line plots, scatter plots, and bar charts help in interpreting data effectively.

- **A line for visualizing the variation of temperature with time:** A line plot is an effective way to visualize daily temperature variations over time.



```
plt.figure(figsize=(12,6))
sns.lineplot(x=df['Date'].dt.strftime('%d-%m-%Y'),y=df['Temperature_C'],marker='o',label='Daily Temperature')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.title('Temperature trends over time')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.show()
```

Output:

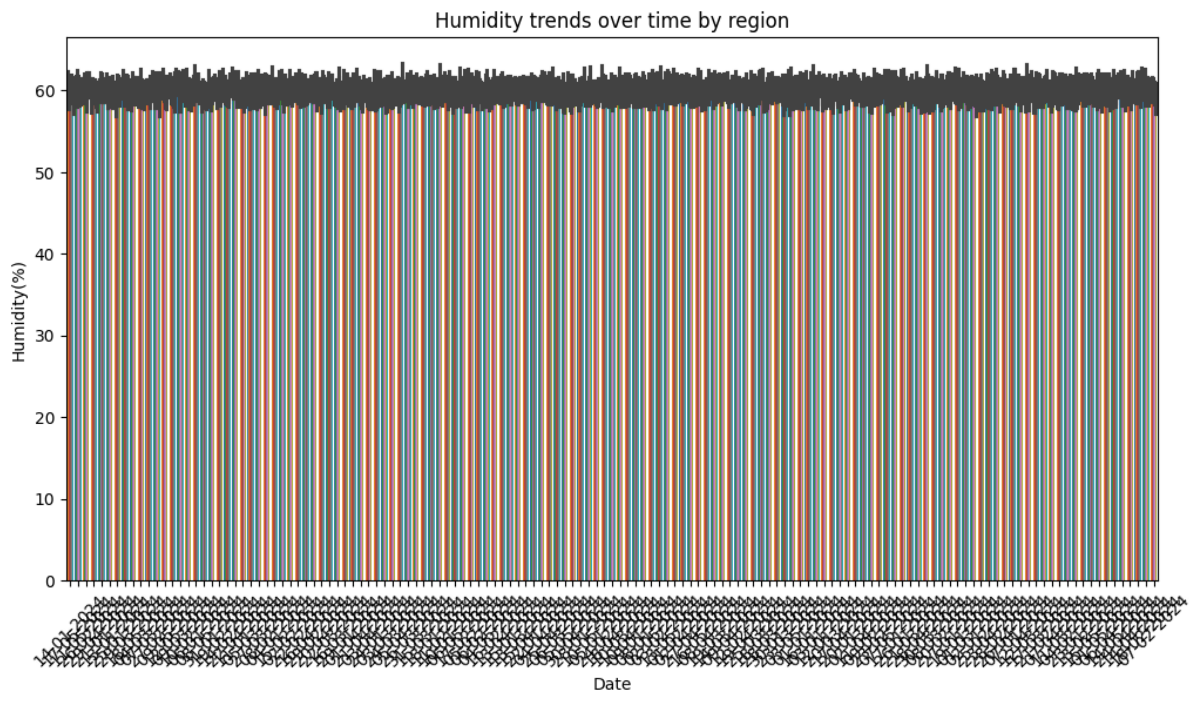


- **A bar plot for visualizing humidity trends over time by region:**

  A bar plot is an effective way of visualizing humidity trends over time by region.

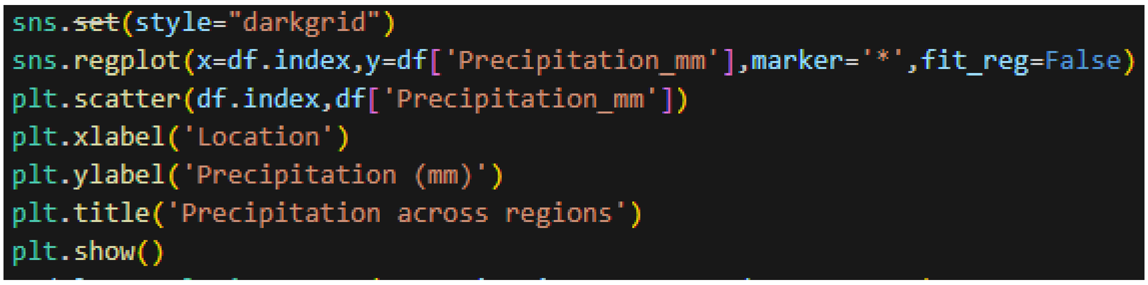
```
plt.figure(figsize=(12,6))
sns.barplot(x=df['Date'].dt.strftime('%d-%m-%Y'),y=df['Humidity_pct'],hue=df.index)
plt.xlabel('Date')
plt.ylabel('Humidity(%)')
plt.title('Humidity trends over time by region')
plt.xticks(rotation=45)
plt.legend([],[], frameon=False)
plt.show()
```

