Optimizing Omnichannel Order Fulfillment with AI and Advanced Analytics

Presented By: Jubin Thomas
Senior Member of IEEE
Technical Architect, Signet Jewelers

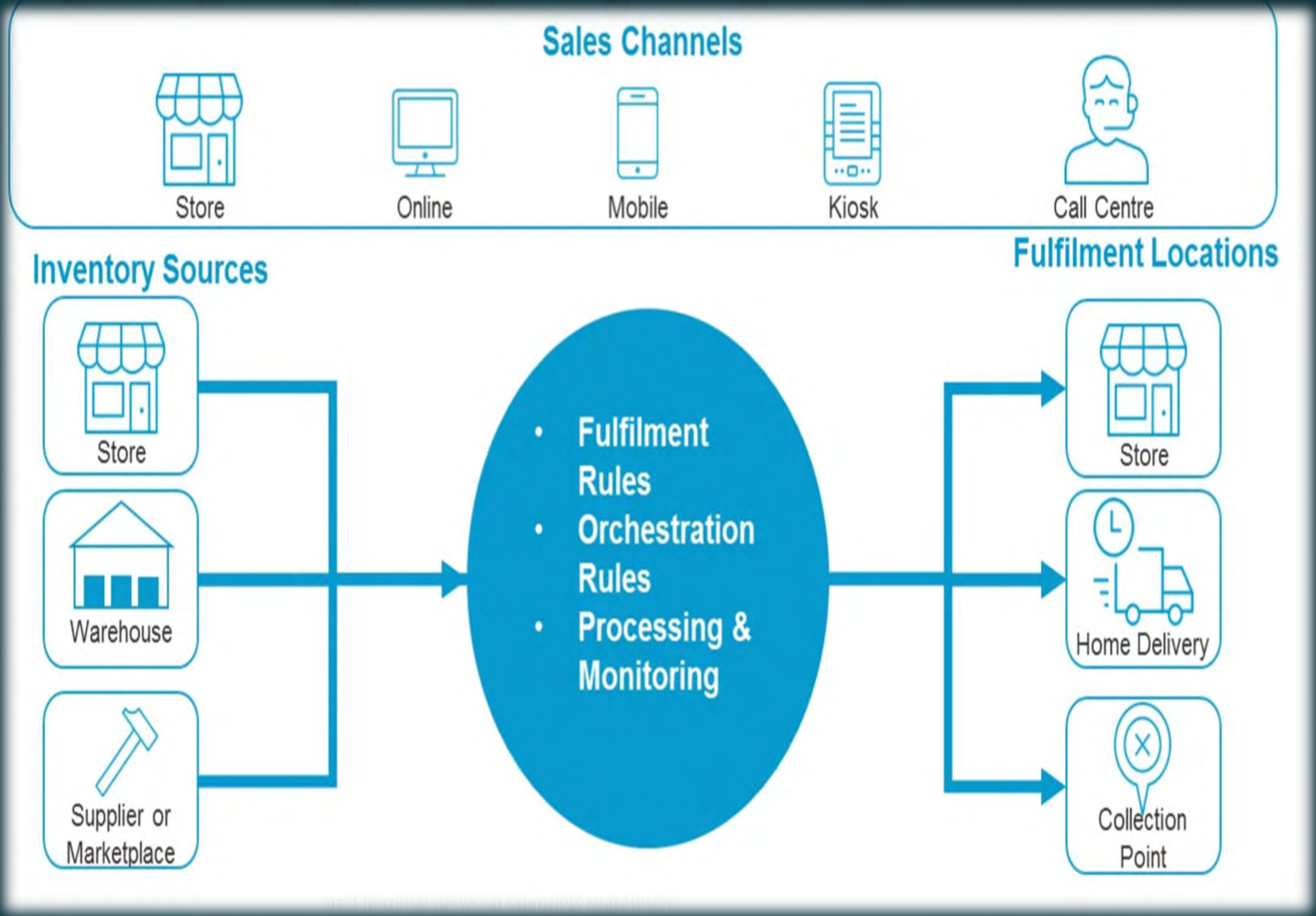
Ecommerce Industry

The e-commerce industry encompasses businesses that operate on the internet to sell goods and services directly to consumers, without a physical storefront.

ExpertMarketResearch.com

USD 1.1T	**USD 3.85T**	**14.8%**
2024	2032	increase

OMNI Channel Retailing



Problem Statement

One of the biggest problem statements in omnichannel retailing is providing a seamless and consistent customer experience across all channels. Customers today expect to be able to interact with retailers through multiple channels such as brick-and-mortar stores, websites, mobile apps, social media, and more. However, ensuring a consistent experience across these channels can be challenging due to factors such as inventory management, pricing consistency, and personalized marketing.

Real-Time Inventory Visibility

- Overstocking and Understocking
- Fulfillment Delays and Errors
- Inefficient Use of Inventory
- Increased Operational Costs

Sub-optimal Sourcing

- High Shipping Costs
- Impact on Sustainability
- Supplier and Stock Management Issues
- Lost Sales and Customer Dissatisfaction

Agenda

- Building a central inventory visibility system using AI and predictive algorithms
- Using machine learning to optimize order sourcing and routing
- Predictive analytics to allocate omni-channel inventory dynamically.
- Reinforcement learning to optimize allocation across fulfillment centers.

Building a central inventory visibility system using AI and predictive algorithms

DEMAND FORECASTING

Demand forecasting involves predicting the future demand for a product or service over a specific period. It considers factors such as historical sales data, market trends, economic conditions, customer preferences, and seasonality.

SALES FORECASTING

Sales forecasting specifically focuses on predicting future sales volumes or revenues generated by selling products or services. It typically relies on historical sales data, market analysis, customer feedback, and sales pipeline information.

Forecasting Use Case

Data Description:
The dataset contains historical sales data for various products sold by Company A. Key columns include:

Transaction Date: The date of the sale.

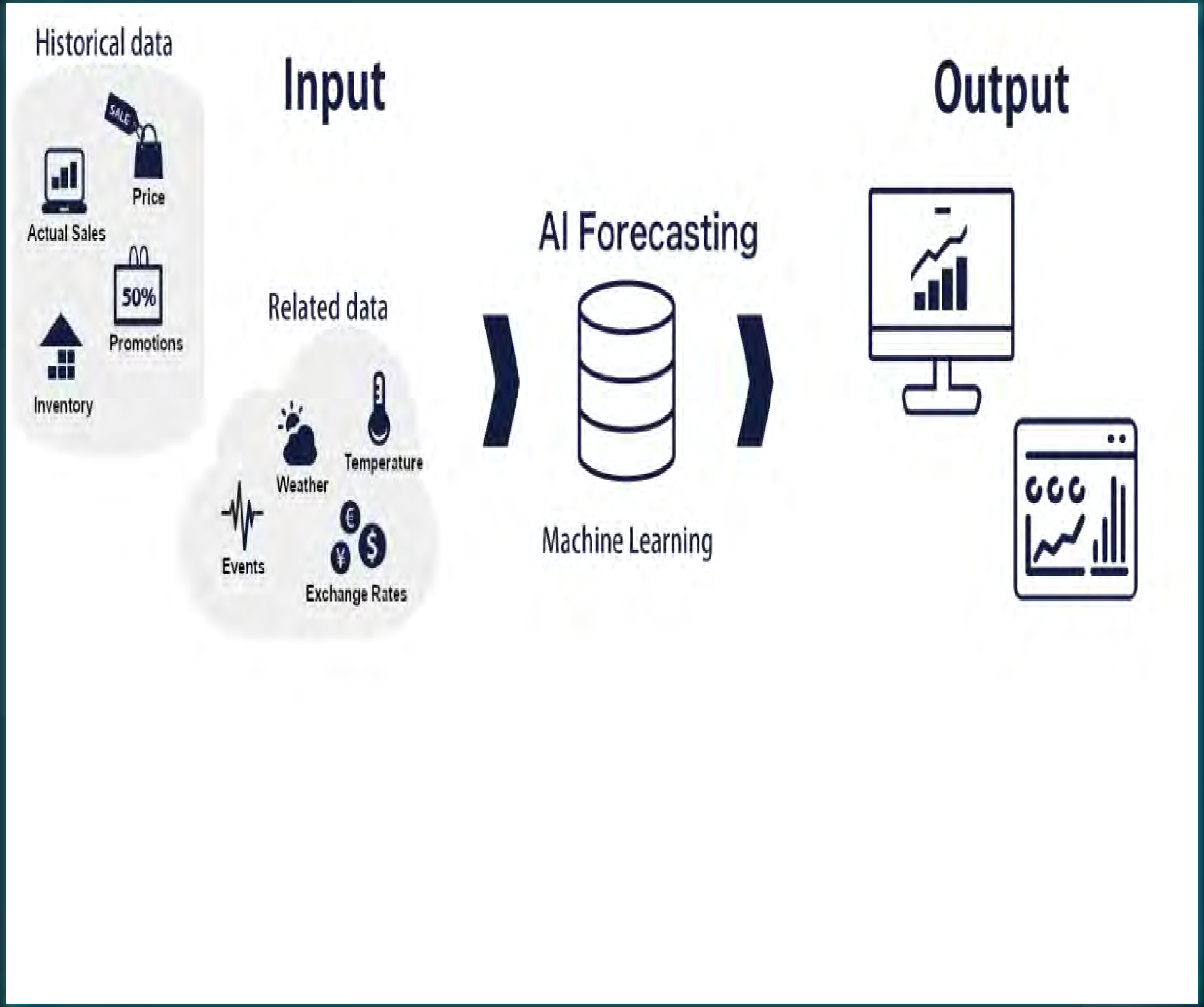
Sales Quantity: The quantity of the product sold.

