ESTIMATING STOCK PREDICTION USING MACHINE LEARNING

Estimating the Stock Keeping Units Using ML

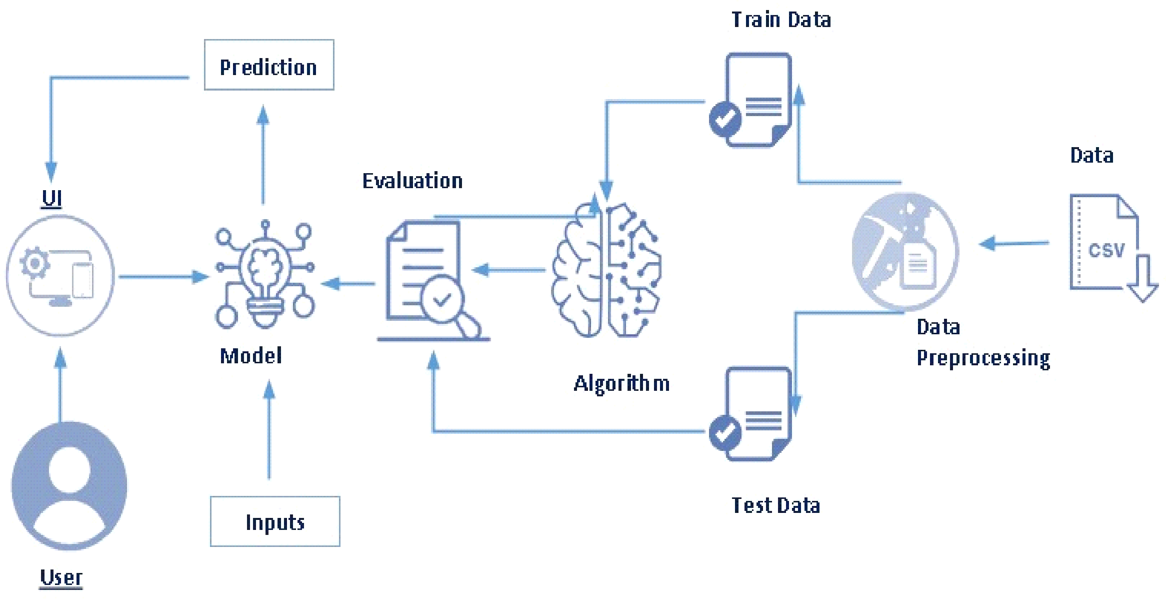
Estimating the Stock Keeping Units (SKUs) Using Machine Learning (ML) is a data-driven approach aimed at predicting and managing inventory levels for different products or items in a warehouse or retail setting. By leveraging historical sales data, seasonality trends, and other relevant variables, machine learning models are employed to forecast demand for each SKU. This enables businesses to optimize stock levels, reduce overstock or stockouts, and improve overall inventory management efficiency.

Scenario 1: Inventory Optimization Retailers and wholesalers can use SKU estimation to optimize their inventory levels. By accurately predicting demand for each SKU, they can ensure they have the right amount of stock on hand to meet customer demand while minimizing excess inventory holding costs.

Scenario 2: Demand Planning Manufacturers and distributors can leverage SKU estimation to improve demand planning processes. By forecasting SKU demand, they can streamline production schedules, procurement activities, and supply chain operations, leading to better resource utilization and reduced lead times.

Scenario 3: Marketing and Promotion Strategies Marketing teams can benefit from SKU estimation by aligning their promotions and marketing campaigns with predicted demand patterns. By targeting high-demand SKUs during peak periods, they can maximize sales opportunities and enhance overall marketing effectiveness.

Technical Architecture:



Pre requisites

To complete this project, you must require following software’s, concepts and packages

- Anaconda navigator and PyCharm:
    - Refer the link below to download anaconda navigator
    - Link: Here
- Python packages:
    - Open anaconda prompt as administrator
    - Type “pip install NumPy” and click enter.
    - Type “pip install pandas” and click enter.
    - Type “pip install scikit-learn” and click enter.
    - Type” pip install matplotlib” and click enter.
    - Type” pip install SciPy” and click enter.
    - Type” pip install pickle-mixin” and click enter.
    - Type” pip install seaborn” and click enter.
    - Type “pip install Flask” and click enter.

Let's start building the project.

Project Objectives

By the end of this project, you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/transformation techniques on outlier and some visualization concepts.

Project Flow

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

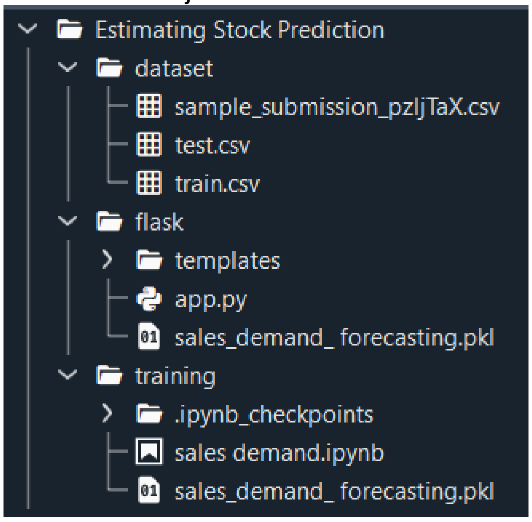
To accomplish this, we have to complete all the activities listed below,

