Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

FINAL REPORT

MACHINE LEARNING

Instructor: **Mr. LÊ ANH CƯỜNG**

Student: **PHAN THÀNH ĐẠT - 521H0218**

NGUYỄN LÊ PHƯỚC TIẾN: 521H0514

NGUYỄN MẠNH CƯỜNG - 521H0496

Class :**21H50302**

Year **: 2023-2024**

HO CHI MINH CITY, 2023

Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

FINAL REPORT

MACHINE LEARNING

Instructor: **Mr. LÊ ANH CƯỜNG**

Student: **PHAN THÀNH ĐẠT - 521H0218**

NGUYỄN LÊ PHƯỚC TIẾN: 521H0514

NGUYỄN MẠNH CƯỜNG - 521H0496

Class :**21H50302**

Year **: 2023-2024**

HO CHI MINH CITY, 2023

ACKNOWLEDGEMENT

I would like to express my deep gratitude to Mr. Le Anh Cuong for his valuable support in the past project. The teacher's knowledge and guidance have played an important role in our work process. What he shared and guided us helped us overcome challenges, expand our knowledge and approach the project more confidently. His dedication and help not only helped us understand the project better, but also encouraged and inspired us.

Ho Chi Minh city, 22th December, 2023

Author

(Sign and write full name)

Phan Thành Đạt

Nguyễn Lê Phước Tiến

Nguyễn Mạnh Cường

CONFIRMATION AND ASSESSMENT SECTION

Instructor confirmation section

Ho Chi Minh 22 December, 2023

(Sign and write full name)

Evaluation section for grading instructor

Ho Chi Minh, 22 December 2023

(Sign and write full name)

THE PROJECT IS COMPLETED AT TON DUC THANG UNIVERSITY

Our team would like to assure that this is our own research project and is under the scientific guidance of Lê Anh Cường Teacher. The research content and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments, and evaluation were collected by the author from different sources and clearly stated in the reference section.

In addition, the report also uses a number of comments, assessments as well as data from other authors and other organizations, all with citations and source notes.

If any fraud is detected, our team will take full responsibility for the content of our IT Project Report 2. Ton Duc Thang University is not involved in copyright violations caused by us during the implementation process (if any).

Ho Chi Minh city, 22th December, 2023

Author

(Sign and write full name)

Phan Thành Đạt

Nguyễn Lê Phước Tiến

Nguyễn Mạnh Cường

SUMMARY

Use machine learning to predict with diverse data, including both numerical and categorical features. Analyze the data, apply underlying models and Ensemble Learning, then use Neural Network. Avoid Overfitting and improve accuracy by analyzing errors and applying solutions.

INDEX

CHAPTER 1: DATASET INFORMATION

Dataset used: Sample Superstore Dataset

1.1 Introduction

This is an example of a superstore dataset, which you can use to run a simulation in which you analyze a lot of data to find patterns that will help the business maximize revenues and minimize losses.

1.2 Attribute

Attribute	Description
Ship Mode	Mode of shipping used for shipment delivery
Segment	(Categorical) Customer segment product was shipped to
Country	Country in which the shipment was delivered
City	City in which shipment was delivered
State	State in which the shipment was delivered
Postal Code	Postal code the shipment was delivered to
Region	Country region
Category	The category product belongs to
Sub-Category	Sub-category of the product
Sales	Sale made in USD
Quantity	Product quantity
Discount	Discount given on the product
Profit	Profit/loss made on the sale

1.3 Library

Library	Purpose of using
Pandas	Use to read and explore data, perform descriptive statistics, and draw simple graphs
Seaborn, Matplotlib	Draw a diagram to better understand the relationship between the characteristics and the problem goal. Use to understand correlation and distribution of data.
NumPy	Handling arithmetic data.
Scikit-learn (sklearn)	Apply basic models and Ensemble Learning models to predict targets. Use techniques such as Cross-Validation,... to avoid overfitting.
XGBoost	Build and tune boosting models to improve prediction accuracy and performance. Evaluate the importance of the features.
TensorFlow, Keras	Use neural networks to predict problem goals. Apply overfitting avoidance techniques such as Dropout and Early Stopping during model training.

These libraries provide powerful tools to approach and solve prediction problems on Sample Superstore Dataset data. The flexible combination of them will help us better understand the data, build and optimize machine learning models and neural networks to predict the goal of the problem.

CHAPTER 2: STATISTICAL ANALYSIS OF DATA

2.1 Statistical analysis on data.

- Read data form datasets: Display the first 5 rows of data, information about data structure, data type, calculate basic descriptive statistics.

```
        Ship Mode    Segment        Country            City       State  \
0    Second Class   Consumer  United States       Henderson    Kentucky
1    Second Class   Consumer  United States       Henderson    Kentucky
2    Second Class  Corporate  United States     Los Angeles  California
3  Standard Class   Consumer  United States  Fort Lauderdale    Florida
4  Standard Class   Consumer  United States  Fort Lauderdale    Florida

   Postal Code Region         Category Sub-Category     Sales  Quantity  \
0        42420  South        Furniture    Bookcases  261.9600         2
1        42420  South        Furniture       Chairs  731.9400         3
2        90036   West  Office Supplies       Labels   14.6200         2
3        33311  South        Furniture       Tables  957.5775         5
4        33311  South  Office Supplies      Storage   22.3680         2

   Discount    Profit
0      0.00   41.9136
1      0.00  219.5820
2      0.00    6.8714
3      0.45 -383.0310
4      0.20    2.5164
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Ship Mode     9994 non-null   object
 1   Segment       9994 non-null   object
 2   Country       9994 non-null   object
 3   City          9994 non-null   object
 4   State         9994 non-null   object
 5   Postal Code   9994 non-null   int64
 6   Region        9994 non-null   object
 7   Category      9994 non-null   object
 8   Sub-Category  9994 non-null   object
 9   Sales         9994 non-null   float64
 10  Quantity      9994 non-null   int64
 11  Discount      9994 non-null   float64
 12  Profit        9994 non-null   float64
dtypes: float64(3), int64(2), object(8)
memory usage: 1015.1+ KB
None
        Postal Code         Sales     Quantity     Discount       Profit
count   9994.000000   9994.000000  9994.000000  9994.000000  9994.000000
mean   55190.379428    229.858001     3.789574     0.156203    28.656896
std    32063.693350    623.245101     2.225110     0.206452   234.260108
min     1040.000000      0.444000     1.000000     0.000000 -6599.978000
25%    23223.000000     17.280000     2.000000     0.000000     1.728750
50%    56430.500000     54.490000     3.000000     0.200000     8.666500
75%    90008.000000    209.940000     5.000000     0.200000    29.364000
max    99301.000000  22638.480000    14.000000     0.800000  8399.976000
```

