Project Report

INTRODUCTION:

1.1 Project Overview:

The project involves analyzing sales data from 45 Walmart stores, incorporating store information and monthly sales. The goal is to assess the impact of holidays on store sales. To achieve this, various algorithms such as ARIMA, Random Forest, and XgBoost will be employed. Additionally, the project includes Flask integration and deployment on IBM.

1.2 Purpose:

1. Sales Forecasting: Utilize historical sales data to predict future sales accurately. This involves understanding patterns, trends, and the influence of holidays on sales.

2. Algorithm Implementation: Apply advanced algorithms, including ARIMA, Random Forest, and XgBoost, to train and test the data. These algorithms will aid in generating reliable sales forecasts.

3. Holiday Impact Assessment: Specifically, evaluate the impact of holidays (Christmas, Thanksgiving, Super Bowl, and Labor Day) on store sales. Weighting these holiday weeks higher in the evaluation process acknowledges their significance.

4. Flask Integration: Implement Flask, a web framework, to create a user-friendly interface for interacting with the sales forecasting model. This allows users to input data and receive predictions conveniently.

5. IBM Deployment: Deploy the developed model on IBM, making it accessible to a wider audience. This ensures scalability, reliability, and availability of the sales forecasting system.

2. LITERATURE SURVEY

2.1 Existing problem: The existing problem revolves around accurately forecasting sales for Walmart stores, taking into account the impact of promotional markdown events, especially those related to holidays like Christmas, Thanksgiving, Super Bowl, and Labor Day. The challenge lies in developing a reliable model that considers the specific characteristics of these holiday weeks, which are weighted five times higher in evaluation than non-holiday weeks. Additionally, integrating and deploying this solution using Flask and IBM is part of the existing problem.

2.2 References: To address the sales forecasting problem for Walmart, references could include relevant literature on sales forecasting methodologies, time series analysis, and the application of machine learning algorithms such as ARIMA, Random Forest, and XgBoost in retail forecasting. Existing research papers, industry best practices, and documentation on Flask integration and IBM deployment can serve as valuable references.

2.3 Problem Statement Definition: The problem statement is to develop a robust sales forecasting solution for Walmart, leveraging data from 45 stores that includes store information and monthly sales. The emphasis is on accurately capturing the impact of promotional markdown events, particularly during holiday weeks (Christmas, Thanksgiving,

Super Bowl, and Labor Day). The goal is to implement machine learning **algorithms,** including ARIMA, Random Forest, and XgBoost, to train and test the data for effective sales predictions. Furthermore, the solution involves integrating the developed model into Flask for a user-friendly interface and deploying it on the IBM platform for scalability and accessibility.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

enhance sales forecasting for 45 stores. By combining store information and monthly sales data, the focus is on understanding the impact of promotional markdown events, particularly around major holidays like Super Bowl, Labor Day, Thanksgiving, and Christmas. These holiday weeks carry a fivefold weight in evaluation. The provided weekly data will be analyzed using advanced algorithms—ARIMA, Random Forest, and XgBoost—for training and testing.

Reference:

https://shorturl.at/cxDST

3.2 Ideation & Brainstorming

During the ideation phase, the team focused on defining clear objectives, exploring data handling solutions, and brainstorming methods for quantifying the impact of holidays on sales. Discussions included algorithm selection criteria, considering ARIMA, Random Forest, and XgBoost, and exploring features for the Flask application to ensure a user-friendly interface. The advantages of IBM deployment were highlighted, emphasizing scalability and reliability. Ideas were generated for visualizing sales forecasts and incorporating user feedback into model refinement. Future scopes included advanced analytics for markdown optimization and collaborations with inventory management and supply chain systems. Ethical considerations and risk mitigation strategies were also addressed during the brainstorming process.

https://shorturl.at/bjt47

4.1 Functional Requirements:

Data Ingestion:

The system should be able to ingest and preprocess the provided weekly sales data for 45 stores, including store information and promotional markdown events.

Holiday Impact Analysis:

The system should incorporate a feature to identify and analyze the impact of holidays (Christmas, Thanksgiving, Super Bowl, Labor Day) on the weekly sales data.

The impact of holiday weeks should be weighted five times higher in the evaluation compared to non-holiday weeks.

Algorithm Integration:

Implement ARIMA, Random Forest, and XgBoost algorithms for sales forecasting.
The system should allow training and testing of the data using these algorithms.

Evaluation Metrics:

Define and implement evaluation metrics to assess the accuracy and performance of the forecasting algorithms.

Metrics could include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), or other relevant measures.

Flask Integration:

Develop a web interface using Flask to interact with the sales forecasting system.
Users should be able to input data, initiate model training, and view forecasting results through the Flask web application.

IBM Deployment:

Integrate the system with IBM Cloud for deployment.
Ensure seamless deployment and accessibility of the sales forecasting application on the IBM Cloud platform.

4.2 Non-Functional Requirements:

Performance:

Considering the large dataset and complex algorithms, the system should provide efficient and timely sales forecasts.
Response times for user interactions through the Flask interface should be within acceptable limits.

Scalability:

The system should be designed to handle an increasing volume of data as the number of stores or the duration of historical data grows.
Algorithms should scale appropriately to accommodate future data expansion.

Reliability:

Ensure the reliability of the forecasting models by conducting rigorous testing on historical data.
The system should be resilient to potential errors and handle exceptions gracefully.

Security:

Implement secure data handling practices, ensuring the confidentiality and integrity of the sales data.
Access to sensitive data and forecasting models should be restricted to authorized users.

Usability:

The Flask web interface should be user-friendly, providing clear instructions for data input, model training, and result interpretation.
Proper documentation should be available for users to understand the functionality and usage of the system.

Maintainability:

Develop the system with modular and well-documented code to facilitate ease of maintenance and future enhancements.

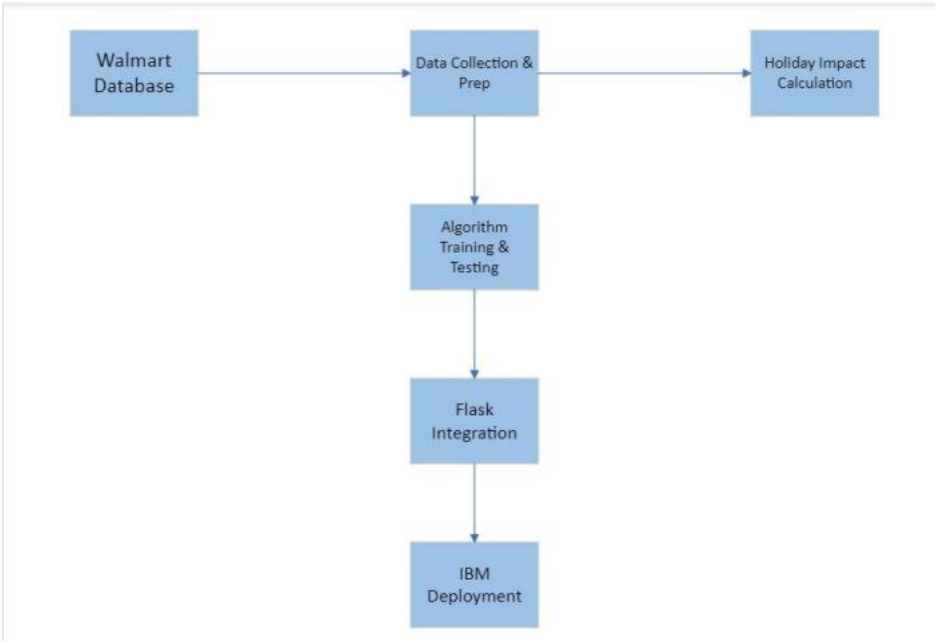
Regularly update and retrain the forecasting models to adapt to changing trends in the retail industry.

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

Data Flow Diagram:



User Stories:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance Criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account/dashboard.	High	Sprint-1
Customer (Mobile user)	Registration	USN-2	As a user, I will receive a confirmation email once I have registered for the application.	I can receive a confirmation email & click confirm.	High	Sprint-1
Customer (Mobile user)	Registration	USN-3	As a user, I can register for the application through Facebook.	I can register & access the dashboard with Facebook Login.	Low	Sprint-2
Customer (Mobile user)	Registration	USN-4	As a user, I can register for the application through Gmail.	-	Medium	Sprint-1

Customer (Mobile user)	Login	USN-5	As a user, I can log into the application by entering email & password.	-	High	Sprint-1
Customer (Web user)	Dashboard	USN-6	As a user, I can view the sales forecast dashboard.	-	High	Sprint-1
Customer Care Executive	-	USN-7	As a Customer Care Executive, I can access customer information and sales forecasts.	-	Medium	Sprint-1
Administrator	-	USN-8	As an Administrator, I can manage user accounts and system settings.	-	Low	Sprint-1

5.2 Solution Architecture

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

• Find the best tech solution to solve existing business problems.

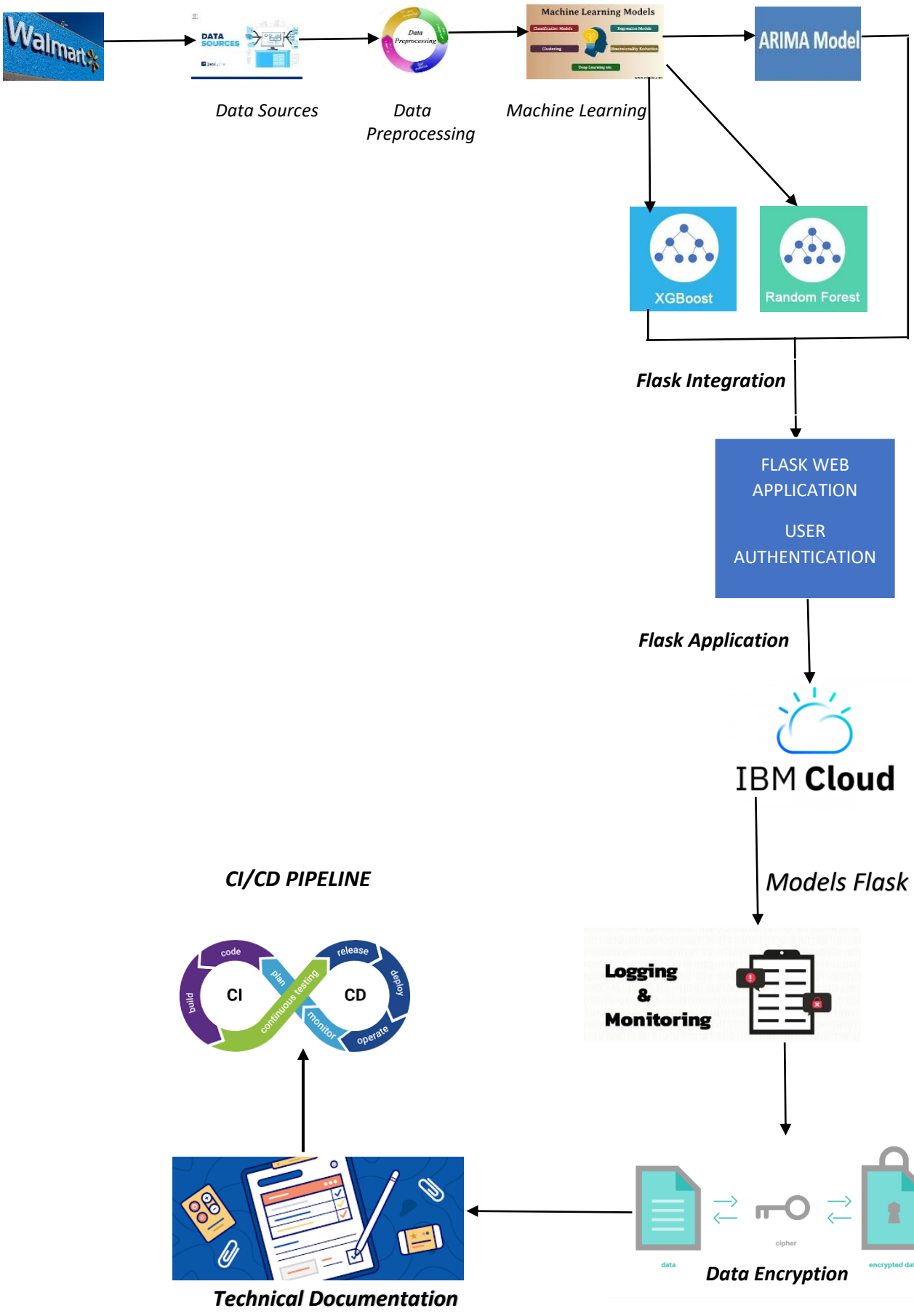
• Describe the structure, characteristics, behavior, and other aspects of the

software to project stakeholders.

• Define features, development phases, and solution requirements.

• Provide specifications according to which the solution is defined, managed,

and delivered.


Walmart
DATA SOURCES
Data Preprocessing
Machine Learning Models
ARIMA Model
Data Sources
Data Preprocessing
Machine Learning
XGBoost
Random Forest
Flask Integration
FLASK WEB APPLICATION
USER AUTHENTICATION
Flask Application
IBM Cloud
Models Flask
CI/CD PIPELINE
code
release
plan
deploy
build
CI
continuous testing
CD
monitor
operate
Logging & Monitoring
data
cipher
encrypted data
Data Encryption
Technical Documentation

6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

Architectural Diagram:

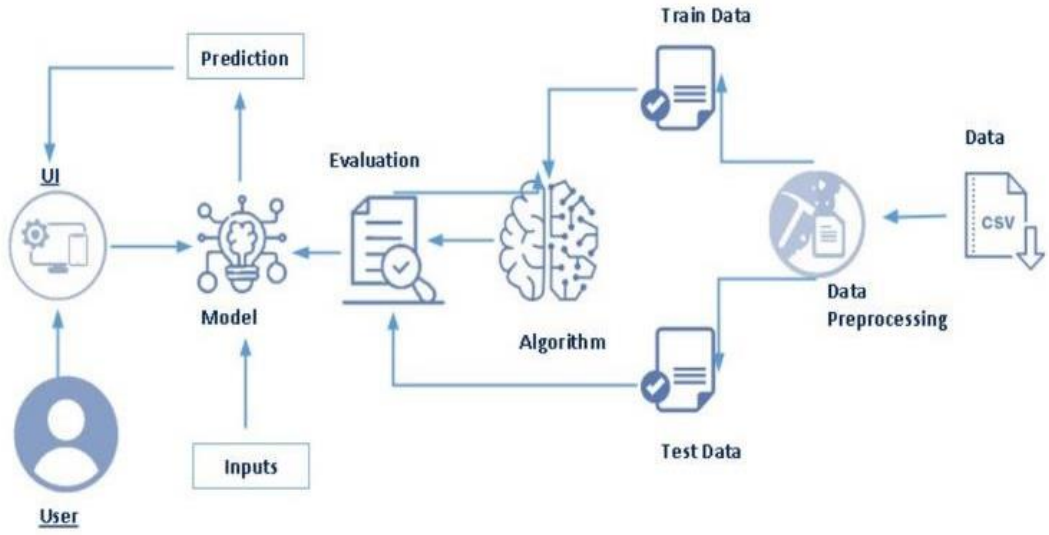


Table-1: Components & Technologies

S.No	Component	Description	Technology
1.	User Interface	Web UI for user interaction	HTML, CSS, JavaScript / Angular Js / React Js
2.	Application Logic-1	Main application logic for processing	Java / Python
3.	Application Logic-2	IBM Watson Speech-to-Text (STT) service	IBM Watson STT service
4.	Application Logic-3	IBM Watson Assistant for chatbot	IBM Watson Assistant
5.	Database	Data storage (e.g., store information)	MySQL, NoSQL, etc.
6.	Cloud Database	Cloud-based database service	IBM DB2, IBM Cloudant, etc.
7.	File Storage	File storage requirements	IBM Block Storage or Other Storage Service or Local Filesystem
8.	External API-1	External API for weather information	IBM Weather API, etc.
9.	External API-2	External API for additional information	Aadhar API, etc.
10.	Machine Learning Model	ML model for sales forecasting	ARIMA, Random Forest, XgBoost
11.	Infrastructure (Server / Cloud)	Application deployment	Local Server Configuration, Cloud Server Configuration (Cloud Foundry, Kubernetes, etc.)