- Data collection
    - Collect the dataset or create the dataset
- Visualizing and Analysing the Data
    - Importing the libraries
    - Read the Dataset
    - Data pre-processing

    - Handling Missing Values
    - Data Visualization
    - Splitting x and y
    - Splitting data into train and test
- Model building
    - Import the model building libraries
    - Initializing the model
    - Training and testing the model
    - Evaluating performance of model
    - Save the model
- Application Building
    - Create an HTML file
    - Build python code

Project Structure

Create the Project folder which contains files as shown below



```
Estimating Stock Prediction
├─ dataset
│  ├─ sample_submission_pzljTaX.csv
│  ├─ test.csv
│  └─ train.csv
├─ flask
│  ├─ templates
│  ├─ app.py
│  └─ sales_demand_ forecasting.pkl
└─ training
   ├─ .ipynb_checkpoints
   ├─ sales demand.ipynb
   └─ sales_demand_ forecasting.pkl
```

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- Sales_demand_forecasting.pkl is our saved model.
- Further we will use this model for flask integration.
- Training folder contains model training files

Data Collection

ML depends heavily on data; it is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Download the dataset

There are many popular open sources for collecting the data. E.g.: kaggle.com, UCI repository, etc.

In this project we have used drug200.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: Demand Forecasting

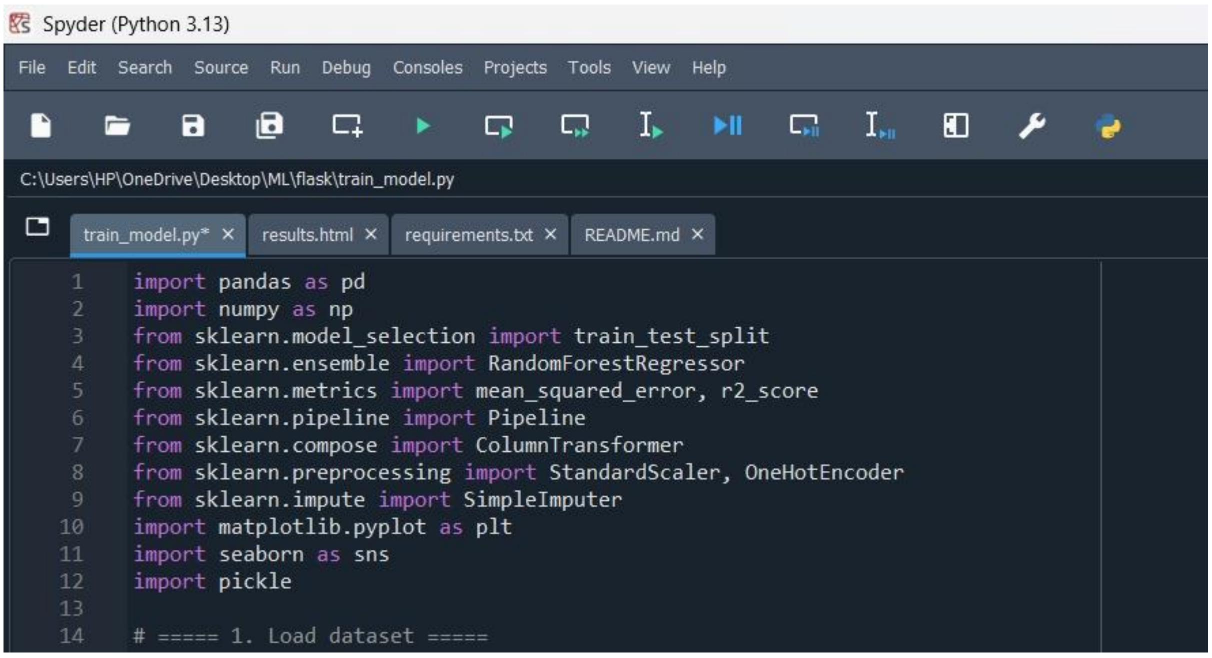
Visualizing and analysing the data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Importing the libraries

Import the necessary libraries as shown in the image.

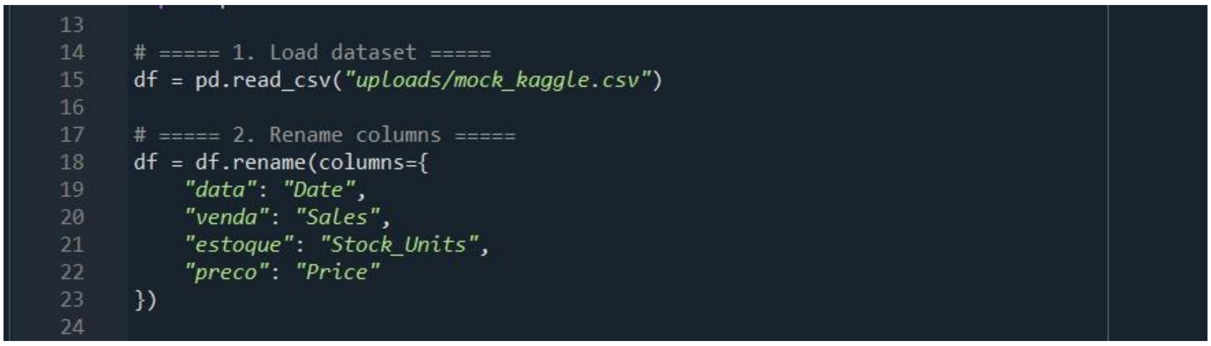

Spyder (Python 3.13)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\HP\OneDrive\Desktop\ML\flask\train_model.py
train_model.py*
results.html
requirements.txt
README.md

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
import seaborn as sns
import pickle

# ===== 1. Load dataset =====
```

Read the Dataset

Our dataset format might be in .csv, excel files, txt, json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv () to read the dataset. As a parameter we have to give the directory of the csv file.


