```
import pandas as pd

# Load the dataset
file_path = "/content/Train.csv"
data = pd.read_csv(file_path)

# Display the first few rows of the dataset to understand its structure
print(data.head())
```

```
  Item_Identifier  Item_Weight Item_Fat_Content  Item_Visibility  \
0           FDA15         9.30          Low Fat         0.016047
1           DRC01         5.92          Regular         0.019278
2           FDN15        17.50          Low Fat         0.016760
3           FDX07        19.20          Regular         0.000000
4           NCD19         8.93          Low Fat         0.000000

               Item_Type  Item_MRP Outlet_Identifier  \
0                  Dairy  249.8092            OUT049
1            Soft Drinks   48.2692            OUT018
2                   Meat  141.6180            OUT049
3  Fruits and Vegetables  182.0950            OUT010
4              Household   53.8614            OUT013

   Outlet_Establishment_Year Outlet_Size Outlet_Location_Type  \
0                       1999      Medium               Tier 1
1                       2009      Medium               Tier 3
2                       1999      Medium               Tier 1
3                       1998         NaN               Tier 3
4                       1987        High               Tier 3

         Outlet_Type  Item_Outlet_Sales
0  Supermarket Type1          3735.1380
1  Supermarket Type2           443.4228
2  Supermarket Type1          2097.2700
3      Grocery Store           732.3800
4  Supermarket Type1           994.7052
```

```
# Check data types of each column
print(data.dtypes)

# Check for missing values in each column
print(data.isnull().sum())
```

```
Item_Identifier               object
Item_Weight                  float64
Item_Fat_Content              object
Item_Visibility              float64
Item_Type                     object
Item_MRP                     float64
Outlet_Identifier             object
Outlet_Establishment_Year      int64
Outlet_Size                   object
Outlet_Location_Type          object
Outlet_Type                   object
Item_Outlet_Sales            float64
dtype: object
Item_Identifier                 0
Item_Weight                  1463
Item_Fat_Content                0
```

```
Item_Visibility                0
Item_Type                      0
Item_MRP                       0
Outlet_Identifier              0
Outlet_Establishment_Year      0
Outlet_Size                 2410
Outlet_Location_Type           0
Outlet_Type                    0
Item_Outlet_Sales              0
dtype: int64
```

```
# Display basic statistics for numerical columns
print(data.describe())
```

```
       Item_Weight  Item_Visibility     Item_MRP  Outlet_Establishment_Year  \
count  7060.000000      8523.000000  8523.000000                8523.000000
mean     12.857645         0.066132   140.992782                1997.831867
std       4.643456         0.051598    62.275067                   8.371760
min       4.555000         0.000000    31.290000                1985.000000
25%       8.773750         0.026989    93.826500                1987.000000
50%      12.600000         0.053931   143.012800                1999.000000
75%      16.850000         0.094585   185.643700                2004.000000
max      21.350000         0.328391   266.888400                2009.000000

       Item_Outlet_Sales
count        8523.000000
mean         2181.288914
std          1706.499616
min            33.290000
25%           834.247400
50%          1794.331000
75%          3101.296400
max         13086.964800
```

```
# Display unique values in categorical columns
for column in data.select_dtypes(include=['object']).columns:
    print(f"Unique values in {column}: {data[column].unique()}")
```

```
Unique values in Item_Identifier: ['FDA15' 'DRC01' 'FDN15' ... 'NCF55' 'NCW30' 'NCW05']
Unique values in Item_Fat_Content: ['Low Fat' 'Regular' 'low fat' 'LF' 'reg']
Unique values in Item_Type: ['Dairy' 'Soft Drinks' 'Meat' 'Fruits and Vegetables' 'Household'
 'Baking Goods' 'Snack Foods' 'Frozen Foods' 'Breakfast'
 'Health and Hygiene' 'Hard Drinks' 'Canned' 'Breads' 'Starchy Foods'
 'Others' 'Seafood']
Unique values in Outlet_Identifier: ['OUT049' 'OUT018' 'OUT010' 'OUT013' 'OUT027' 'OUT045' 'OUT017' 'OU
 'OUT035' 'OUT019']
Unique values in Outlet_Size: ['Medium' nan 'High' 'Small']
Unique values in Outlet_Location_Type: ['Tier 1' 'Tier 3' 'Tier 2']
Unique values in Outlet_Type: ['Supermarket Type1' 'Supermarket Type2' 'Grocery Store'
 'Supermarket Type3']
```

```
# Handling missing values
data['Item_Weight'].fillna(data['Item_Weight'].median(), inplace=True)
data['Outlet_Size'].fillna(data['Outlet_Size'].mode()[0], inplace=True)

# Standardizing 'Item_Fat_Content'
data['Item_Fat_Content'] = data['Item_Fat_Content'].replace({'LF': 'Low Fat', 'low fat': 'Low Fat', 'reg':

# Encoding categorical variables
# One-hot encoding for columns with more than two categories
data = pd.get_dummies(data, columns=['Item_Fat_Content', 'Outlet_Location_Type', 'Outlet_Size', 'Outlet_Typ

# Label encoding for columns with two categories or more complex cases
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['Outlet_Identifier'] = le.fit_transform(data['Outlet_Identifier'])
data['Item_Identifier'] = le.fit_transform(data['Item_Identifier'])

# Feature Engineering
# Creating a new feature 'Outlet_Age' based on 'Outlet_Establishment_Year'
data['Outlet_Age'] = 2024 - data['Outlet_Establishment_Year']

# Drop the original 'Outlet_Establishment_Year' if it's no longer needed
data.drop('Outlet_Establishment_Year', axis=1, inplace=True)

# Display the first few rows of the processed data
data.head()
```