6.2 Sprint Planning & Estimation

Sprint Planning & Estimation is a crucial part of Agile project management. It involves breaking down the project into manageable chunks, known as “sprints”, and estimating the amount of work that can be completed in each sprint.

Product Backlog:

Product Backlog:

Sprint	Epic	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	V Gowtham, G. Sri Sai Rohith, J. Sai Vsyhanv Reddy, E. Hemanth
Sprint-1	Registration	USN-2	As a user, I will receive a confirmation email once I have registered for the application.	1	High	V Gowtham, G. Sri Sai Rohith, J. Sai Vsyhanv Reddy, E. Hemanth
Sprint-2	Registration	USN-3	As a user, I can register for the application through Facebook.	2	Low	V Gowtham, G. Sri Sai Rohith, J. Sai Vsyhanv Reddy, E. Hemanth
Sprint-1	Registration	USN-4	As a user, I can register for the application through Gmail.	2	Medium	V Gowtham, G. Sri Sai Rohith, J. Sai Vsyhanv Reddy, E. Hemanth
Sprint-1	Login	USN-5	As a user, I can log into the application by entering email & password.	1	High	V Gowtham, G. Sri Sai Rohith, J. Sai Vsyhanv Reddy, E. Hemanth
Sprint-1	Dashboard		Project Tracker, Velocity & Burndown Chart:			V Gowtham, G. Sri Sai Rohith, J. Sai Vsyhanv Reddy, E. Hemanth

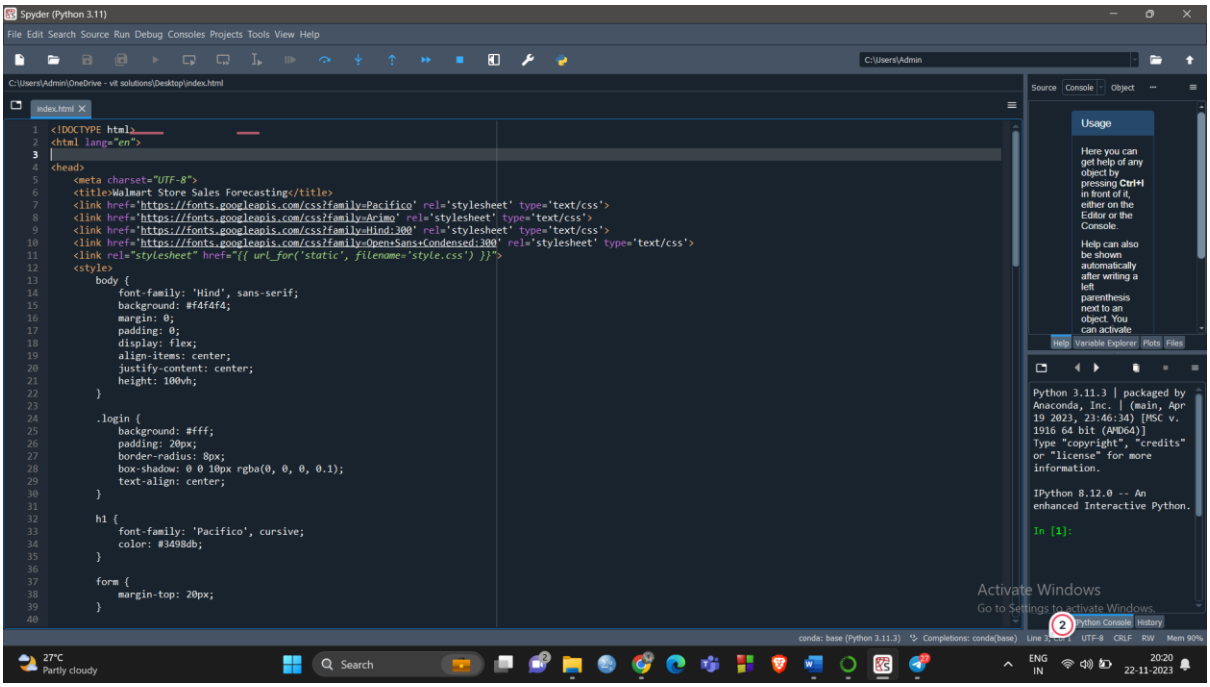
6.3 Sprint Delivery Schedule

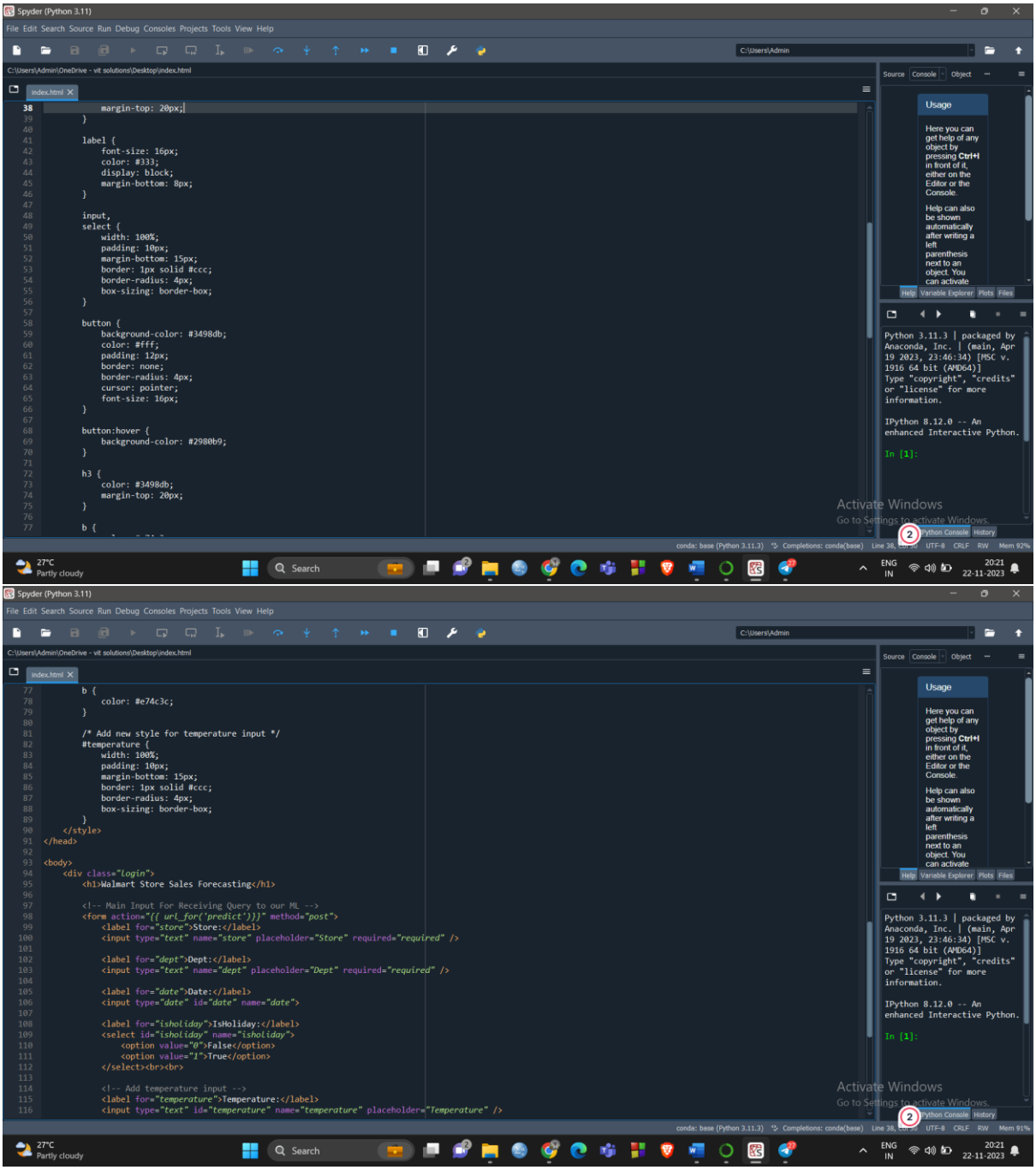
Sprint Schedule:

Sprint	Duration	Start Date	End Date	Planned	Story Points Completed (as on Planned End Date)	Actual Release Date
Sprint-1	6 Days	30 Oct 2023	4 Nov 2023	20	20	4 Nov 2023
Sprint-2	6 Days	4 Nov 2023	10 Nov 2022	20	20	10 Nov 2022
Sprint-3	6 Days	10 Nov 2022	16 Nov 2022	20	20	16 Nov 2022
Sprint-4	6 Days	16 Nov 2022	22 Nov 2022	20	20	22 Nov 2022

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

Index.html:

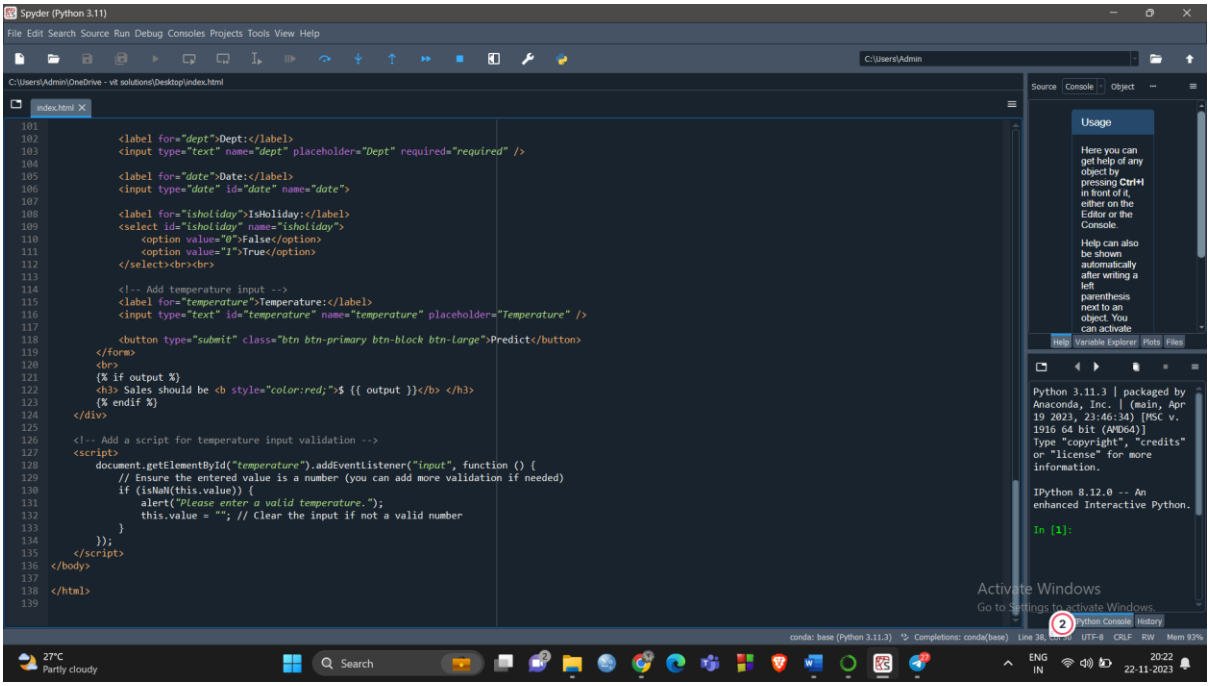



Spyder (Python 3.11)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Admin\OneDrive - vit solutions\Desktop\index.html
index.html
margin-top: 20px;
}
label {
font-size: 16px;
color: #333;
display: block;
margin-bottom: 8px;
}
input,
select {
width: 100%;
padding: 10px;
margin-bottom: 15px;
border: 1px solid #ccc;
border-radius: 4px;
box-sizing: border-box;
}
button {
background-color: #3498db;
color: #fff;
padding: 12px;
border: none;
border-radius: 4px;
cursor: pointer;
font-size: 16px;
}
button:hover {
background-color: #2980b9;
}
h3 {
color: #3498db;
margin-top: 20px;
}
b {
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate
Python 3.11.3 | packaged by Anaconda, Inc. | (main, Apr 19 2023, 23:46:34) [MSC v. 1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 8.12.0 -- An enhanced Interactive Python.
In [1]:
Activate Windows
Go to Settings to activate Windows.
27°C
Partly cloudy
Search
ENG
IN
20:21
22-11-2023

Spyder (Python 3.11)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Admin\OneDrive - vit solutions\Desktop\index.html
index.html
b {
color: #e74c3c;
}
/* Add new style for temperature input */
#temperature {
width: 100%;
padding: 10px;
margin-bottom: 15px;
border: 1px solid #ccc;
border-radius: 4px;
box-sizing: border-box;
}
</style>
</head>
<body>

<h1>Walmart Store Sales Forecasting</h1>
<!-- Main Input For Receiving Query to our ML -->
<form action="{{ url_for('predict')}}" method="post">
<label for="store">Store:</label>
<input type="text" name="store" placeholder="Store" required="required" />
<label for="dept">Dept:</label>
<input type="text" name="dept" placeholder="Dept" required="required" />
<label for="date">Date:</label>
<input type="date" id="date" name="date">
<label for="isholiday">IsHoliday:</label>
<select id="isholiday" name="isholiday">
<option value="0">False</option>
<option value="1">True</option>
</select>

<!-- Add temperature input -->
<label for="temperature">Temperature:</label>
<input type="text" id="temperature" name="temperature" placeholder="Temperature" />
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate
Python 3.11.3 | packaged by Anaconda, Inc. | (main, Apr 19 2023, 23:46:34) [MSC v. 1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 8.12.0 -- An enhanced Interactive Python.
In [1]:
Activate Windows
Go to Settings to activate Windows.
27°C
Partly cloudy
Search
ENG
IN
20:21
22-11-2023



Features:

Responsive Design:

The page is designed to be responsive and centered both vertically and horizontally on the screen.
Fonts:

Different Google Fonts are imported and used, such as Pacifico, Arimo, Hind, and Open Sans Condensed.
Form Input:

The form includes input fields for store, department, date, and a dropdown for IsHoliday.
Added a new input field for temperature.
Styling:

The styling is clean, with a white background for the form and a subtle box shadow for a modern look.
Labels and input fields have consistent styling for a uniform appearance.
Button:

The "Predict" button is styled with a blue background color that changes on hover.
Output Display:

If there is an output (sales prediction), it is displayed with a red-colored amount.
Temperature Input Validation:

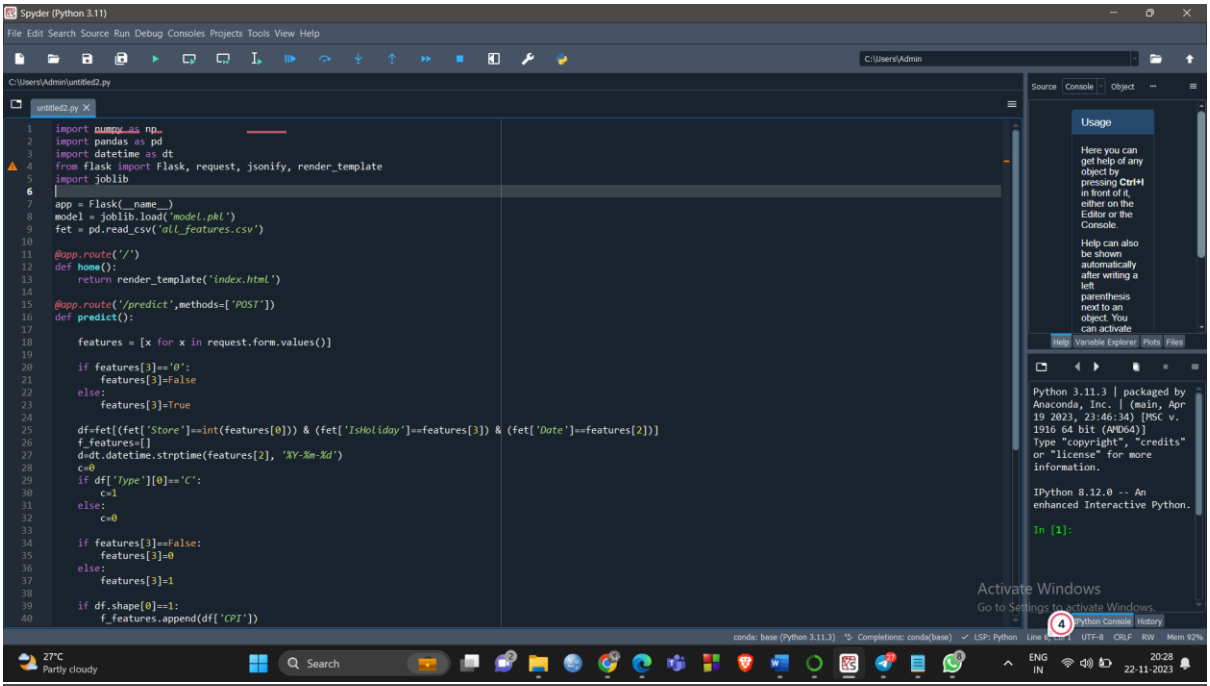
Added a simple JavaScript script to validate the temperature input. It shows an alert if the entered value is not a number and clears the input.
Title:

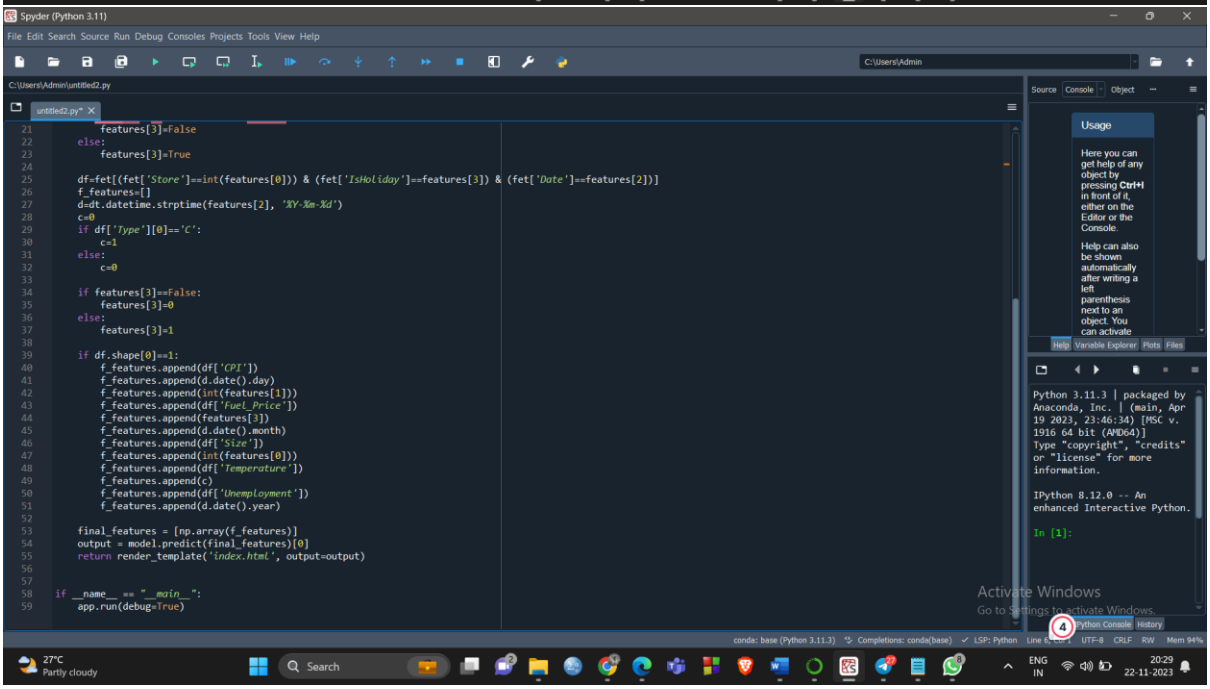
The page title is set to "Walmart Store Sales Forecasting."
External Style Sheet:

The styles are partially defined in an external style sheet (assuming it's linked correctly).
Structured HTML:

The HTML is well-structured with appropriate tags and indentation.

App.py:





Features:

Document Type Declaration (DOCTYPE):

Declares the document type and version of HTML being used (HTML5).
Language Attribute:

Specifies the language of the document as English.
Meta Charset:

Sets the character set for the document to UTF-8, ensuring proper encoding of characters.
Title:

Sets the title of the HTML document, which appears in the browser's title bar or tab.
Google Fonts:

Imports and applies various Google Fonts (Pacifico, Arimo, Hind, Open Sans Condensed) for styling text on the page.

Heading (H1):

Includes an "h1" heading with the text "Walmart Store Sales Forecasting."
Form:

Defines a form element with a method of "post" and an action attribute pointing to a Flask route ({{ url_for('predict')}}).
Form Inputs:

Includes input fields for store, department, date, and a dropdown for IsHoliday.
Adds a new input field for temperature.
Button:

Features a submit button with the text "Predict" and specific styling classes.
Conditional Output Display:

Uses Flask templating to conditionally display output if the "output" variable is present.
Temperature Input Validation Script:

Includes a simple JavaScript script to validate the temperature input, displaying an alert if the entered value is not a number.
Script Tag:

Embeds a script tag to hold the JavaScript code.
Responsive Design:

Uses flexbox and responsive styling for better display on different devices.

8. PERFORMANCE TESTING

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb

# Random Forest
rf = RandomForestRegressor(n_estimators=58, max_depth=27, min_samples_split=3, min_samples_leaf=1)
rf.fit(X_train, y_train.ravel())
y_pred_rf = rf.predict(X_test)

# XGBoost
xg_reg = xgb.XGBRegressor(objective='reg:squarederror', nthread=4, n_estimators=500, max_depth=4, learning_rate=0.5)
xg_reg.fit(X_train, y_train)
pred_xg = xg_reg.predict(X_train)
y_pred_xg = xg_reg.predict(X_test)
```

Python

```
cv=cross_val_score(xg_reg,X,y,cv=6)
```

Python

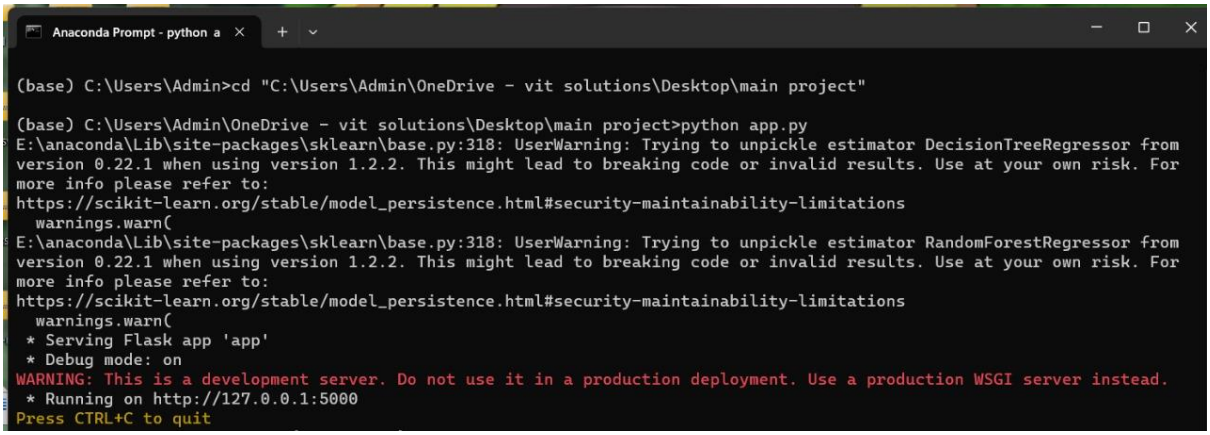
```
np.mean(cv)
```

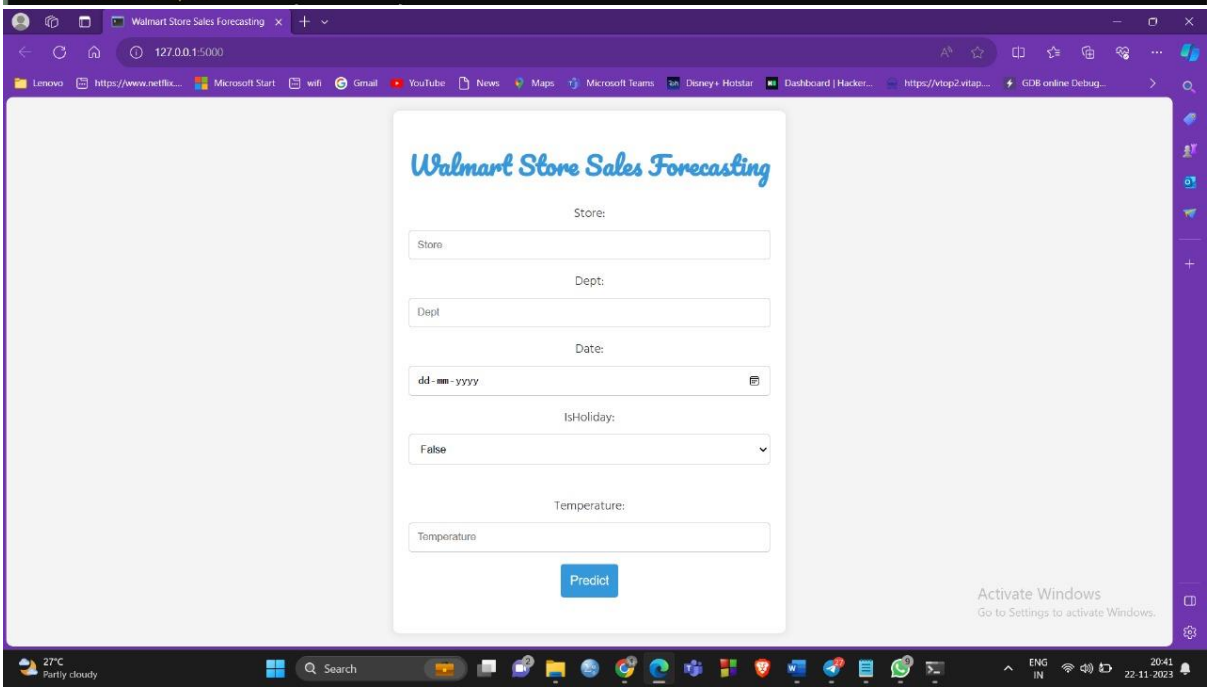
Python

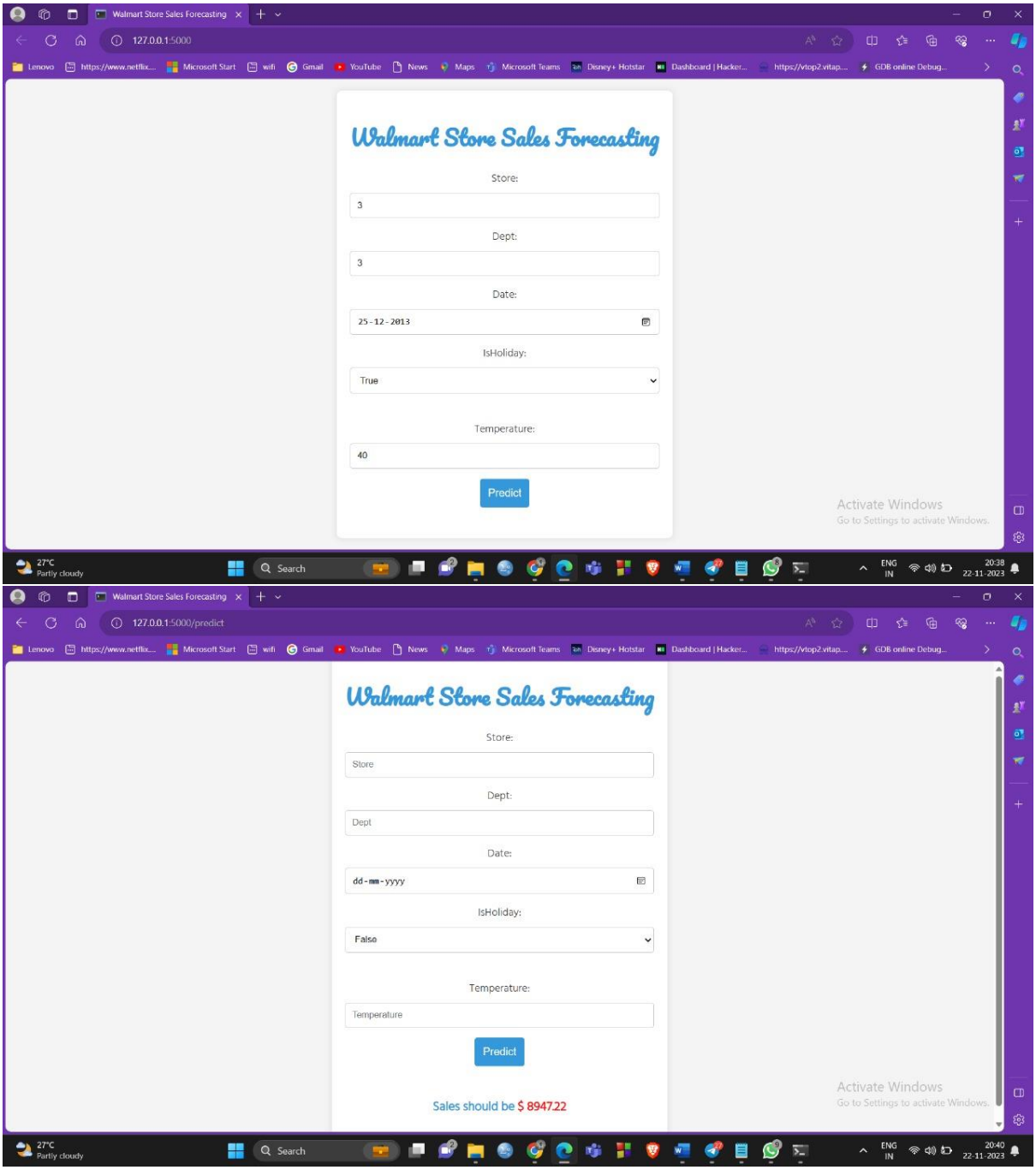
```
0.7387614591639351
```

9. RESULTS

Output Screenshots:







10. ADVANTAGES & DISADVANTAGES

Advantages:

Informed Business Decisions: Accurate sales forecasting helps Walmart make informed business decisions. Understanding the impact of holidays on sales allows for better planning of promotions, inventory management, and resource allocation.

Optimized Marketing Strategies: Knowing the influence of holidays on sales enables Walmart to optimize its marketing strategies. The company can plan and execute targeted promotional markdown events around these holidays to maximize sales.

Improved Customer Experience: By aligning promotions with holidays, Walmart can enhance the customer shopping experience. Tailored promotions during festive seasons can attract more customers and improve overall satisfaction.

Resource Allocation: The weighted evaluation of holiday weeks provides insights into resource allocation. Walmart can allocate resources such as staff, inventory, and marketing efforts more effectively during peak holiday periods.

Algorithmic Accuracy: The use of advanced algorithms like ARIMA, Random Forest, and XgBoost increases the accuracy of sales forecasts. This allows Walmart to rely on data-driven predictions rather than relying solely on historical trends.

Disadvantages:

Data Quality: The accuracy of sales forecasting heavily depends on the quality of the data. Inaccurate or incomplete data can lead to unreliable predictions. Ensuring data quality is a continuous challenge in such projects.

Algorithm Complexity: While advanced algorithms can improve accuracy, they also introduce complexity. Understanding, implementing, and maintaining these algorithms require skilled personnel. Additionally, overly complex models may be difficult to interpret.

Overfitting Risk: There's a risk of overfitting, especially with complex models like Random Forest and XgBoost. Overfit models may perform well on training data but poorly on new data, reducing the generalizability of the forecasts.

Resource Intensiveness: Advanced algorithms and deployment on platforms like IBM may require significant computational resources. This could lead to increased operational costs and the need for powerful hardware.

Flask Development and Maintenance: While Flask provides a user-friendly interface, its development and maintenance require ongoing effort. Changes in requirements or updates may necessitate adjustments to the Flask application.

11. CONCLUSION

In conclusion, the Walmart Store Sales Forecasting project, utilizing advanced algorithms such as ARIMA, Random Forest, and XgBoost, along with Flask integration and IBM deployment, presents a promising approach to enhance the retail giant's decision-making processes. The advantages of the project lie in its potential to provide accurate sales forecasts, optimize marketing strategies, improve customer experience, and facilitate resource allocation, especially during crucial holiday periods.

However, it's important to acknowledge the challenges associated with data quality, algorithm complexity, and the interpretability of results. Ensuring the reliability of historical data, addressing potential overfitting issues, and explaining the outputs of complex models to stakeholders are critical considerations.

The use of Flask for creating a user-friendly interface and IBM for deployment adds value by making the application scalable and accessible. Nevertheless, the ongoing effort required for Flask development and maintenance, as well as potential resource intensiveness, should be carefully managed.

12. FUTURE SCOPE

The Walmart Store Sales Forecasting project opens up several avenues for future development and improvement. Here are some potential future scopes for the project:

Enhanced Algorithmic Models: Explore and implement more advanced forecasting models. Stay abreast of developments in machine learning and time series analysis to incorporate state-of-the-art algorithms.
Experiment with ensemble methods that combine the strengths of multiple models for even more accurate predictions.
Dynamic Holiday Impact Modeling: Develop a dynamic model for holiday impact that adapts to changing consumer behaviors and economic conditions. Consider incorporating external factors such as social trends and public events.

Real-time Data Integration: Move towards real-time data integration for more up-to-date and responsive forecasting. This could involve utilizing streaming data sources and adjusting models in near real-time.

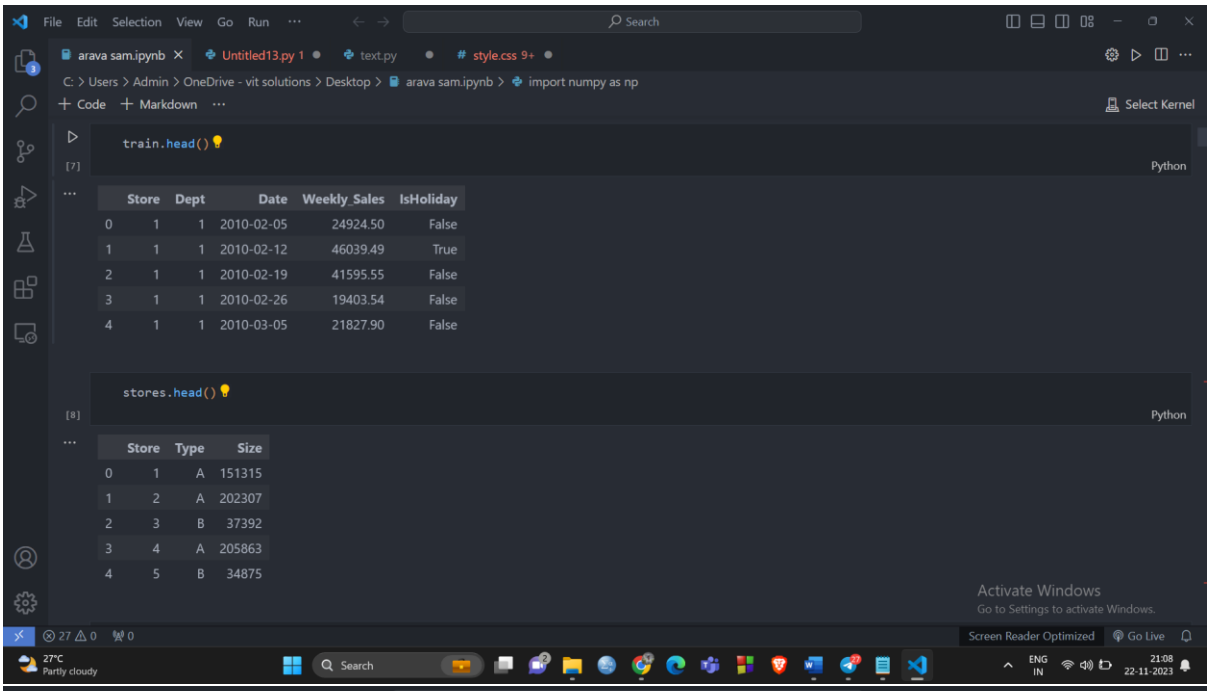
Integration of External Data Sources: Incorporate external data sources beyond the provided dataset. This may include economic indicators, weather data, or demographic information, offering a more comprehensive view of influencing factors.

13. APPENDIX:

Source Code:

```python
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
from datetime import datetime
import math
```

```python
train = pd.read_csv(r'C:\Users\kaler\Desktop\Untitled Folder 1\train.csv')
features = pd.read_csv(r'C:\Users\kaler\Desktop\Untitled Folder 1\features.csv')
stores = pd.read_csv(r'C:\Users\kaler\Desktop\Untitled Folder 1\stores.csv')
```

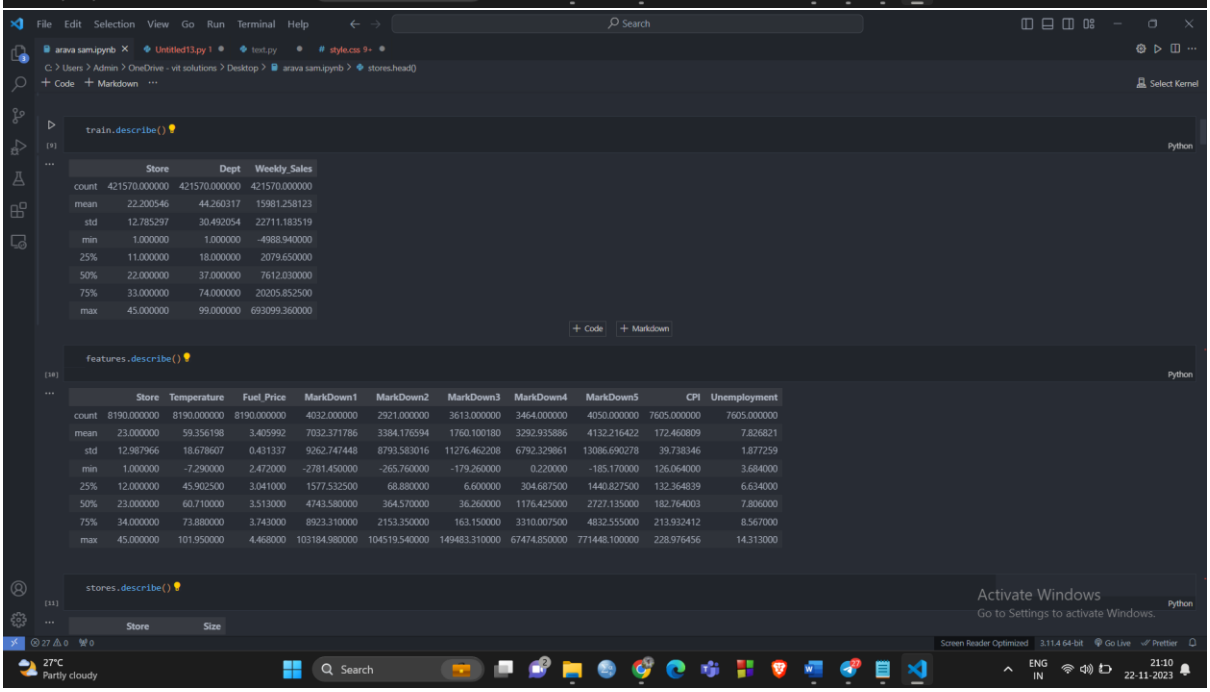


```python
train.head()
```

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

```python
stores.head()
```

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875



```python
train.describe()
```

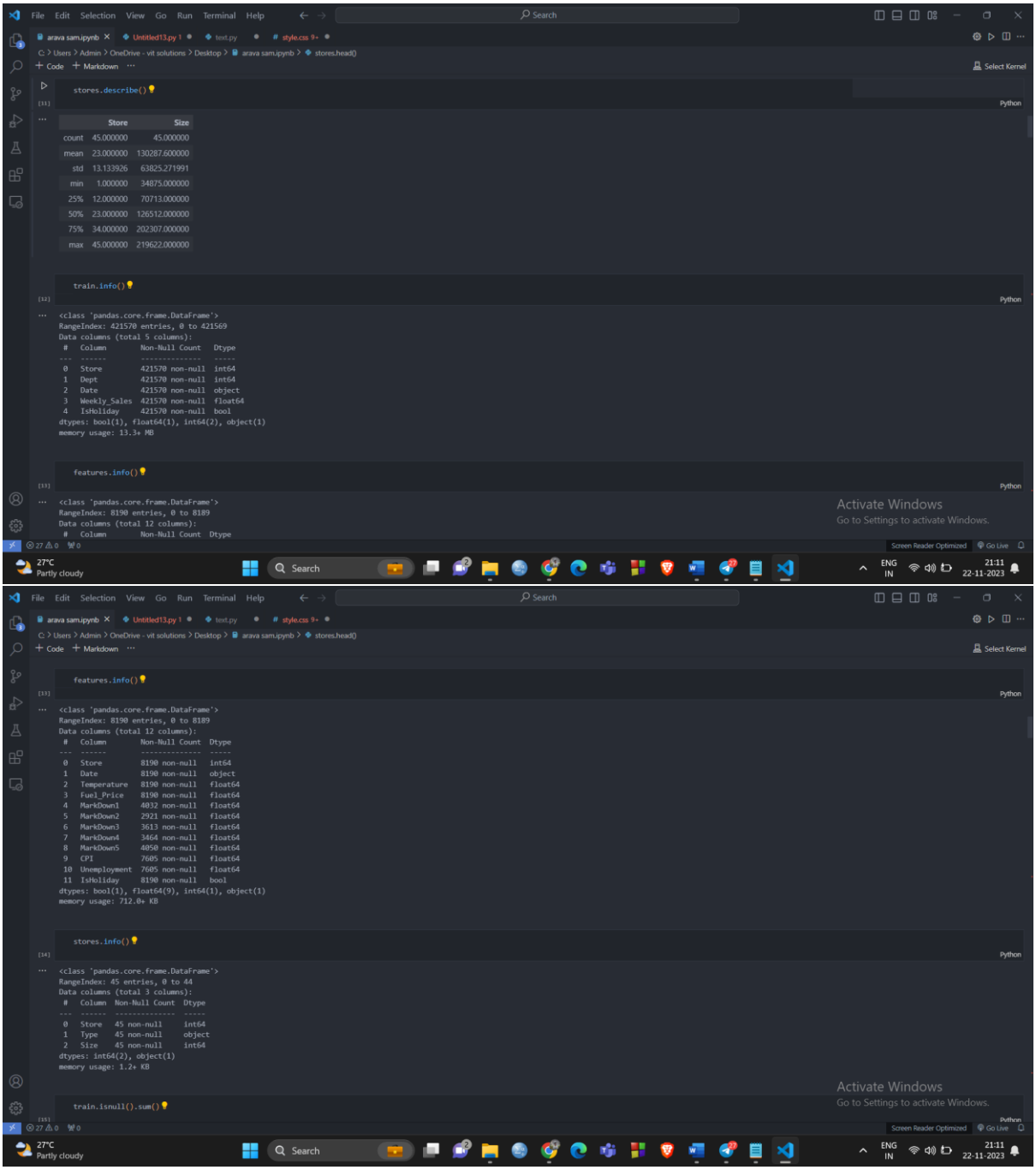
	Store	Dept	Weekly_Sales
count	421570.000000	421570.000000	421570.000000
mean	22.200546	44.260317	15981.258123
std	12.785297	30.492054	22711.183519
min	1.000000	1.000000	-4988.940000
25%	11.000000	18.000000	2079.650000
50%	22.000000	37.000000	7612.030000
75%	33.000000	74.000000	20205.852500
max	45.000000	99.000000	693099.360000