```
# ===== 1. Load dataset =====
df = pd.read_csv("uploads/mock_kaggle.csv")

# ===== 2. Rename columns =====
df = df.rename(columns={
    "data": "Date",
    "venda": "Sales",
    "estoque": "Stock_Units",
    "preco": "Price"
})
```

Data Pre-Processing

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Handling Missing Values

For checking the null values, data. is null () function is used. To sum those null values, we use. sum () function to it. From the below image we found that there are no null values present in our dataset. So, we can skip handling the missing values step.

```
prep.categorical.fields
```

```
['week']
```

```
prep.numerical.fields
```

```
['record_ID',
 'store_id',
 'sku_id',
 'total_price',
 'base_price',
 'is_featured_sku',
 'is_display_sku',
 'units_sold']
```

```
prep.missing_values
```

```
record_ID          0
week               0
store_id           0
sku_id             0
total_price        1
base_price         0
is_featured_sku    0
is_display_sku     0
units_sold         0
dtype: int64
```

```
prep.numerical.impute()
```

```
Numerical features imputed successfully.
```

Data Description:

Variable Definition record_ID: Unique ID for each week store sku combination week: Starting Date of the week store_id: Unique ID for each store (no numerical order to be assumed) sku_id: Unique ID for each product (no numerical order to be assumed) total_price: Sales Price of the product base_price: Base price of the product is_featured_sku: Was part of the featured item of the week is_display_sku: Product was on display at a prominent place at the store units_sold(Target): Total Units sold for that week-store-sku combination.

```
prep.df
```

	record_ID	week	store_id	sku_id	total_price	base_price	is_featured_sku	is_display_sku	units_sold
0	1.0	17/01/11	8091.0	216418.0	99.0375	111.8625	0.0	0.0	20.0
1	2.0	17/01/11	8091.0	216419.0	99.0375	99.0375	0.0	0.0	28.0
2	3.0	17/01/11	8091.0	216425.0	133.9500	133.9500	0.0	0.0	19.0
3	4.0	17/01/11	8091.0	216233.0	133.9500	133.9500	0.0	0.0	44.0
4	5.0	17/01/11	8091.0	217390.0	141.0750	141.0750	0.0	0.0	52.0
...	...	...	...	...	...	...	...	...	...
150145	212638.0	09/07/13	9984.0	223245.0	235.8375	235.8375	0.0	0.0	38.0
150146	212639.0	09/07/13	9984.0	223153.0	235.8375	235.8375	0.0	0.0	30.0
150147	212642.0	09/07/13	9984.0	245338.0	357.6750	483.7875	1.0	1.0	31.0
150148	212643.0	09/07/13	9984.0	547934.0	141.7875	191.6625	0.0	1.0	12.0
150149	212644.0	09/07/13	9984.0	679023.0	234.4125	234.4125	0.0	0.0	15.0

150150 rows × 9 columns

Create a new column 'key' for unique identification. This is done in order to handle the duplicate data in 'week' column.

```
prep.dataset['key'] = prep.df['week'].astype(str) + '_' + prep.df['store_id'].astype(str)
```

Removing columns that are not helpful for time series predictions.

```
prep.dataset = prep.df.drop(['record_ID', 'week', 'store_id', 'sku_id', 'total_price', 'base_price', 'is_featured_sku', 'is_displ
```

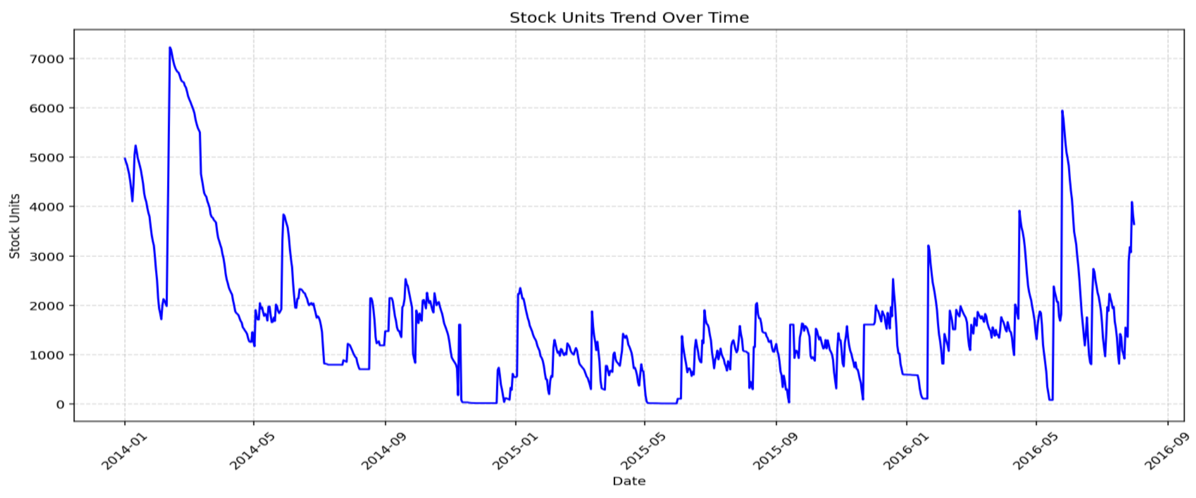
Summing units_sold group by key.

