## Sprint 5

***Day 9 Task 2:***

# Sprints for AI
# Use Case For Demand Forecasting Using POS Transaction Data



## Team Members:

| |
|---|
| Tufail Irshad |
| Amjad Ali Malik |
| Zameer Ahmad Mir |
| Asif Ahmad Najar |
| Irfan Ahmad Mutoo |

# Sprint 5 Day 9 Task 2

# AI Finalized Code for Machine Learning Model Ready for Deployment

## Identifying Missing Values and Detecting Outliers

The code provided focuses on loading a synthetic dataset, checking for missing values, and handling those missing values appropriately. Here is a detailed explanation of each step involved, including identifying which fields had missing values.

Importing Pandas and Loading the Dataset:

- The pandas library is imported as pd.
- The synthetic dataset pos_dataset.csv is read into a DataFrame named dff.

```
import pandas as pd

# Load the synthetic dataset
dff = pd.read_csv('/content/sample_data/pos_dataset1.csv')
dff.head(5)
dff.describe()
```

| | Date of Transaction | Item Purchased | Quantity | Unit Price | Tax | Transaction Amount | Transaction Mode | Transaction Currency | Total Amount In INR |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 20/01/2020 | Printer | 1.0 | 329.25 | 32.93 | 362.18 | Mobile Payment | USD | 30133.38 |
| 1 | 05/05/2021 | Smartphone | 3.0 | 645.78 | 193.74 | 2131.08 | Mobile Payment | INR | 710.36 |
| 2 | 15/05/2024 | Smartwatch | 3.0 | 727.45 | 218.25 | 2400.60 | Cash | USD | 66576.64 |
| 3 | 06/12/2022 | Smartphone | NaN | 552.72 | 110.54 | 1215.98 | NaN | USD | 50584.77 |
| 4 | 02/03/2023 | Headphones | NaN | 281.03 | 28.10 | 309.13 | NaN | INR | 309.13 |

| | Quantity | Unit Price | Tax | Transaction Amount | Total Amount In INR |
|---|---|---|---|---|---|
| **count** | 9970.000000 | 9998.000000 | 10000.000000 | 10000.000000 | 10000.000000 |
| **mean** | 3.010130 | 508.858579 | 152.662426 | 1679.277699 | 23669.625429 |
| **std** | 1.419805 | 286.705135 | 119.262316 | 1311.881747 | 29802.671677 |

| | Quantity | Unit Price | Tax | Transaction Amount | Total Amount In INR |
|---|---|---|---|---|---|
| **min** | 1.000000 | 10.000000 | 1.090000 | 12.040000 | 11.340000 |
| **25%** | 2.000000 | 260.392500 | 57.435000 | 631.700000 | 545.710000 |
| **50%** | 3.000000 | 509.205000 | 119.890000 | 1318.750000 | 1090.450000 |
| **75%** | 4.000000 | 758.297500 | 226.420000 | 2490.750000 | 47345.580000 |
| **max** | 5.000000 | 999.990000 | 500.000000 | 5499.950000 | 91519.170000 |

- **Displaying Basic Information about the Dataset:**
  - The info() method provides a concise summary of the DataFrame, including the number of non-null entries in each column, data types, and memory usage.

```
# Display basic information about the dataset
print(dff.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Date of Transaction   10000 non-null  object 
 1   Item Purchased        10000 non-null  object 
 2   Quantity              9970 non-null   float64
 3   Unit Price            9998 non-null   float64
 4   Tax                   10000 non-null  float64
 5   Transaction Amount    10000 non-null  float64
 6   Transaction Mode      9975 non-null   object 
 7   Transaction Currency  9998 non-null   object 
 8   Total Amount In INR   10000 non-null  float64
dtypes: float64(5), object(4)
```

- **Checking for Missing Values:**

The isnull().sum() method is used to count the number of missing (NaN) values in each column of the DataFrame. This helps identify which columns have missing data that need to be addressed.