2.2 Draw graphs to understand the problem and understand the data.

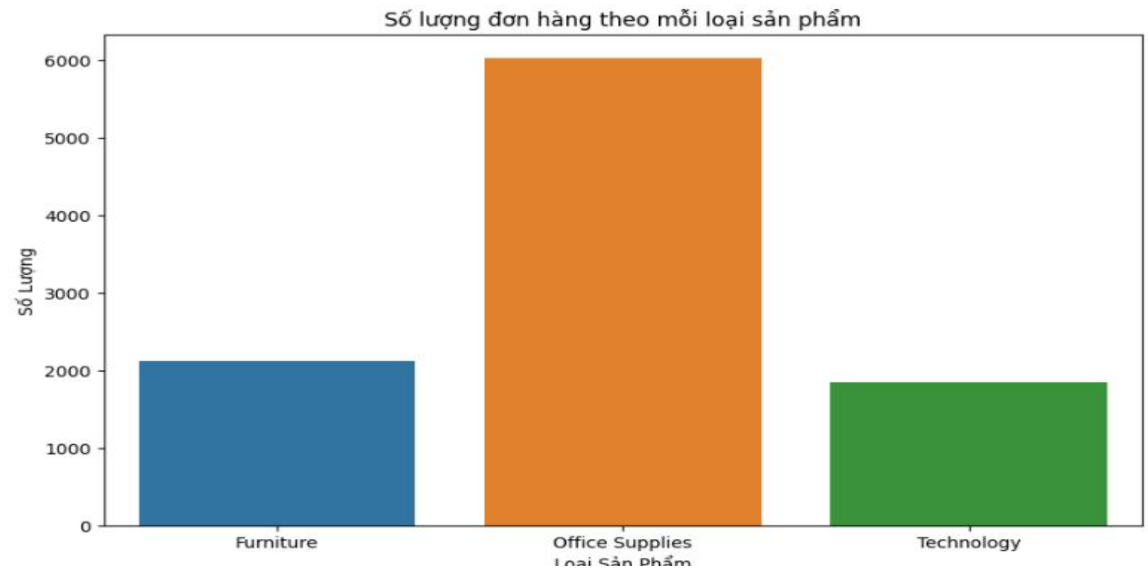
In this section we perform data visualization, information discovery and lay the foundation for data preprocessing, data analysis and building a machine learning model.

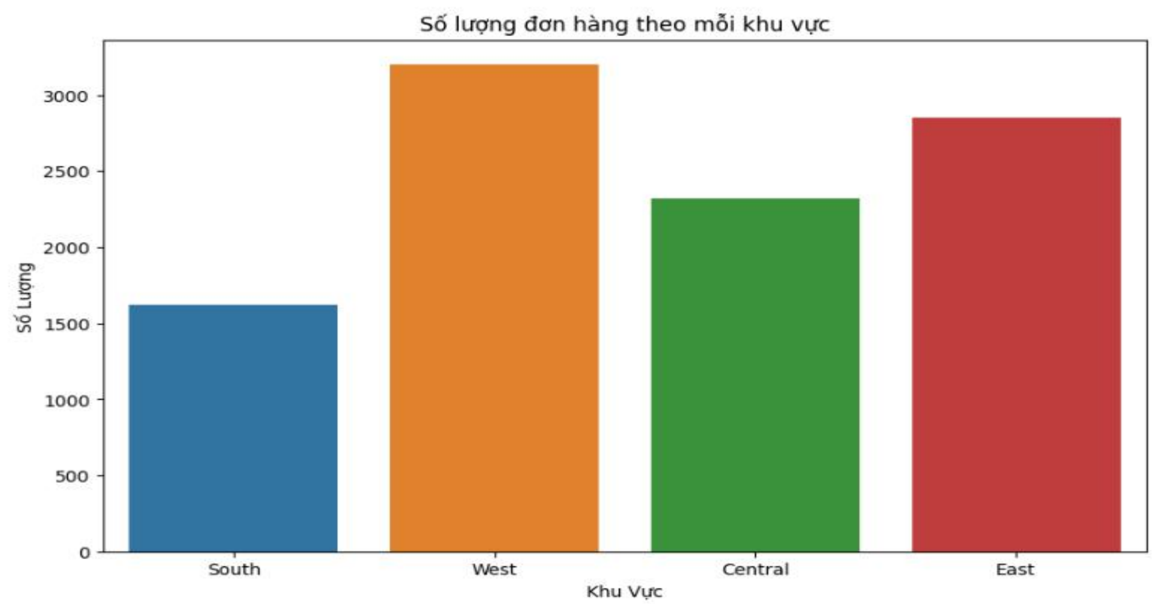
- By using the Seaborn library to create an order count chart and the matplotlib library to display the chart:

```python
#Hiển thị số lượng đơn hàng cho mỗi loại sản phẩm.
plt.figure(figsize=(10, 6))
sns.countplot(x='Category', data=df)
plt.title('Số lượng đơn hàng theo mỗi loại sản phẩm')
plt.xlabel('Loại Sản Phẩm')
plt.ylabel('Số Lượng')
plt.show()
```

```python
#Hiển thị số lượng đơn hàng cho mỗi khu vực.
plt.figure(figsize=(10, 6))
sns.countplot(x='Region', data=df)
plt.title('Số lượng đơn hàng theo mỗi khu vực')
plt.xlabel('Khu Vực')
plt.ylabel('Số Lượng')
plt.show()
```

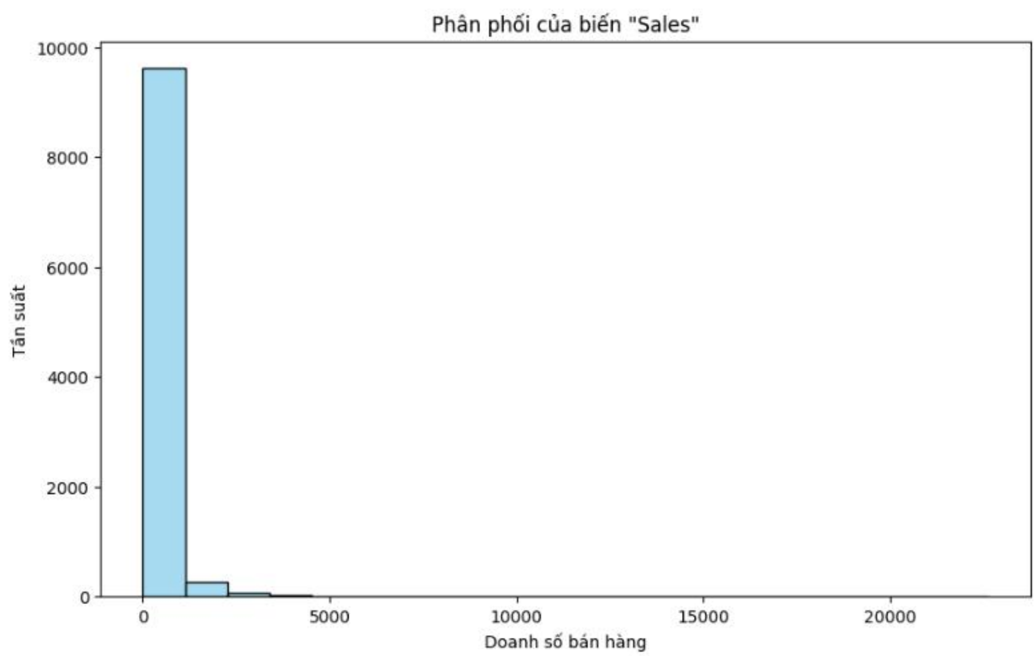
- Below is the output:





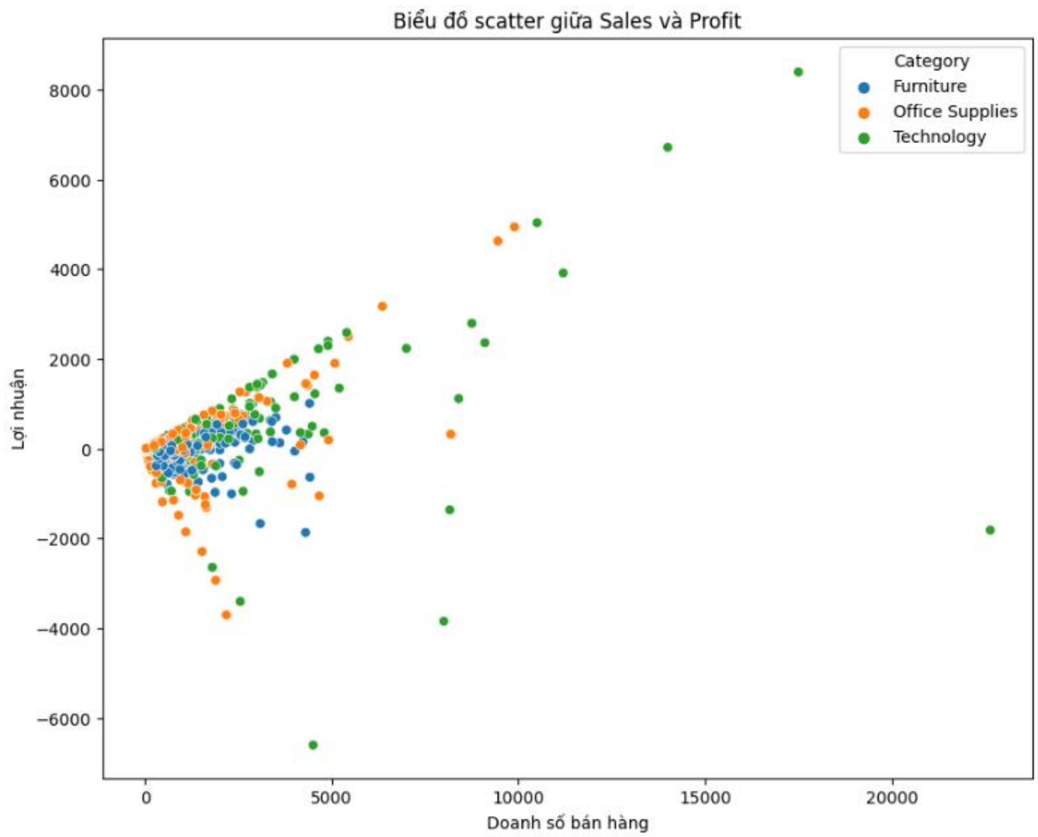
This also uses the Seaborn library but this time creates a histogram showing the distribution of the variable 'Sales'. The bins=20 argument determines the number of bins in the histogram, i.e. the number of value intervals divided to calculate frequency. The kde=False argument specifies that the Kernel Density Estimation curve should not be displayed. The color='skyblue' argument specifies the color of the histogram:

```
#Hiển thị phân phối của biến 'Sales'.
plt.figure(figsize=(10, 6))
sns.histplot(df['Sales'], bins=20, kde=False, color='skyblue')
plt.title('Phân phối của biến "Sales"')
plt.xlabel('Doanh số bán hàng')
plt.ylabel('Tần suất')
plt.show()
```



- This time using the Seaborn library to create a scatter chart showing the relationship between sales and profits:

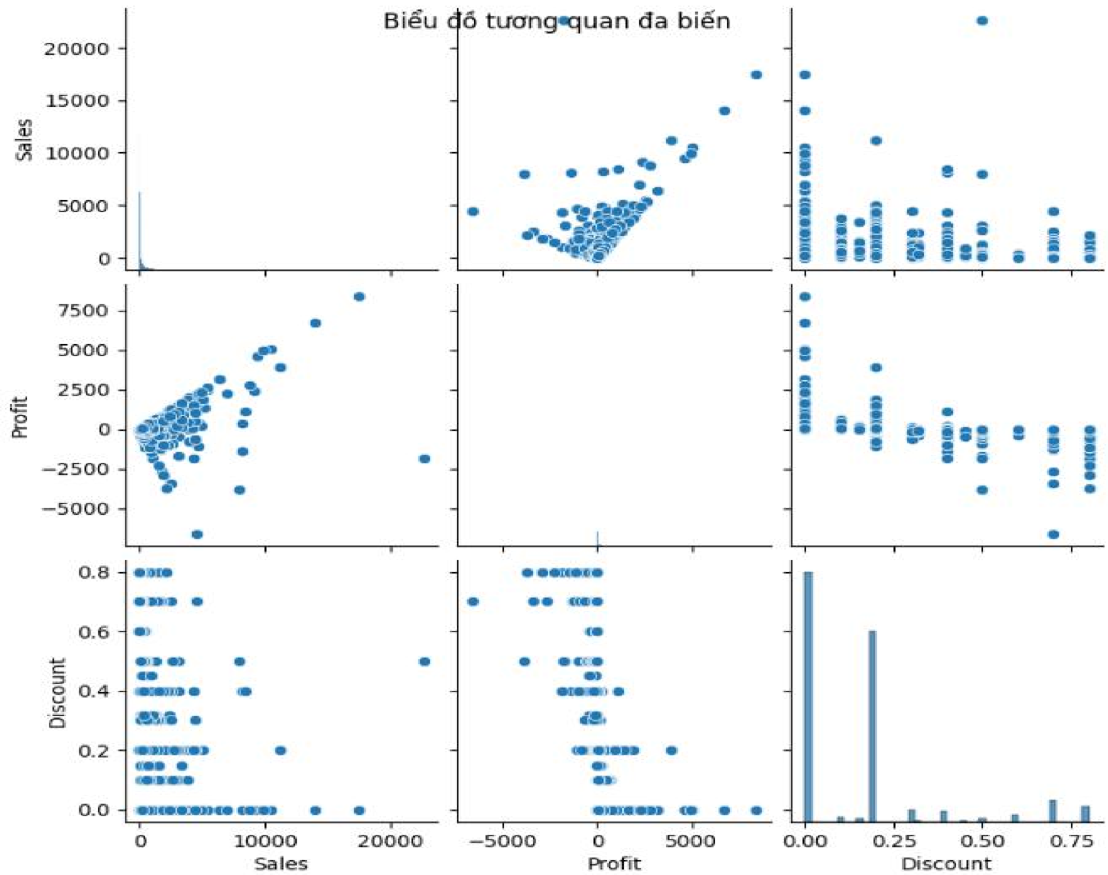
```
#Hiển thị mối quan hệ giữa doanh số bán hàng và lợi nhuận.
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Sales', y='Profit', data=df, hue='Category')
plt.title('Biểu đồ scatter giữa Sales và Profit')
plt.xlabel('Doanh số bán hàng')
plt.ylabel('Lợi nhuận')
plt.show()
```



- Shows the relationship between 'Sales', 'Profit' and 'Discount'. This chart helps us observe and analyze the degree of correlation between variables and also shows their distribution. This can help us gain a deeper understanding of the relationship between sales, profits and discounts and can provide observations and insights in data analysis and model building. machine.