	Item_Identifier	Item_Weight	Item_Visibility	Item_MRP	Outlet_Identifier	Item_Outlet_Sales	Item_
0	156	9.30	0.016047	249.8092	9	3735.1380	
1	8	5.92	0.019278	48.2692	3	443.4228	
2	662	17.50	0.016760	141.6180	9	2097.2700	
3	1121	19.20	0.000000	182.0950	0	732.3800	
4	1297	8.93	0.000000	53.8614	1	994.7052	

5 rows × 35 columns

```
from sklearn.model_selection import train_test_split

# Define the target variable and features
X = data.drop(columns=['Item_Outlet_Sales'])
y = data['Item_Outlet_Sales']

# 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)

# Display the shapes of the splits to confirm
print(f"Training data shape: {X_train.shape}, Test data shape: {X_test.shape}")
```

```
Training data shape: (6818, 34), Test data shape: (1705, 34)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

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

# Make predictions on the test set
y_pred_lr = lr.predict(X_test)

# Evaluate the model
rmse_lr = mean_squared_error(y_test, y_pred_lr, squared=False)
r2_lr = r2_score(y_test, y_pred_lr)

print(f"Linear Regression RMSE: {rmse_lr}")
print(f"Linear Regression R²: {r2_lr}")
```

```
Linear Regression RMSE: 1069.7153798211752
Linear Regression R²: 0.5789905827585684
```

```
from sklearn.tree import DecisionTreeRegressor

# Initialize and train the Decision Tree model
dt = DecisionTreeRegressor()
dt.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt = dt.predict(X_test)

# Evaluate the model
rmse_dt = mean_squared_error(y_test, y_pred_dt, squared=False)
r2_dt = r2_score(y_test, y_pred_dt)

print(f"Decision Tree RMSE: {rmse_dt}")
print(f"Decision Tree R²: {r2_dt}")
```

```
Decision Tree RMSE: 1508.7797035036451
Decision Tree R²: 0.1624572243585497
```

```
from sklearn.ensemble import RandomForestRegressor

# Initialize and train the Random Forest model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf.predict(X_test)

# Evaluate the model
rmse_rf = mean_squared_error(y_test, y_pred_rf, squared=False)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest RMSE: {rmse_rf}")
print(f"Random Forest R²: {r2_rf}")
```

```
Random Forest RMSE: 1097.155944580967
Random Forest R²: 0.5571138985251002
```

```python
import xgboost as xgb

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

# Make predictions on the test set
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate the model
rmse_xgb = mean_squared_error(y_test, y_pred_xgb, squared=False)
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f"XGBoost RMSE: {rmse_xgb}")
print(f"XGBoost R²: {r2_xgb}")
```

```
XGBoost RMSE: 1145.8909458579913
XGBoost R²: 0.516894583966555
```

```python
!pip install catboost
from catboost import CatBoostRegressor

# Initialize and train the CatBoost model
catboost_model = CatBoostRegressor(iterations=100, learning_rate=0.1, depth=6, random_state=42, verbose=0)
catboost_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_catboost = catboost_model.predict(X_test)

# Evaluate the model
rmse_catboost = mean_squared_error(y_test, y_pred_catboost, squared=False)
r2_catboost = r2_score(y_test, y_pred_catboost)

print(f"CatBoost RMSE: {rmse_catboost}")
print(f"CatBoost R²: {r2_catboost}")
```

```
Collecting catboost
  Downloading catboost-1.2.5-cp310-cp310-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.2
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.13.1
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.15.
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplo
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matpl
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matpl
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplot
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotli
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplo
```