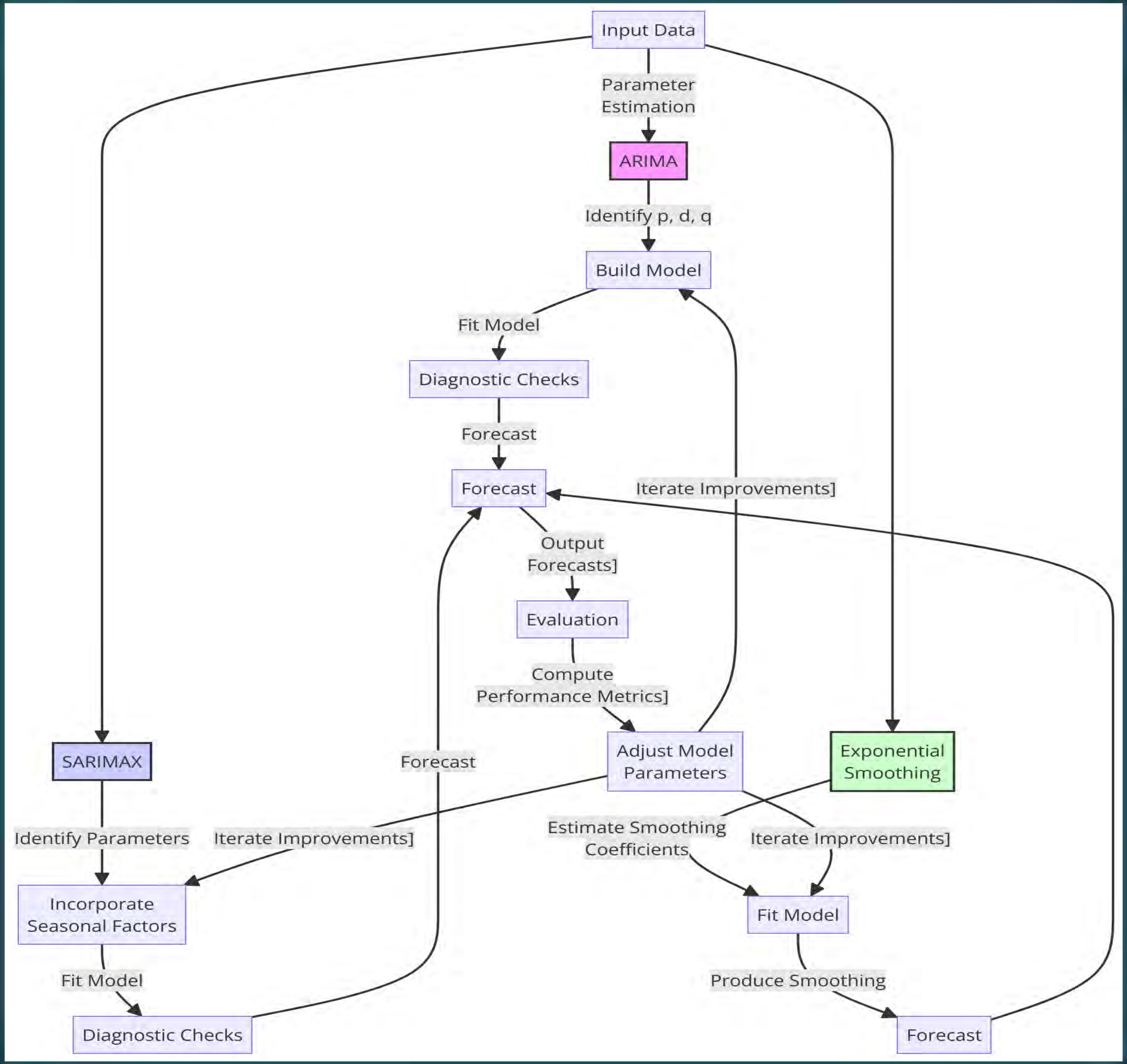
Transaction Date	Sales Quantity	Price	Discounts	Inventory	Economic	Holiday	Competitor Price	Market Trend
11/7/1996	79	466.39	5	81	0.78	1	489.7	0.85
11/8/1996	86	344.83	15	252	1.19	0	365.3	1.11
11/9/1996	80	138.61	0	322	1.15	1	141.2	0.88
11/10/1996	81	326.07	20	223	0.7	1	340.5	0.82
11/11/1996	81	425.8	10	358	1.19	0	462.3	1.16
11/12/1996	79	206.8	5	287	0.79	0	205.4	0.93
11/13/1996	78	416.68	15	405	0.7	0	421.3	1
11/14/1996	83	85.66	15	437	1.25	0	89.7	0.84
11/15/1996	84	413.16	5	159	0.86	1	415.9	0.99
11/16/1996	84	274.74	15	461	1.07	0	278.6	0.89
11/17/1996	88	119.26	20	120	0.7	1	132.3	1.14
11/18/1996	85	209.5	15	193	1.11	0	215.9	1.09
11/19/1996	79	494.12	0	255	0.71	1	500.9	1.07
11/20/1996	75	393.94	10	406	0.67	0	395	1.11

Forecasting Models

Time Series Models:

ARIMA, SARIMAX and Exponential Smoothing





Auto ARIMA Process

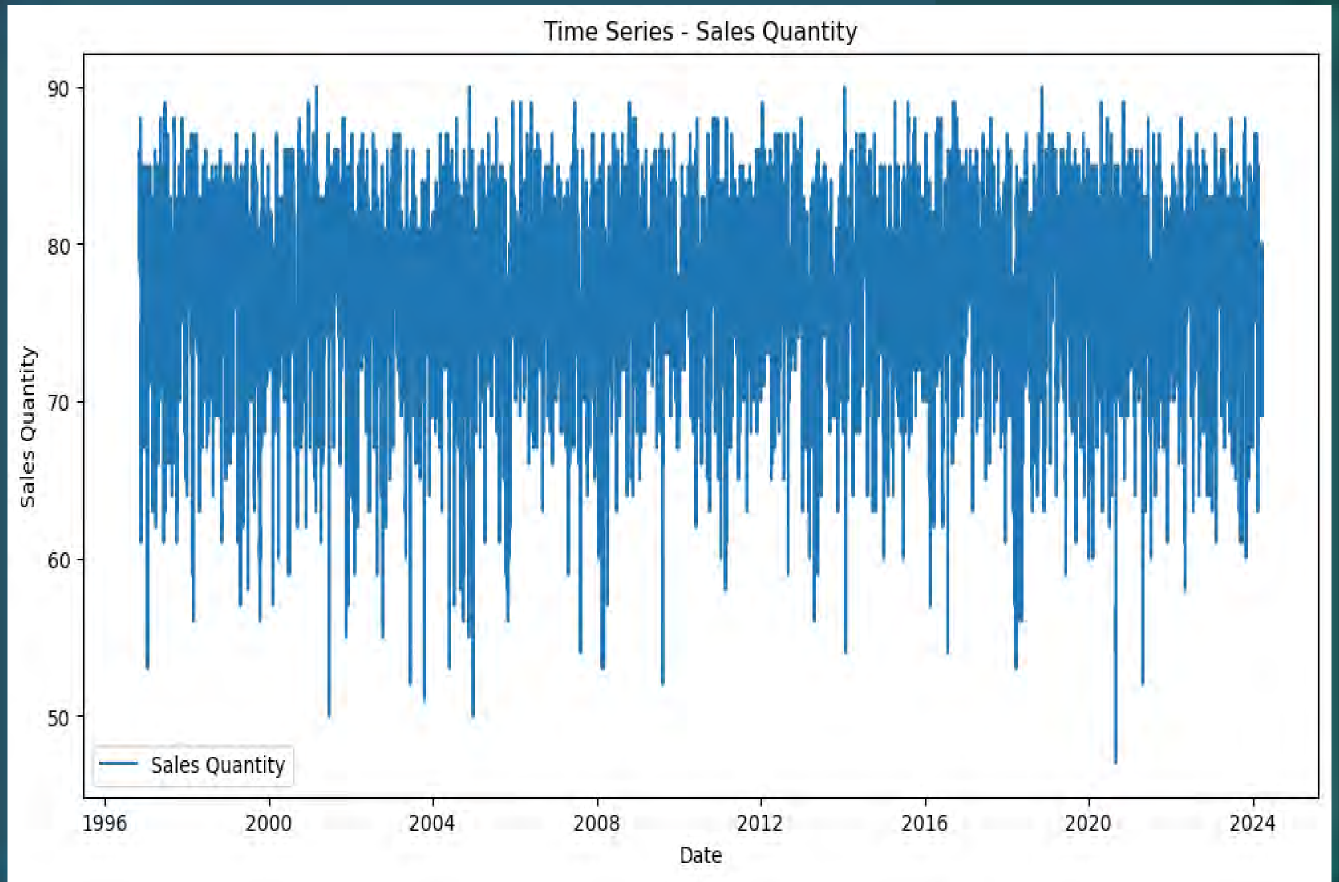
Auto ARIMA, or Automatic Autoregressive Integrated Moving Average, is a statistical algorithm used for time series forecasting. It automatically selects the best parameters (p, d, q) for an ARIMA model.