```
prep.dataset = prep.df.groupby('key').sum()
```

```
prep.df
```

```
                 units_sold
             key
01/01/13_8023.0      2025.0
01/01/13_8058.0       682.0
01/01/13_8063.0       535.0
01/01/13_8091.0       210.0
01/01/13_8094.0       782.0
             ...        ...
31/10/11_9890.0       531.0
31/10/11_9909.0       551.0
31/10/11_9954.0       431.0
31/10/11_9961.0       820.0
31/10/11_9984.0       506.0

9880 rows × 1 columns
```

Data Visualization



Splitting data into x and y

Here, units_sold becomes target variable

day_1, day_2, day_3, day_4 becomes input
We will train our model to predict sales based on the previous 4 days.

```
prep.df['day_1'] = prep.df['units_sold'].shift(-1)
prep.df['day_2'] = prep.df['units_sold'].shift(-2)
prep.df['day_3'] = prep.df['units_sold'].shift(-3)
prep.df['day_4'] = prep.df['units_sold'].shift(-4)
```

```
prep.df
```

	units_sold	day_1	day_2	day_3	day_4
key					
01/01/13_8023.0	2025.0	682.0	535.0	210.0	782.0
01/01/13_8058.0	682.0	535.0	210.0	782.0	1357.0
01/01/13_8063.0	535.0	210.0	782.0	1357.0	524.0
01/01/13_8091.0	210.0	782.0	1357.0	524.0	362.0
01/01/13_8094.0	782.0	1357.0	524.0	362.0	1546.0
...	...	...	...	...	...
31/10/11_9890.0	531.0	551.0	431.0	820.0	506.0
31/10/11_9909.0	551.0	431.0	820.0	506.0	NaN
31/10/11_9954.0	431.0	820.0	506.0	NaN	NaN
31/10/11_9961.0	820.0	506.0	NaN	NaN	NaN
31/10/11_9984.0	506.0	NaN	NaN	NaN	NaN

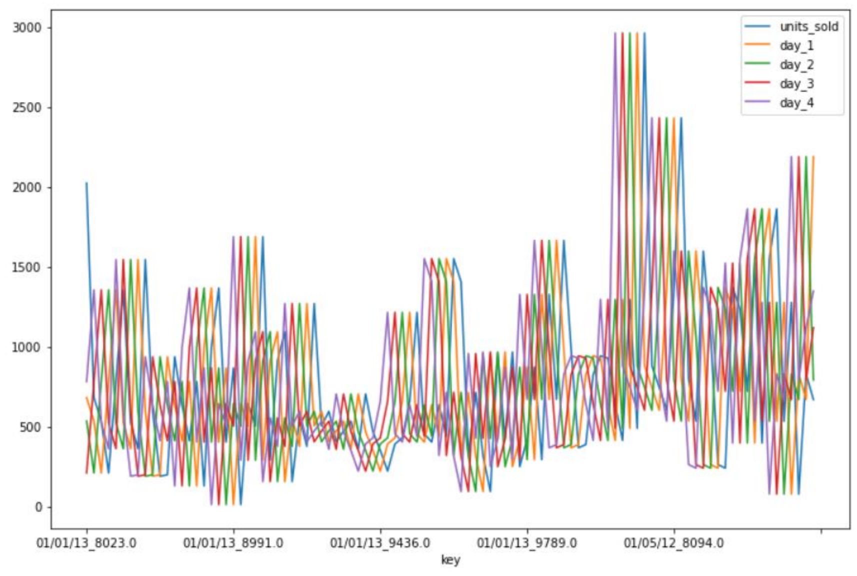
9880 rows × 5 columns

Removing NaN data

```
df = prep.df.dropna()
```

```
df[:100].plot(figsize=(12,8))
```

```
<AxesSubplot:xlabel='key'>
```



Splitting data into train and test

Now let’s split the Dataset into train and test sets

Changes: first split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, data is passed with dropping the target variable. And on y target variable is passed.

```
x1, x2, x3, x4, y = df['day_1'], df['day_2'], df['day_3'], df['day_4'], df['units_sold']
x1, x2, x3, x4, y = np.array(x1), np.array(x2), np.array(x3), np.array(x4), np.array(y)
x1, x2, x3, x4, y = x1.reshape(-1,1), x2.reshape(-1,1), x3.reshape(-1,1), x4.reshape(-1,1), y.reshape(-1,1)