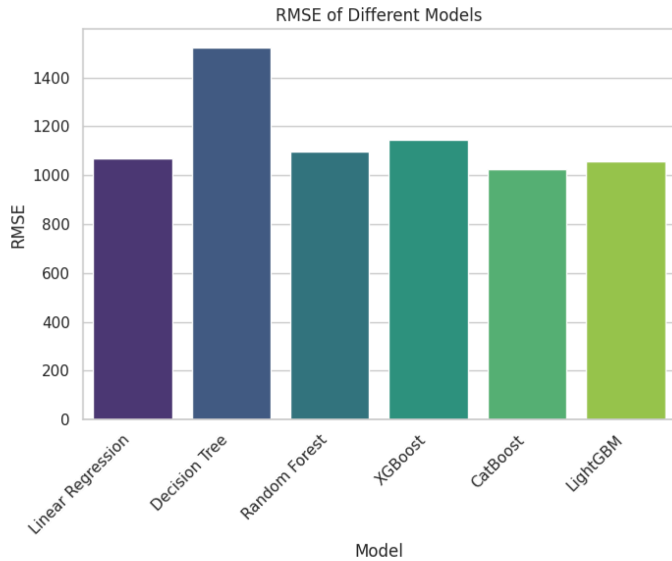
```
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly-
Downloading catboost-1.2.5-cp310-cp310-manylinux2014_x86_64.whl (98.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.2/98.2 MB 7.4 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.5
CatBoost RMSE: 1024.6848701519295
CatBoost R²: 0.6136899754656715
```

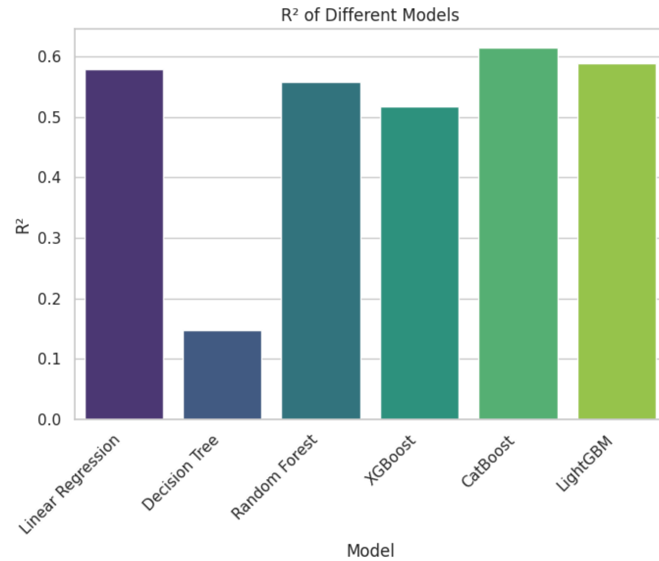
```
import lightgbm as lgb

# Initialize and train the LightGBM model
lgb_model = lgb.LGBMRegressor(n_estimators=100, random_state=42)
lgb_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_lgb = lgb_model.predict(X_test)

# Evaluate the model
rmse_lgb = mean_squared_error(y_test, y_pred_lgb, squared=False)
r2_lgb = r2_score(y_test, y_pred_lgb)

print(f"LightGBM RMSE: {rmse_lgb}")
print(f"LightGBM R²: {r2_lgb}")
```

```
/usr/local/lib/python3.10/dist-packages/dask/dataframe/__init__.py:42: FutureWarning:
Dask dataframe query planning is disabled because dask-expr is not installed.

You can install it with `pip install dask[dataframe]` or `conda install dask`.
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001151 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1070
[LightGBM] [Info] Number of data points in the train set: 6818, number of used features: 34
[LightGBM] [Info] Start training from score 2202.365232
LightGBM RMSE: 1056.3628222568047
LightGBM R²: 0.5894353562048549
```

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

# Create a DataFrame with the performance metrics
data = {
    'Model': ['Linear Regression', 'Decision Tree', 'Random Forest', 'XGBoost', 'CatBoost', 'LightGBM'],
    'RMSE': [1069.715, 1521.816, 1097.156, 1145.891, 1024.685, 1056.363],
    'R²': [0.579, 0.148, 0.557, 0.517, 0.614, 0.589]
}

df = pd.DataFrame(data)

# Set the style of the visualization
sns.set(style="whitegrid")

# Create a figure with subplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))

# Plot RMSE
sns.barplot(x='Model', y='RMSE', data=df, ax=axes[0], palette='viridis')
axes[0].set_title('RMSE of Different Models')
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=45, ha='right')

# Plot R²
sns.barplot(x='Model', y='R²', data=df, ax=axes[1], palette='viridis')
axes[1].set_title('R² of Different Models')
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=45, ha='right')