- AIC (Akaike Information Criterion)
- MAE (Mean Absolute Error)

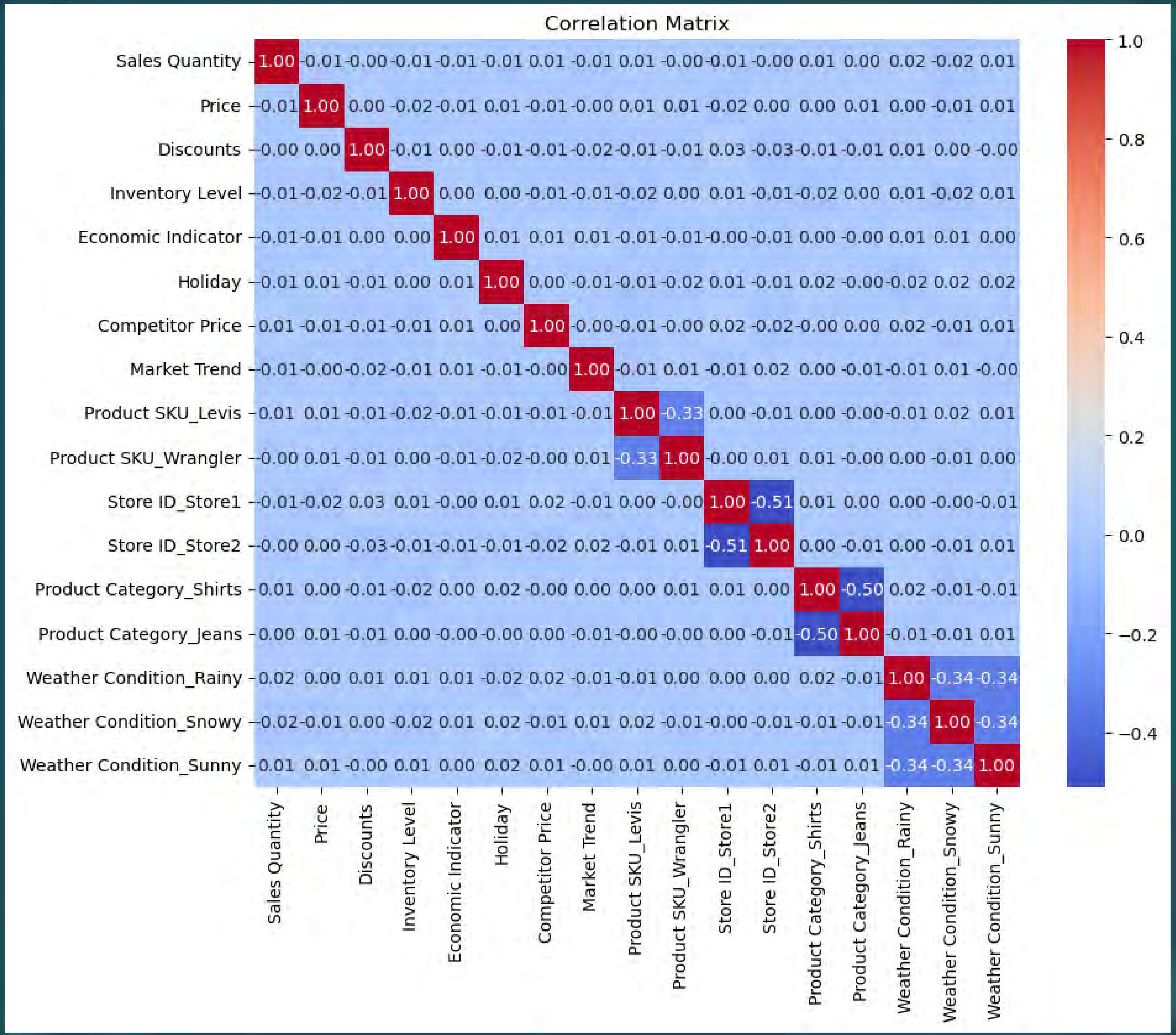
```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=53116.011, Time=24.55 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=62700.306, Time=0.12 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=53209.721, Time=14.88 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=57015.634, Time=3.89 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=115431.584, Time=0.13 sec
 ARIMA(2,0,2)(0,0,1)[12] intercept   : AIC=52973.517, Time=20.84 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=52932.323, Time=10.76 sec
 ARIMA(2,0,2)(1,0,0)[12] intercept   : AIC=53199.570, Time=29.64 sec
 ARIMA(1,0,2)(0,0,0)[12] intercept   : AIC=53083.688, Time=5.65 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=53073.279, Time=9.29 sec
 ARIMA(3,0,2)(0,0,0)[12] intercept   : AIC=52829.668, Time=14.68 sec
 ARIMA(3,0,2)(1,0,0)[12] intercept   : AIC=inf, Time=33.14 sec
 ARIMA(3,0,2)(0,0,1)[12] intercept   : AIC=53084.263, Time=25.79 sec
 ARIMA(3,0,2)(1,0,1)[12] intercept   : AIC=inf, Time=33.06 sec
 ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=52835.850, Time=12.69 sec
 ARIMA(4,0,2)(0,0,0)[12] intercept   : AIC=52743.740, Time=15.37 sec
 ARIMA(4,0,2)(1,0,0)[12] intercept   : AIC=inf, Time=29.71 sec
 ARIMA(4,0,2)(0,0,1)[12] intercept   : AIC=52819.652, Time=27.60 sec
 ARIMA(4,0,2)(1,0,1)[12] intercept   : AIC=inf, Time=37.95 sec
 ARIMA(4,0,1)(0,0,0)[12] intercept   : AIC=52719.267, Time=13.15 sec
 ARIMA(4,0,1)(1,0,0)[12] intercept   : AIC=53458.334, Time=28.87 sec
 ARIMA(4,0,1)(0,0,1)[12] intercept   : AIC=52720.786, Time=22.46 sec
 ARIMA(4,0,1)(1,0,1)[12] intercept   : AIC=inf, Time=31.80 sec
 ARIMA(4,0,0)(0,0,0)[12] intercept   : AIC=52765.229, Time=1.34 sec
 ARIMA(5,0,1)(0,0,0)[12] intercept   : AIC=51947.519, Time=16.15 sec
 ARIMA(5,0,1)(1,0,0)[12] intercept   : AIC=54691.712, Time=39.03 sec
 ARIMA(5,0,1)(0,0,1)[12] intercept   : AIC=52073.883, Time=26.98 sec
 ARIMA(5,0,1)(1,0,1)[12] intercept   : AIC=inf, Time=38.15 sec
 ARIMA(5,0,0)(0,0,0)[12] intercept   : AIC=52500.208, Time=2.20 sec
 ARIMA(5,0,2)(0,0,0)[12] intercept   : AIC=52262.814, Time=20.45 sec
 ARIMA(5,0,1)(0,0,0)[12]             : AIC=53922.838, Time=1.90 sec

Best model:  ARIMA(5,0,1)(0,0,0)[12] intercept
Total fit time: 592.269 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                10000
Model:               SARIMAX(5, 0, 1)   Log Likelihood              -25965.760
Date:                Fri, 12 Apr 2024   AIC                          51947.519
Time:                        20:31:31   BIC                          52005.202
Sample:                             0   HQIC                         51967.044
                              - 10000
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     37.5853      0.899     41.814      0.000      35.824      39.347
ar.L1          0.2404      0.021     11.429      0.000       0.199       0.282
ar.L2          0.5018      0.019     26.738      0.000       0.465       0.539
ar.L3         -0.0043      0.011     -0.393      0.694      -0.026       0.017
ar.L4          0.0913      0.011      8.034      0.000       0.069       0.114
ar.L5         -0.3154      0.011    -29.289      0.000      -0.336      -0.294
ma.L1          0.5917      0.022     27.048      0.000       0.549       0.635
sigma2        10.9274      0.141     77.463      0.000      10.651      11.204
===================================================================================
Ljung-Box (L1) (Q):                  26.54   Jarque-Bera (JB):               609.40
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.91   Skew:                            -0.44
Prob(H) (two-sided):                  0.01   Kurtosis:                         3.83
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
Mean Absolute Error: 2.6730952813582376
```

EDA

Time Series Plot



Correlation Matrix



Code Snippet

Data Preparation and cleaning

```python
# Load the datasets
train_df = pd.read_csv('C:/Personal/Demand Prediction/forecasting/trainingData.csv')
test_df = pd.read_csv('C:/Personal/Demand Prediction/forecasting/testData.csv')

# Convert date columns and set as index, then explicitly set frequency
train_df['Transaction Date'] = pd.to_datetime(train_df['Transaction Date'])
test_df['Transaction Date'] = pd.to_datetime(test_df['Transaction Date'])
train_df.set_index('Transaction Date', inplace=True)
test_df.set_index('Transaction Date', inplace=True)
train_df.index.freq = 'D'
test_df.index.freq = 'D'
```

EDA

```python
# Plotting the time series of the target variable
plt.figure(figsize=(12, 6))
plt.plot(train_df['Sales Quantity'], label='Sales Quantity')
plt.title('Time Series - Sales Quantity')
plt.xlabel('Date')
plt.ylabel('Sales Quantity')
plt.legend()
plt.show()

# Plotting histograms for all numeric variables to understand distributions
train_df.hist(bins=20, figsize=(15, 10))
plt.tight_layout()
plt.show()

# Heatmap for correlation analysis
plt.figure(figsize=(10, 8))
sns.heatmap(train_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

Defining Models and Fitting

```python
# Define models
models = {
    'ARIMA': ARIMA(y_train, order=(5,0,1)),
    'SARIMAX': SARIMAX(y_train, exog=X_train, order=(5,0,1), seasonal_order=(1,1,1,12)),  # Adding seasonal components
    'ExpSmoothing': ExponentialSmoothing(y_train, trend='add', seasonal='add', seasonal_periods=12)
}