split_percentage = 15
test_split = int(len(df)*(split_percentage/100))
x = np.concatenate((x1, x2, x3, x4), axis=1)
X_train,X_test,y_train,y_test = x[:-test_split],x[-test_split:],y[:-test_split],y[-test_split:]
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(8395, 4)
(1481, 4)
(8395, 1)
(1481, 1)
```

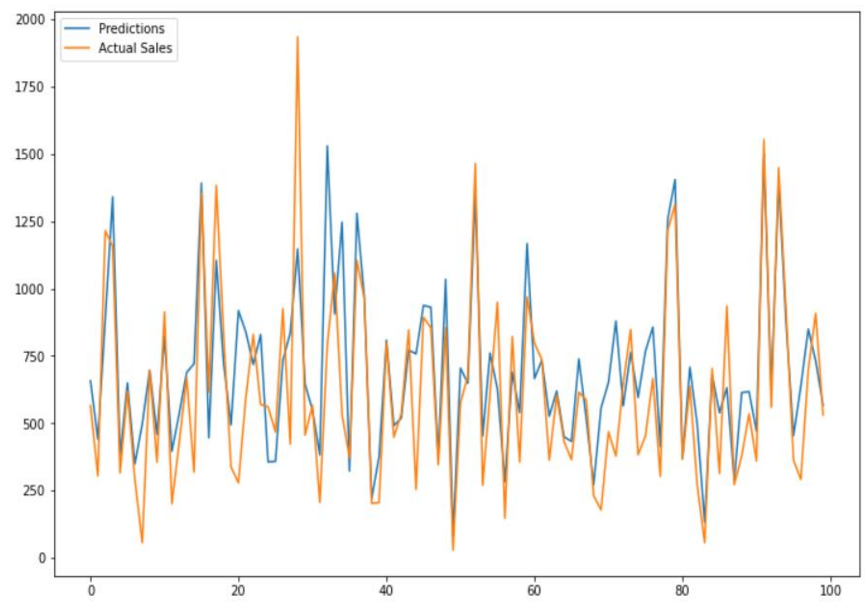
```
from sklearn.ensemble import RandomForestRegressor
rf_regressor = RandomForestRegressor()
rf_regressor.fit(X_train, y_train)

y_pred = rf_regressor.predict(X_test)
```

```
C:\Users\parig\AppData\Local\Temp\ipykernel_13336\3771703173.py:3: DataConversionWarning: A column-vector y was passed when a 1
d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  rf_regressor.fit(X_train, y_train)
```

```
print("R Sq. Score for Random Forest Regression :", rf_regressor.score(X_test, y_test))
```

```
R Sq. Score for Random Forest Regression : 0.6892176049902299
```

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12,8)
plt.plot(y_pred[-100:], label='Predictions')
plt.plot(y_test[-100:], label='Actual Sales')
plt.legend(loc="upper left")
plt.show()
```



Model Building

Now that the data is cleaned, it’s time to build the model. We can train our data on different algorithms. For this project we will be using Linear Regression, Decision tree, Random forest, and XgBoost algprithms.

Random Forest model

- A function named Random Forest model is created and train and test data are passed as the parameters. Inside the function, Random Forest algorithm is initialized and training data is passed to the model with. fit () function. Test data is predicted with. predict () function and save in new variable. For evaluating the model, accuracy score, classification_report (precision, recall, f1 score, support). It gives the best results.

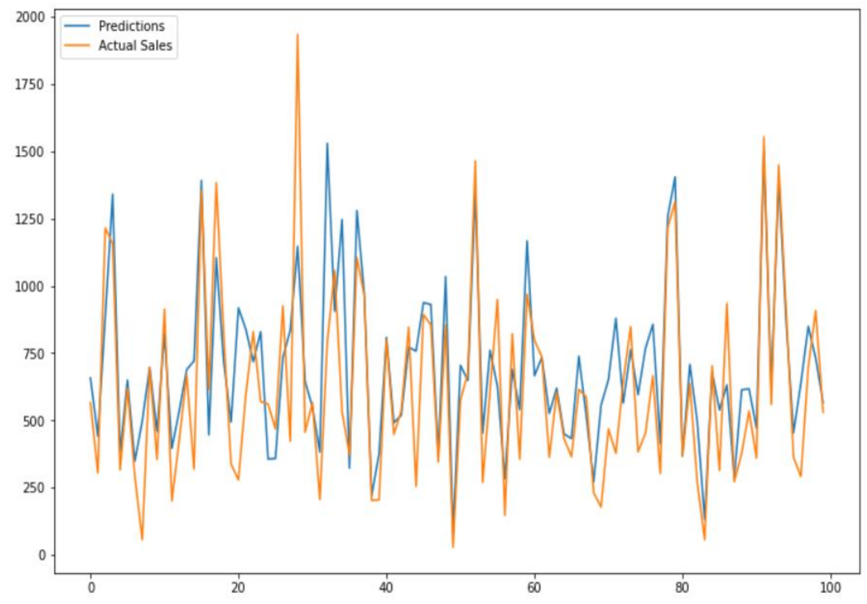
```
from sklearn.ensemble import RandomForestRegressor
rf_regressor = RandomForestRegressor()
rf_regressor.fit(X_train, y_train)

y_pred = rf_regressor.predict(X_test)
```

```
C:\Users\parig\AppData\Local\Temp\ipykernel_13336\3771703173.py:3: DataConversionWarning: A column-vector y was passed when a 1
d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  rf_regressor.fit(X_train, y_train)
```

```
print("R Sq. Score for Random Forest Regression :", rf_regressor.score(X_test, y_test))
```

```
R Sq. Score for Random Forest Regression : 0.6892176049902299
```

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12,8)
plt.plot(y_pred[-100:], label='Predictions')
plt.plot(y_test[-100:], label='Actual Sales')
plt.legend(loc="upper left")
plt.show()
```



XGBClassifier model

A function named XGBClassifier model is created and train and test data are passed as the parameters. Inside the function, XGBClassifier algorithm is initialized and training data is passed to the model with. fit () function. Test data is predicted with. predict () function and save in new variable. For evaluating the model, accuracy score and classification report is done.

