**YS&OP Solutions**

This is Documentation of various Tasks performed by Saifuddin during his internship at YS&OP Solutions it contains GitHub repository of code , steps of execution and some more details about the tasks.

**Login and Sign Up**

**Step 1: Backend Setup**

* Install Node.js and npm (Node Package Manager) on your machine.
* Set up a MySQL server and client.
* Create a new Node.js project using the command **npm init -y** in your desired project directory.
* Install the required dependencies using the command **npm install express mysql jsonwebtoken bcrypt**.
* Create a **server.js** file in the project's root directory to implement the backend API.

**Theory:**

* Node.js: A JavaScript runtime environment that allows running JavaScript code outside the browser.
* npm: The package manager for Node.js. It helps in installing and managing external libraries and dependencies.
* Express.js: A popular web framework for Node.js that simplifies building web applications.
* MySQL: An open-source relational database management system used to store data.
* JSON Web Tokens (JWT): A standard for securely transmitting information between parties as a JSON object. In this case, it will be used for user authentication.
* bcrypt: A library used to hash passwords securely.

**Step 2: Frontend Setup**

* Install create-react-app globally using the command **npm install -g create-react-app**.
* Create a new React.js project using the command **npx create-react-app frontend**.
* Navigate to the project directory using **cd frontend**.
* Install the required dependencies using the command **npm install axios react-router-dom tailwindcss**.

**Theory:**

* React.js: A JavaScript library for building user interfaces. It allows you to create reusable UI components.
* create-react-app: A tool that sets up a new React.js project with a basic file structure and configuration.
* Axios: A library used for making HTTP requests from the frontend to the backend.
* react-router-dom: A library for handling routing and navigation in React.js applications.
* Tailwind CSS: A utility-first CSS framework that provides a set of pre-built styles and classes to quickly build UI components.

**Data Description:**

* Almost three years of data ranging from 01-04-2020 to 27-02-2023 describing the sales transaction of nutrition type. (agri related)
* There are in total 15 columns which are the various features of the dataset.
* Critical elements which uniquely define the sales are product code and material description.

**Machine Learning Models:**

**Neural Prophet:**

Neural Prophet is a Python library developed by Facebook Research that combines the power of neural networks with traditional time series forecasting techniques. It offers an easy-to-use interface for training and evaluating neural network models for time series forecasting tasks.

**Step 1: Installation**

* Ensure you have Python installed on your machine. You can download Python from the official website ([https://www.python.org](https://www.python.org/)) and follow the installation instructions for your operating system.
* Open a terminal or command prompt and install Neural Prophet using pip, a package installer for Python. Run the following command: **pip install neuralprophet.**
* Neural prophet relies on the ‘pystan’ package, which requires additional dependencies to be installed.To install these dependencies, run the following command: **pip install pystan==2.19.1.1.**
* Once the installation is complete, you can import Neural Prophet in your Python environment by including the following line at the beginning of your script: **from neuralprophet import NeuralProphet.**

**Step 2: Data Preparation**

* Load your sales data into a pandas DataFrame. Ensure that the DataFrame contains two columns: 'ds' (the timestamps) and 'y' (the sales values). The 'ds' column should be of type **datetime**.
* Example:

**import pandas as pd**

**sales\_data = pd.read\_csv('sales\_data.csv')**

**sales\_data['ds'] = pd.to\_datetime(sales\_data['ds'])**

**Step 3: Model Training**

* Load your sales data into a pandas DataFrame. Ensure that the DataFrame contains two columns: 'ds' (the timestamps) and 'y' (the sales values). The 'ds' column should be of type **datetime**.
* You can refer to the Neural Prophet documentation ([https://neuralprophet.com](https://neuralprophet.com/)) for a full list of available parameters and their descriptions.
* Fit the model to your data using the **fit** method. Pass in the prepared DataFrame containing the 'ds' and 'y' columns.
* Example : **m.fit(sales\_data)**

**Step 4: Forecasting**

* Generate future timestamps for which you want to forecast sales. You can use the **make\_future\_dataframe** method to create a DataFrame with the desired range of future timestamps.
* Example: **future = m.make\_future\_dataframe(periods=30)**
* Use the **predict** method to obtain sales forecasts for the future timestamps.
* Example: **forecast = m.predict(future)**
* The resulting forecast DataFrame will contain the predicted sales values in the 'yhat' column, along with additional columns such as the lower and upper bounds of the uncertainty interval.

**Step 5: Evaluation**

* Split our dataset into a train and validation set. We will use the validation set to check the performance of our model. The size of the validation set is 20% of our total dataset. Adapt the size with the parameter **valid\_p** in **split\_df**
* Example: **df\_train, df\_val = m.split\_df(df, valid\_p=0.2)**
* Validation is performed by passing the validation set to the fit method during training. The resulting metrics show the performance of the model compared to our validation set.