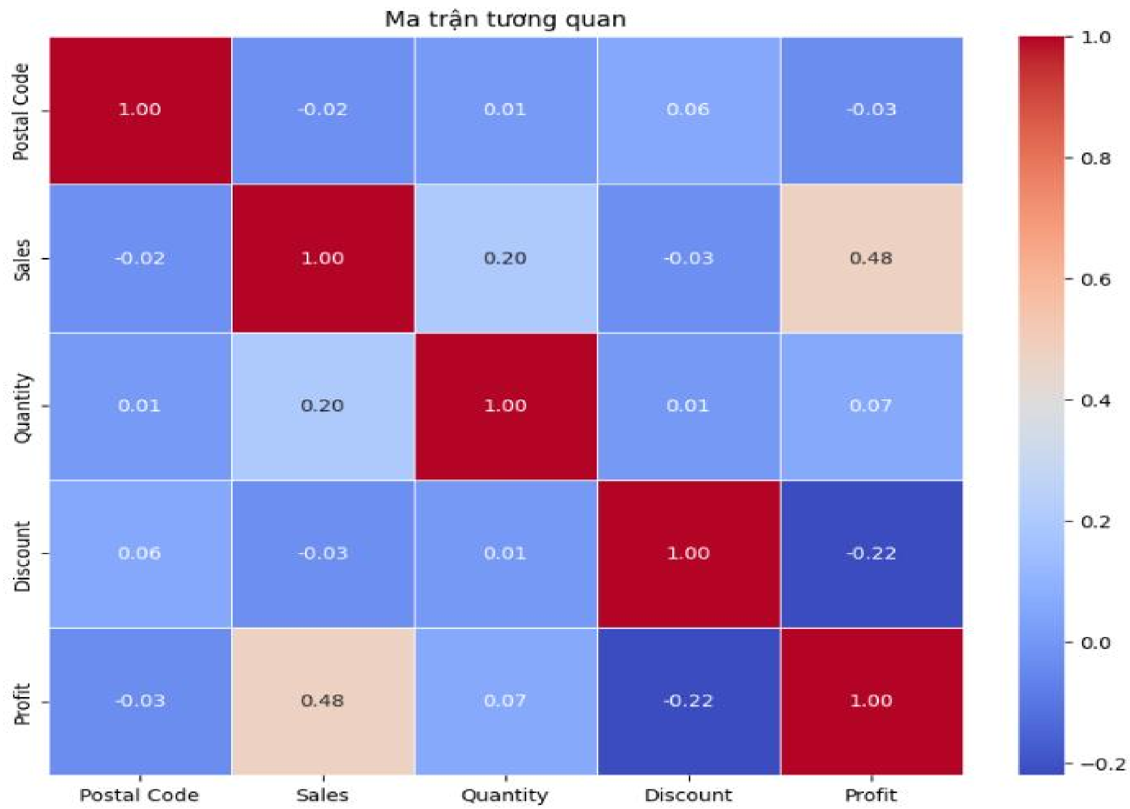
```
#Hiển thị mối quan hệ giữa 'Sales', 'Profit', và 'Discount'.
plt.figure(figsize=(12, 8))
sns.pairplot(df[['Sales', 'Profit', 'Discount']])
plt.suptitle('Biểu đồ tương quan đa biến')
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```



calculate the correlation matrix and display the correlation matrix in a heat map plot to evaluate the degree of correlation between variables in the DataFrame. By using the Seaborn library to create a heatmap plot from a correlation matrix and the matplotlib library to display it.

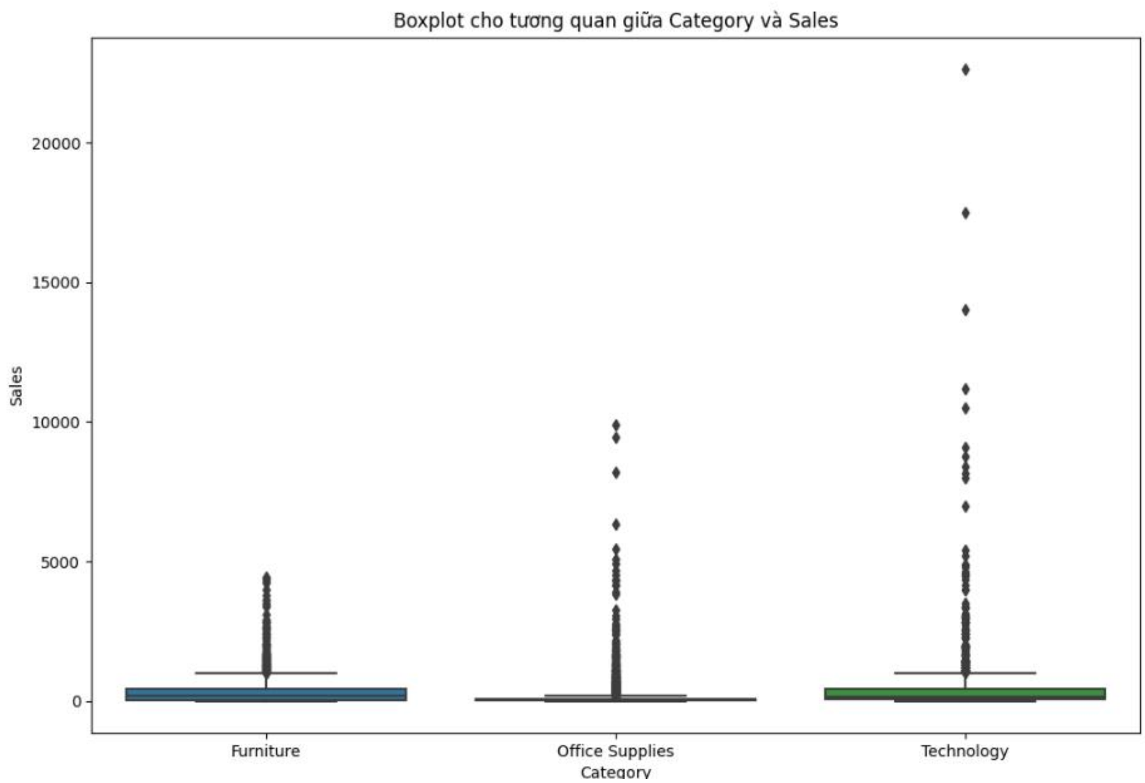
```
#Tính toán ma trận tương quan để đánh giá mức độ tương quan giữa các biến số.
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Ma trận tương quan')
plt.show()
```

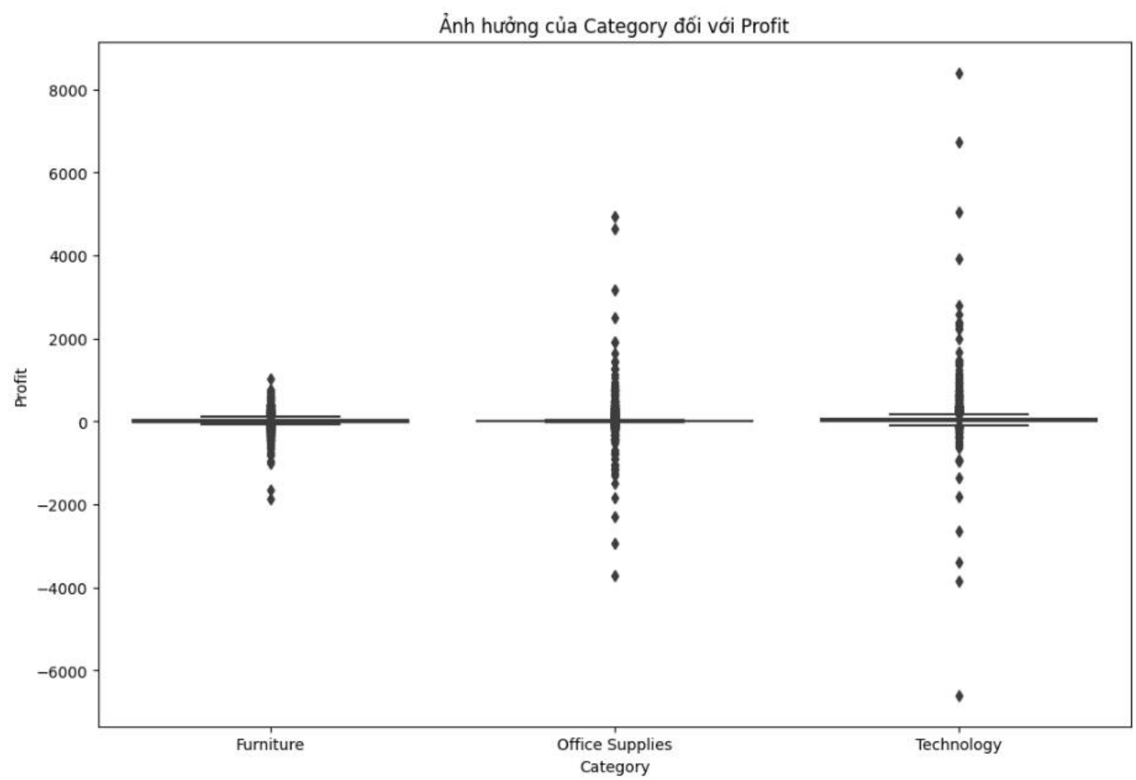


Use boxplot to show correlation between numeric and categorical variables in a DataFrame. Boxplot charts help observe the distribution and level of correlation between variables according to each value of the categorical variable. This helps us evaluate the influence of categorical variables on numerical variables and analyze differences in subgroups of categorical variables.