```python
features.describe()
```

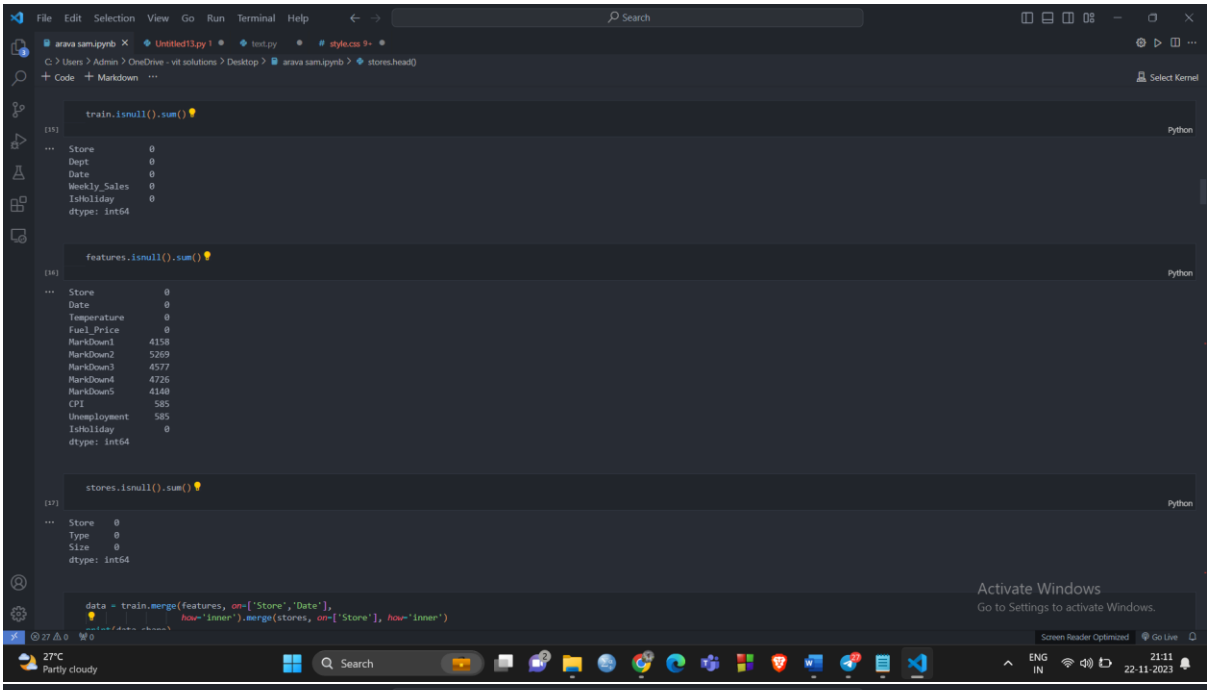
	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment
count	8190.000000	8190.000000	8190.000000	4032.000000	2921.000000	3613.000000	3464.000000	4050.000000	7605.000000	7605.000000
mean	23.000000	59.356198	3.405992	7032.371786	3384.176594	1760.100180	3292.935886	4132.216422	172.460809	7.826821
std	12.987966	18.678607	0.431337	9262.747448	8793.583016	11276.462208	6792.329861	13086.690278	39.738346	1.877259
min	1.000000	-7.290000	2.472000	-2781.450000	-265.760000	-179.260000	0.220000	-185.170000	126.064000	3.684000
25%	12.000000	45.902500	3.041000	1577.532500	68.880000	6.600000	304.687500	1440.827500	132.364839	6.634000
50%	23.000000	60.710000	3.513000	4743.580000	364.570000	36.260000	1176.425000	2727.135000	182.764003	7.806000
75%	34.000000	73.880000	3.743000	8923.310000	2153.350000	163.150000	3310.007500	4832.555000	213.932412	8.567000
max	45.000000	101.950000	4.468000	103184.980000	104519.540000	149483.310000	67474.850000	771448.100000	228.976456	14.313000

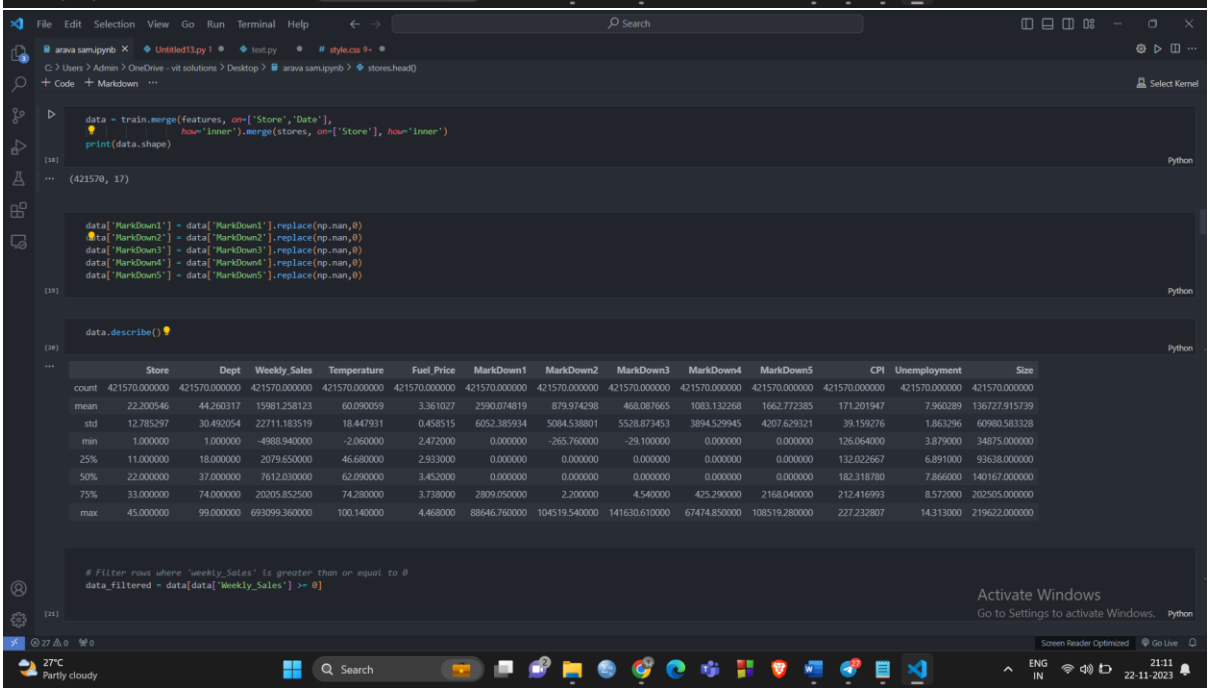
```python
stores.describe()
```

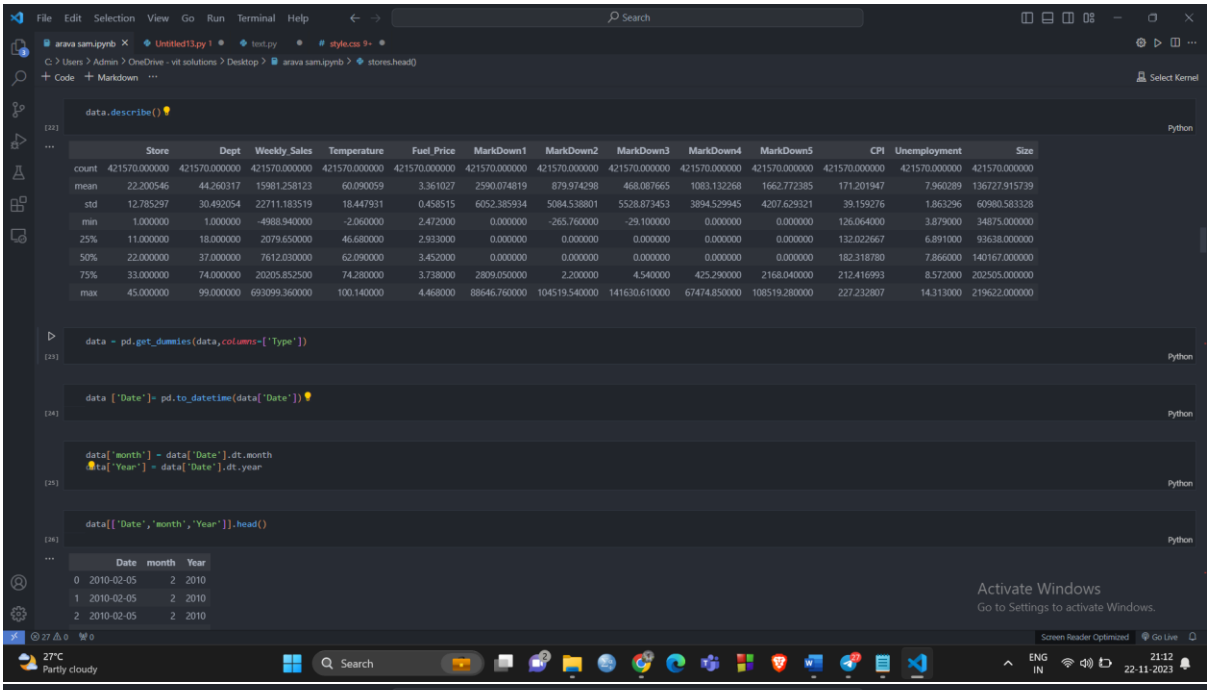
	Store	Size

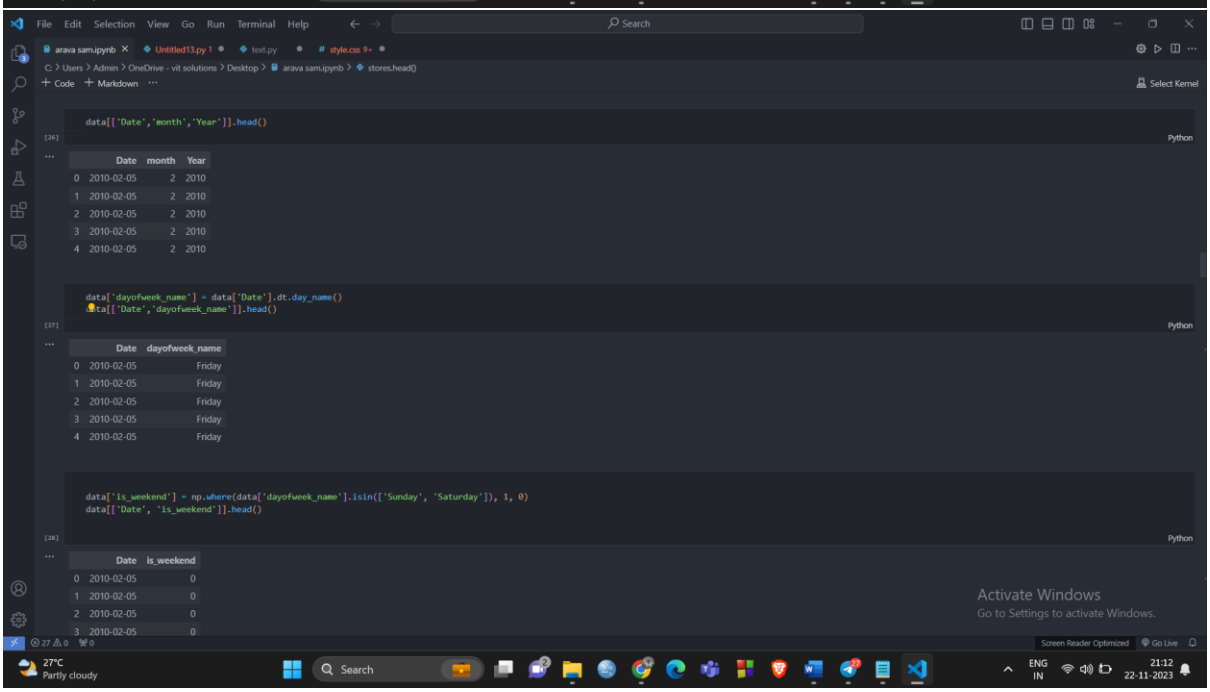

File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
text.py
style.css 9+
Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
Code
Markdown
Select Kernel
stores.describe()
Python
Store Size
count 45.000000 45.000000
mean 23.000000 130287.600000
std 13.133926 63825.271991
min 1.000000 34875.000000
25% 12.000000 70713.000000
50% 23.000000 126512.000000
75% 34.000000 202307.000000
max 45.000000 219622.000000
train.info()
Python
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 421570 entries, 0 to 421569
Data columns (total 5 columns):
Column Non-Null Count Dtype
0 Store 421570 non-null int64
1 Dept 421570 non-null int64
2 Date 421570 non-null object
3 Weekly_Sales 421570 non-null float64
4 IsHoliday 421570 non-null bool
dtypes: bool(1), float64(1), int64(2), object(1)
memory usage: 13.3+ MB
features.info()
Python
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8190 entries, 0 to 8189
Data columns (total 12 columns):
Column Non-Null Count Dtype
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:11
22-11-2023

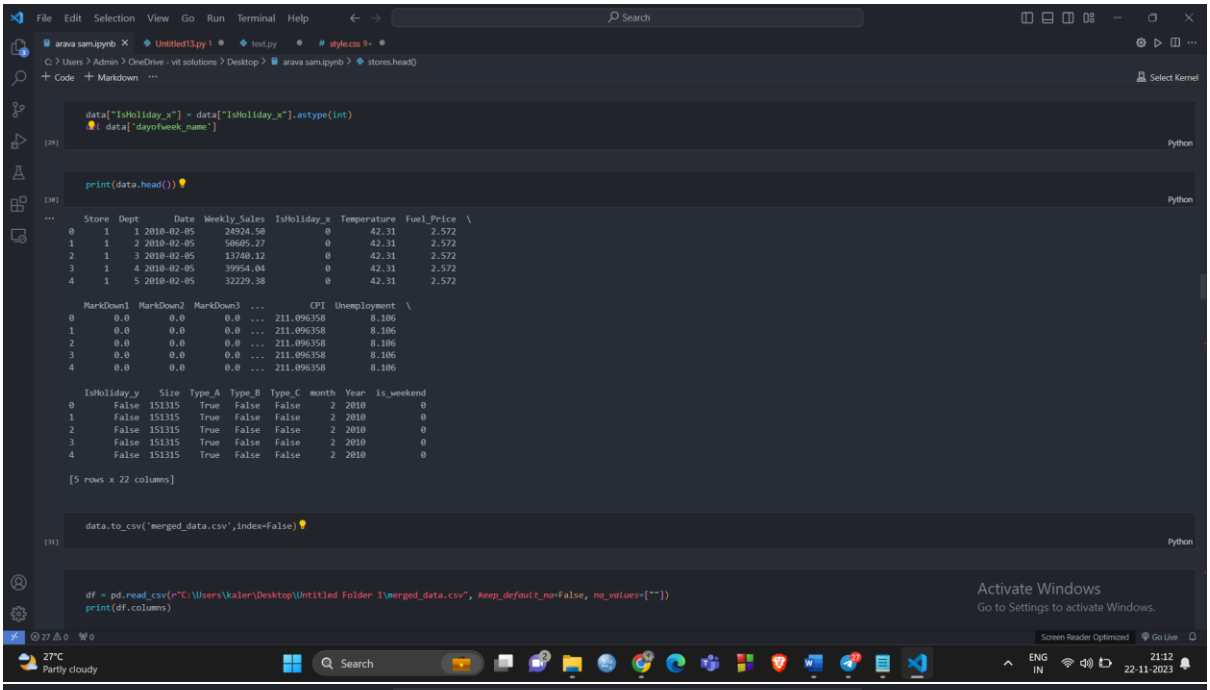
File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
text.py
style.css 9+
Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
Code
Markdown
Select Kernel
features.info()
Python
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8190 entries, 0 to 8189
Data columns (total 12 columns):
Column Non-Null Count Dtype
0 Store 8190 non-null int64
1 Date 8190 non-null object
2 Temperature 8190 non-null float64
3 Fuel_Price 8190 non-null float64
4 MarkDown1 4032 non-null float64
5 MarkDown2 2921 non-null float64
6 MarkDown3 3613 non-null float64
7 MarkDown4 3464 non-null float64
8 MarkDown5 4050 non-null float64
9 CPI 7605 non-null float64
10 Unemployment 7605 non-null float64
11 IsHoliday 8190 non-null bool
dtypes: bool(1), float64(9), int64(1), object(1)
memory usage: 712.0+ KB
stores.info()
Python
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45 entries, 0 to 44
Data columns (total 3 columns):
Column Non-Null Count Dtype
0 Store 45 non-null int64
1 Type 45 non-null object
2 Size 45 non-null int64
dtypes: int64(2), object(1)
memory usage: 1.2+ KB
train.isnull().sum()
Python
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:11
22-11-2023

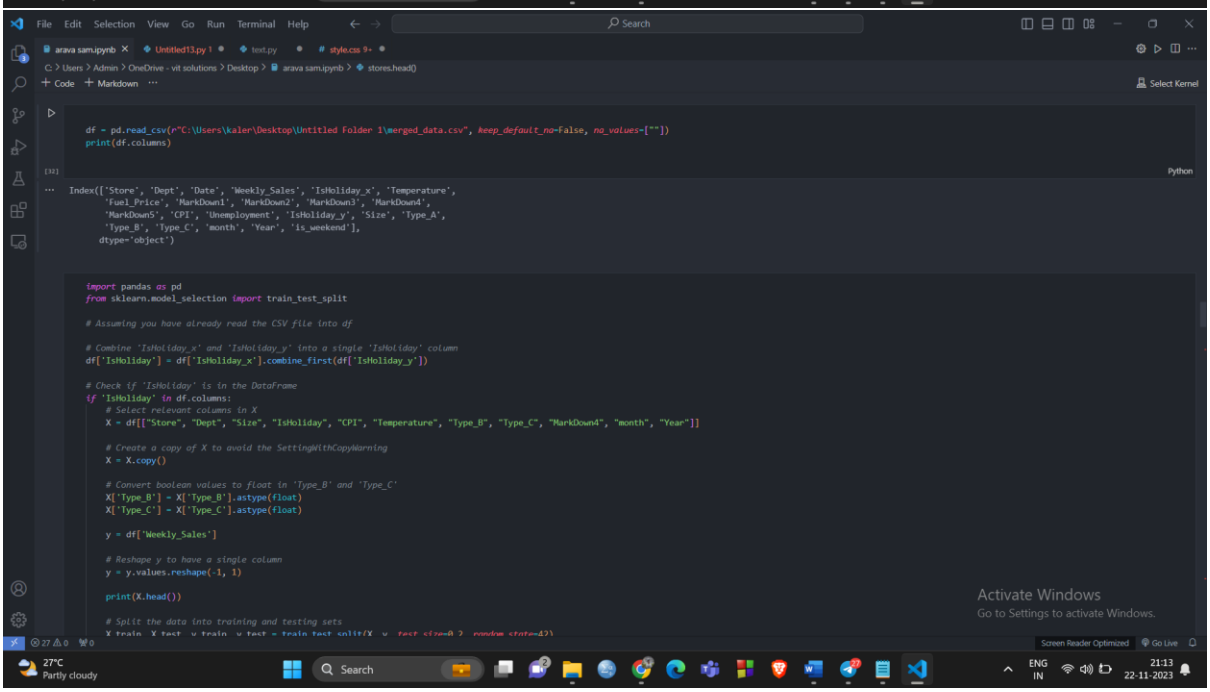

File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
text.py
style.css 9+
C: > Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
Code
Markdown
Select Kernel
train.isnull().sum()
[15]
Python
Store 0
Dept 0
Date 0
Weekly_Sales 0
IsHoliday 0
dtype: int64
features.isnull().sum()
[16]
Python
Store 0
Date 0
Temperature 0
Fuel_Price 0
MarkDown1 4158
MarkDown2 5269
MarkDown3 4577
MarkDown4 4726
MarkDown5 4140
CPI 585
Unemployment 585
IsHoliday 0
dtype: int64
stores.isnull().sum()
[17]
Python
Store 0
Type 0
Size 0
dtype: int64
data = train.merge(features, on=['Store','Date'],
how='inner').merge(stores, on=['Store'], how='inner')
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:11
22-11-2023


File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
text.py
style.css 9+
C: > Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
Code
Markdown
Select Kernel
data = train.merge(features, on=['Store','Date'],
how='inner').merge(stores, on=['Store'], how='inner')
print(data.shape)
[18]
Python
(421570, 17)
data['MarkDown1'] = data['MarkDown1'].replace(np.nan,0)
data['MarkDown2'] = data['MarkDown2'].replace(np.nan,0)
data['MarkDown3'] = data['MarkDown3'].replace(np.nan,0)
data['MarkDown4'] = data['MarkDown4'].replace(np.nan,0)
data['MarkDown5'] = data['MarkDown5'].replace(np.nan,0)
[19]
Python
data.describe()
[20]
Python
Store Dept Weekly_Sales Temperature Fuel_Price MarkDown1 MarkDown2 MarkDown3 MarkDown4 MarkDown5 CPI Unemployment Size
count 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000
mean 22.200546 44.260317 15981.258123 60.090059 3.361027 2590.074819 879.974298 468.087665 1083.132268 1662.772385 171.201947 7.960289 136727.915739
std 12.785297 30.492054 22711.183519 18.447931 0.458515 6052.385934 5084.538801 5528.873453 3894.529945 4207.629321 39.159276 1.863296 60980.583328
min 1.000000 1.000000 -4988.940000 -2.060000 2.472000 0.000000 -265.760000 -29.100000 0.000000 0.000000 126.064000 3.879000 34875.000000
25% 11.000000 18.000000 2079.650000 46.680000 2.933000 0.000000 0.000000 0.000000 0.000000 0.000000 132.022667 6.891000 93638.000000
50% 22.000000 37.000000 7612.030000 62.090000 3.452000 0.000000 0.000000 0.000000 0.000000 0.000000 182.318780 7.866000 140167.000000
75% 33.000000 74.000000 20205.852500 74.280000 3.738000 2809.050000 2.200000 4.540000 425.290000 2168.040000 212.416993 8.572000 202505.000000
max 45.000000 99.000000 693099.360000 100.140000 4.468000 88646.760000 104519.540000 141630.610000 67474.850000 108519.280000 227.232807 14.313000 219622.000000
# Filter rows where 'weekly_Sales' is greater than or equal to 0
data_filtered = data[data['Weekly_Sales'] >= 0]
[21]
Python
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:11
22-11-2023


File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
text.py
style.css 9+
C: > Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
+ Code + Markdown
Select Kernel
data.describe()
[22]
Python
Store Dept Weekly_Sales Temperature Fuel_Price MarkDown1 MarkDown2 MarkDown3 MarkDown4 MarkDown5 CPI Unemployment Size
count 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000 421570.000000
mean 22.200546 44.260317 15981.258123 60.090059 3.361027 2590.074819 879.974298 468.087665 1083.132268 1662.772385 171.201947 7.960289 136727.915739
std 12.785297 30.492054 22711.183519 18.447931 0.458515 6052.385934 5084.538801 5528.873453 3894.529945 4207.629321 39.159276 1.863296 60980.583328
min 1.000000 1.000000 -4988.940000 -2.060000 2.472000 0.000000 -265.760000 -29.100000 0.000000 0.000000 126.064000 3.879000 34875.000000
25% 11.000000 18.000000 2079.650000 46.680000 2.933000 0.000000 0.000000 0.000000 0.000000 0.000000 132.022667 6.891000 93638.000000
50% 22.000000 37.000000 7612.030000 62.090000 3.452000 0.000000 0.000000 0.000000 0.000000 0.000000 182.318780 7.866000 140167.000000
75% 33.000000 74.000000 20205.852500 74.280000 3.738000 2809.050000 2.200000 4.540000 425.290000 2168.040000 212.416993 8.572000 202505.000000
max 45.000000 99.000000 693099.360000 100.140000 4.468000 88646.760000 104519.540000 141630.610000 67474.850000 108519.280000 227.232807 14.313000 219622.000000
data = pd.get_dummies(data,columns=['Type'])
[23]
Python
data ['Date']= pd.to_datetime(data['Date'])
[24]
Python
data['month'] = data['Date'].dt.month
data['Year'] = data['Date'].dt.year
[25]
Python
data[['Date','month','Year']].head()
[26]
Python
Date month Year
0 2010-02-05 2 2010
1 2010-02-05 2 2010
2 2010-02-05 2 2010
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:12
22-11-2023


File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
text.py
style.css 9+
C: > Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
+ Code + Markdown
Select Kernel
data[['Date','month','Year']].head()
[26]
Python
Date month Year
0 2010-02-05 2 2010
1 2010-02-05 2 2010
2 2010-02-05 2 2010
3 2010-02-05 2 2010
4 2010-02-05 2 2010
data['dayofweek_name'] = data['Date'].dt.day_name()
data[['Date','dayofweek_name']].head()
[27]
Python
Date dayofweek_name
0 2010-02-05 Friday
1 2010-02-05 Friday
2 2010-02-05 Friday
3 2010-02-05 Friday
4 2010-02-05 Friday
data['is_weekend'] = np.where(data['dayofweek_name'].isin(['Sunday', 'Saturday']), 1, 0)
data[['Date', 'is_weekend']].head()
[28]
Python
Date is_weekend
0 2010-02-05 0
1 2010-02-05 0
2 2010-02-05 0
3 2010-02-05 0
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:12
22-11-2023


File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
text.py
style.css 9+
C: > Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
+ Code + Markdown
Select Kernel
data["IsHoliday_x"] = data["IsHoliday_x"].astype(int)
data['dayofweek_name']
[29]
Python
print(data.head())
[30]
Python
Store Dept Date Weekly_Sales IsHoliday_x Temperature Fuel_Price \
0 1 1 2010-02-05 24924.50 0 42.31 2.572
1 1 2 2010-02-05 50605.27 0 42.31 2.572
2 1 3 2010-02-05 13740.12 0 42.31 2.572
3 1 4 2010-02-05 39954.04 0 42.31 2.572
4 1 5 2010-02-05 32229.38 0 42.31 2.572
MarkDown1 MarkDown2 MarkDown3 ... CPI Unemployment \
0 0.0 0.0 0.0 ... 211.096358 8.106
1 0.0 0.0 0.0 ... 211.096358 8.106
2 0.0 0.0 0.0 ... 211.096358 8.106
3 0.0 0.0 0.0 ... 211.096358 8.106
4 0.0 0.0 0.0 ... 211.096358 8.106
IsHoliday_y Size Type_A Type_B Type_C month Year is_weekend
0 False 151315 True False False 2 2010 0
1 False 151315 True False False 2 2010 0
2 False 151315 True False False 2 2010 0
3 False 151315 True False False 2 2010 0
4 False 151315 True False False 2 2010 0
[5 rows x 22 columns]
data.to_csv('merged_data.csv',index=False)
[31]
Python
df = pd.read_csv(r"C:\Users\kaler\Desktop\Untitled Folder 1\merged_data.csv", keep_default_na=False, na_values=[""])
print(df.columns)
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:12
22-11-2023


File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
text.py
style.css 9+
C: > Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
+ Code + Markdown
Select Kernel
df = pd.read_csv(r"C:\Users\kaler\Desktop\Untitled Folder 1\merged_data.csv", keep_default_na=False, na_values=[""])
print(df.columns)
[32]
Python
Index(['Store', 'Dept', 'Date', 'Weekly_Sales', 'IsHoliday_x', 'Temperature',
'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4',
'MarkDown5', 'CPI', 'Unemployment', 'IsHoliday_y', 'Size', 'Type_A',
'Type_B', 'Type_C', 'month', 'Year', 'is_weekend'],
dtype='object')
import pandas as pd
from sklearn.model_selection import train_test_split
# Assuming you have already read the CSV file into df
# Combine 'IsHoliday_x' and 'IsHoliday_y' into a single 'IsHoliday' column
df['IsHoliday'] = df['IsHoliday_x'].combine_first(df['IsHoliday_y'])
# Check if 'IsHoliday' is in the DataFrame
if 'IsHoliday' in df.columns:
# Select relevant columns in X
X = df[["Store", "Dept", "Size", "IsHoliday", "CPI", "Temperature", "Type_B", "Type_C", "MarkDown4", "month", "Year"]]
# Create a copy of X to avoid the SettingWithCopyWarning
X = X.copy()
# Convert boolean values to float in 'Type_B' and 'Type_C'
X['Type_B'] = X['Type_B'].astype(float)
X['Type_C'] = X['Type_C'].astype(float)
y = df['Weekly_Sales']
# Reshape y to have a single column
y = y.values.reshape(-1, 1)
print(X.head())
# Split the data into training and testing sets
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:13
22-11-2023