# Forecasting and evaluating
forecast_results = {}
mae_scores = {}
pearson_correlations = {}

for model_name, model in models.items():
    fitted_model = model.fit()
    if model_name == 'SARIMAX':
        forecast = fitted_model.get_forecast(steps=len(y_test), exog=X_test).predicted_mean
    else:
        forecast = fitted_model.forecast(steps=len(y_test))

    forecast_results[model_name] = forecast
    mae_scores[model_name] = mean_absolute_error(y_test, forecast)
    pearson_correlations[model_name] = pearsonr(y_test, forecast)[0]
```

Forecasting and Plotting

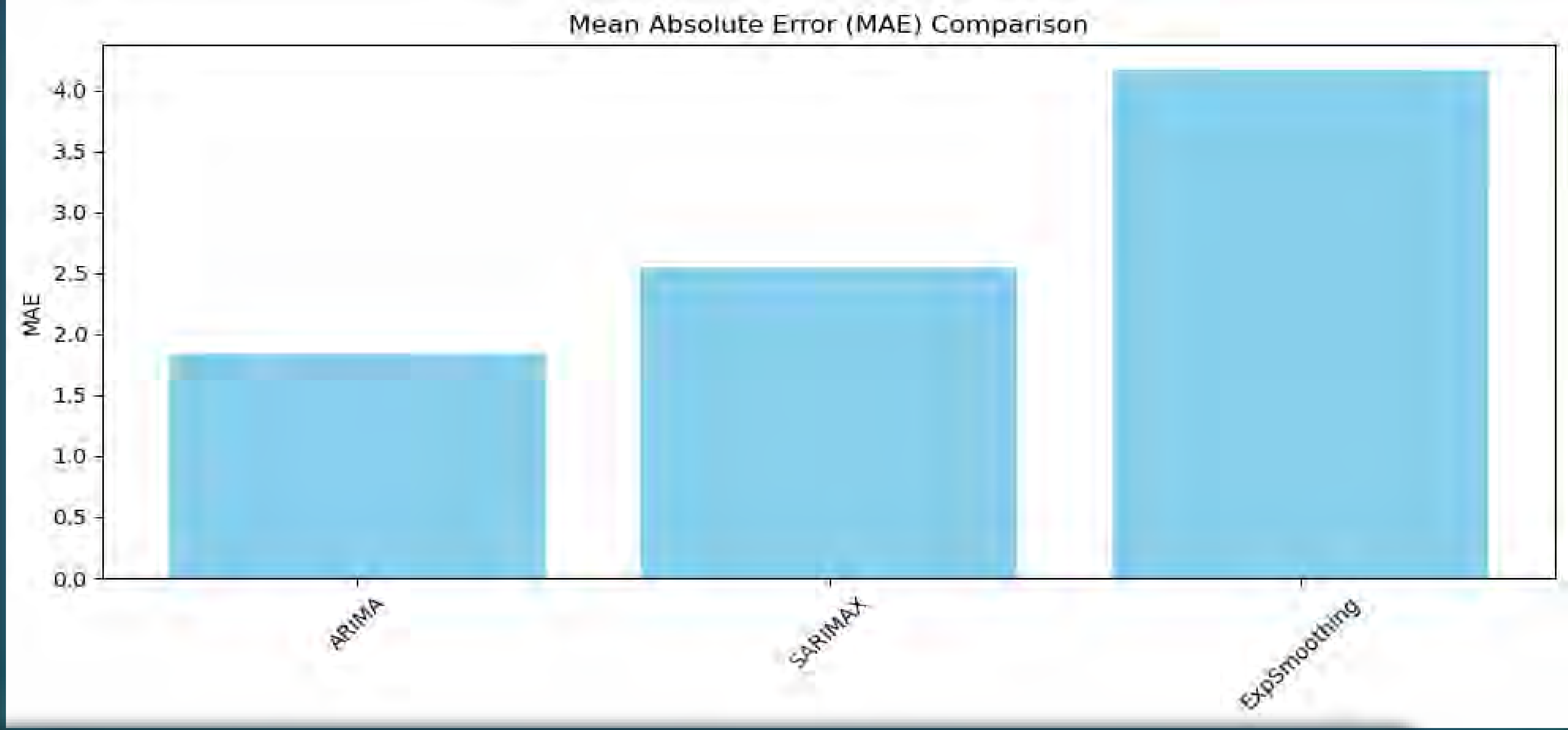
```python
# Print forecasted values
for model_name, forecast in forecast_results.items():
    print(f"Forecast from {model_name}:")
    print(forecast.to_string(), "\n")  # Using to_string() for nicer formatting

# Outputting the forecasts and errors
print("Mean Absolute Errors:")
for model, mae in mae_scores.items():
    print(f"{model}: {mae}")

print("\nPearson Correlation Coefficients:")
for model, corr in pearson_correlations.items():
    print(f"{model}: {corr}")

# Plotting MAE Scores
plt.figure(figsize=(10, 5))
plt.bar(mae_scores.keys(), mae_scores.values(), color='skyblue')
plt.title('Mean Absolute Error (MAE) Comparison')
plt.ylabel('MAE')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Plotting Pearson Correlation
plt.figure(figsize=(10, 5))
plt.bar(pearson_correlations.keys(), pearson_correlations.values(), color='lightgreen')
plt.title('Pearson Correlation Coefficient Comparison')
plt.ylabel('Pearson Correlation Coefficient')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

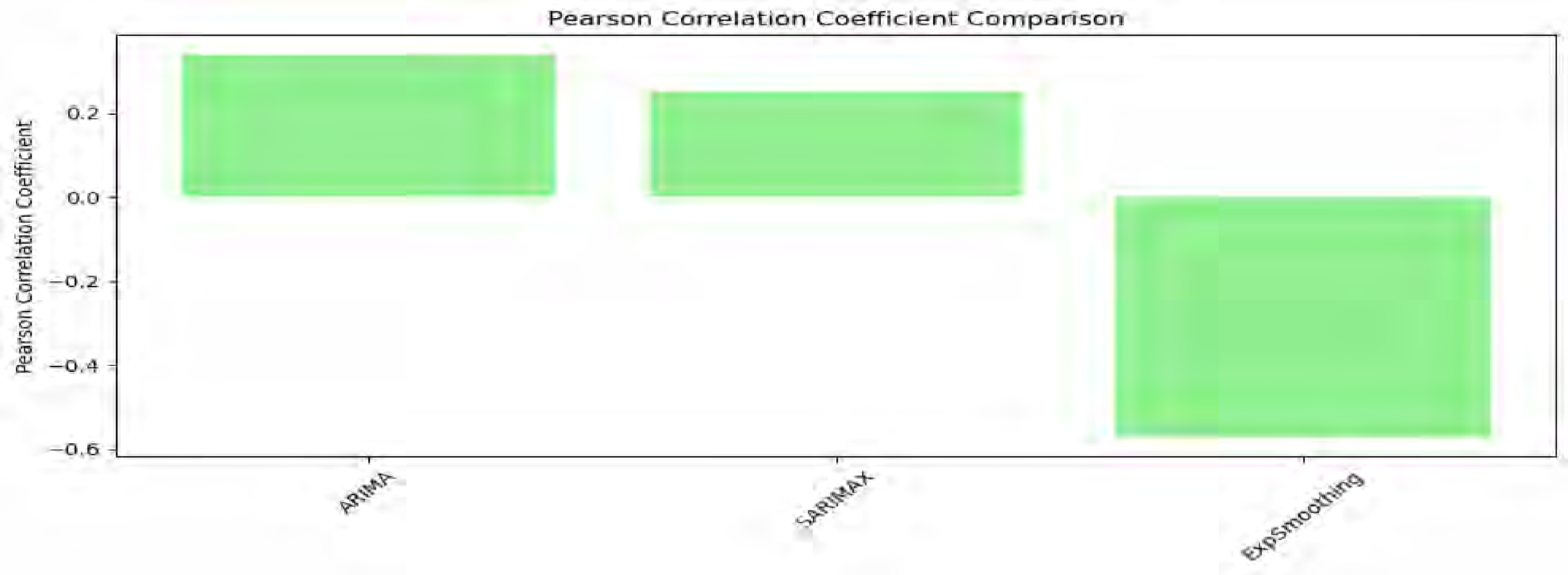
Results

Forecasting Results

Forecast - ARIMA		Forecast - SARIMAX		Forecast - Exponential Smoothing		
2024-03-25	80.121864	2024-03-25	78.694814	2024-03-25	79.689730	
2024-03-26	80.572541	2024-03-26	79.784029	2024-03-26	79.971674	
2024-03-27	82.103513	2024-03-27	84.176492	2024-03-27	80.621531	
2024-03-28	81.065124	2024-03-28	83.536920	2024-03-28	80.248024	
2024-03-29	79.886376	2024-03-29	80.551936	2024-03-29	79.602761	Mean Absolute Errors:
2024-03-30	79.111195	2024-03-30	80.866445	2024-03-30	80.338572	ARIMA: 1.8419574455117966
2024-03-31	78.301879	2024-03-31	81.059235	2024-03-31	80.425297	SARIMAX: 2.546420453744377
2024-04-01	77.174891	2024-04-01	76.024025	2024-04-01	80.329099	ExpSmoothing: 4.160054782239185
2024-04-02	76.769915	2024-04-02	75.918407	2024-04-02	81.264618	
2024-04-03	76.480582	2024-04-03	75.989085	2024-04-03	81.264901	
2024-04-04	76.411456	2024-04-04	74.602263	2024-04-04	80.661868	Pearson Correlation Coefficients:
2024-04-05	76.455309	2024-04-05	74.782415	2024-04-05	81.278708	ARIMA: 0.33961288574213533
2024-04-06	76.771467	2024-04-06	74.935686	2024-04-06	81.160147	SARIMAX: 0.2530820893056085
2024-04-07	76.988242	2024-04-07	74.812789	2024-04-07	81.442091	ExpSmoothing: -0.5720288204039103
2024-04-08	77.277617	2024-04-08	76.964718	2024-04-08	82.091948	
2024-04-09	77.480551	2024-04-09	78.300250	2024-04-09	81.718440	
2024-04-10	77.670732	2024-04-10	77.458779	2024-04-10	81.073178	
2024-04-11	77.726689	2024-04-11	78.617665	2024-04-11	81.808989	
2024-04-12	77.776080	2024-04-12	80.034087	2024-04-12	81.895714	
2024-04-13	77.735111	2024-04-13	78.109778	2024-04-13	81.799516	