```
#Sử dụng boxplot để hiển thị tương quan giữa các biến số và phân loại.
plt.figure(figsize=(12, 8))
sns.boxplot(x='Category', y='Sales', data=df)
plt.title('Boxplot cho tương quan giữa Category và Sales')
plt.show()
```

```
# Xem xét mức độ ảnh hưởng của loại sản phẩm ('Category') đối với lợi nhuận ('Profit').
plt.figure(figsize=(12, 8))
sns.boxplot(x='Category', y='Profit', data=df)
plt.title('Ảnh hưởng của Category đối với Profit')
plt.show()
```

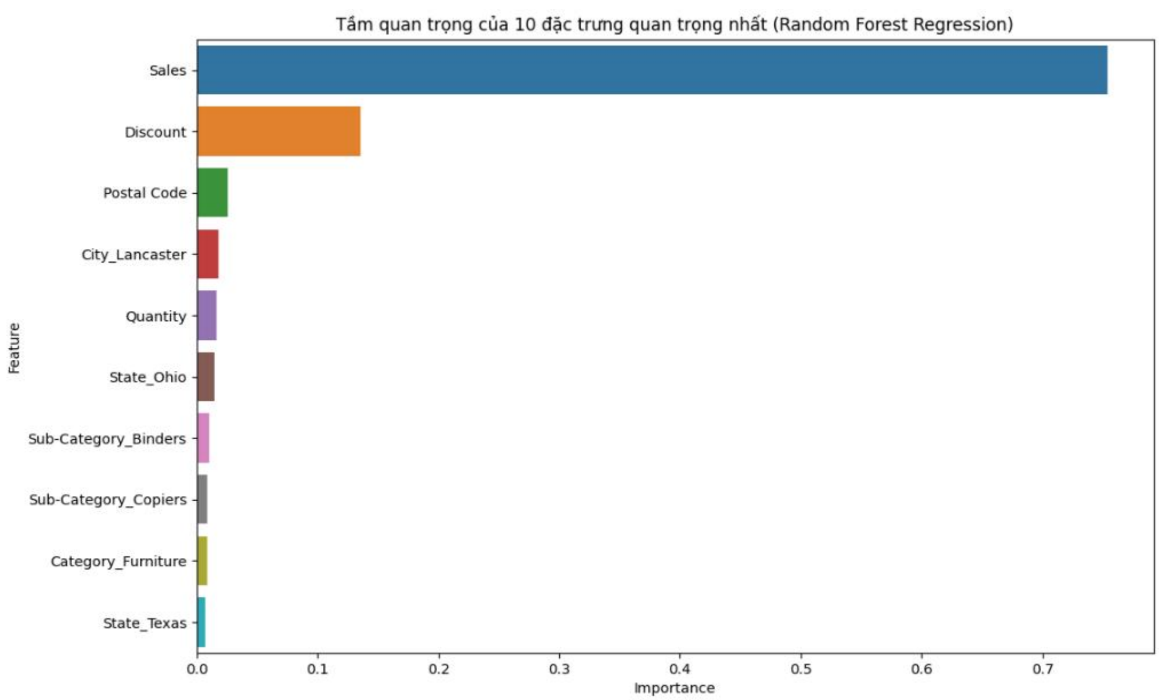

Boxplot cho tương quan giữa Category và Sales
Sales
20000
15000
10000
5000
0
Furniture
Office Supplies
Technology
Category


Ảnh hưởng của Category đối với Profit
Profit
8000
6000
4000
2000
0
−2000
−4000
−6000
Furniture
Office Supplies
Technology
Category

2.3 Find out the characteristics and evaluate the role of the characteristics for the problem goal.

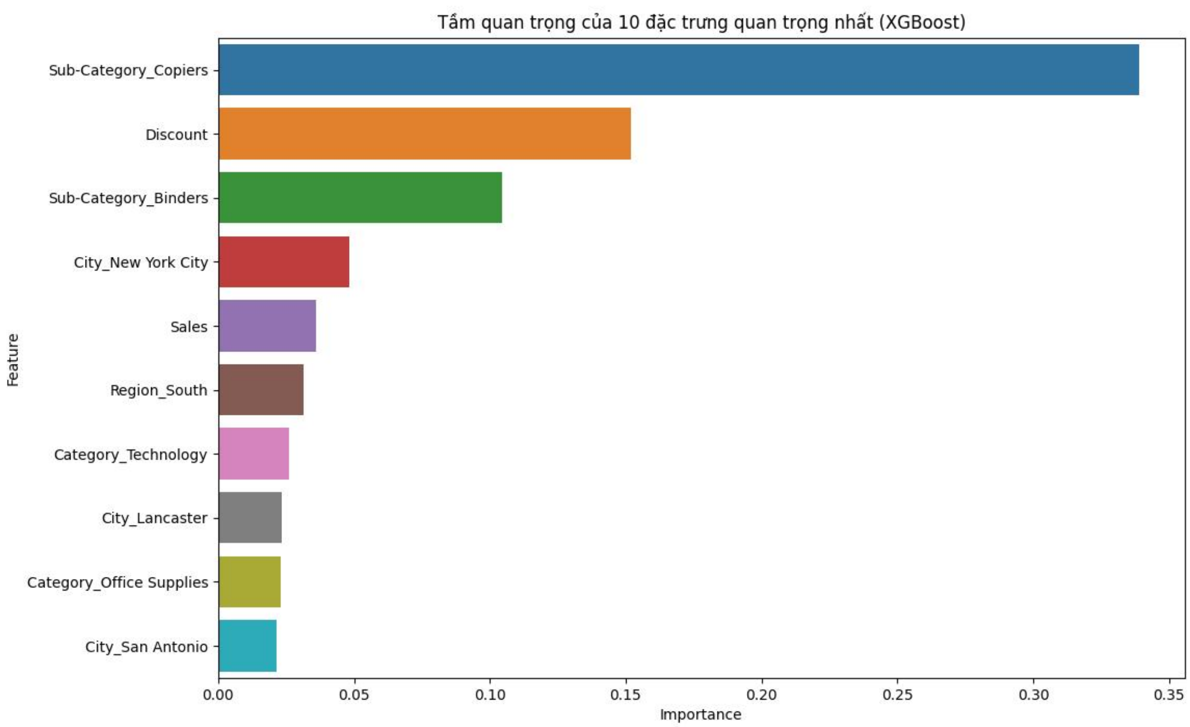
2.3.1 Random Forest Regression:

- After building the initial Random Forest Regression model, some important features are selected from this model. The purpose of important feature selection is to reduce the number of features to increase model performance and minimize the influence of features that are not important or have a small impact on profit prediction. Then, a new Random Forest Regression model is built using only the selected important features.
- Evaluate the accuracy of the model on the test set, using the mean squared error (MSE) index. This accuracy indicates the degree of deviation between the actual profit value and the profit value predicted from the model on the test set. The goal is to have a model with high accuracy, i.e. low MSE value, that can predict profits with good accuracy from selected important features.



2.3.2 XGBoost:

- Using XGBoost, this code aims to identify the most important features in the data, i.e. those features that have a large influence on the target variable (in this case the variable "Profit"). Evaluating the importance of features helps us better understand the role and contribution of each feature to the prediction model.
- After the XGBoost model is trained on the training data and evaluated on the test data, we use XGBoost's feature_importances_ method to calculate the importance of each feature. Importance is measured by looking at how much the model improves when a feature is used in the decision tree construction process.
- The end result is a bar plot that displays the importance of the 10 most important features. This chart helps us see the influence of each feature on profit prediction and make decisions about the use of features in the prediction model.



CHAPTER 3: APPLYING MACHINE LEARNING MODELS

3.1 Method

To apply basic machine learning models to solve the profit prediction problem, you can do the following steps:

- Prepare Data:

- Divide the data into training set and test set. Prepare features and target variables from the data. Choose Model:

- Choose basic machine learning models like Linear Regression, Decision Tree, Random Forest, Gradient Boosting, and Support Vector Regression. Model Training:

- Train each model on the training set. Model Rating:

- Evaluate the performance of each model on the test set using metrics such as Mean Squared Error (MSE) or Root Mean Squared Error (RMSE). Combining Ensemble Model:

- Combine models using Ensemble Learning, such as using Random Forest or Gradient Boosting models. Prediction and Evaluation:

- Predicting profit on test set using hybrid model. Compare Results:

- Compare the results of the Ensemble model with the basic models to see the improvement.

3.2 Code example

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

# Chuẩn bị dữ liệu (X, y là features và target)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Chọn và huấn luyện mô hình cơ bản
models = {
```

```
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(),
    'Gradient Boosting': GradientBoostingRegressor(),
    'Support Vector Regression': SVR()
}
# Đánh giá hiệu suất và chọn mô hình tốt nhất
best_model = None
best_mse = float('inf')  # Đặt giá trị ban đầu là vô cùng lớn
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f'{name} Mean Squared Error: {mse}')
    # Lưu mô hình tốt nhất nếu có hiệu suất tốt hơn
    if mse < best_mse:
        best_mse = mse
        best_model = model
# Kết hợp các mô hình bằng cách sử dụng mô hình Random Forest
ensemble_model = RandomForestRegressor()
ensemble_model.fit(X_train, y_train)
y_pred_ensemble = ensemble_model.predict(X_test)
mse_ensemble = mean_squared_error(y_test, y_pred_ensemble)
print(f'Ensemble Model Mean Squared Error: {mse_ensemble}')
print(f'Best Model: {type(best_model).__name__} with MSE: {best_mse}')
```

