Superstore Analytics Project Architecture

High-Level Architecture and Workflow

Author Name : Ayan Biswas

Date:23.01.2025

Sample Superstore

Introduction to the Project

Objective:

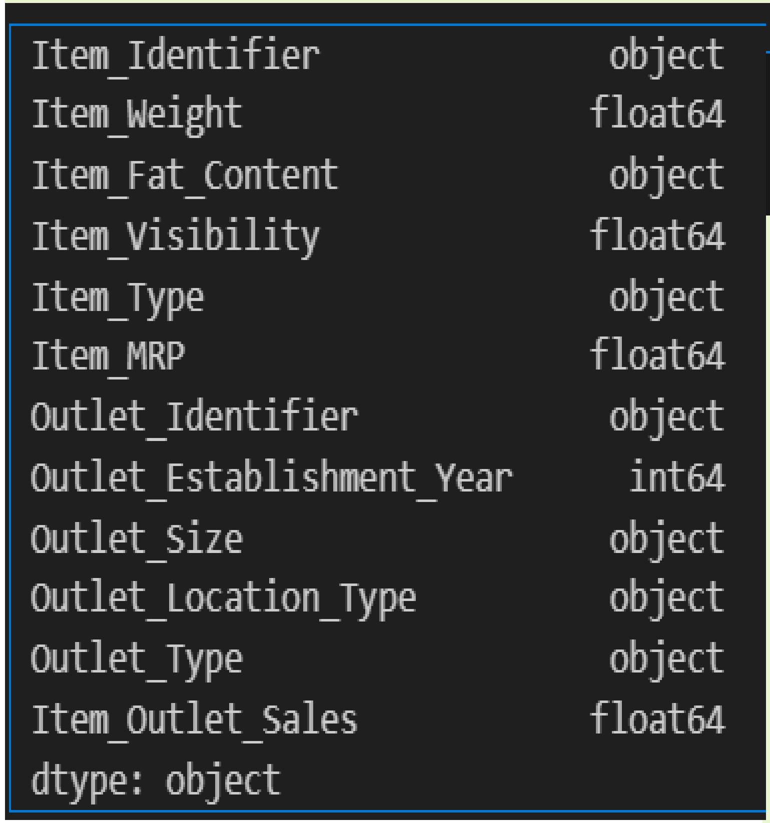
To analyze superstore sales data to identify trends, improve decision-making, and enhance operational efficiency.

Key Deliverables:

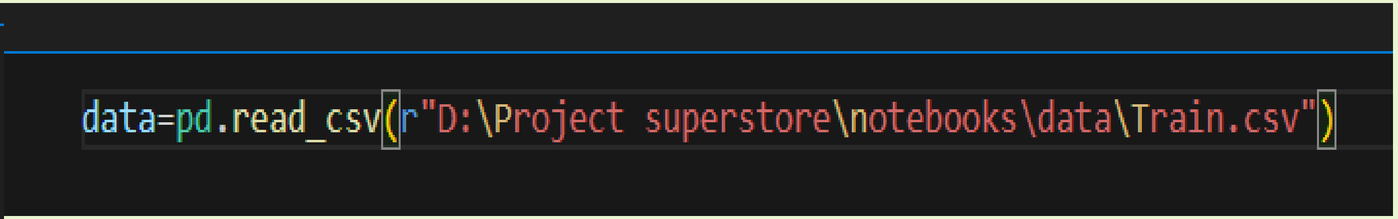
- Sales insights
- Customer segmentation
- Forecasting
- Inventory optimization

Data Sources

Data Types:



```
Item_Identifier               object
Item_Weight                  float64
Item_Fat_Content              object
Item_Visibility              float64
Item_Type                     object
Item_MRP                     float64
Outlet_Identifier             object
Outlet_Establishment_Year      int64
Outlet_Size                   object
Outlet_Location_Type          object
Outlet_Type                   object
Item_Outlet_Sales            float64
dtype: object
```