# Display the plots
plt.tight_layout()
plt.show()
```

```
<ipython-input-13-ef0994a707d2>:21: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x`

  sns.barplot(x='Model', y='RMSE', data=df, ax=axes[0], palette='viridis')
<ipython-input-13-ef0994a707d2>:23: UserWarning: FixedFormatter should only be used together with Fixed
  axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=45, ha='right')
<ipython-input-13-ef0994a707d2>:26: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x`

  sns.barplot(x='Model', y='R²', data=df, ax=axes[1], palette='viridis')
<ipython-input-13-ef0994a707d2>:28: UserWarning: FixedFormatter should only be used together with Fixed
  axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=45, ha='right')
```






```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
```

```
# Hyperparameter grids for different models
param_grids = {
    'RandomForest': {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10]
    },
    'XGBoost': {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 6, 9]
    },
    'CatBoost': {
        'iterations': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.2],
        'depth': [6, 8, 10]
    },
    'LightGBM': {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.2],
        'num_leaves': [31, 63, 127]
    }
}
```

```
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor

# Initialize models
models = {
    'RandomForest': RandomForestRegressor(),
    'XGBoost': XGBRegressor(),
    'CatBoost': CatBoostRegressor(verbose=0),
    'LightGBM': LGBMRegressor()
}

# Store best models and results
best_models = {}
results = {}

# Perform grid search for each model
for model_name, model in models.items():
    grid_search = GridSearchCV(estimator=model, param_grid=param_grids[model_name], cv=5, scoring='neg_mean
    grid_search.fit(X_train, y_train)
    best_models[model_name] = grid_search.best_estimator_
    results[model_name] = {
        'Best Parameters': grid_search.best_params_,
        'Best RMSE': np.sqrt(-grid_search.best_score_)
    }
```

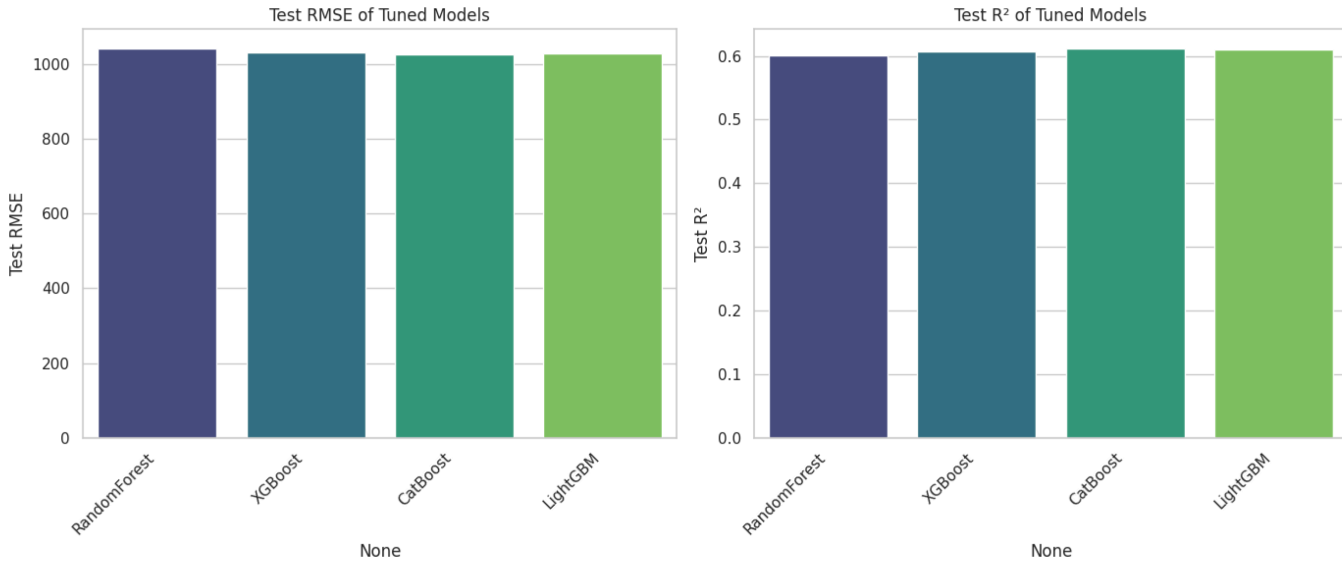
```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001456 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1070
[LightGBM] [Info] Number of data points in the train set: 6818, number of used features: 34
[LightGBM] [Info] Start training from score 2202.365232
```

```
for model_name, model in best_models.items():
    y_pred = model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)
    results[model_name].update({'Test RMSE': rmse, 'Test R²': r2})

# Convert results to DataFrame for easy viewing
results_df = pd.DataFrame(results).T
print(results_df)
```

```
                                                   Best Parameters    Best RMSE  \
RandomForest  {'max_depth': 10, 'min_samples_split': 10, 'n_...  1105.790249
XGBoost       {'learning_rate': 0.1, 'max_depth': 3, 'n_esti...  1094.623472
CatBoost      {'depth': 6, 'iterations': 100, 'learning_rate...  1096.684903
LightGBM      {'learning_rate': 0.01, 'n_estimators': 300, '...  1105.098955