MAE and Pearson Graphs





Using machine learning to optimize order sourcing and routing

PREDICTIVE DELIVERY DATE

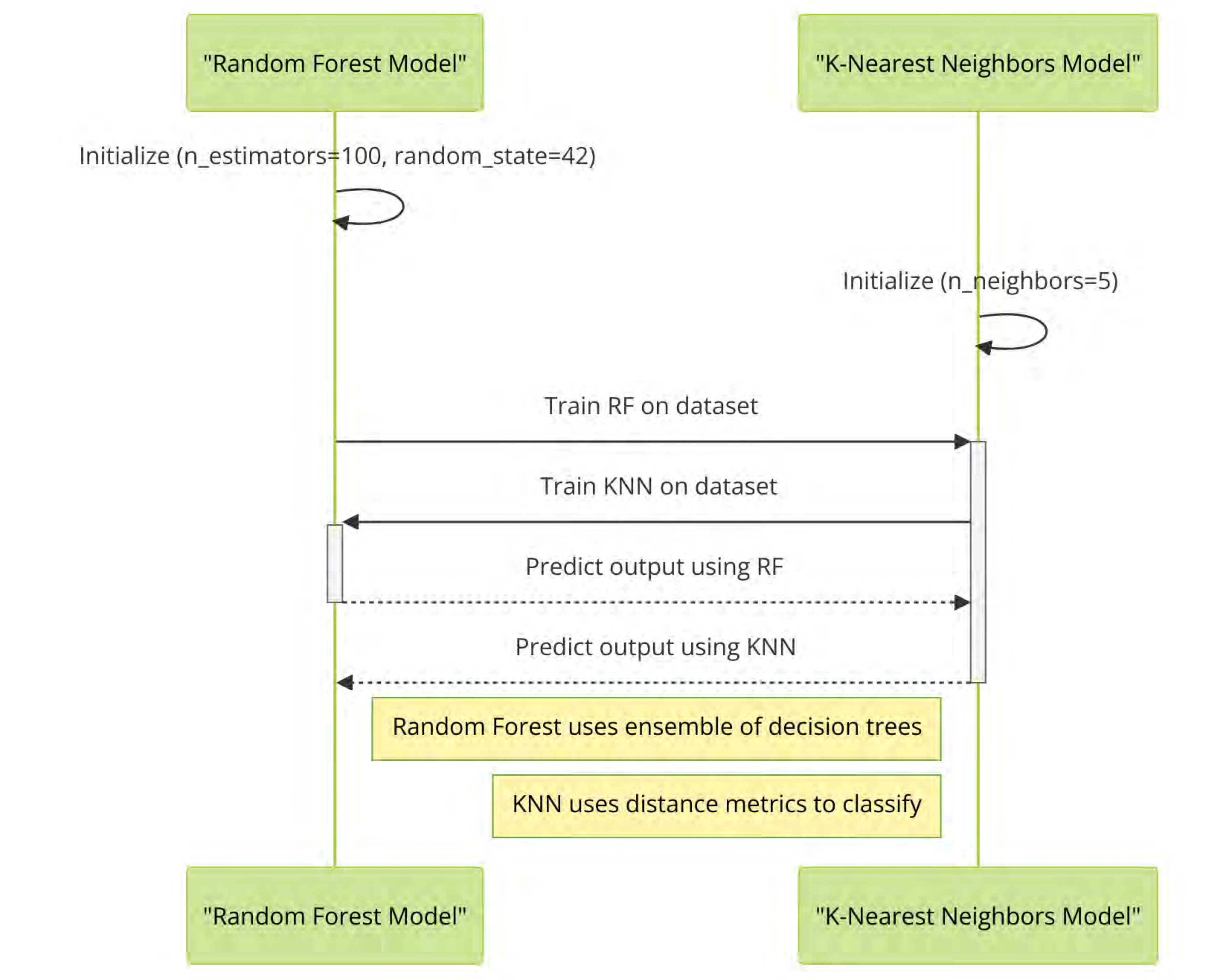
A predicted delivery date is an estimate of when a particular item or service will be delivered to the customer or recipient based on various factors such as shipping time, processing time, and logistical considerations.

SOURCING AND ROUTING

- Sourcing of orders refers to the process of determining where products or services will be obtained to fulfill customer orders. It can be stores, warehouse, vendors etc.
- The routing decision may be based on factors such as proximity to the customer, inventory availability, and shipping method selected by the customer.

Predictive Models

- **RandomForestRegressor**
- **KNeighborsClassifier**



Use Case

Data Description:
The dataset contains historical delivery data of Company A. Key columns include:

Customer Zip Code: Zip Code of the customer.
Store Zip Code: Zip Code of the store which fulfilled the order.
Shipping Option: Shipping Option selected for the order.
Fulfillment Success: Whether delivery was successful or not.

DayOfWeek	TimeOfDay	Season	Customer	StoreZipC	OrderedQ	ItemID	ShippingOption	HasDiscount	IsPeakSeason	IsWeekend	Distance	Inventory	FulfillmentSuccess
Sun	Afternoon	Winter	64656	61814	39	Item4	Overnight	TRUE	FALSE	TRUE	326.2809461	84	1
Thu	Evening	Summer	32940	98380	29	Item2	Overnight	FALSE	FALSE	TRUE	2612.493703	71	0
Fri	Morning	Winter	41039	91208	15	Item3	Standard	TRUE	TRUE	TRUE	1922.568494	86	0
Sun	Afternoon	Summer	65591	80137	43	Item2	Standard	FALSE	TRUE	FALSE	655.8263626	97	0
Wed	Afternoon	Fall	1068	30656	8	Item4	Expedited	FALSE	TRUE	FALSE	861.9275401	50	0
Fri	Evening	Spring	12166	60620	21	Item4	Standard	TRUE	FALSE	FALSE	677.5486561	36	0
Fri	Evening	Spring	45674	70462	39	Item3	Standard	FALSE	TRUE	FALSE	749.4038736	74	1
Sun	Evening	Fall	88256	67122	19	Item4	Expedited	FALSE	TRUE	FALSE	564.4064236	55	0
Tue	Evening	Summer	63384	76431	23	Item3	Standard	TRUE	FALSE	TRUE	530.9476708	45	1
Wed	Evening	Winter	22718	14012	11	Item4	Standard	FALSE	FALSE	FALSE	337.8130216	72	1
Sun	Night	Summer	48335	59484	11	Item1	Overnight	FALSE	FALSE	TRUE	1429.41548	36	0
Wed	Evening	Spring	27872	79915	24	Item3	Overnight	FALSE	TRUE	FALSE	1689.288813	44	0
Wed	Morning	Fall	30082	48721	36	Item2	Expedited	FALSE	TRUE	TRUE	758.6494335	21	0
Fri	Afternoon	Fall	57059	75929	40	Item3	Overnight	TRUE	TRUE	FALSE	843.7684662	24	0
Thu	Morning	Summer	76311	41564	24	Item4	Standard	TRUE	TRUE	FALSE	936.8152641	23	0
Wed	Morning	Summer	56567	37030	3	Item4	Overnight	FALSE	FALSE	TRUE	859.4987025	55	1
Sat	Night	Fall	63028	24378	22	Item4	Overnight	FALSE	FALSE	FALSE	501.5343452	25	1
Fri	Evening	Spring	34482	12546	2	Item2	Expedited	TRUE	TRUE	FALSE	1004.562257	40	0
Tue	Afternoon	Winter	65604	67853	24	Item2	Standard	TRUE	TRUE	TRUE	380.5002614	96	1
Thu	Morning	Winter	32680	45776	44	Item4	Expedited	FALSE	FALSE	TRUE	665.9569462	45	0
Sat	Evening	Summer	98650	56547	30	Item1	Standard	FALSE	TRUE	TRUE	1185.265987	74	1
Sat	Evening	Fall	34251	44671	38	Item3	Standard	TRUE	TRUE	FALSE	917.1656037	58	0

Code Snippet

Data Preparation

```python
# Load the data
data_path = 'C:/Personal/EDD1/EDD/trainingData.csv'
data = pd.read_csv(data_path)
us_zips = pd.read_csv('C:/Personal/EDD1/EDD/uszips.csv')

# Prepare the features and labels
features = data_encoded.drop(['CustomerZipCode', 'StoreZipCode', 'FulfillmentSuccess'], axis=1)
labels = data_encoded['FulfillmentSuccess'].astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features.values, labels.values, test_size=0.2, random_state=42)
X_train = np.ascontiguousarray(X_train)
X_test = np.ascontiguousarray(X_test)
```

Training & Modelling

```python
# Initialize and train models
models = {
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=5)
}

# Train models
for name, model in models.items():
    model.fit(X_train, y_train)
```