Output:



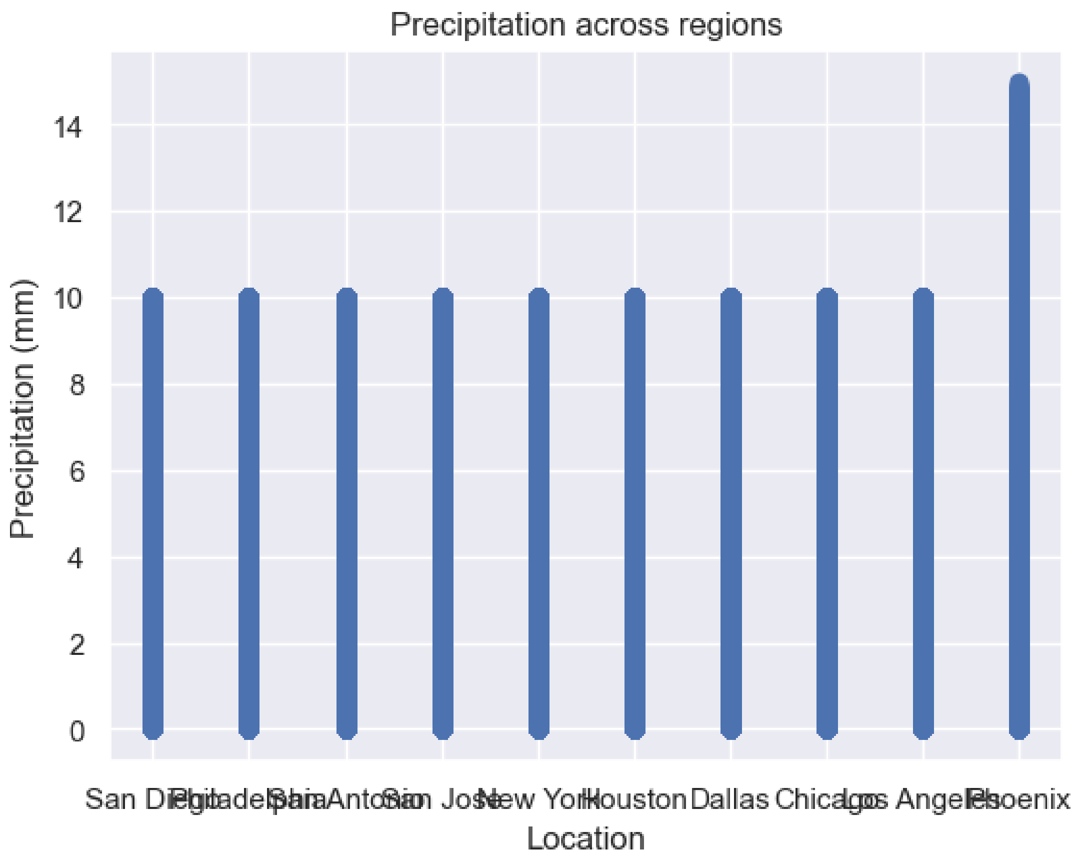
- **A bar plot for visualizing precipitation over regions:**

  A bar plot helps visualize precipitation across various regions.



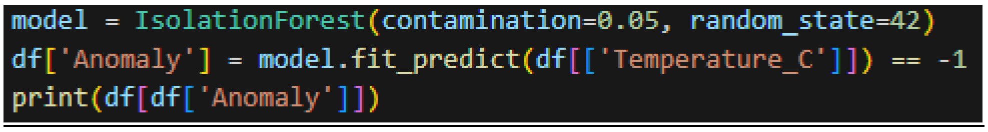
```
sns.~~set~~(style="darkgrid")
sns.regplot(x=df.index,y=df['Precipitation_mm'],marker='*',fit_reg=False)
plt.scatter(df.index,df['Precipitation_mm'])
plt.xlabel('Location')
plt.ylabel('Precipitation (mm)')
plt.title('Precipitation across regions')
plt.show()
```

Output:



3) Identify anomalies like extreme temperatures.

Anomaly detection helps in identifying extreme temperature values that deviate significantly from normal trends. You can use statistical methods, visualization techniques, and machine learning to detect these anomalies.



```
model = IsolationForest(contamination=0.05, random_state=42)
df['Anomaly'] = model.fit_predict(df[['Temperature_C']]) == -1
print(df[df['Anomaly']])
```

Here **Isolation Forest** algorithm is used to detect anomalies (outliers) in temperature data.