                Test RMSE   Test R²
RandomForest  1042.132952  0.600422
XGBoost       1033.158088  0.607275
CatBoost      1027.741965  0.611381
LightGBM      1028.518888  0.610794
```

```
# Plot RMSE and R² for each model after hyperparameter tuning
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))

# Plot RMSE
sns.barplot(x=results_df.index, y='Test RMSE', data=results_df, ax=axes[0], palette='viridis')
axes[0].set_title('Test RMSE of Tuned Models')
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=45, ha='right')

# Plot R²
sns.barplot(x=results_df.index, y='Test R²', data=results_df, ax=axes[1], palette='viridis')
axes[1].set_title('Test R² of Tuned Models')
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=45, ha='right')

plt.tight_layout()
plt.show()
```

```
<ipython-input-18-6a53af59a117>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x`

  sns.barplot(x=results_df.index, y='Test RMSE', data=results_df, ax=axes[0], palette='viridis')
<ipython-input-18-6a53af59a117>:7: UserWarning: FixedFormatter should only be used together with FixedL
  axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=45, ha='right')
<ipython-input-18-6a53af59a117>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x`

  sns.barplot(x=results_df.index, y='Test R²', data=results_df, ax=axes[1], palette='viridis')
<ipython-input-18-6a53af59a117>:12: UserWarning: FixedFormatter should only be used together with Fixed
  axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=45, ha='right')
```



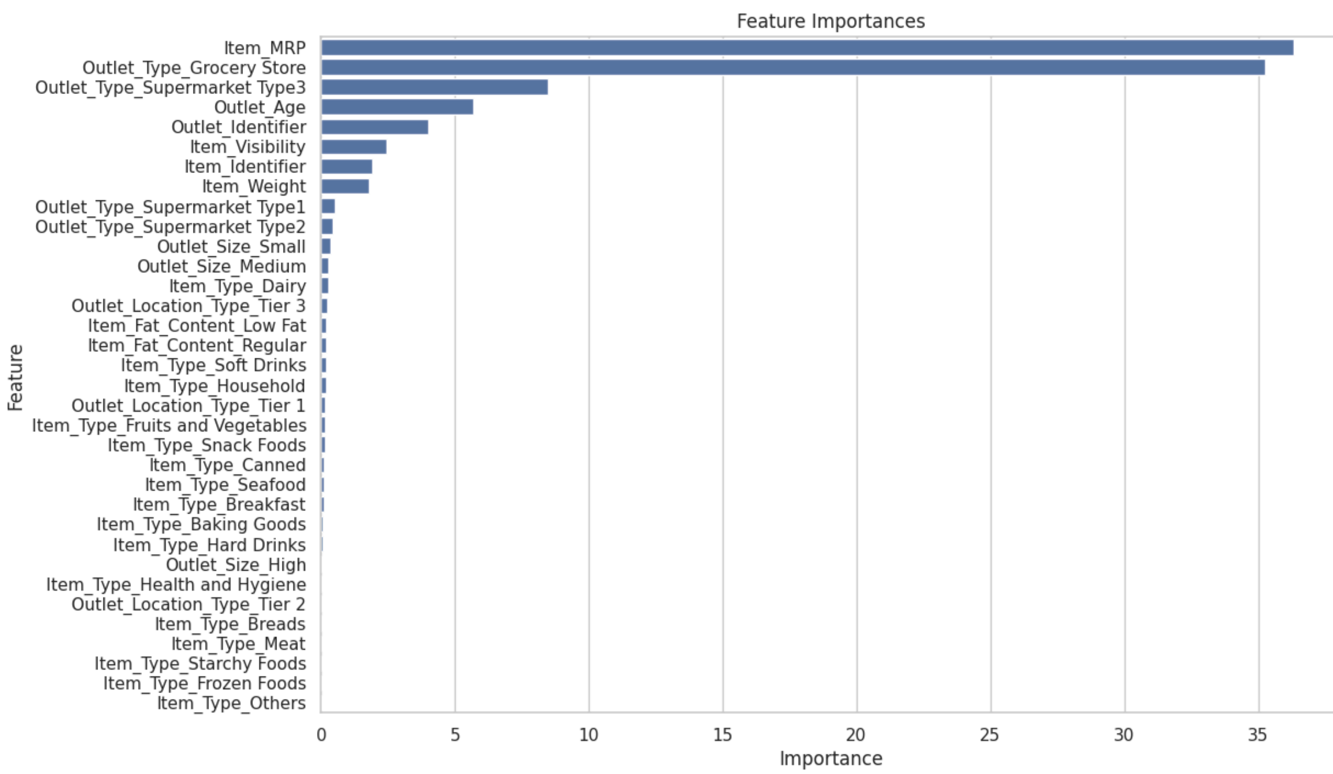