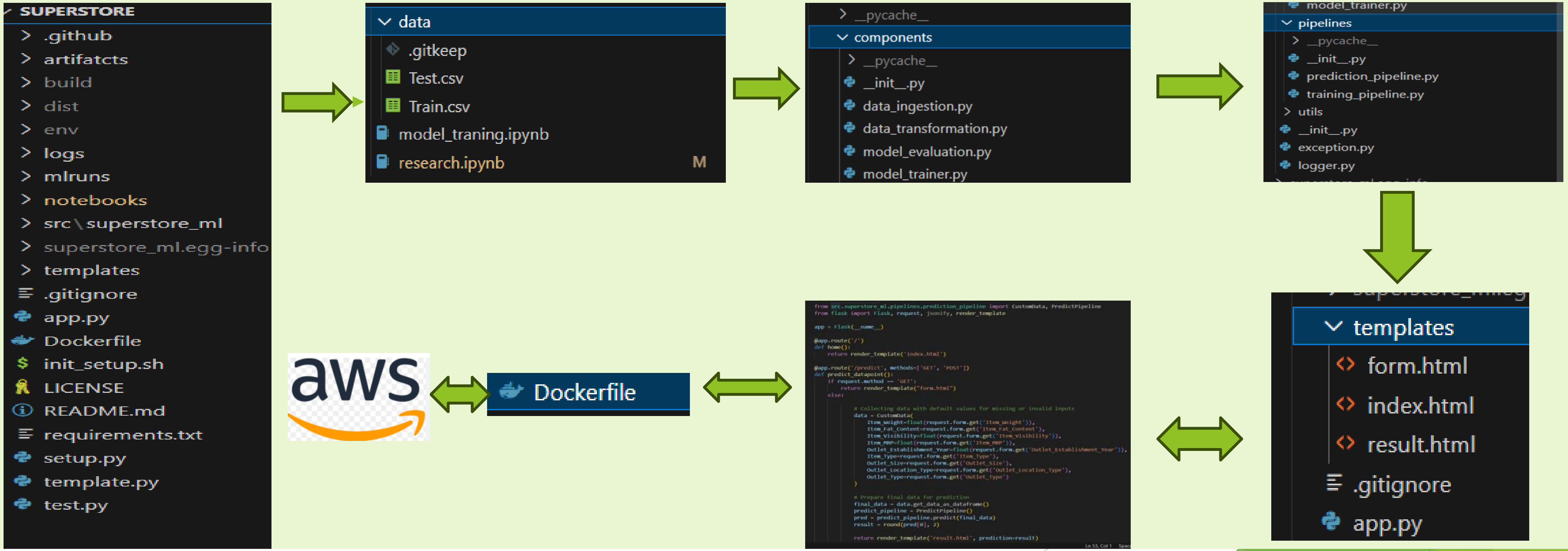
```
data=pd.read_csv(r"D:\Project superstore\notebooks\data\Train.csv")
```

This is the way how I import data

Architecture Overview

High-Level System Architecture

Diagram Elements:



```
SUPERSTORE
> .github
> artifacts
> build
> dist
> env
> logs
> mlruns
> notebooks
> src \ superstore_ml
> superstore_ml.egg-info
> templates
.gitignore
app.py
Dockerfile
init_setup.sh
LICENSE
README.md
requirements.txt
setup.py
template.py
test.py
```

```
∨ data
    .gitkeep
    Test.csv
    Train.csv
  model_traning.ipynb
  research.ipynb                M
```

```
> __pycache__
∨ components
  > __pycache__
    __init__.py
    data_ingestion.py
    data_transformation.py
    model_evaluation.py
    model_trainer.py
```

```
    model_trainer.py
∨ pipelines
  > __pycache__
    __init__.py
    prediction_pipeline.py
    training_pipeline.py
> utils
  __init__.py
  exception.py
  logger.py
```

```
∨ templates
    form.html
    index.html
    result.html
  .gitignore
  app.py
```

```python
from src.superstore_ml.pipelines.prediction_pipeline import CustomData, PredictPipeline
from flask import Flask, request, jsonify, render_template

app = Flask(__name__)

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

@app.route('/predict', methods=['GET', 'POST'])
def predict_datapoint():
    if request.method == 'GET':
        return render_template("form.html")
    else:

            # Collecting data with default values for missing or invalid inputs
            data = CustomData(
                Item_Weight=float(request.form.get('Item_Weight')),
                Item_Fat_Content=request.form.get('Item_Fat_Content'),
                Item_Visibility=float(request.form.get('Item_Visibility')),
                Item_MRP=float(request.form.get('Item_MRP')),
                Outlet_Establishment_Year=float(request.form.get('Outlet_Establishment_Year')),
                Item_Type=request.form.get('Item_Type'),
                Outlet_Size=request.form.get('Outlet_Size'),
                Outlet_Location_Type=request.form.get('Outlet_Location_Type'),
                Outlet_Type=request.form.get('Outlet_Type')
            )

            # Prepare final data for prediction
            final_data = data.get_data_as_dataframe()
            predict_pipeline = PredictPipeline()
            pred = predict_pipeline.predict(final_data)
            result = round(pred[0], 2)

            return render_template('result.html', prediction=result)
```

Directory Structure
Core Folders:

Notebooks:

Data:

- Research.ipynb
- Model_traning.ipynb
- Train.csv

Components:

- Data Ingestion
- Data Transformation
- Model_Trainer

pipelines:

- Traning pipeline
- prediction pipeline

Templates

- Form.html
- Index.html
- Result.html

Main files

- app.py
- requirements.txt
- setup.py

Workflow Diagram

1. **Data Ingestion:** Load and clean data.
2. **Data Transformation:** Prepare data for modeling.
3. **Model Training:** Build and train ML models.
4. **Evaluation:** Assess model performance.
5. **Prediction Pipeline:** Generate predictions.
6. **Web App:** Accept inputs, display predictions.

Core Modules

Explain the role of key Python scripts:

- **data_ingestion.py:** Reads and preprocesses the dataset.
- **data_transformation.py:** Handles feature engineering.
- **model_trainer.py:** Trains and saves the model.
- **model_evaluation.py:** Validates model performance.
- **prediction_pipeline.py:** Loads the model for predictions.

Web Interface

Templates

- Form.html : input page
- Result.html : output page

Flask Integration

- app.py : handle the request and hosts the web server

Deployment

Docker:

Contains Dockerfile for containerization.

Scalability:

The structure supports easy deployment and updates.