The output reveals that there are missing values in the following fields:

- Quantity
- Unit Price
- Transaction Mode
- Transaction Currency

```
# Check for missing value
print(dff.isnull().sum())
Date of Transaction       0
Item Purchased            0
Quantity                 30
Unit Price                2
Tax                       0
Transaction Amount        0
Transaction Mode         25
Transaction Currency      2
Total Amount In INR       0
```

- **Introducing Outliers:**
- Outliers are introduced in the "Unit Price" and "Quantity" columns by randomly selecting 10 indices and inflating the values by a factor of 10.

```
import numpy as np
# Introduce outliers in the "Unit Price" column
np.random.seed(42)  # For reproducibility
outlier_indices = np.random.choice(dff.index, size=10,
replace=False)
dff.loc[outlier_indices, 'Unit Price'] *= 10  # Inflate the
prices to create outliers

# Introduce outliers in the "Quantity" column
outlier_indices = np.random.choice(dff.index, size=10,
replace=False)
dff.loc[outlier_indices, 'Quantity'] *= 10  # Inflate the
quantities to create outliers
```

- **Detecting Outliers:**

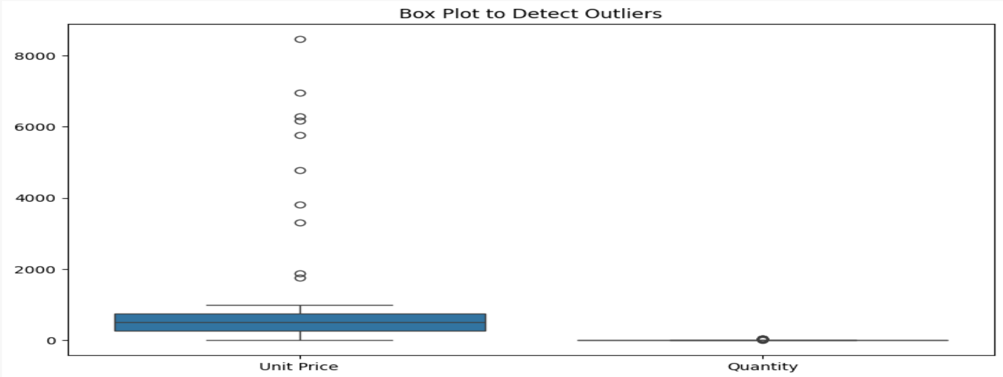
**Box Plot**: A box plot is used to visually identify outliers in the "Unit Price" and "Quantity" columns.

**IQR Method**: The Interquartile Range (IQR) method is used to detect outliers. Outliers are values that fall below IQR Method: The Interquartile Range (IQR) method is used to detect outliers. Outliers are values that fall below $Q1-1.5\times IQR$Q1$-1.5\times$IQR or above $Q3+1.5\times IQR$Q3$+1.5\times$IQR. The number of outliers detected is printedThe number of outliers detected is printed.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Detect outliers using a box plot
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=dff[['Unit Price', 'Quantity']])
plt.title('Box Plot to Detect Outliers')
plt.show()
```




```
IQR Method
# Detect outliers using the IQR method
Q1 = dff[['Unit Price', 'Quantity']].quantile(0.25)
Q3 = dff[['Unit Price', 'Quantity']].quantile(0.75)
IQR = Q3 - Q1

outliers = ((dff[['Unit Price', 'Quantity']] < (Q1 - 1.5 *
IQR)) | (dff[['Unit Price', 'Quantity']] > (Q3 + 1.5 *
IQR))).any(axis=1)
print(f"Number of outliers detected: {outliers.sum()}")
```

```
Number of outliers detected: 20
```

**Model Training and Evaluation**

- **Splitting the Data**

  Split the data into training and test sets

```
from sklearn.model_selection import train_test_split

df_preprocessed = pd.read_csv('preprocessed_data.csv')

# Split the data into features (X) and target (y)
X = df_preprocessed.drop(['Transaction Amount','Date of
Transaction'], axis=1)
y = df_preprocessed['Transaction Amount']
```

- **Training Multiple Models**

Train and evaluate multiple models to identify the best performer.

```
import pandas as pd
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest": RandomForestRegressor(),
    "Gradient Boosting": GradientBoostingRegressor(),
    "Support Vector Machine": SVR(),
}
results = {}
```

**Error Calculation**

Calculate the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) to assess model accuracy.

