```
# Jeffery Dirden
# W214801986

# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# Generate synthetic dataset (simulate a smart electronics store)
np.random.seed(42)

# Simulating features for the dataset
n_samples = 1000
quantity_ordered = np.random.randint(1, 10, size=n_samples)  # Quantity ordered between 1 and 10
price_each = np.random.uniform(100, 500, size=n_samples)  # Price each between $100 and $500
discount = np.random.uniform(0, 0.2, size=n_samples)  # Discount applied between 0 and 20%
marketing_spend = np.random.uniform(1000, 10000, size=n_samples)  # Marketing spend between $1000 and $10000
customer_rating = np.random.uniform(1, 5, size=n_samples)  # Customer rating between 1 and 5

# Sales revenue (Turnover) calculated as: Quantity Ordered * Price Each * (1 - Discount) + Marketing Spend * Effect
# Assume that marketing spend and customer rating have some influence on sales
sales_revenue = (quantity_ordered * price_each * (1 - discount)) + (marketing_spend * np.random.uniform(0.5, 1.5, size=n_samples)

# Creating the DataFrame
data = pd.DataFrame({
    'Quantity Ordered': quantity_ordered,
    'Price Each': price_each,
    'Discount': discount,
    'Marketing Spend': marketing_spend,
    'Customer Rating': customer_rating,
    'Turnover': sales_revenue  # Target variable
})

# Preprocessing: Separate features and target variable
X = data.drop('Turnover', axis=1)  # Features
y = data['Turnover']  # Target variable (Sales Revenue)

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize and train the Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X_train_scaled, y_train)

# Predict on the test set
linear_predictions = linear_model.predict(X_test_scaled)

# Evaluate the Linear Regression model
linear_mse = mean_squared_error(y_test, linear_predictions)
linear_r2 = r2_score(y_test, linear_predictions)
print(f'Linear Regression - MSE: {linear_mse}, R2: {linear_r2}')

# Initialize and train the Lasso Regression model (with regularization)
lasso_model = Lasso(alpha=0.1)  # Adjust alpha for regularization strength
lasso_model.fit(X_train_scaled, y_train)

# Predict on the test set
lasso_predictions = lasso_model.predict(X_test_scaled)

# Evaluate the Lasso Regression model
lasso_mse = mean_squared_error(y_test, lasso_predictions)
lasso_r2 = r2_score(y_test, lasso_predictions)
print(f'Lasso Regression - MSE: {lasso_mse}, R2: {lasso_r2}')

# Initialize and train the Random Forest model
```

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

# Predict on the test set
rf_predictions = rf_model.predict(X_test)

# Evaluate the Random Forest model
rf_mse = mean_squared_error(y_test, rf_predictions)
rf_r2 = r2_score(y_test, rf_predictions)
print(f'Random Forest - MSE: {rf_mse}, R2: {rf_r2}')

# Initialize and train the Gradient Boosting model
gbm_model = GradientBoostingRegressor(n_estimators=100, random_state=42)
gbm_model.fit(X_train, y_train)

# Predict on the test set
gbm_predictions = gbm_model.predict(X_test)

# Evaluate the Gradient Boosting model
gbm_mse = mean_squared_error(y_test, gbm_predictions)
gbm_r2 = r2_score(y_test, gbm_predictions)
print(f'Gradient Boosting - MSE: {gbm_mse}, R2: {gbm_r2}')

# Initialize and train the XGBoost model
xgb_model = xgb.XGBRegressor(n_estimators=100, random_state=42)
xgb_model.fit(X_train, y_train)

# Predict on the test set
xgb_predictions = xgb_model.predict(X_test)

# Evaluate the XGBoost model
xgb_mse = mean_squared_error(y_test, xgb_predictions)
xgb_r2 = r2_score(y_test, xgb_predictions)
print(f'XGBoost - MSE: {xgb_mse}, R2: {xgb_r2}')
```

```
Linear Regression - MSE: 3099647.713358311, R2: 0.723729316402499
Lasso Regression - MSE: 3099668.5043177214, R2: 0.7237274633104382
Random Forest - MSE: 2955937.261280524, R2: 0.7365381864765179
Gradient Boosting - MSE: 3211276.1788769066, R2: 0.7137799042983896
XGBoost - MSE: 3405206.60923854, R2: 0.6964949424185385
```