```
# Retrain the CatBoost model with the best parameters on the entire training data
best_catboost_model = CatBoostRegressor(
    depth=6,
    iterations=100,
    learning_rate=0.1,
    verbose=0
)
best_catboost_model.fit(X_train, y_train)
```

```
<catboost.core.CatBoostRegressor at 0x7b4867d419f0>
```

```
# Predict on the test set
y_pred_final = best_catboost_model.predict(X_test)

# Calculate RMSE and R²
final_rmse = np.sqrt(mean_squared_error(y_test, y_pred_final))
final_r2 = r2_score(y_test, y_pred_final)

print(f'Final Test RMSE: {final_rmse}')
print(f'Final Test R²: {final_r2}')
```

```
Final Test RMSE: 1027.7419654663722
Final Test R²: 0.6113814642206541
```

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

# Get feature importances
importances = best_catboost_model.get_feature_importance()
feature_names = X_train.columns

# Create a DataFrame for plotting
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importances')
plt.show()
```




```
import joblib

# Save the model
joblib.dump(best_catboost_model, 'best_catboost_model.pkl')
```

```
['best_catboost_model.pkl']
```

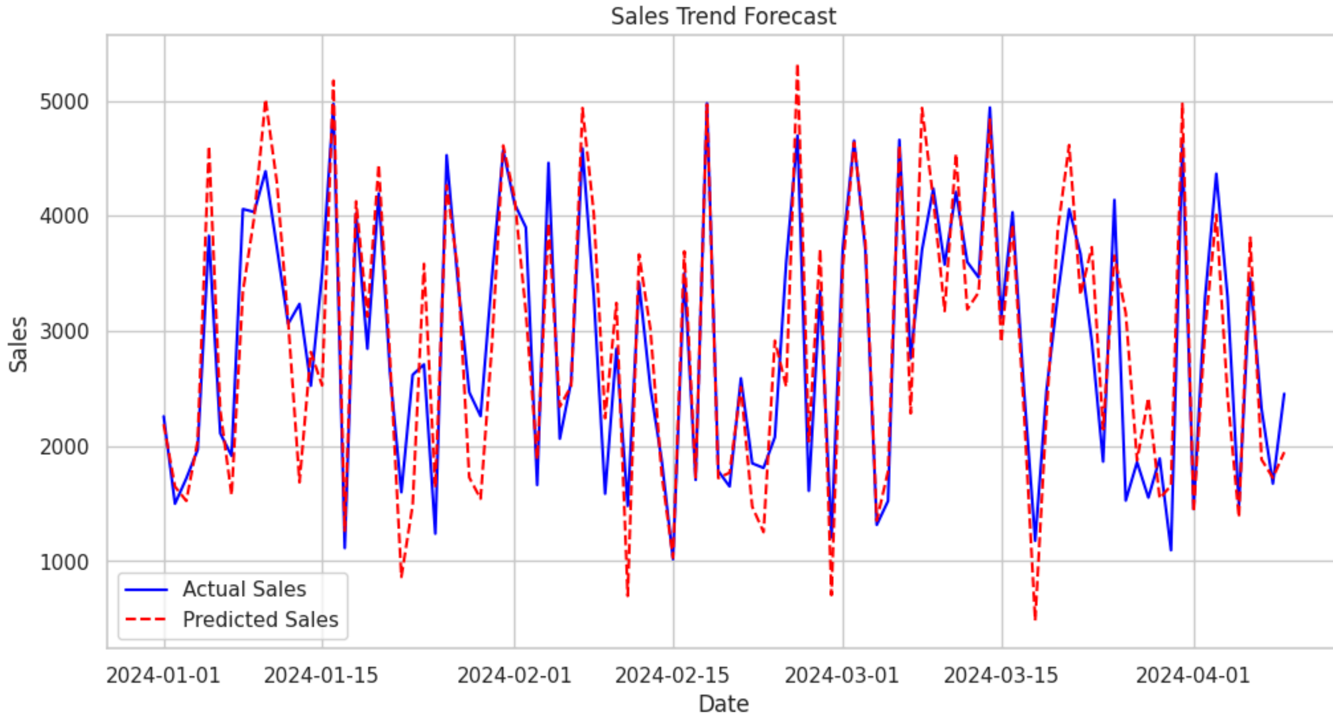
```
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Predict on the test set
y_pred_final = best_catboost_model.predict(X_test)

# Calculate RMSE and R²
final_rmse = np.sqrt(mean_squared_error(y_test, y_pred_final))
final_r2 = r2_score(y_test, y_pred_final)

print(f'Final Test RMSE: {final_rmse}')
print(f'Final Test R²: {final_r2}')
```