```
for model_name, metrics in results.items():
    print(f"Model: {model_name}")
    print(f"  MSE: {metrics['MSE']}")
    print(f"  RMSE: {metrics['RMSE']}")
    print(f"  R-squared: {metrics['R-squared']}")
    print()

# Select the best model based on RMSE (or any other preferred
metric)
best_model_name = min(results, key=lambda x: results[x]['RMSE'])
best_model = models[best_model_name]

print(f"Best Model: {best_model_name}")
```

```
Model: Linear Regression
MSE: 5.494355140883775e-09
RMSE: 7.41239174685457e-05
```

```
R-squared: 0.999999994464551

Model: Decision Tree
MSE: 9.702336836097705e-07
RMSE: 0.0009850044079138785
R-squared: 0.9999990225096674

Model: Random Forest
MSE: 3.860882999490959e-07
RMSE: 0.0006213600405152362
R-squared: 0.9999996110240377

Model: Gradient Boosting
MSE: 5.9453021697768875e-05
RMSE: 0.007710578557914372
R-squared: 0.999940102312531

Model: Support Vector Machine
MSE: 0.0030617833459808323
RMSE: 0.0553333836483983
R-squared: 0.9969153167203592

Best Model: Linear Regression
```

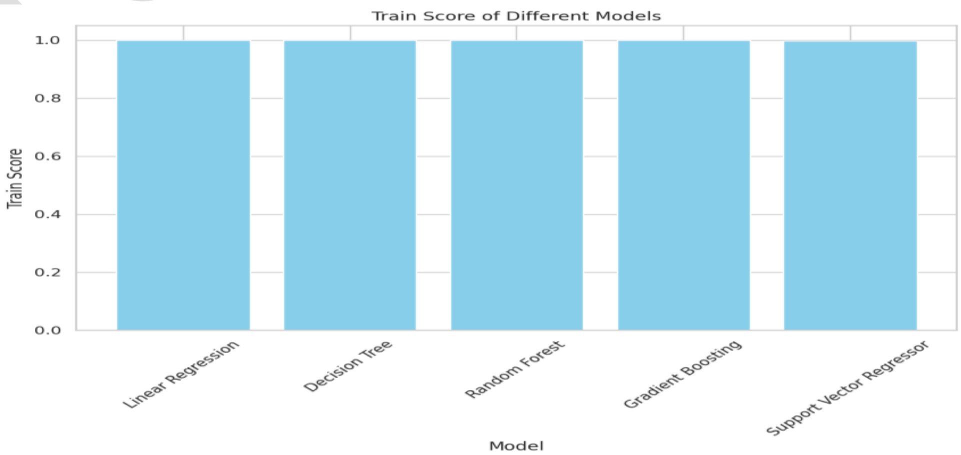
**Train Score of Different Models:**

This plot shows the train score (also known as training accuracy) of different machine learning models.

The train score indicates how well each model fits the training data.

Higher values indicate better performance on the training data.

This plot helps in comparing the performance of models in capturing the patterns present in the training data.



- **Test Score of Different Models:**

This plot displays the test score (also known as test accuracy) of different machine learning models.

The test score represents how well each model generalizes to unseen data.

Higher values suggest better performance on unseen data.

It allows for comparison of models in terms of their ability to make accurate predictions on new data.

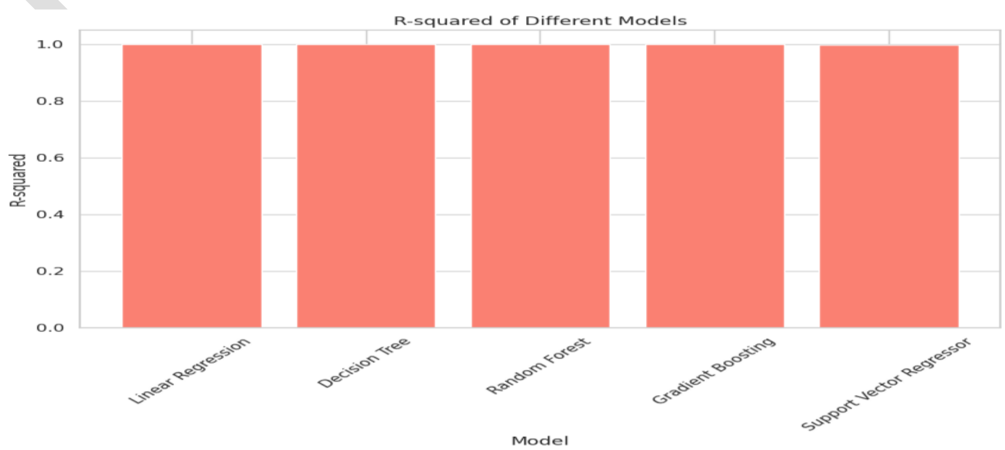