Predicting Delivery Date and finding best route

```python
# Input shipping option
shipping_option = 'Standard'

# Finding inventory for Item2 at the nearest suitable store
customer_zip_code = 19063
customer_location = us_zips[us_zips['zip'] == customer_zip_code][['lat', 'lng']].iloc[0].apply(radians)
us_zips_rad = us_zips[['lat', 'lng']].applymap(radians)
distances = haversine_distances(customer_location.to_numpy().reshape(1, -1), us_zips_rad).flatten() * 6371000 / 1609.34  # Conver

# Rank stores by distance and check inventory sequentially
sorted_store_indices = np.argsort(distances)
for idx in sorted_store_indices:
    store_zip = us_zips.iloc[idx]['zip']
    inventory_filter = (data['ItemID'] == 'Item2') & (data['Inventory'] >= requested_quantity) & (data['StoreZipCode'] == store_
    nearest_inventory = data[inventory_filter]
    if not nearest_inventory.empty:
        store_city = us_zips[us_zips['zip'] == store_zip]['city'].iloc[0]
        distance = distances[idx]
        print(Fore.GREEN + "Inventory Information" + Style.RESET_ALL)
        print(tabulate([["CustomerZip", "ReqQty", "AvailableQty", "StoreZip", "City", "Distance to Customer(miles)", "Shipping Op
                        [customer_zip_code, requested_quantity, nearest_inventory['Inventory'].iloc[0], store_zip, store_city, f'

        # Prepare test data for prediction
        test_features = pd.get_dummies(nearest_inventory.drop(['CustomerZipCode', 'StoreZipCode', 'FulfillmentSuccess'], axis=1))
        test_features = test_features.reindex(columns=features.columns, fill_value=0).astype(float)  # Align columns with trainin
        test_features = np.ascontiguousarray(test_features.values)  # Ensure C-contiguity

        today = datetime.now()
        predictions = np.mean([model.predict_proba(test_features)[:, 1] for model in models.values()], axis=0)  # Average predict
        average_prediction = np.mean(predictions)

        print(Fore.BLUE + "Estimated Delivery Dates and MAE for Different Models" + Style.RESET_ALL)
        table_data = []
        for name, model in models.items():
            y_pred = model.predict(test_features)
            if not shipping_option:
                # If shipping option is not provided, use the most frequent shipping option from the training data
                shipping_option = data['ShippingOption'].mode()[0]
            estimated_date = estimate_delivery_date(today, shipping_option, distance, average_prediction)  # Use input shipping

            # Ensure y_pred has the same length as y_test
            y_pred = np.array([y_pred] * len(y_test))

            # Calculate MAE
            mae = mean_absolute_error(y_test, y_pred)
            table_data.append([name, estimated_date.strftime("%Y-%m-%d")])

        print(tabulate(table_data, headers=['Model', 'Predicted Delivery Date'], tablefmt='grid'))
        break
else:
    print("No available inventory for Item2 at any nearby store.")
```

Results

Prediction when Shipping option is Standard

Inventory Information

CustomerZip	ReqQty	AvailableQty	StoreZip	City	Distance to Customer(miles)	Shipping Option
19063	100	110	89402	Crystal Bay	2349.88	Standard

Estimated Delivery Dates and MAE for Different Models

Model	Predicted Delivery Date
Random Forest	2024-05-26
KNN	2024-05-26

Prediction when Shipping Option in input is Overnight

Inventory Information

CustomerZip	ReqQty	AvailableQty	StoreZip	City	Distance to Customer(miles)	Shipping Option
19063	100	110	89402	Crystal Bay	2349.88	Overnight

Estimated Delivery Dates and MAE for Different Models

Model	Predicted Delivery Date
Random Forest	2024-05-21
KNN	2024-05-21

Prediction when Quantity is reduced to 10 and no Shipping Option

Inventory Information

CustomerZip	ReqQty	AvailableQty	StoreZip	City	Distance to Customer(miles)	Shipping Option
19063	10	37	19052	Lenni	2.4	

Estimated Delivery Dates and MAE for Different Models

Model	Predicted Delivery Date
Random Forest	2024-05-22
KNN	2024-05-22

When there is no inventory

No available inventory for Item2 at any nearby store.

Predictive analytics to allocate omni-channel inventory dynamically

Improve Inventory Management

Inadequate inventory management results in reduced sales, creating a misleading impression of decreased demand for specific items. Consequently, future order forecasts based on this flawed historical data are inherently unreliable.

Smart Prediction

Smart retailers leverage real-time data to strategically distribute inventory to high-demand areas preemptively. Furthermore, they utilize predictive analytics to determine optimal stock levels and distribution locations, informed by insights into regional preferences, weather patterns, and other relevant factors.

Use Case

Data Description:
The dataset contains historical product data of Company A. Key columns include:

Product ID: Item ID for the product.
Cost per unit: Unit price for the item.
Revenue: Revenue that product has generated for Company A.
Number of product units: Inventory level of the product.

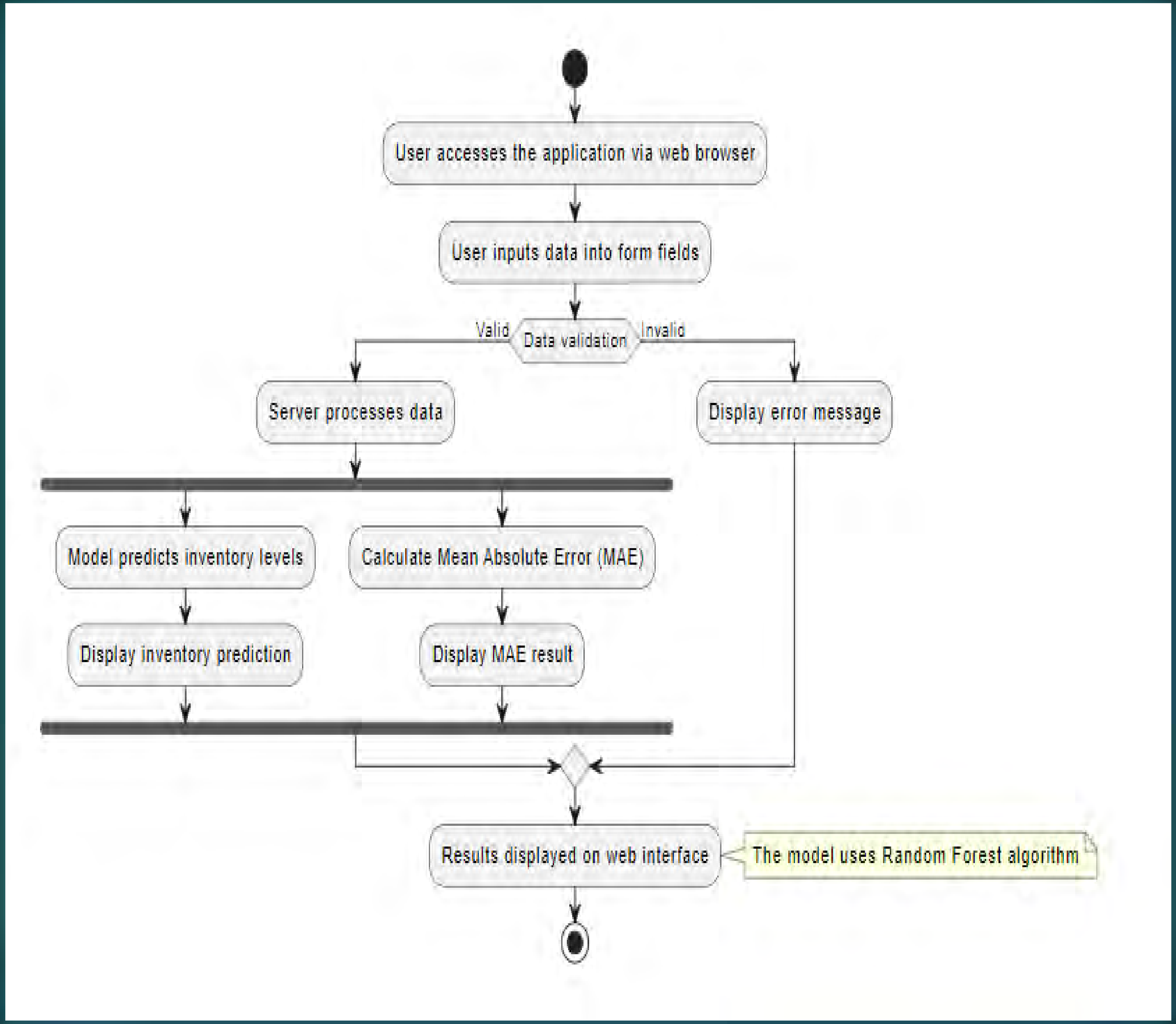
product_id	cost_per_unit	time_delivery	revenue	generic_holiday	day_of_week	number_of_product_units
30	275	7	5784	0	2	24
493	882	9	8987	0	6	12
825	1107	10	3791	0	3	4
449	1672	14	16802	0	1	12
867	1308	8	5153	0	2	5
209	593	5	2757	0	7	6
901	80	13	4516	0	6	64
491	405	7	4439	0	4	13
914	277	14	10021	0	5	41
612	1419	12	8260	0	2	7
584	1239	13	4032	0	2	4
1104	508	14	4388	0	1	10
667	400	13	11364	0	5	32
473	298	5	4959	0	2	19
760	992	6	8786	0	2	10
662	182	14	14005	0	4	87
924	1646	13	3178	0	3	3
242	1541	8	4074	0	5	3
363	1985	12	5911	0	2	4
732	1653	6	6330	0	2	5
135	177	9	4392	0	3	28
611	1718	12	6155	0	4	5
518	1278	5	6753	0	6	6
892	945	14	6066	0	1	8
478	1446	8	10973	0	2	9
1032	1737	7	8512	0	2	6
914	277	14	14264	0	2	58