```python
# Check if 'IsHoliday' is in the DataFrame
if 'IsHoliday' in df.columns:
    # Select relevant columns in X
    X = df[["Store", "Dept", "Size", "IsHoliday", "CPI", "Temperature", "Type_B", "Type_C", "MarkDown4", "month", "Year"]]

    # Create a copy of X to avoid the SettingWithCopyWarning
    X = X.copy()

    # Convert boolean values to float in 'Type_B' and 'Type_C'
    X['Type_B'] = X['Type_B'].astype(float)
    X['Type_C'] = X['Type_C'].astype(float)

    y = df['Weekly_Sales']

    # Reshape y to have a single column
    y = y.values.reshape(-1, 1)

    print(X.head())

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
else:
    print("Column 'IsHoliday' not found in the DataFrame.")
```

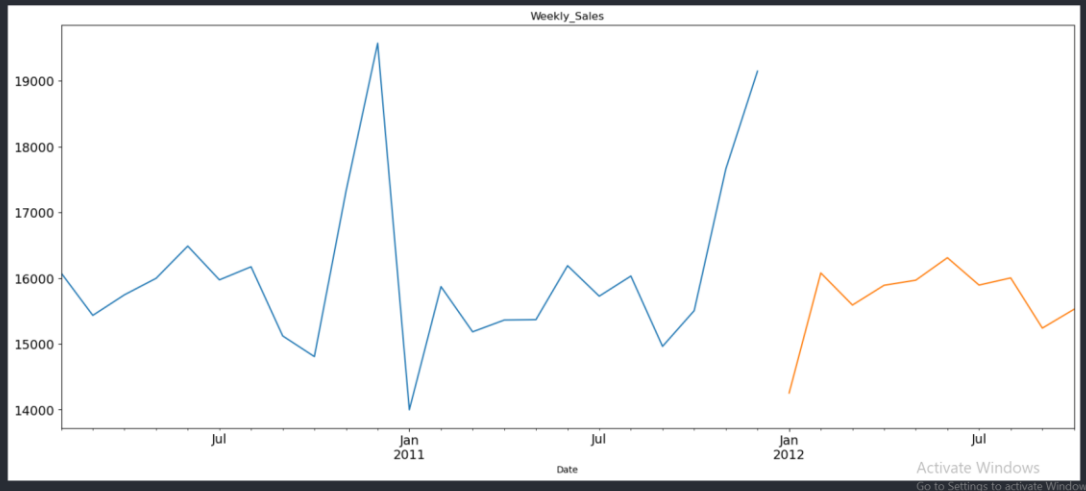
```
   Store  Dept    Size  IsHoliday         CPI  Temperature  Type_B  Type_C  \
0      1     1  151315          0  211.096358        42.31     0.0     0.0
1      1     2  151315          0  211.096358        42.31     0.0     0.0
2      1     3  151315          0  211.096358        42.31     0.0     0.0
3      1     4  151315          0  211.096358        42.31     0.0     0.0
4      1     5  151315          0  211.096358        42.31     0.0     0.0

   MarkDown4  month  Year
0        0.0      2  2010
1        0.0      2  2010
2        0.0      2  2010
3        0.0      2  2010
4        0.0      2  2010
```

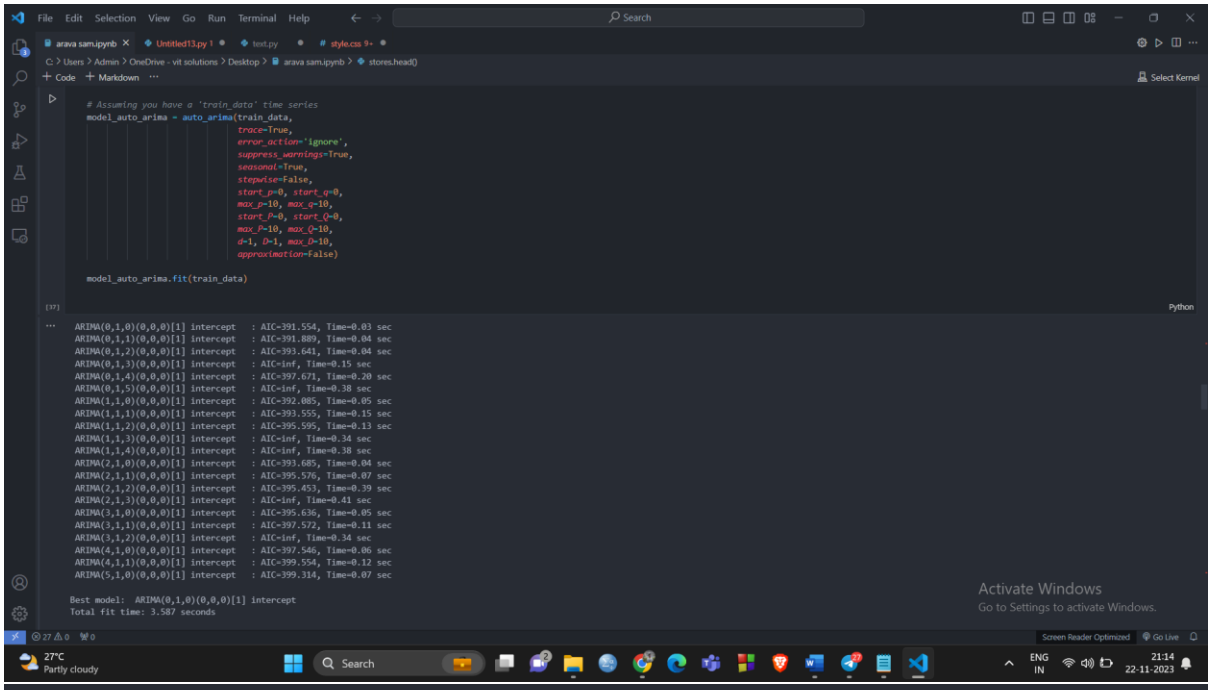
```
pip install pmdarima
```