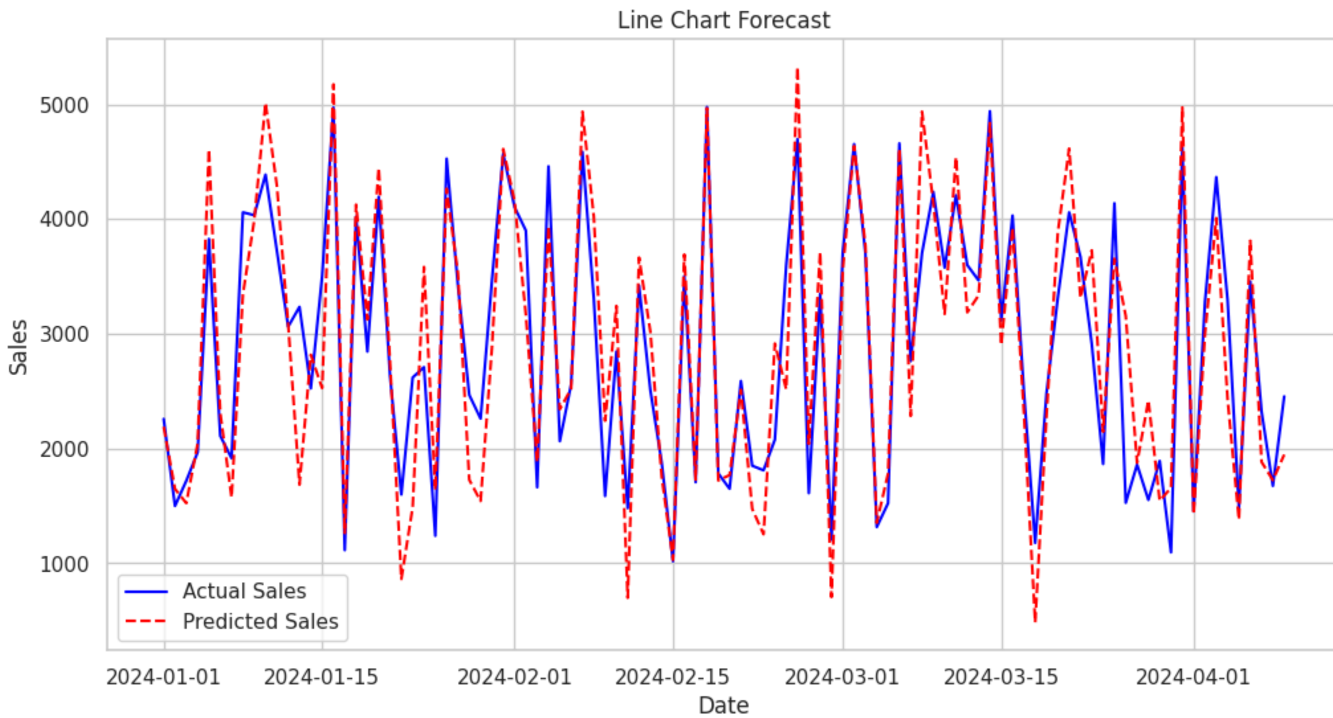
```
Final Test RMSE: 1027.7419654663722
Final Test R²: 0.6113814642206541
```