```
pip install xgboost
```

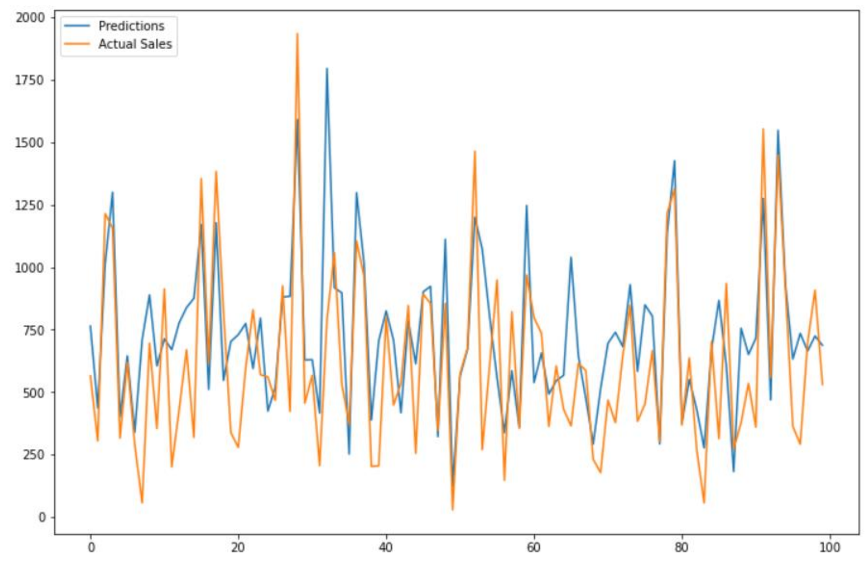
```
Collecting xgboost
  Downloading xgboost-1.7.2-py3-none-win_amd64.whl (89.1 MB)
Requirement already satisfied: scipy in c:\users\parig\anaconda3\lib\site-packages (from xgboost) (1.7.3)
Requirement already satisfied: numpy in c:\users\parig\anaconda3\lib\site-packages (from xgboost) (1.21.5)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.2
Note: you may need to restart the kernel to use updated packages.
```

```
import xgboost
xgb_regressor = xgboost.XGBRegressor()
xgb_regressor.fit(X_train, y_train)

y_pred = xgb_regressor.predict(X_test)
```

```
print("R Sq. Score for XGBoost :", xgb_regressor.score(X_test, y_test))
```

```
R Sq. Score for XGBoost : 0.5560478646422438
```

```
import matplotlib.pyplot as plt
plt.plot(y_pred[-100:], label='Predictions')
plt.plot(y_test[-100:], label='Actual Sales')
plt.legend(loc="upper left")
plt.show()
```



Evaluating performance of the model Using Randomized Search CV and saving the model

From sclera, accuracy is used to evaluate the score of the model. On the parameters, we have given rf (model name). Our model is performing well. So, we are saving the model by pickle. dump ().
Note: To understand cross validation, refer to this link - Link

```
from sklearn.model_selection import RandomizedSearchCV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 50, stop = 250, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(0, 120, num = 20)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [50, 72, 94, 116, 138, 161, 183, 205, 227, 250], 'max_features': ['auto', 'sqrt'], 'max_depth': [0, 6, 12, 18,
25, 31, 37, 44, 50, 56, 63, 69, 75, 82, 88, 94, 101, 107, 113, 120, None], 'min_samples_split': [2, 5, 10], 'min_samples_leaf':
[1, 2, 4], 'bootstrap': [True, False]}
```

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 10, cv = 3, verbose=2, random_state=0,
```

```
rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
C:\Users\parig\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:926: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  self.best_estimator_.fit(X, y, **fit_params)
```

```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [0, 6, 12, 18, 25, 31, 37,
                                                      44, 50, 56, 63, 69, 75,
                                                      82, 88, 94, 101, 107, 113,
                                                      120, None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [50, 72, 94, 116, 138,
                                                         161, 183, 205, 227,
                                                         250]},
                   random_state=0, verbose=2)
```

```
rf_random.best_params_
```

```
{'n_estimators': 116,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_features': 'auto',
 'max_depth': 56,
 'bootstrap': True}
```

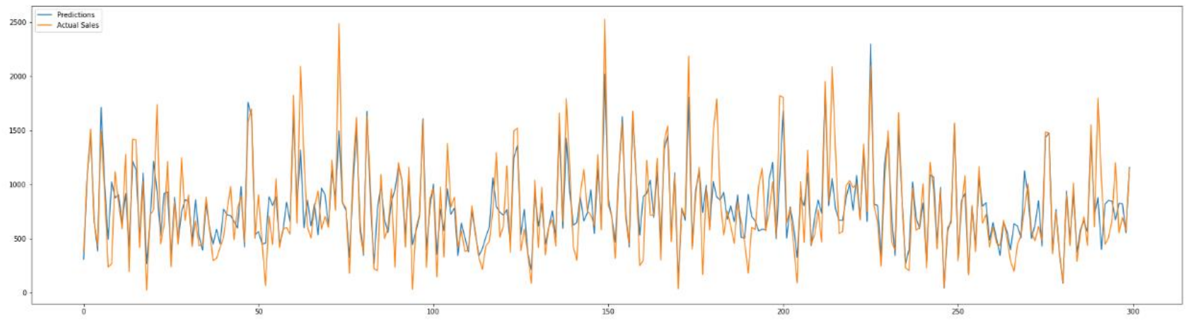
```
best_random = rf_random.best_estimator_
```

```
y_pred = best_random.predict(X_test)
```

```
print("R Sq. Score for Random Forest Regression :", best_random.score(X_test, y_test))
print("Adj. R Sq. Score for Random Forest Regression :", 1 - (1 - best_random.score(X_test, y_test) ) * ( len(y_test) - 1 ) / ( l
```

```
R Sq. Score for Random Forest Regression : 0.6821773483490408
Adj. R Sq. Score for Random Forest Regression : 0.6813160403499867
```