Demo

Technical Stack:
Python, Flask

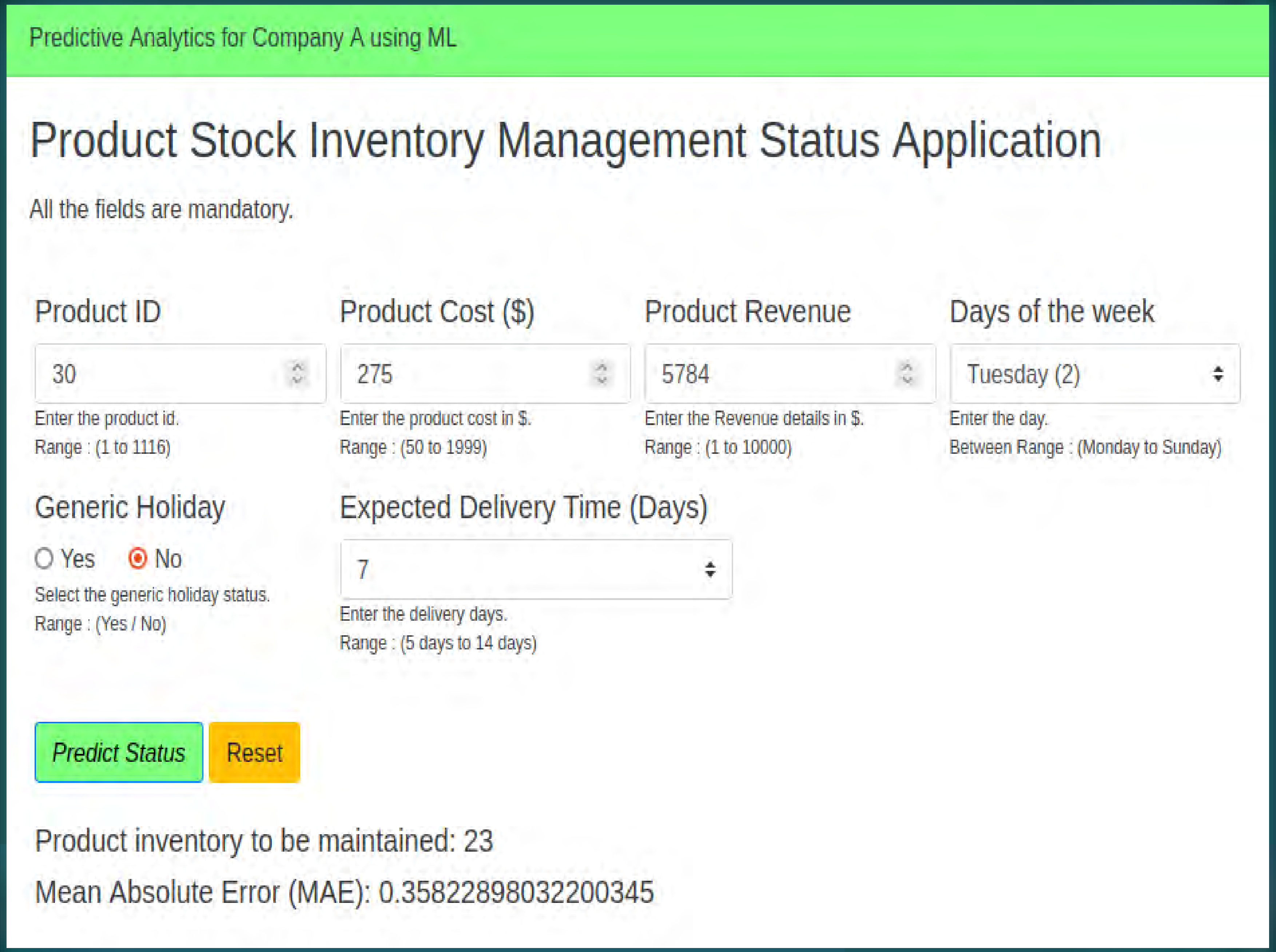
Model used to train: RandomForestRegressor

Output: Predicts the inventory level to be maintained for a particular Product ID based on prediction done by the model. It also prints the MAE for each prediction.

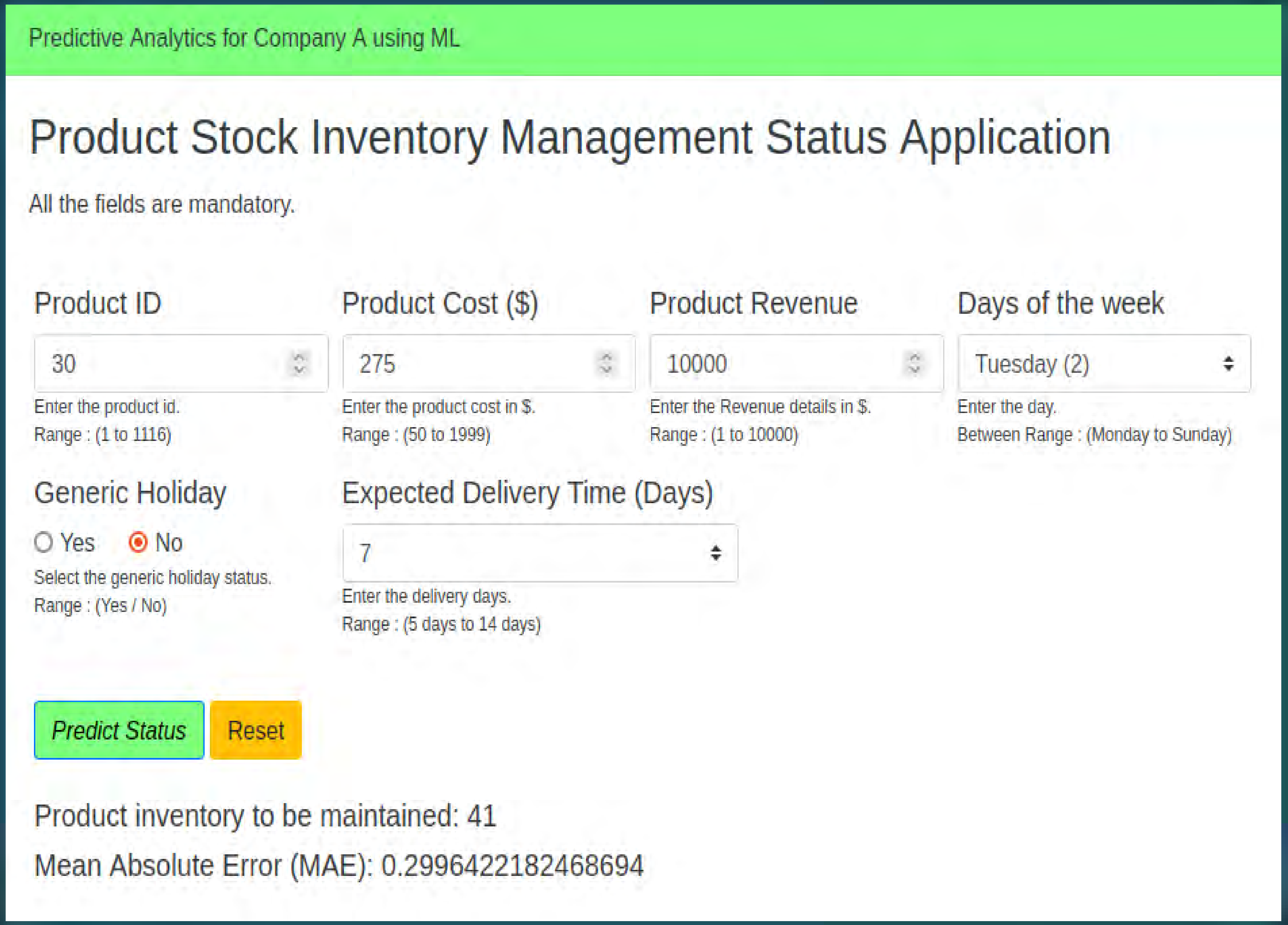


Results

Model Prediction with same revenue as training data:



Model Prediction with increased revenue:

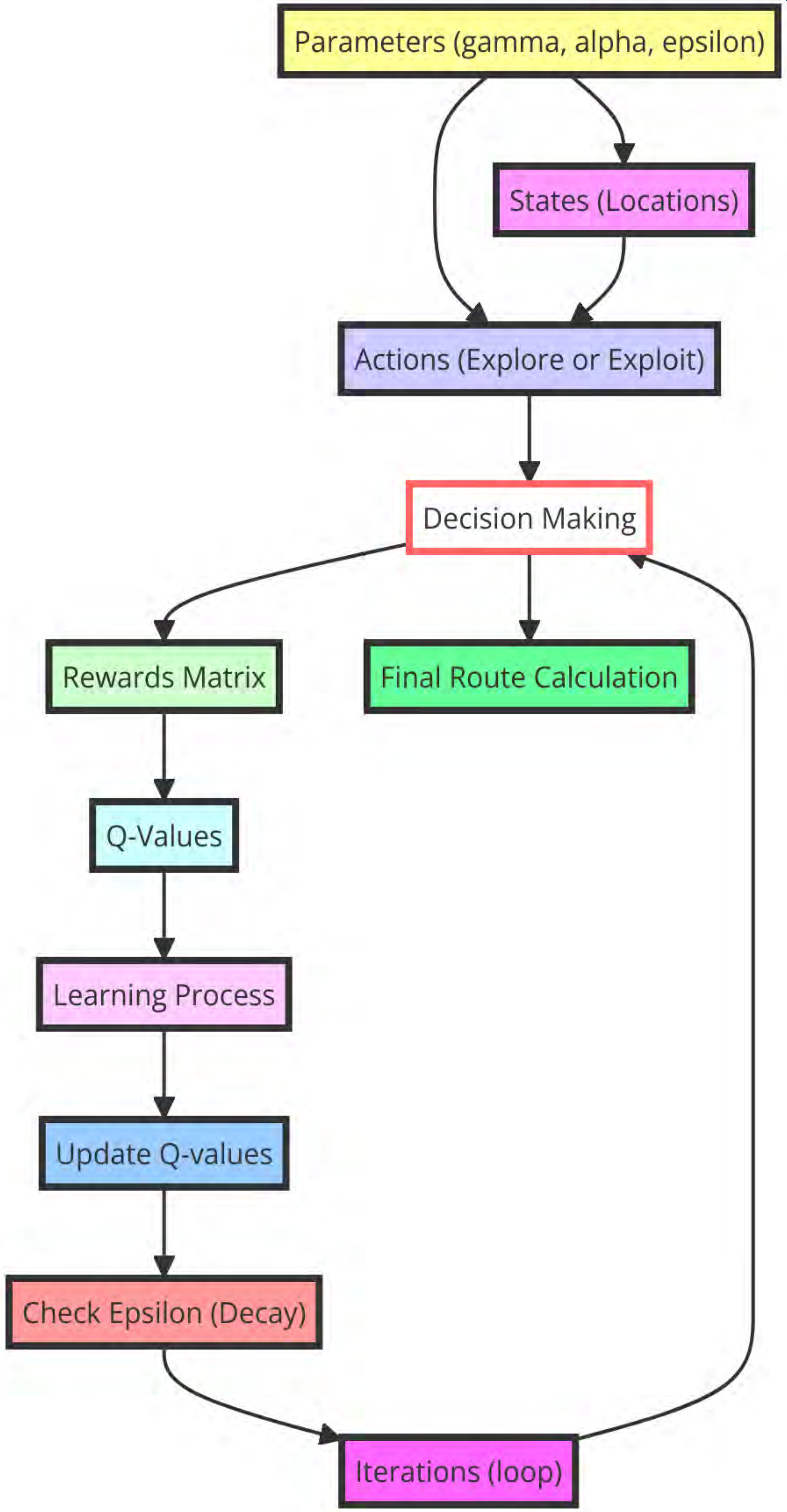


Reinforcement learning to optimize allocation across fulfillment centers

Reinforcement learning (RL) is a branch of machine learning concerned with making sequential decisions in an environment to maximize cumulative rewards. In the context of optimizing allocation across fulfillment centers

RL Algorithm

Q-Learning: Q-Learning is one of the simplest RL algorithms. It learns the optimal action-selection policy for a Markov decision process (MDP) by iteratively updating the Q-values of state-action pairs based on the observed rewards and transitions.



Q-Learning

- **Agent** - The decision-maker
- **Environment** – The system in which agent operates
- **Actions** – The decisions which agent can take
- **Rewards** – Feedback provided to agent based on it’s actions
- **Learning** – The agent learns from interactions over time.

Code Snippet

Parameters

```python
# Parameters
gamma = 0.85  # Discount factor
alpha = 0.9   # Learning rate
initial_epsilon = 1.0  # Starting exploration factor
decay = 0.999  # Decay rate for epsilon
min_epsilon = 0.1  # Minimum value for epsilon

# States and actions
location_to_state = {
    'NewYork': 0, 'NewJersey': 1, 'Pennsylvania': 2, 'Delaware': 3, 'Maryland': 4,
    'Baltimore': 5, 'Washington DC': 6, 'Virginia': 7, 'North Carolina': 8,
    'South Carolina': 9, 'Georgia': 10, 'Florida': 11
}
state_to_location = {state: location for location, state in location_to_state.items()}
```

Reward

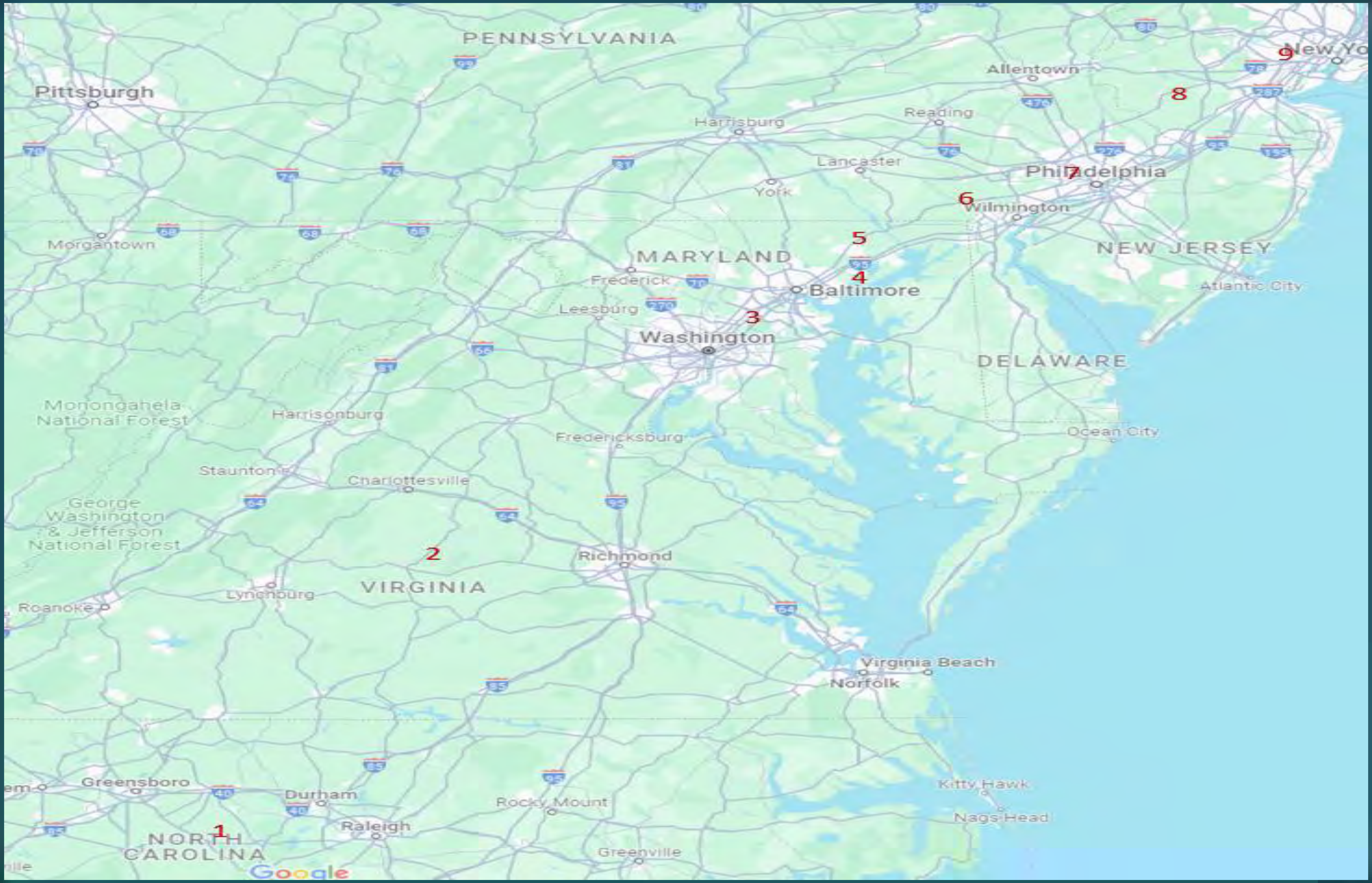
```python
# Reward matrix
R = np.array([
      [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],   # NY
      [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],   # NJ
      [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],   # PA
      [0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],   # DE
      [0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0],   # MD
      [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0],   # BM
      [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],   # DC
      [0, 0, 0, 0, 0, 0, 1, 0, 5, 0, 0, 1],   # VA
      [0, 0, 0, 0, 0, 0, 0, 5, 0, 5, 0, 0],   # NC
      [0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 5, 0],   # SC
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 1],   # GA
      [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0]    # FL
])
```

Q-Learning Modelling

```python
def route(starting_location, ending_location):
    start_state = location_to_state[starting_location]
    end_state = location_to_state[ending_location]
    R_new = np.copy(R)
    R_new[end_state, end_state] = 1000  # High reward for reaching the goal

    Q = np.zeros([12, 12])
    epsilon = initial_epsilon
    for i in range(50000):  # Increased iterations for more thorough learning
        current_state = np.random.randint(0, 12)
        if random.uniform(0, 1) < epsilon:
            playable_actions = np.where(R_new[current_state] > 0)[0]
            next_state = np.random.choice(playable_actions)
        else:
            next_state = np.argmax(Q[current_state])

        # Q-value update
        TD = R_new[current_state, next_state] + gamma * np.max(Q[next_state]) - Q[current_state, next_state]
        Q[current_state, next_state] += alpha * TD
        epsilon = max(min_epsilon, epsilon * decay)

    # Generate the route
    route = [starting_location]
    next_location = starting_location
    while next_location != ending_location:
        starting_state = location_to_state[starting_location]
        next_state = np.argmax(Q[starting_state])
        next_location = state_to_location[next_state]
        route.append(next_location)
        starting_location = next_location

    return route

final_route = route('North Carolina', 'NewYork')
print('Route:')
print(final_route)
```

Results

Map



Output

```
Route:
['North Carolina', 'Virginia', 'Washington DC', 'Baltimore', 'Maryland', 'Delaware', 'Pennsylvania', 'NewJersey', 'NewYork']
```

Q & A

Seeking answers to pressing questions or interested in exploring a specific topic? You've come to the right place!

THANK YOU!

EMAIL ADDRESS
jubin.thomas@ieee.org

LINKEDIN
Linkedin.com/in/contactjubinthomas

GITHUB
https://github.com/jubingithubid