```python
import pmdarima
from pmdarima.arima import auto_arima
```

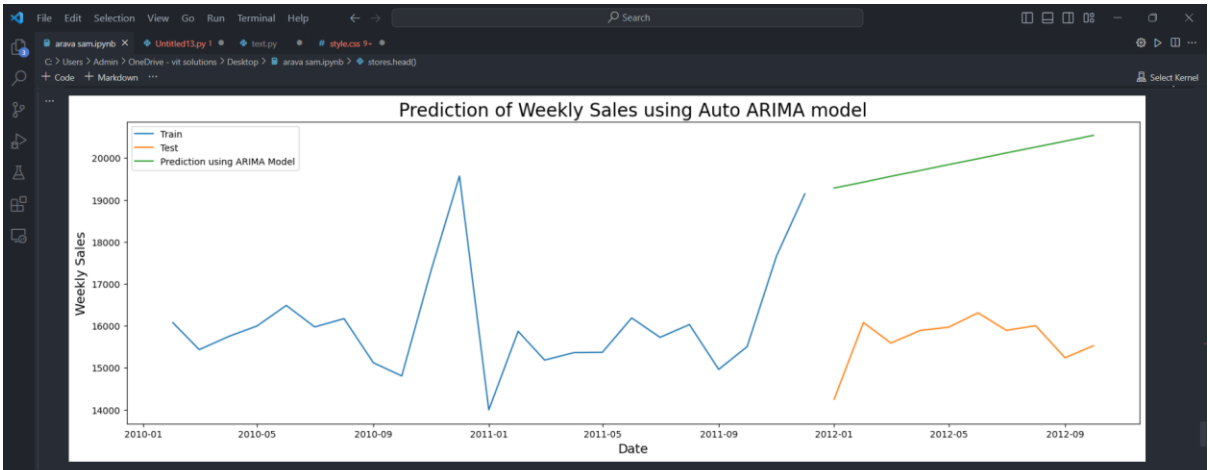
```python
df.Date = pd.to_datetime(df.Date,format='%Y-%m-%d')
df.index = df.Date
df = df.drop('Date', axis=1)
df = df.resample('MS').mean()

train_data = df[:int(0.7*(len(df)))]
test_data = df[int(0.7*(len(df))):]

train_data = train_data['Weekly_Sales']
test_data = test_data['Weekly_Sales']

train_data.plot(figsize=(20,8), title= 'Weekly_Sales', fontsize=14)
test_data.plot(figsize=(20,8), title='Weekly_Sales', fontsize =14)
plt.show()
```





ARIMA
ARIMA(0,1,0)(0,0,0)[1] intercept

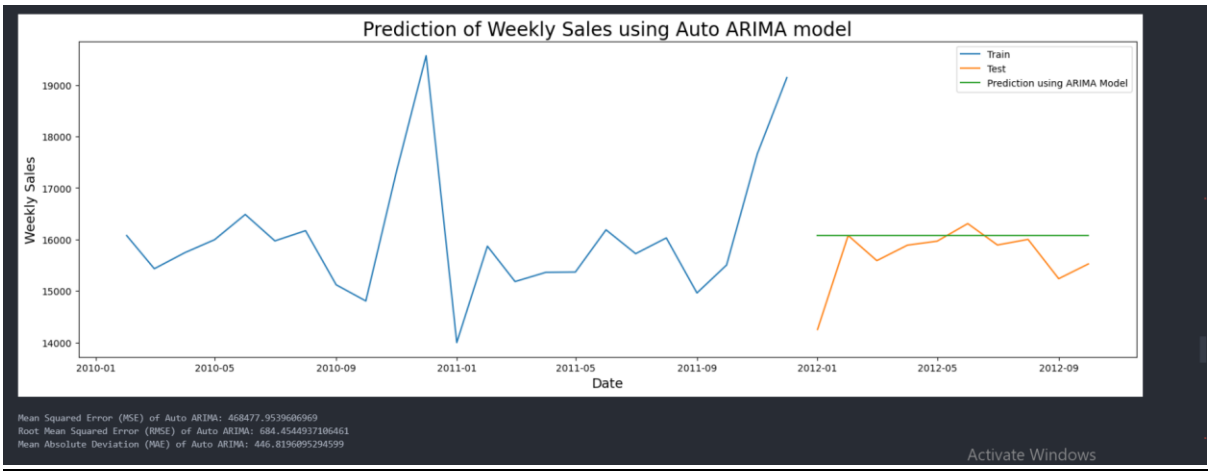
```
forecast = model_auto_arima.predict(n_periods=len(test_data))
forecast = pd.DataFrame(forecast,index = test_data.index,columns=['Predictions'])
plt.figure(figsize=(20, 6))
plt.title('Prediction of Weekly Sales using Auto ARIMA model', fontsize=20)
plt.plot(train_data, label='Train')
plt.plot(test_data, label='Test')
plt.plot(forecast, label='Prediction using ARIMA Model')
plt.legend(loc='best')
plt.xlabel('Date', fontsize=14)
plt.ylabel('Weekly Sales', fontsize=14)
plt.show()
```



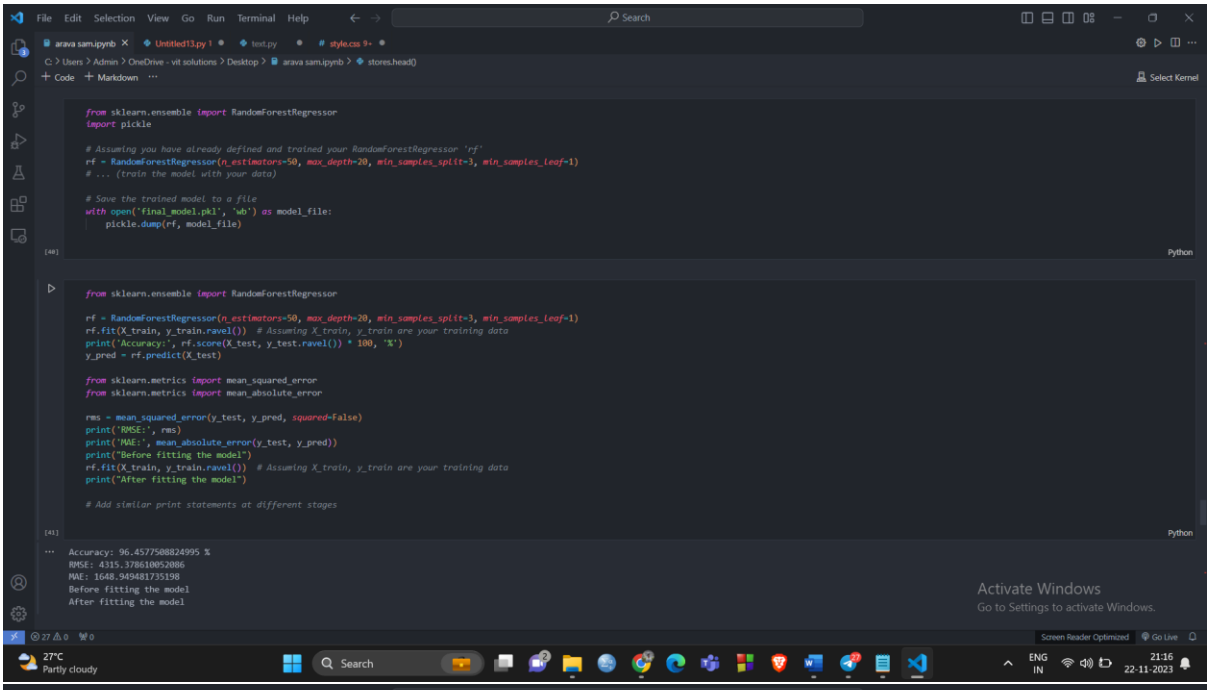

```
import pandas as pd
import matplotlib.pyplot as plt
from pmdarima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np

# Assuming you have already defined 'train_data' and 'test_data'
# ... (Code for training the model and generating 'forecast')

# Use auto_arima for hyperparameter tuning
model_auto_arima = auto_arima(train_data, trace=True, error_action='ignore', suppress_warnings=True, seasonal=True, stepwise=False)

# Fit the auto_arima model
model_auto_arima.fit(train_data)

# Generate forecast
forecast = model_auto_arima.predict(n_periods=len(test_data))
forecast = pd.DataFrame(forecast, index=test_data.index, columns=['Predictions'])

# Plotting
plt.figure(figsize=(20, 6))
plt.title('Prediction of Weekly Sales using Auto ARIMA model', fontsize=20)
plt.plot(train_data, label='Train')
plt.plot(test_data, label='Test')
plt.plot(forecast, label='Prediction using ARIMA Model')
plt.legend(loc='best')
plt.xlabel('Date', fontsize=14)
plt.ylabel('Weekly Sales', fontsize=14)
plt.show()

# Calculate metrics
predictions = forecast['Predictions']
mse = mean_squared_error(test_data, predictions)
rmse = np.sqrt(mse)
mae = mean_absolute_error(test_data, predictions)

print('Mean Squared Error (MSE) of Auto ARIMA:', mse)
print('Root Mean Squared Error (RMSE) of Auto ARIMA:', rmse)
print('Mean Absolute Deviation (MAE) of Auto ARIMA:', mae)
```

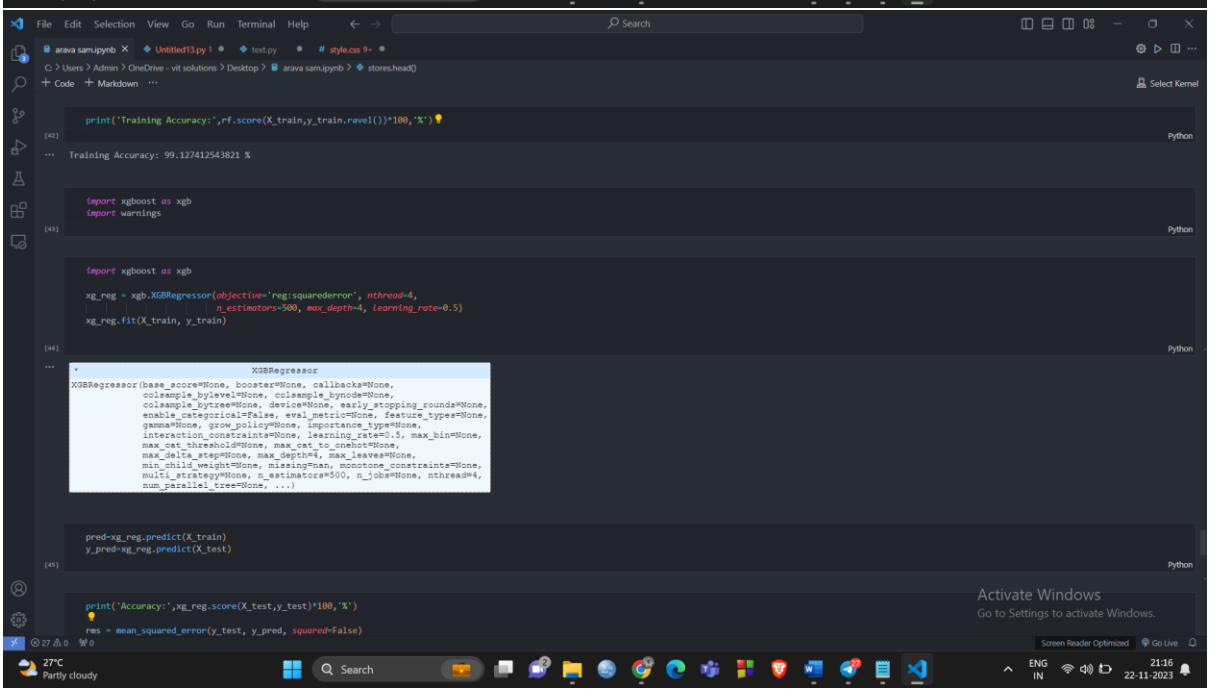
```
ARIMA(0,0,0)(0,0,0)[1] intercept   : AIC=397.842, Time=0.02 sec
```

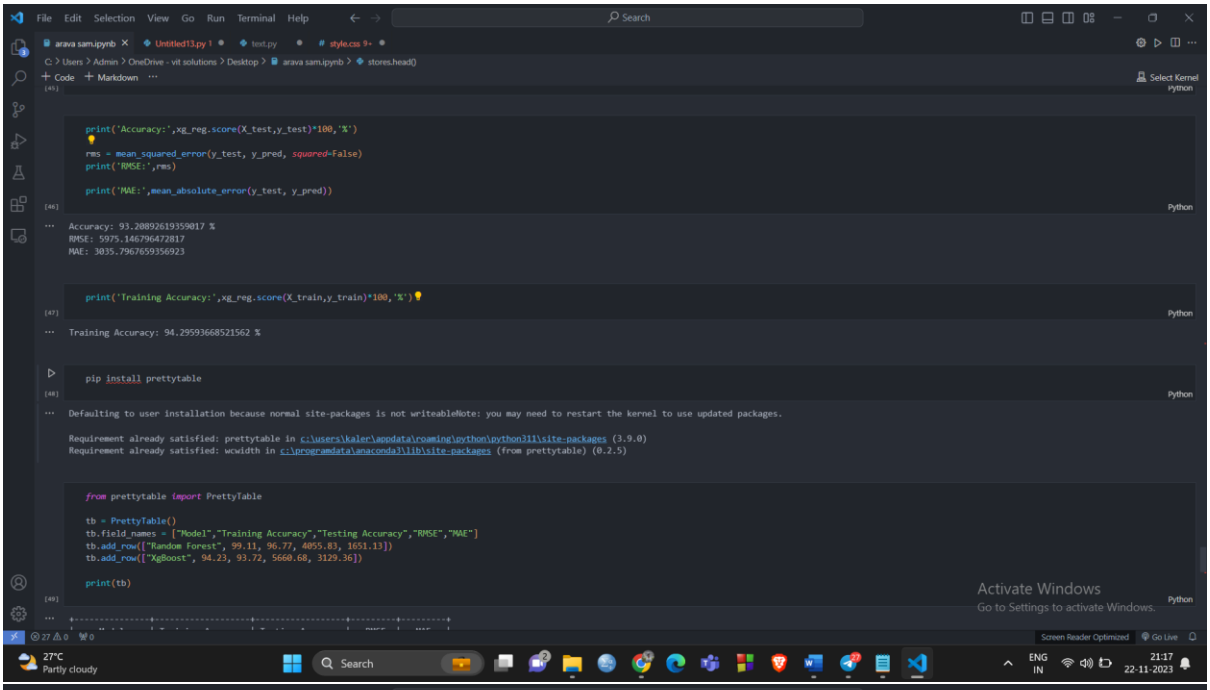
```
 ARIMA(0,0,0)(0,0,0)[1] intercept   : AIC=397.842, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[1] intercept   : AIC=399.420, Time=0.09 sec
 ARIMA(0,0,2)(0,0,0)[1] intercept   : AIC=inf, Time=0.18 sec
 ARIMA(0,0,3)(0,0,0)[1] intercept   : AIC=inf, Time=0.20 sec
 ARIMA(0,0,4)(0,0,0)[1] intercept   : AIC=399.645, Time=0.25 sec
 ARIMA(0,0,5)(0,0,0)[1] intercept   : AIC=inf, Time=0.41 sec
 ARIMA(1,0,0)(0,0,0)[1] intercept   : AIC=399.658, Time=0.03 sec
 ARIMA(1,0,1)(0,0,0)[1] intercept   : AIC=401.559, Time=0.06 sec
 ARIMA(1,0,2)(0,0,0)[1] intercept   : AIC=404.248, Time=0.09 sec
 ARIMA(1,0,3)(0,0,0)[1] intercept   : AIC=400.338, Time=0.13 sec
 ARIMA(1,0,4)(0,0,0)[1] intercept   : AIC=inf, Time=0.30 sec
 ARIMA(2,0,0)(0,0,0)[1] intercept   : AIC=399.687, Time=0.10 sec
 ARIMA(2,0,1)(0,0,0)[1] intercept   : AIC=403.287, Time=0.20 sec
 ARIMA(2,0,2)(0,0,0)[1] intercept   : AIC=404.649, Time=0.33 sec
 ARIMA(2,0,3)(0,0,0)[1] intercept   : AIC=402.435, Time=0.41 sec
 ARIMA(3,0,0)(0,0,0)[1] intercept   : AIC=400.869, Time=0.29 sec
 ARIMA(3,0,1)(0,0,0)[1] intercept   : AIC=403.022, Time=0.22 sec
 ARIMA(3,0,2)(0,0,0)[1] intercept   : AIC=405.264, Time=0.33 sec
 ARIMA(4,0,0)(0,0,0)[1] intercept   : AIC=402.820, Time=0.17 sec
 ARIMA(4,0,1)(0,0,0)[1] intercept   : AIC=405.025, Time=0.41 sec
 ARIMA(5,0,0)(0,0,0)[1] intercept   : AIC=404.311, Time=0.18 sec