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (30,8)
plt.plot(y_pred[500:800], label='Predictions')
plt.plot(y_test[500:800], label='Actual Sales')
plt.legend(loc="upper left")
plt.savefig('final.png')
plt.show()
```



Here, we are saving the model with Random Forest Regressor only because after applying Hyperparameter Tuning also because we are getting same result before and after applying Hyperparameter Tuning

Saving the model as sales_demand_ forecasting.pkl

```
# saving the model
import pickle
pickle.dump(rf_regressor,open('sales_demand_ forecasting.pkl','wb'))
```

```
# rf_regressor
features = np.array([[682.0,535.0,210.0,782.0]])
print(rf_regressor.predict(features))
```

```
[1968.33]
```

Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script

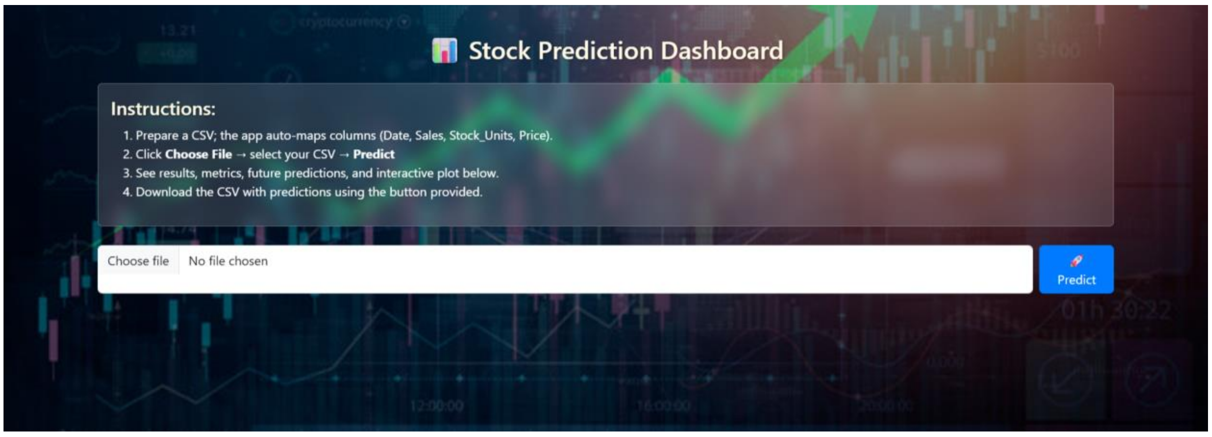
Building Html Pages

For this project create three HTML files namely

- index.html
- resut.html

and save them in templates folder.

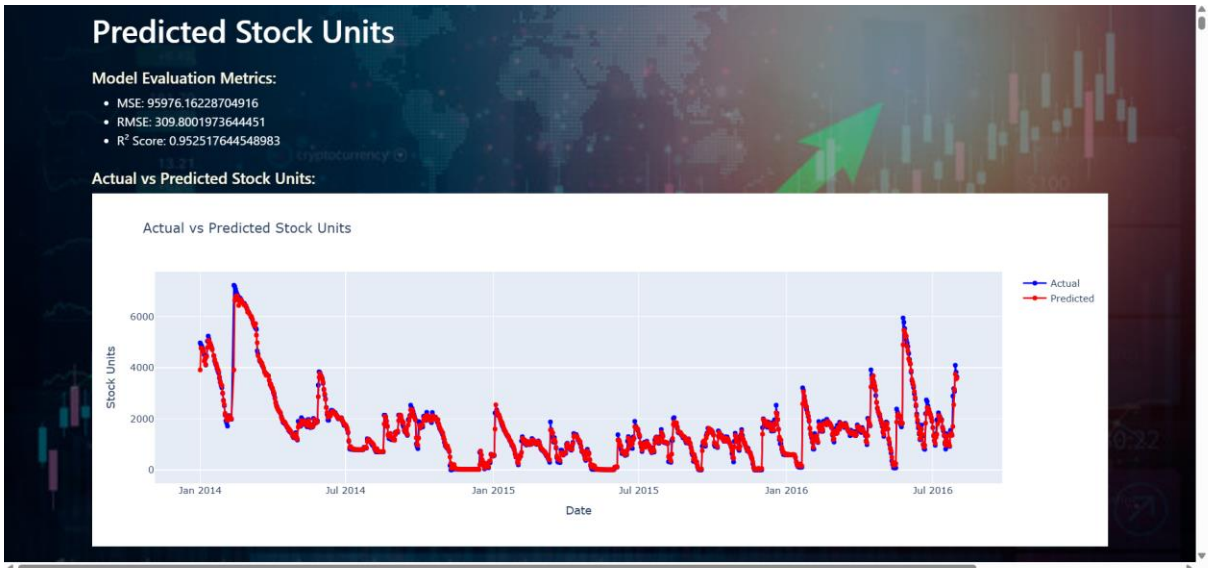
Let’s see how our index.html page looks like:



📊 Stock Prediction Dashboard

Instructions:

1. Prepare a CSV; the app auto-maps columns (Date, Sales, Stock_Units, Price).
2. Click **Choose File** → select your CSV → **Predict**
3. See results, metrics, future predictions, and interactive plot below.
4. Download the CSV with predictions using the button provided.



Predicted Stock Units

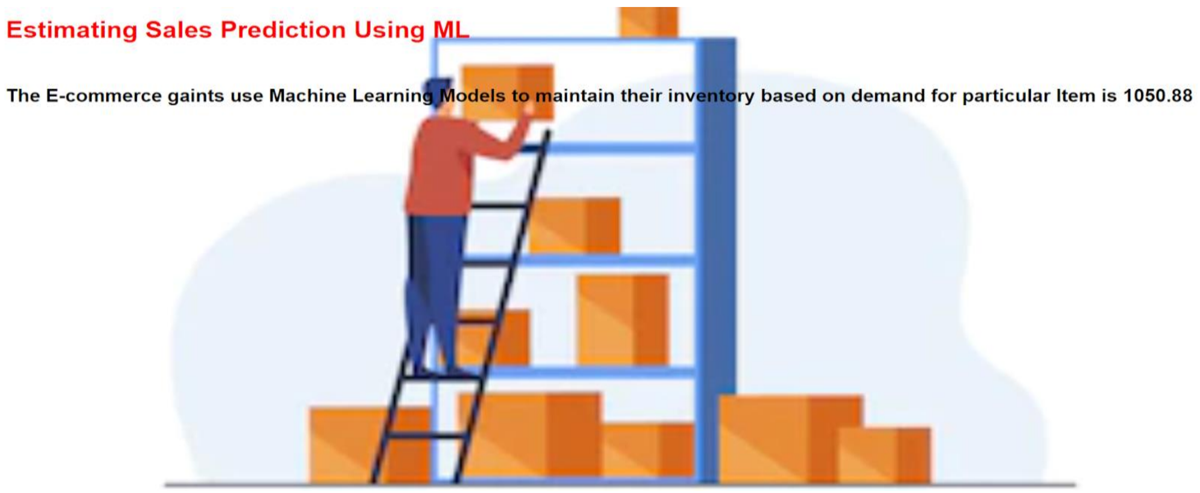
Model Evaluation Metrics:

- MSE: 95976.16228704916
- RMSE: 309.8001973644451
- R² Score: 0.952517644548983

Actual vs Predicted Stock Units:

Now when you click on submit button at bottom and it transfers to result.html page to give the results.

Let’s look how our result.html file looks like:



Estimating Sales Prediction Using ML

The E-commerce gaints use Machine Learning Models to maintain their inventory based on demand for particular Item is 1050.88

Build Python code

Import the libraries