3.3 Output

```
Linear Regression Mean Squared Error: 78987.01408508827
Decision Tree Mean Squared Error: 79169.03158911642
Random Forest Mean Squared Error: 53130.77734732711
Gradient Boosting Mean Squared Error: 48320.740843018444
Support Vector Regression Mean Squared Error: 48517.363638872186
Ensemble Model Mean Squared Error: 52269.418184081245
Best Model: GradientBoostingRegressor with MSE: 48320.740843018444
```

CHAPTER 4: USE FEED FORWARD NEURAL NETWORK AND RECURRENT NEURAL NETWORK.

4.1 Simple example with TensorFlow and Keras for Feed Forward Neural Network

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
# Giả sử df là DataFrame chứa dữ liệu của bạn
# Chuyển đổi biến hạng mục thành dạng số bằng phương pháp One-Hot Encoding
df_encoded = pd.get_dummies(df, columns=['Ship Mode', 'Segment', 'Country', 'City',
'State', 'Region', 'Category', 'Sub-Category'])
# Chia dữ liệu thành X (đặc trưng) và y (biến mục tiêu)
X = df_encoded.drop('Profit', axis=1)
y = df_encoded['Profit']
# Chuẩn hóa dữ liệu đặc trưng
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Chia dữ liệu thành tập huấn luyện và tập kiểm tra
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
# Kiểm tra giá trị NaN trong dữ liệu đầu vào
print("Before handling NaN values:")
print(np.isnan(X_train).any())
# Xử lý giá trị NaN (ví dụ: sử dụng fillna với giá trị trung bình)
X_train = pd.DataFrame(X_train).fillna(pd.DataFrame(X_train).mean()).to_numpy()
# Kiểm tra lại sau khi xử lý
print("\nAfter handling NaN values:")
print(np.isnan(X_train).any())
# Sử dụng Feed Forward Neural Network (FFNN)
model_ffnn = Sequential()
model_ffnn.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
```

```
model_ffnn.add(Dense(64, activation='relu'))
model_ffnn.add(Dense(1, activation='linear'))
model_ffnn.compile(optimizer='adam', loss='mean_squared_error')
model_ffnn.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
# Sử dụng mô hình Random Forest để so sánh
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
print(f'Random Forest Mean Squared Error on Test Set: {mse_rf}')
# Sử dụng Recurrent Neural Network (RNN) với LSTM
model_rnn = Sequential()
model_rnn.add(LSTM(50, input_shape=(X_train.shape[1], 1), activation='relu'))
model_rnn.add(Dense(1, activation='linear'))
model_rnn.compile(optimizer='adam', loss='mean_squared_error')
model_rnn.fit(X_train.reshape(X_train.shape[0], X_train.shape[1], 1), y_train,
epochs=10, batch_size=32, validation_split=0.2)
# Đánh giá độ chính xác trên tập kiểm tra
X_test_rnn = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
y_pred_rnn = model_rnn.predict(X_test_rnn)
mse_rnn = mean_squared_error(y_test, y_pred_rnn)
print(f'RNN Mean Squared Error on Test Set: {mse_rnn}')
```

=> The purpose of the above code is to train and compare the performance between a neural network (FFNN) and a Random Forest model on data divided into features (X) and target variables (y). The model is trained to predict the 'Profit' value based on other features in the data set. Then, the accuracy of both models is evaluated by calculating the mean squared error between the predicted value and the actual value on the test set. The aim is to see which model performs better in predicting 'Profit' on the given data.

CHAPTER 5: OVERFITTING

5.1 Apply Overfitting avoidance techniques on sentence models (2)

5.1.1 Regularization in Random Forest

While the Random Forest model is less likely to overfit than some other models, we can still experiment with hyperparameter tuning to adjust tree depth, such as max_deep:

Code:

```
from sklearn.ensemble import RandomForestRegressor

rf_model = RandomForestRegressor(n_estimators=100, max_depth=10,
random_state=42)
rf_model.fit(X_train, y_train)
```

By limiting the complexity of the tree, as in this case 10, max_deep can help reduce the possibility of overfitting and improve the generalization ability of the model. We can achieve a balance between model complexity and performance on data that has never been seen before by fine-tuning these hyperparameters.

```
RandomForestRegressor
RandomForestRegressor(max_depth=10, random_state=42)
```

5.1.2 Early Stopping

Although Random Forest does not directly enable Early Stopping, we can still manage the training process with the help of this technique. To evaluate the training and avoid overfitting, we divided the data into a test set and a training set using this technique

Code:

```
# Divide the data into training set and test set
X_train, X_val, y_train, y_val = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Use Random Forest model to calculate feature importance
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
training_errors, validation_errors = [], []
for i in range(1, 101):
   rf_model.fit(X_train, y_train)
   y_train_pred = rf_model.predict(X_train)
   y_val_pred = rf_model.predict(X_val)
   training_errors.append(mean_squared_error(y_train, y_train_pred))
   validation_errors.append(mean_squared_error(y_val, y_val_pred))
   if i > 2 and validation_errors[-1] > validation_errors[-2] > validation_errors[-3]:
      break
```

This code uses a training set to train the model and a test set to evaluate performance. We verify the error on both the training set and test set each training session. To avoid overfitting, we pause the training process if, after three consecutive iterations, the error on the test set increases. This improves the generalization ability of the model and helps tune the training process.

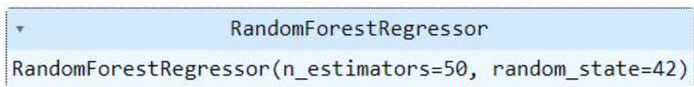
5.1.3 Reduce the number of trees

Reducing the number of trees (n_estimators) in the RandomForestRegressor model can be a useful strategy to reduce the risk of overfitting:

Code:

```
rf_model = RandomForestRegressor(n_estimators=50, random_state=42)
rf_model.fit(X_train, y_train)
```

n_estimators is now set to 50. As a result there are fewer trees in the forest, which reduces model complexity and reduces the possibility of overfitting. We can fine-tune the model to excellent accuracy on test data while maintaining generality by tuning this hyperparameter.



5.2 Apply Overfitting avoidance techniques on sentence models (3)

5.2.1 Regularization in FFNN

Add Dropout layers to randomly drop some neurons during training

Code:

```
from tensorflow.keras.layers import Dropout
model_ffnn = Sequential()
model_ffnn.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model_ffnn.add(Dropout(0.5))  # Thêm Dropout layer
model_ffnn.add(Dense(64, activation='relu'))
model_ffnn.add(Dropout(0.5))  # Thêm Dropout layer
model_ffnn.add(Dense(1, activation='linear'))
model_ffnn.compile(optimizer='adam', loss='mean_squared_error')
model_ffnn.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

50% of the neurons in the network will be randomly removed during each training session before the count is calculated and updated. Dropout(0.5) is introduced after each Density layer at a rate of 0.5. By randomly "turning off" neurons, this prevents the model from overfitting the training set and improves its generalization ability.

```
Epoch 1/10
200/200 [==============================] - 5s 13ms/step - loss: 61888.5586 - val_loss: 32126.1445
Epoch 2/10
200/200 [==============================] - 2s 11ms/step - loss: 57861.4648 - val_loss: 30515.4648
Epoch 3/10
200/200 [==============================] - 2s 11ms/step - loss: 53610.6172 - val_loss: 28431.9219
Epoch 4/10
200/200 [==============================] - 2s 11ms/step - loss: 49724.2773 - val_loss: 26903.6719
Epoch 5/10
200/200 [==============================] - 1s 5ms/step - loss: 44341.8633 - val_loss: 26098.6270
Epoch 6/10
200/200 [==============================] - 1s 6ms/step - loss: 41483.2930 - val_loss: 27867.0879
Epoch 7/10
200/200 [==============================] - 1s 7ms/step - loss: 35687.2383 - val_loss: 32774.6602
Epoch 8/10
200/200 [==============================] - 1s 7ms/step - loss: 32790.0898 - val_loss: 45996.7891
Epoch 9/10
200/200 [==============================] - 2s 9ms/step - loss: 32822.5352 - val_loss: 68214.6328
Epoch 10/10
200/200 [==============================] - 2s 12ms/step - loss: 28060.7754 - val_loss: 109305.6484
<keras.src.callbacks.History at 0x7eda74b73100>
```

5.2.2 Regularization in RNN

Boost RNN's dropout layers:

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
# Identify the target variable column
target_column = 'Sales'
# Extract features and target
features = df.drop(columns=[target_column])
target = df[target_column]
# Handle categorical variables (one-hot encoding)
categorical_cols = features.select_dtypes(include='object').columns
features = pd.get_dummies(features, columns=categorical_cols, drop_first=True)
```

```
# Convert to numpy arrays
X = features.values
y = target.values.reshape(-1, 1)
# Normalize the data (if needed)
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
y = scaler.fit_transform(y)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Reshape the input for LSTM
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
# Build the model with Dropout layers
model_rnn = Sequential()
model_rnn.add(LSTM(50, input_shape=(X_train.shape[1], 1), activation='relu'))
model_rnn.add(Dropout(0.5))  # Add Dropout after the LSTM layer
model_rnn.add(Dense(1, activation='linear'))
model_rnn.compile(optimizer='adam', loss='mean_squared_error')
# Train the model
model_rnn.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

By randomly removing certain neurons during training, the Dropout layer, which is added after the LSTM layer and has a dropout rate of 0.5, helps the model prevent overfitting. As a result, the model can generalize better and make accurate predictions on experimental data that have never been seen before.

```
Epoch 1/10
200/200 [==============================] - 54s 252ms/step - loss: 7.3262e-04 - val_loss: 4.5458e-04
Epoch 2/10
200/200 [==============================] - 50s 251ms/step - loss: 7.0952e-04 - val_loss: 4.5327e-04
Epoch 3/10
200/200 [==============================] - 51s 256ms/step - loss: 7.0731e-04 - val_loss: 4.5405e-04
Epoch 4/10
200/200 [==============================] - 53s 263ms/step - loss: 7.0715e-04 - val_loss: 4.5198e-04
Epoch 5/10
200/200 [==============================] - 49s 245ms/step - loss: 7.0415e-04 - val_loss: 4.5374e-04
Epoch 6/10
200/200 [==============================] - 50s 249ms/step - loss: 7.0613e-04 - val_loss: 4.5092e-04
Epoch 7/10
200/200 [==============================] - 50s 248ms/step - loss: 7.0491e-04 - val_loss: 4.7831e-04
Epoch 8/10
200/200 [==============================] - 50s 251ms/step - loss: 7.0466e-04 - val_loss: 4.5108e-04
Epoch 9/10
200/200 [==============================] - 49s 243ms/step - loss: 7.0254e-04 - val_loss: 4.5373e-04
Epoch 10/10
200/200 [==============================] - 51s 258ms/step - loss: 7.0451e-04 - val_loss: 4.5519e-04
<keras.src.callbacks.History at 0x7f02c26b23b0>
```

5.2.3 Early Stopping

Use the EarlyStopping callback to stop the training process when there is no improvement long enough:

<u>Code:</u>

```
from tensorflow.keras.callbacks import EarlyStopping


early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
#During the fitting process
model_ffnn.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2,
callbacks=[early_stopping])
```

```
# Or
model_rnn.fit(X_train.reshape(X_train.shape[0], X_train.shape[1], 1), y_train,
epochs=10, batch_size=32, validation_split=0.2, callbacks=[early_stopping])
```

The training procedure is terminated if the patience of the next loop does not improve, and the EarlyStopping callback is used to monitor the improvement of the val_loss measure (error on the validation set). When the workout ends, the best weight will be restored if the Restore_best_weights=True option is set. By ending training early when the model is no longer learning much, this callback improves training by preventing overfitting and saving training time.

```
Epoch 1/10
200/200 [==============================] - 2s 8ms/step - loss: 27626.7188 - val_loss: 203167.7344
Epoch 2/10
200/200 [==============================] - 2s 9ms/step - loss: 32001.7207 - val_loss: 296639.4375
Epoch 3/10
200/200 [==============================] - 2s 11ms/step - loss: 25379.8516 - val_loss: 423803.6875
Epoch 4/10
200/200 [==============================] - 2s 11ms/step - loss: 23332.1660 - val_loss: 541794.6250
Epoch 1/10
200/200 [==============================] - 49s 246ms/step - loss: nan - val_loss: nan
Epoch 2/10
200/200 [==============================] - 48s 241ms/step - loss: nan - val_loss: nan
Epoch 3/10
200/200 [==============================] - 47s 237ms/step - loss: nan - val_loss: nan
<keras.src.callbacks.History at 0x7cf4e6201180>
```

5.2.4 Batch Normalization

Add a BatchNormalization layer to help stabilize the training process

Code:

```
from tensorflow.keras.layers import BatchNormalization
model_ffnn = Sequential()
model_ffnn.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model_ffnn.add(BatchNormalization())  # Thêm BatchNormalization layer
model_ffnn.add(Dense(64, activation='relu'))
model_ffnn.add(BatchNormalization())  # Thêm BatchNormalization layer
model_ffnn.add(Dense(1, activation='linear'))
model_ffnn.compile(optimizer='adam', loss='mean_squared_error')
model_ffnn.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

The BatchNormalization layer is added after each Dense layer in the FFNN model. This layer helps adjust and stabilize the input value for each layer during the training process, thereby improving the network's learning ability and helping the model converge faster. BatchNormalization can help reduce vanishing/exploding gradients and stabilize the training process, increasing the model's accuracy and generalization ability.