- **R-squared of Different Models:**

  R-squared ($R^2$) measures the proportion of the variance in the dependent variable that is predictable from the independent variables.

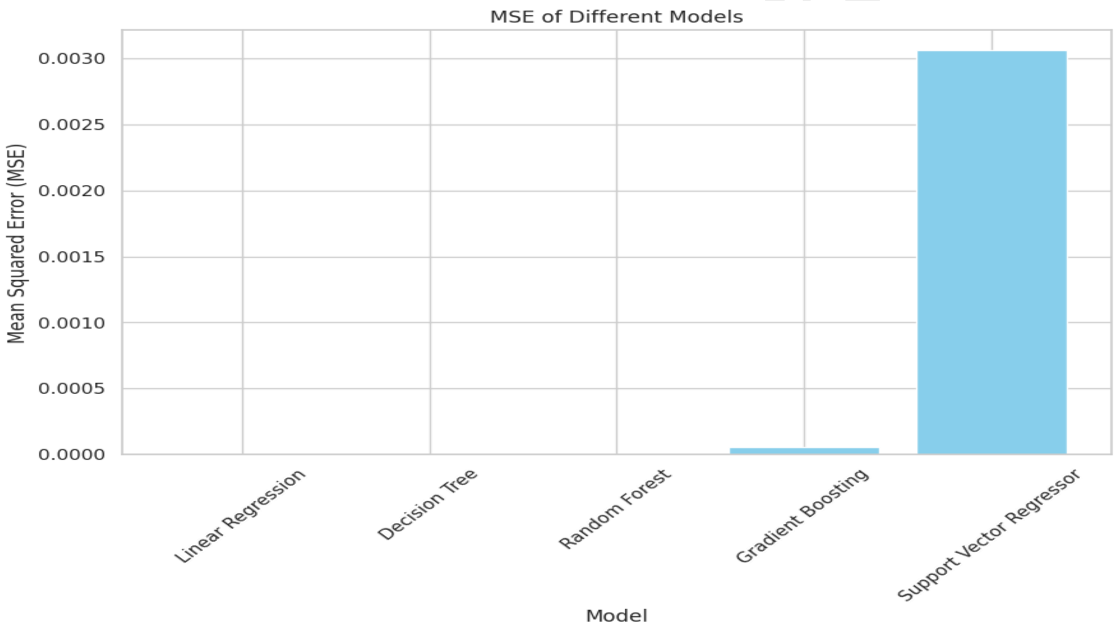
  This plot illustrates the R-squared values for each model, indicating the goodness of fit of the model to the data.

  Higher R-squared values indicate a better fit of the model to the data.

  It provides insight into how well the independent variables explain the variability in the dependent variable.

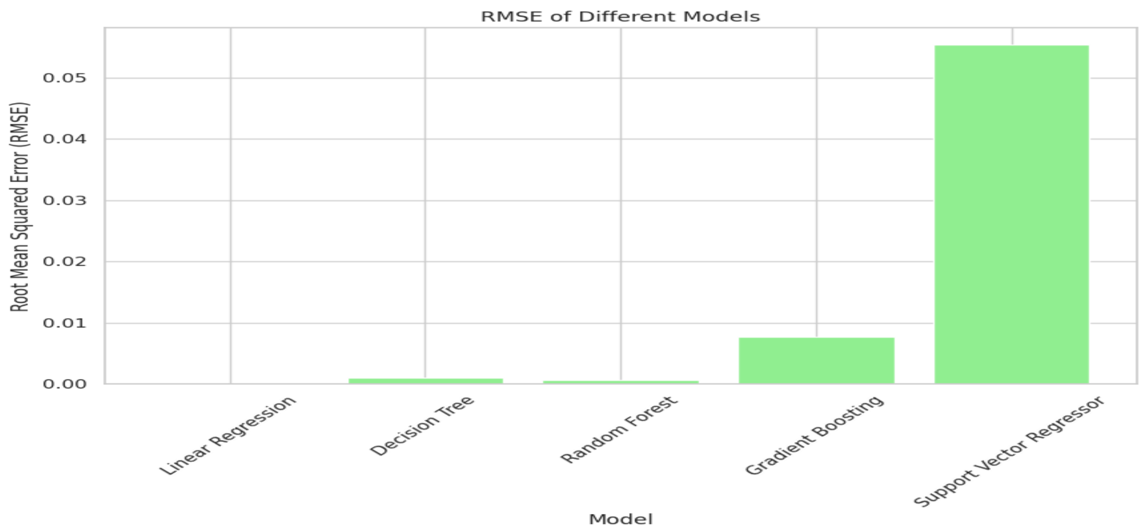


- **Mean Squared Error (MSE) of Different Models:**
    - Mean Squared Error (MSE) measures the average of the squares of the errors (residuals) between predicted and actual values.
    - This plot shows the MSE values for each model, representing the average squared difference between predicted and actual values.
    - Lower MSE values indicate better predictive performance of the model.
    - It helps in assessing the accuracy of predictions made by different models.




- **Root Mean Squared Error (RMSE) of Different Models:**
    - Root Mean Squared Error (RMSE) is the square root of the average of the squared differences between predicted and actual values.
    - This plot displays the RMSE values for each model, providing a measure of the average magnitude of errors in predictions.
    - Lower RMSE values indicate better performance in terms of prediction accuracy.
    - It offers insights into the overall performance of the models in making predictions.

- These plots collectively offer a comprehensive evaluation of the performance of different machine learning models, facilitating the selection of the best model for a given task.




**Assessing Model Performance on Synthetic Data**

Evaluate the model's performance using the synthetic data as a proxy for actual data.