- **IsolationForest(contamination=0.05, random_state=42)**: Initializes the Isolation Forest model, setting contamination=0.05, meaning 5% of the data points are expected to be anomalies. The random_state=42 ensures reproducibility.
- **model.fit_predict(df[['Temperature_C']]) == -1**: Trains the model on the temperature data and predicts whether each data point is an anomaly. The model assigns -1 to anomalies and 1 to normal values.
- **df['Anomaly'] =** Creates a new column named 'Anomaly', storing True for anomalies (-1) and False for normal data points.
- **print(df[df['Anomaly']])**: Displays the rows where anomalies were detected.

This approach helps identify extreme temperature fluctuations, which may indicate errors, unusual weather patterns, or other significant events.

Conclusion:

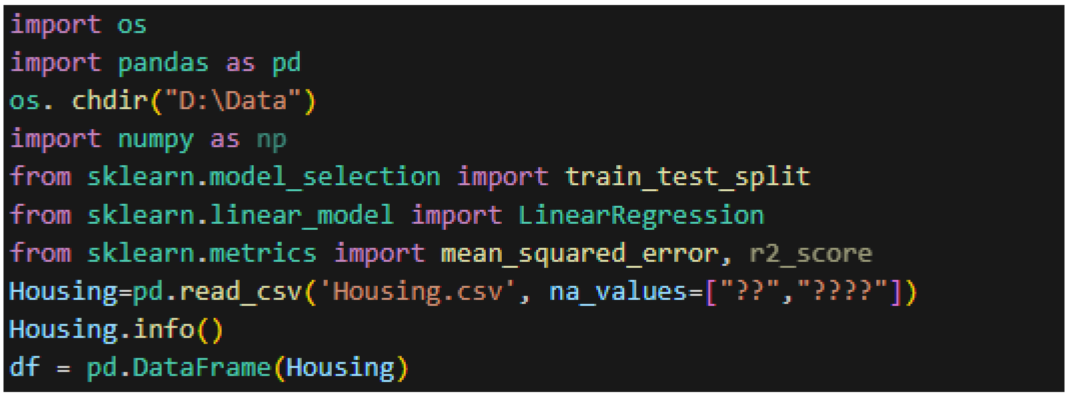
Weather data visualization plays a crucial role in understanding trends, detecting anomalies, and improving forecasting accuracy. By analyzing temperature, humidity, wind speed, and precipitation n through various visual tools, we can identify long-term climate patterns and seasonal variations. This information is essential for making informed decisions in agriculture, disaster preparedness, and daily activities. Additionally, visualizing weather data helps in detecting extreme conditions such as heat waves, storms, and cold snaps, allowing for timely precautions. Regional comparisons through bar charts and maps provide valuable insights into climate differences, supporting research and forecasting. Moreover, historical weather trends contribute to the accuracy of predictive models, enhancing weather forecasting.

Week-3

Machine learning foundations

Project 5: Predicting house prices

1) Load a dataset with house features (e.g., size, location) and prices.



```
import os
import pandas as pd
os. chdir("D:\Data")
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
Housing=pd.read_csv('Housing.csv', na_values=["??","????"])
Housing.info()
df = pd.DataFrame(Housing)
```

1. Importing Required Libraries

The script begins by importing essential libraries:

- os for handling directory changes.
- pandas for data manipulation and analysis.
- numpy for numerical operations.
- sklearn.model_selection for splitting the dataset into training and testing sets.
- sklearn.linear_model for building a linear regression model.
- sklearn.metrics for evaluating model performance using mean squared error and R-squared score.

2. Setting Up the Working Directory

- The script changes the working directory to "D:\Data" using os.chdir("D:\Data").
- This ensures that the dataset is correctly loaded from the specified location.

3. Loading the Dataset

- The dataset "Housing.csv" is read into a Pandas Data Frame using pd.read_csv().
- Missing values represented by "??" and "????" are identified and treated as NaN.
- The dataset's structure and missing values are examined using Housing.info().