**Step 6: Visualization**

* Prophet provides convenient plotting functionalities to visualize the forecast results. You can use the **plot** method to generate plots of the forecasted sales.
* Example:
* **m.plot(forecast)**
* This will generate a plot showing the historical sales data, the forecasted values, and the uncertainty interval.

**Auto-TS Library:**

AutoTS is a Python library designed for automated time series forecasting. It aims to simplify the process of building accurate forecasting models by automating the selection and tuning of various forecasting techniques. The library provides a high-level interface that allows users to easily train models, generate forecasts, and evaluate their performance.

* Prophet **Automated Model Selection:** AutoTS automatically searches through a wide range of forecasting models and algorithms to identify the most suitable one for a given time series. It includes popular models like ARIMA, SARIMA, Exponential Smoothing, Prophet, and more.
* **Model Hyperparameter Tuning:** The library leverages optimization algorithms to search for the optimal hyperparameters of the selected forecasting models. It intelligently explores different combinations of hyperparameters to find the best configuration for each model.
* **Ensemble Methods:** AutoTS implements ensemble methods to combine multiple forecasting models and generate more accurate predictions. It uses techniques like weighted averaging and stacking to leverage the strengths of different models and mitigate individual model weaknesses.
* **Automatic Data Pre-processing:** The library handles common data pre-processing tasks, such as handling missing values, handling outliers, and automatically applying transformations like differencing or logarithmic scaling. This saves users the effort of performing these steps manually.
* **Evaluation Metrics:** AutoTS provides various evaluation metrics to assess the performance of forecasting models. It calculates metrics like mean absolute error (MAE), root mean squared error (RMSE), mean absolute percentage error (MAPE), and more, allowing users to compare and evaluate different models easily.
* **Visualization Tools:** The library offers convenient plotting functions to visualize time series data, forecasted values, uncertainty intervals, and individual forecast components like trends and seasonality. These visualizations help users gain insights into the data and assess the quality of the forecasts.
* **Forecasting Configuration:** AutoTS allows users to customize various aspects of the forecasting process, such as the forecast length, frequency of the time series, and inclusion/exclusion of uncertainty intervals. This flexibility enables users to adapt the forecasting process to their specific requirements.

**Step 1: Installation**

* Ensure you have Python installed on your machine. You can download Python from the official website ([https://www.python.org](https://www.python.org/)) and follow the installation instructions for your operating system.
* Open a terminal or command prompt and install Neural Prophet using pip, a package installer for Python. Run the following command: **pip install autots.**
* Once the installation is complete, you can import AutoTS in your Python environment by including the following line at the beginning of your script: **from autots import AutoTS**

**Step 2: Data Preparation**

* Load your sales data into a pandas DataFrame. Ensure that the DataFrame contains two columns: 'ds' (the timestamps) and 'y' (the sales values). The 'ds' column should be of type **datetime**.
* Example:
* **import pandas as pd**
* **sales\_data = pd.read\_csv('sales\_data.csv')**
* **sales\_data['ds'] = pd.to\_datetime(sales\_data['ds'])**

**Step 3: Model Training and Forecasting**

* Create an instance of AutoTS class.
* Example: **model = AutoTS(forecast\_length=30, frequency='D')**
* Here, **forecast\_length** is the number of future time steps to forecast, and **frequency** is the frequency of the time series ('D' for daily, 'W' for weekly, 'M' for monthly, etc.).
* Train the model on your sales data using the **fit** method. Pass in the prepared DataFrame containing the 'ds' and 'y' columns.
* Example : **model.fit(sales\_data)**
* Generate forecasts for the future timestamps using the **predict** method. The **predict** method takes an optional parameter **return\_pred\_int** to specify whether to return the uncertainty intervals along with the point forecasts.
* Example : **forecast = model.predict(return\_pred\_int=True)**
* The **forecast** variable will contain the predicted sales values in the 'yhat' column, along with additional columns such as the lower and upper bounds of the uncertainty interval, if **return\_pred\_int** is set to **True**.

**Step 4: Evaluation**

* If you have ground truth data for the period covered by your forecast, you can evaluate the performance of the model using various metrics. AutoTS provides the **evaluate** method for this purpose.
* Example: **model.evaluate(sales\_data)**
* This will print out the evaluation metrics, including mean absolute error (MAE), root mean squared error (RMSE), and mean absolute percentage error (MAPE).

**Step 5: Visualisation**

* AutoTS provides built-in plotting functions to visualize the forecast results. You can use the **plot** method to generate plots of the forecasted sales.
* Example: **model.plot()**
* This will generate a plot showing the historical sales data, the forecasted values, and the uncertainty interval.
* You can also access the individual forecast components, such as trend, seasonality, and residuals, using the **get\_forecast\_component** method.
* Example: **trend = model.get\_forecast\_component('trend')**

**seasonality = model.get\_forecast\_component('seasonality')**

**residuals = model.get\_forecast\_component('residuals')**

* These variables will contain the forecasted values for the respective components.