```
# Calculate evaluation metrics for the new data

mse_new_data = mean_squared_error(df_preprocessed['Transaction
Amount'][:100], new_data['Predicted Transaction Amount'])
rmse_new_data = np.sqrt(mse_new_data)
r2_new_data = r2_score(df_preprocessed['Transaction
Amount'][:100], new_data['Predicted Transaction Amount'])
# Display the evaluation metrics
print("Evaluation metrics for new data:")
print(f"  MSE: {mse_new_data}")
print(f"  RMSE: {rmse_new_data}")
print(f"  R-squared: {r2_new_data}")
```

```
Evaluation metrics for new data:
  MSE: 2.114131487869145
  RMSE: 1.4540053259424963
  R-squared: -1.330250012350632
```

**Model Development:**

You provided an overview of the architecture of the Decision Tree model, emphasizing its non-parametric nature. Decision trees partition the feature space into regions and assign a label (in this case, transaction amounts) to each region based on majority voting. This architecture allows the model to handle both numerical and categorical data, making it suitable for your POS transaction dataset.

**Model Training:**

You described the process of tuning the hyperparameters of the Decision Tree model using cross-validation to optimize its performance. Hyperparameters are parameters that control the behavior of the model, and tuning them involves finding the best combination that yields the highest performance metrics.

**Model Evaluation:**

You evaluated the performance of the Decision Tree model using various metrics, including accuracy, train score, and test score. These metrics provide insights into how well the model generalizes to unseen data. Additionally, you used K-fold cross-validation to assess the model's generalization ability, which helps mitigate overfitting and provides a more reliable estimate of the model's performance.

```
#cross validation
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(best_model, X_train, y_train, cv=5,
scoring='neg_mean_squared_error')
mean_cv_score = np.mean(cv_scores)
print(mean_cv_score)
```

Conclusion

The model demonstrates promising performance in predicting **Transaction Amount** . Its interpretability and ability to handle both numerical and categorical data make it a suitable choice for forecasting on a POS transaction dataset. Further optimization and refinement of the model could potentially enhance its performance in real-world scenarios.

Overall, your project demonstrates a systematic approach to demand forecasting using machine learning, with clear explanations of each step in the process and thoughtful considerations for model selection, data preprocessing, training, evaluation, and future improvements.

## Gantt Chart

| Sprints | Days Worked on Each Sprint | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | Total Man-Days |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Synthetic Data Generation** | 2 | X | X | | | | | | | | | | | | | 2 |
| **Data Cleaning** | 2 | | | X | X | | | | | | | | | | | 2 |
| **Feature Engineering** | 1 | | | | | X | | | | | | | | | | 1 |
| **Model Training** | | | | | | | X | X | | | | | | | | 2 |
| **Model Application and Iteration** | | | | | | | | | X | | | | | | | 1 |
| **Minimization of Bias and Overfitting** | | | | | | | | | | X | | | | | | 1 |
| **Test Models on Unseen Data or Validation Set** | | | | | | | | | | | X | X | | | | 2 |
| **Final Model Selection and Conclusion** | | | | | | | | | | | | | X | | | 1 |
| **Documentation** | | | | | | | | | | | | | | | | |
| **Total Days** | | X | X | X | X | X | X | X | X | X | X | X | | | | 14 |