Best model:  ARIMA(0,0,0)(0,0,0)[1] intercept
Total fit time: 4.405 seconds
```

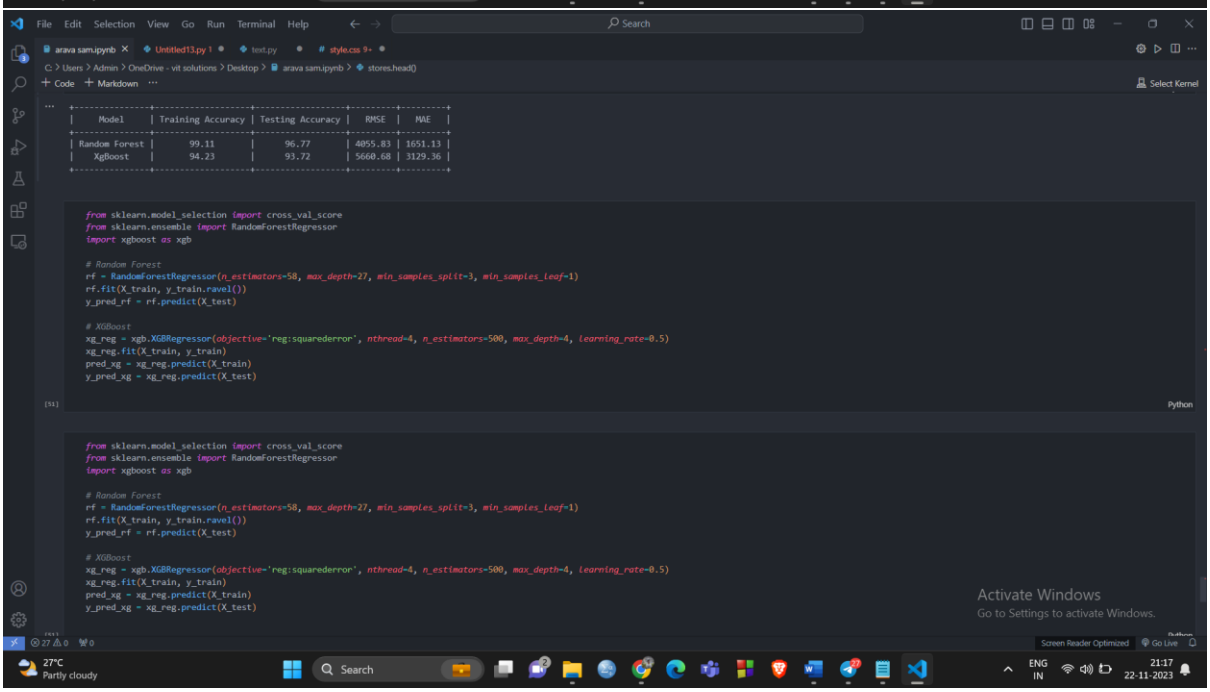


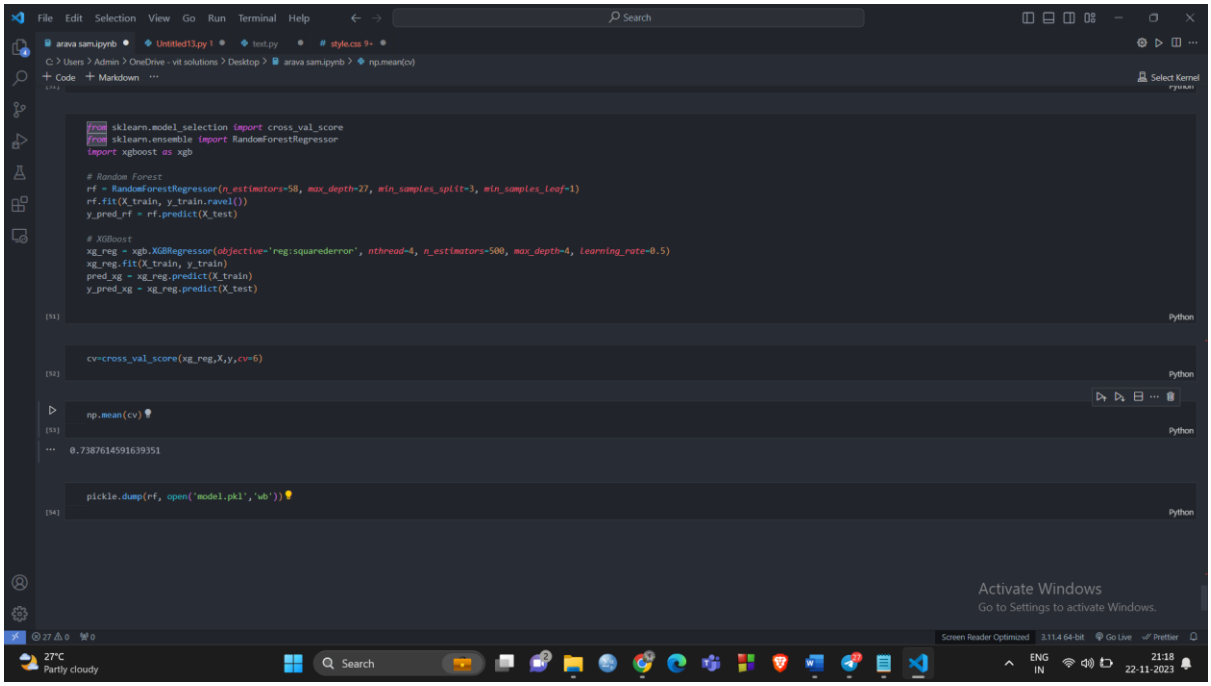

```
Mean Squared Error (MSE) of Auto ARIMA: 468477.9539606969
Root Mean Squared Error (RMSE) of Auto ARIMA: 684.454493706461
Mean Absolute Deviation (MAE) of Auto ARIMA: 446.8196095294599
```


File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
text.py
style.css 9+
C: > Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
Code
Markdown
Select Kernel
from sklearn.ensemble import RandomForestRegressor
import pickle
# Assuming you have already defined and trained your RandomForestRegressor 'rf'
rf = RandomForestRegressor(n_estimators=50, max_depth=20, min_samples_split=3, min_samples_leaf=1)
# ... (train the model with your data)
# Save the trained model to a file
with open('final_model.pkl', 'wb') as model_file:
pickle.dump(rf, model_file)
[40]
Python
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=50, max_depth=20, min_samples_split=3, min_samples_leaf=1)
rf.fit(X_train, y_train.ravel()) # Assuming X_train, y_train are your training data
print('Accuracy:', rf.score(X_test, y_test.ravel()) * 100, '%')
y_pred = rf.predict(X_test)
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
rms = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', rms)
print('MAE:', mean_absolute_error(y_test, y_pred))
print("Before fitting the model")
rf.fit(X_train, y_train.ravel()) # Assuming X_train, y_train are your training data
print("After fitting the model")
# Add similar print statements at different stages
[41]
Python
Accuracy: 96.4577508824995 %
RMSE: 4315.378610052086
MAE: 1648.949481735198
Before fitting the model
After fitting the model
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:16
22-11-2023


File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
text.py
style.css 9+
C: > Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
Code
Markdown
Select Kernel
print('Training Accuracy:',rf.score(X_train,y_train.ravel())*100,'%')
[42]
Python
Training Accuracy: 99.12741254321 %
import xgboost as xgb
import warnings
[43]
Python
import xgboost as xgb
xg_reg = xgb.XGBRegressor(objective='reg:squarederror', nthread=4,
n_estimators=500, max_depth=4, learning_rate=0.5)
xg_reg.fit(X_train, y_train)
[44]
Python
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
colsample_bylevel=None, colsample_bynode=None,
colsample_bytree=None, device=None, early_stopping_rounds=None,
enable_categorical=False, eval_metric=None, feature_types=None,
gamma=None, grow_policy=None, importance_type=None,
interaction_constraints=None, learning_rate=0.5, max_bin=None,
max_cat_threshold=None, max_cat_to_onehot=None,
max_delta_step=None, max_depth=4, max_leaves=None,
min_child_weight=None, missing=nan, monotone_constraints=None,
multi_strategy=None, n_estimators=500, n_jobs=None, nthread=4,
num_parallel_tree=None, ...)
pred=xg_reg.predict(X_train)
y_pred=xg_reg.predict(X_test)
[45]
Python
print('Accuracy:',xg_reg.score(X_test,y_test)*100,'%')
rms = mean_squared_error(y_test, y_pred, squared=False)
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:16
22-11-2023


File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
test.py
style.css 9+
C: > Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
Code
Markdown
Select Kernel
Python
print('Accuracy:',xg_reg.score(X_test,y_test)*100,'%')
rms = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:',rms)
print('MAE:',mean_absolute_error(y_test, y_pred))
Accuracy: 93.20892619359017 %
RMSE: 5975.146796472817
MAE: 3035.796765956923
print('Training Accuracy:',xg_reg.score(X_train,y_train)*100,'%')
Training Accuracy: 94.29593668521562 %
pip install prettytable
Defaulting to user installation because normal site-packages is not writeableNote: you may need to restart the kernel to use updated packages.
Requirement already satisfied: prettytable in c:\users\kaler\appdata\roaming\python\python311\site-packages (3.9.0)
Requirement already satisfied: wcwidth in c:\programdata\anaconda3\lib\site-packages (from prettytable) (0.2.5)
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names = ["Model","Training Accuracy","Testing Accuracy","RMSE","MAE"]
tb.add_row(["Random Forest", 99.11, 96.77, 4055.83, 1651.13])
tb.add_row(["XgBoost", 94.23, 93.72, 5660.68, 3129.36])
print(tb)
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:17
22-11-2023


File Edit Selection View Go Run Terminal Help
Search
arava sam.ipynb
Untitled13.py 1
test.py
style.css 9+
C: > Users > Admin > OneDrive - vit solutions > Desktop > arava sam.ipynb > stores.head()
Code
Markdown
Select Kernel
Model	Training Accuracy	Testing Accuracy	RMSE	MAE
Random Forest	99.11	96.77	4055.83	1651.13
XgBoost	94.23	93.72	5660.68	3129.36
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
# Random Forest
rf = RandomForestRegressor(n_estimators=58, max_depth=27, min_samples_split=3, min_samples_leaf=1)
rf.fit(X_train, y_train.ravel())
y_pred_rf = rf.predict(X_test)
# XGBoost
xg_reg = xgb.XGBRegressor(objective='reg:squarederror', nthread=4, n_estimators=500, max_depth=4, learning_rate=0.5)
xg_reg.fit(X_train, y_train)
pred_xg = xg_reg.predict(X_train)
y_pred_xg = xg_reg.predict(X_test)
Activate Windows
Go to Settings to activate Windows.
Screen Reader Optimized
Go Live
27°C
Partly cloudy
Search
ENG
IN
21:17
22-11-2023



Index.html:

```
<!DOCTYPE html>
<html lang="en">

<head>
   <meta charset="UTF-8">
   <title>Walmart Store Sales Forecasting</title>
   <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet'
type='text/css'>
   <link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet'
type='text/css'>
   <link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet'
type='text/css'>
   <link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300'
rel='stylesheet' type='text/css'>
   <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
   <style>
      body {
         font-family: 'Hind', sans-serif;
```

```
  background: #f4f4f4;
  margin: 0;
  padding: 0;
  display: flex;
  align-items: center;
  justify-content: center;
  height: 100vh;
}

.login {
  background: #fff;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  text-align: center;
}

h1 {
  font-family: 'Pacifico', cursive;
  color: #3498db;
}

form {
  margin-top: 20px;
}

label {
  font-size: 16px;
  color: #333;
  display: block;
  margin-bottom: 8px;
}

input,
select {
  width: 100%;
  padding: 10px;
  margin-bottom: 15px;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;
}

button {
  background-color: #3498db;
  color: #fff;
  padding: 12px;
  border: none;
```

```
      border-radius: 4px;
      cursor: pointer;
      font-size: 16px;
    }

    button:hover {
      background-color: #2980b9;
    }

    h3 {
      color: #3498db;
      margin-top: 20px;
    }

    b {
      color: #e74c3c;
    }

    /* Add new style for temperature input */
    #temperature {
      width: 100%;
      padding: 10px;
      margin-bottom: 15px;
      border: 1px solid #ccc;
      border-radius: 4px;
      box-sizing: border-box;
    }
  </style>
</head>

<body>
  <div class="login">
    <h1>Walmart Store Sales Forecasting</h1>

    <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}" method="post">
      <label for="store">Store:</label>
      <input type="text" name="store" placeholder="Store" required="required" />

      <label for="dept">Dept:</label>
      <input type="text" name="dept" placeholder="Dept" required="required" />

      <label for="date">Date:</label>
      <input type="date" id="date" name="date">

      <label for="isholiday">IsHoliday:</label>
      <select id="isholiday" name="isholiday">
        <option value="0">False</option>
```

```
            <option value="1">True</option>
        </select><br><br>

        <!-- Add temperature input -->
        <label for="temperature">Temperature:</label>
        <input type="text" id="temperature" name="temperature"
placeholder="Temperature" />

        <button type="submit" class="btn btn-primary btn-block btn-
large">Predict</button>
    </form>
    <br>
    {% if output %}
    <h3> Sales should be <b style="color:red;">$ {{ output }}</b> </h3>
    {% endif %}
  </div>

  <!-- Add a script for temperature input validation -->
  <script>
    document.getElementById("temperature").addEventListener("input", function () {
      // Ensure the entered value is a number (you can add more validation if needed)
      if (isNaN(this.value)) {
        alert("Please enter a valid temperature.");
        this.value = ""; // Clear the input if not a valid number
      }
    });
  </script>
</body>

</html>
```

App.py:

```
import numpy as np
import pandas as pd
import datetime as dt
from flask import Flask, request, jsonify, render_template
import joblib

app = Flask(__name__)
model = joblib.load('model.pkl')
fet = pd.read_csv('all_features.csv')

@app.route('/')
def home():
  return render_template('index.html')

@app.route('/predict',methods=['POST'])
```

```
def predict():

    features = [x for x in request.form.values()]

    if features[3]=='0':
        features[3]=False
    else:
        features[3]=True

    df=fet[(fet['Store']==int(features[0])) & (fet['IsHoliday']==features[3]) &
(fet['Date']==features[2])]
    f_features=[]
    d=dt.datetime.strptime(features[2], '%Y-%m-%d')
    c=0
    if df['Type'][0]=='C':
        c=1
    else:
        c=0

    if features[3]==False:
        features[3]=0
    else:
        features[3]=1

    if df.shape[0]==1:
        f_features.append(df['CPI'])
        f_features.append(d.date().day)
        f_features.append(int(features[1]))
        f_features.append(df['Fuel_Price'])
        f_features.append(features[3])
        f_features.append(d.date().month)
        f_features.append(df['Size'])
        f_features.append(int(features[0]))
        f_features.append(df['Temperature'])
        f_features.append(c)
        f_features.append(df['Unemployment'])
        f_features.append(d.date().year)

    final_features = [np.array(f_features)]
    output = model.predict(final_features)[0]
    return render_template('index.html', output=output)


if __name__ == "__main__":
    app.run(debug=True)
```

GitHub :
https://github.com/smartinternz02/SI-GuidedProject-611838-1700048817

Project Demo Link:
https://drive.google.com/drive/folders/1fuCOgO-ON87l-Ve-GCXR1tnan8fkdbYj?usp=sharing