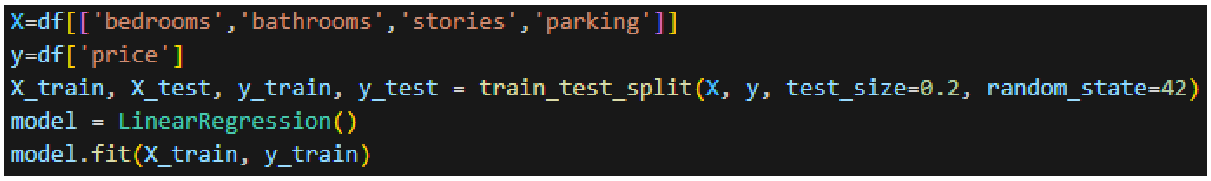
4. Converting to a Pandas Data Frame

- The dataset is explicitly converted into a Data Frame using df = pd.DataFrame(Housing), making it easier to manipulate and analyze.

5. Preparing for Machine Learning

- The script imports necessary functions for machine learning:
  - train_test_split to divide data into training and testing sets.
  - LinearRegression to build a regression model.
  - mean_squared_error and r2_score to evaluate model performance.

2) Train a regression model (e.g., Linear Regression) to predict prices.



```
X=df[['bedrooms','bathrooms','stories','parking']]
y=df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
```

1. Selecting Features and Target Variable

- X = df[['bedrooms', 'bathrooms', 'stories', 'parking']]
  - This line extracts specific columns (bedrooms, bathrooms, stories, and parking) from the Data Frame as independent variables (features) for the model.
- y = df['price']
  - The target variable (dependent variable) is set to price, which the model aims to predict.

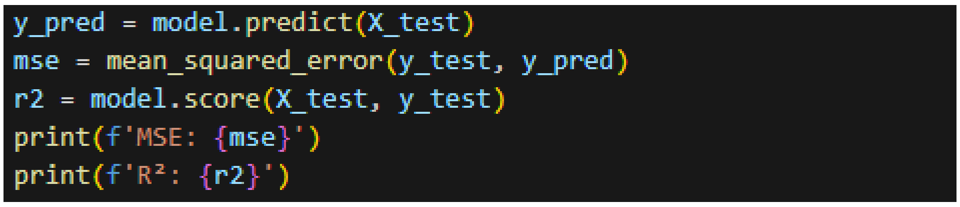
2. Splitting the Dataset

- train_test_split(X, y, test_size=0.2, random_state=42)
    - The dataset is divided into training and testing sets.
    - test_size=0.2 indicates that 20% of the data is allocated for testing, while 80% is used for training.
    - random_state=42 ensures reproducibility by maintaining the same split each time the code runs.
    - X_train, X_test, y_train, and y_test store the training and testing data.

3. Creating and Training the Model

- model = LinearRegression()
    - Initializes a linear regression model.
- model.fit(X_train, y_train)
    - Trains the model using the training data (X_train, y_train).
    - The model learns relationships between the features (X_train) and the target variable (y_train).

3) Evaluate the model's performance using metrics like MAE or R².



```
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = model.score(X_test, y_test)
print(f'MSE: {mse}')
print(f'R²: {r2}')
```

1. Making Predictions

- y_pred = model.predict(X_test)
    - The trained linear regression model is used to predict house prices based on the test set (X_test).
    - The predicted values are stored in y_pred.

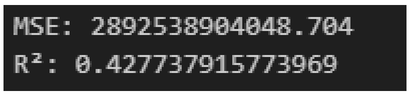
2. Evaluating Model Performance

- mse = mean_squared_error(y_test, y_pred)
    - Calculates the **Mean Squared Error (MSE)**, which measures the average squared difference between actual (y_test) and predicted (y_pred) values.
    - A lower MSE indicates better model accuracy.
- r2 = model.score(X_test, y_test)
    - Computes the **R-squared (R²) score**, which represents the proportion of variance in the target variable explained by the model.
    - An R² value closer to **1** means a better fit, while a value near **0** indicates a poor fit.

3. Displaying the Results

- print(f'MSE: {mse}')
    - Prints the calculated **Mean Squared Error** value.
- print(f'R²: {r2}')
    - Prints the **R-squared score** to indicate the model's predictive power.

Output:



```
MSE: 2892538904048.704
R²: 0.427737915773969
```

Conclusion:

Predicting house prices using machine learning provides valuable insights into real estate trends and helps stakeholders make informed decisions. By analyzing key features such as the number of bedrooms, bathrooms, stories, and parking spaces, a predictive model can estimate property values with reasonable accuracy. Performance metrics like Mean Squared Error (MSE) and R-squared (R²) help evaluate the model's effectiveness, ensuring its reliability in forecasting prices. This approach benefits buyers, sellers, and real estate agents by offering data-driven price estimations, reducing uncertainty, and aiding in financial planning. Additionally, predictive models can identify market patterns, assess the impact of property features on pricing, and contribute to smarter investment strategies.

THANK YOU