```
from flask import Flask, render_template, request
import numpy as np
import pickle
import pandas as pd
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```
model = pickle.load(open( "sales_demand_ forecasting.pkl", "rb"))
app = Flask(__name__)
```

Render HTML page:

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

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, ‘/’ URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.
Retrieves the value from UI:

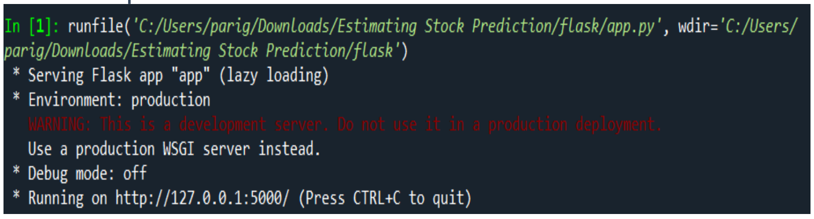
```python
@app.route("/y_predict", methods=['POST', 'GET'])
def y_predict():
    x = [[float(x) for x in request.form.values()]]

    print(x)

    cols=["day_1","day_2","day_3","day_4"]

    print(x)
    pred = model.predict(x)

    print(pred[0])
    return render_template('result.html', prediction_text=pred[0])
```

Here we are routing our app to predict () function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model. Predict () function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

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

Run the application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type "python app.py" command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.
- 

```
In [1]: runfile('C:/Users/parig/Downloads/Estimating Stock Prediction/flask/app.py', wdir='C:/Users/
parig/Downloads/Estimating Stock Prediction/flask')
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Train Model on IBM Cloud

In this milestone, you will learn how to build a Machine Learning Model and deploy it on the IBM Cloud.

Register for IBM Cloud

Watch the below video to register and login into your IBM account https://youtu.be/QuTDhYeJh0k

Train the ML Model on IBM

Watch below video to train the Machine learning model on IBM Watson

https://youtu.be/TysuP3KgSzc

Integrate Flask With Scoring End Point

Follow the below video, to integrate scoring endpoint with flask

https://youtu.be/ST1ZYLmYw2U