```
Epoch 1/10
200/200 [==============================] - 3s 10ms/step - loss: 62214.3789 - val_loss: 32820.8516
Epoch 2/10
200/200 [==============================] - 2s 9ms/step - loss: 59907.6914 - val_loss: 30588.6426
Epoch 3/10
200/200 [==============================] - 2s 10ms/step - loss: 57393.3906 - val_loss: 28864.2480
Epoch 4/10
200/200 [==============================] - 1s 7ms/step - loss: 54882.3320 - val_loss: 26878.2188
Epoch 5/10
200/200 [==============================] - 1s 6ms/step - loss: 52178.2383 - val_loss: 25763.4590
Epoch 6/10
200/200 [==============================] - 1s 6ms/step - loss: 49814.3438 - val_loss: 25486.0547
Epoch 7/10
200/200 [==============================] - 1s 6ms/step - loss: 48203.1055 - val_loss: 24054.9375
Epoch 8/10
200/200 [==============================] - 1s 6ms/step - loss: 45741.2227 - val_loss: 22254.3789
Epoch 9/10
200/200 [==============================] - 1s 5ms/step - loss: 44786.6836 - val_loss: 19994.0273
Epoch 10/10
200/200 [==============================] - 1s 4ms/step - loss: 43335.7891 - val_loss: 19466.1426
<keras.src.callbacks.History at 0x7cf45d968b20>
```

CHAPTER 6: IMPROVE THE ACCURACY

To improve the accuracy of your machine learning model, you can take the following steps:

Test activation functions: Use activation functions that fit your problem, or modify activation functions in hidden layers.

Set the loss function: Choose a loss function that makes sense in a given situation. A good option for predicting continuous values might be 'mean_squared_error'. And additional loss functions like 'mean_absolute_error' can also be taken into account.

Counting layers and neurons: Try building a model with additional layers and neurons to see if that makes predictions more accurate.

Verify and remove noise from the data: Make sure that there is no noise in the data and try to remove any variables that are unnecessary or could introduce noise into the model.

Use BatchNormalization Class: Stability during training can be achieved by providing the model with a BatchNormalization class.

Code:

```
# Giả sử df là DataFrame chứa dữ liệu của bạn
# Chuyển đổi biến hạng mục thành dạng số bằng phương pháp One-Hot Encoding
```

```
df_encoded = pd.get_dummies(df, columns=['Ship Mode', 'Segment', 'Country', 'City',
'State', 'Region', 'Category', 'Sub-Category'])
# Divide data into X (feature) and y (target variable)
X = df_encoded.drop('Profit', axis=1)
y = df_encoded['Profit']
# Normalize characteristic data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Divide the data into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
# Check the NaN value in the input data
print("Before handling NaN values:")
print(np.isnan(X_train).any())
# Handle NaN values (e.g. use fillna with mean value)
X_train = pd.DataFrame(X_train).fillna(pd.DataFrame(X_train).mean()).to_numpy()
X_test = pd.DataFrame(X_test).fillna(pd.DataFrame(X_test).mean()).to_numpy()
# Check again after processing
print("\nAfter handling NaN values:")
print(np.isnan(X_train).any())
# FFNN model before improvement
model_ffnn_initial = Sequential()
model_ffnn_initial.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model_ffnn_initial.add(Dense(64, activation='relu'))
model_ffnn_initial.add(Dense(1, activation='linear'))
model_ffnn_initial.compile(optimizer='adam', loss='mean_squared_error')
model_ffnn_initial.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
# Evaluate the accuracy on the test set
y_pred_ffnn_initial = model_ffnn_initial.predict(X_test)
mse_ffnn_initial = mean_squared_error(y_test, y_pred_ffnn_initial)
print(f'FFNN Mean Squared Error on Test Set (Initial): {mse_ffnn_initial}')
# Improved FFNN model
model_ffnn_improved = Sequential()
model_ffnn_improved.add(Dense(256, input_dim=X_train.shape[1], activation='relu'))
model_ffnn_improved.add(Dense(128, activation='relu'))
model_ffnn_improved.add(Dense(64, activation='relu'))  # Thêm một lớp ẩn nữa
model_ffnn_improved.add(Dense(1, activation='linear'))
model_ffnn_improved.compile(optimizer='adam', loss='mean_squared_error')
model_ffnn_improved.fit(X_train, y_train, epochs=50, batch_size=32,
validation_split=0.2)
# Evaluate the accuracy on the test set
y_pred_ffnn_improved = model_ffnn_improved.predict(X_test)
mse_ffnn_improved = mean_squared_error(y_test, y_pred_ffnn_improved)
print(f'FFNN Mean Squared Error on Test Set (Improved): {mse_ffnn_improved}')
# Draw a comparison chart
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_ffnn_initial, alpha=0.5)
plt.title('FFNN Predictions (Initial)')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred_ffnn_improved, alpha=0.5)
plt.title('FFNN Predictions (Improved)')
```

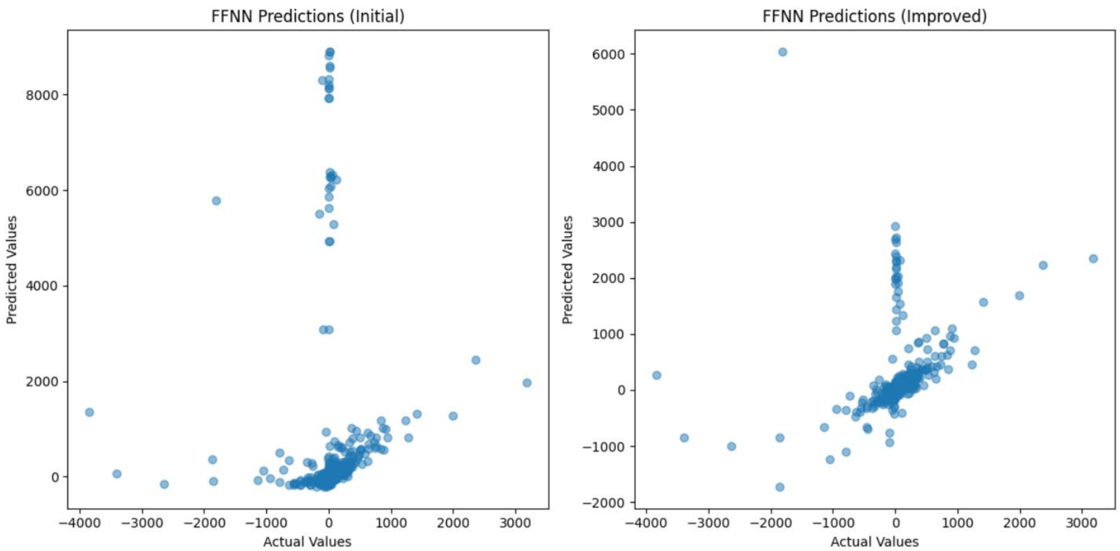
```
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.tight_layout()
plt.show()
```

The results of FFNN Mean Squared Error on Test Set (Initial): **229791.30254812478** and *FFNN Mean Squared Error on Test Set (Improved):* **77817.35174789515** demonstrate a significant improvement in model performance after applying improvement measures. What the results mean and the methods used to achieve improvement are presented below:

Meaning of the results:

FFNN Mean Squared Error on Test Set (Initial): The initial MSE value (high) indicates that the previous FFNN model has inaccurate prediction ability and has poor performance on the test set. FFNN Mean Squared Error on Test Set (Improved): The significant decrease of the MSE value after improvement shows that the FFNN model then has better prediction ability, reducing the error between prediction and actual value. Methods used to improve:

Handling NaN value: Before improving, we checked and processed NaN value in input data using fillna with average value. Refining the FFNN model architecture: The FFNN model has been improved by making architectural changes, such as adding new hidden layers and adjusting the sizes of the layers. This can help the model learn a more complex representation of the relationship between the input features and the target variable.


FFNN Predictions (Initial)
Predicted Values
Actual Values
FFNN Predictions (Improved)
Predicted Values
Actual Values