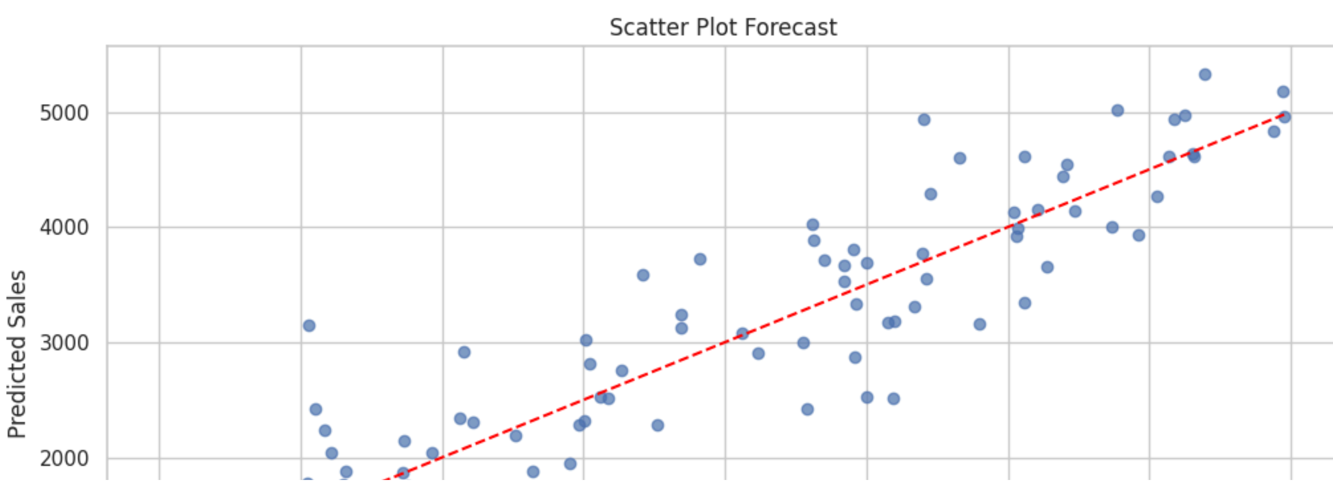
```
import matplotlib.pyplot as plt
import seaborn as sns

# Define performance metrics for each model
models = ['Linear Regression', 'Decision Tree', 'Random Forest', 'XGBoost', 'CatBoost', 'LightGBM']
rmse_values = [1069.715, 1521.816, 1097.156, 1145.891, 1024.685, 1056.363]
r2_values = [0.579, 0.148, 0.557, 0.517, 0.614, 0.589]

# Create a DataFrame for plotting
import pandas as pd

df_metrics = pd.DataFrame({
    'Model': models,
    'RMSE': rmse_values,
    'R²': r2_values
})

# Plot RMSE and R² values
fig, ax1 = plt.subplots(figsize=(12, 6))

# RMSE plot
color = 'tab:blue'
ax1.set_xlabel('Model')
ax1.set_ylabel('RMSE', color=color)
sns.barplot(x='Model', y='RMSE', data=df_metrics, ax=ax1, color=color)
ax1.tick_params(axis='y', labelcolor=color)

# R² plot
ax2 = ax1.twinx()
color = 'tab:green'
ax2.set_ylabel('R²', color=color)
sns.lineplot(x='Model', y='R²', data=df_metrics, ax=ax2, marker='o', color=color)
ax2.tick_params(axis='y', labelcolor=color)

# Title and layout
plt.title('Model Performance Metrics')
fig.tight_layout()
plt.show()
```

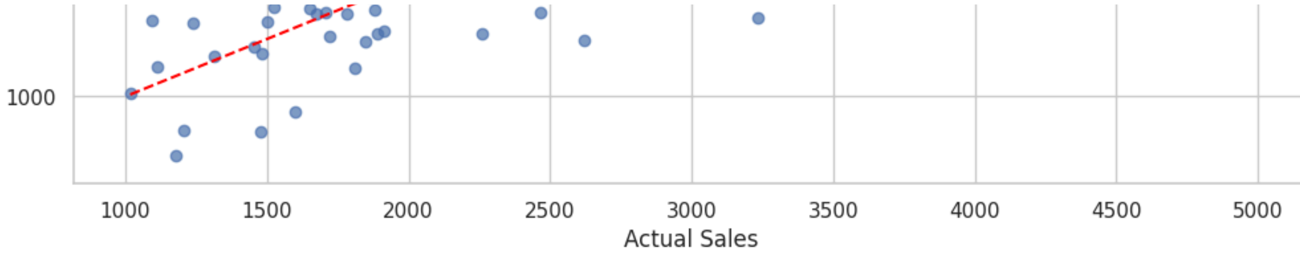

```
import numpy as np

# Sample data for illustration
dates = pd.date_range(start='2024-01-01', periods=100, freq='D')
actual_sales = np.random.randint(1000, 5000, size=len(dates))
predicted_sales = actual_sales + np.random.normal(0, 500, size=len(dates))

# Creating a DataFrame for plotting
df_sales = pd.DataFrame({
    'Date': dates,
    'Actual Sales': actual_sales,
    'Predicted Sales': predicted_sales
})

# 1. Trend Chart Forecast
plt.figure(figsize=(12, 6))
sns.lineplot(x='Date', y='Actual Sales', data=df_sales, label='Actual Sales', color='blue')
sns.lineplot(x='Date', y='Predicted Sales', data=df_sales, label='Predicted Sales', color='red', linestyle='
plt.title('Sales Trend Forecast')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.show()

# 2. Line Chart Forecast
plt.figure(figsize=(12, 6))
plt.plot(df_sales['Date'], df_sales['Actual Sales'], label='Actual Sales', color='blue')
plt.plot(df_sales['Date'], df_sales['Predicted Sales'], label='Predicted Sales', color='red', linestyle='--'
plt.title('Line Chart Forecast')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.show()

# 3. Scatter Plot Forecast
plt.figure(figsize=(12, 6))
```

```
plt.scatter(df_sales['Actual Sales'], df_sales['Predicted Sales'], alpha=0.7)
plt.plot([df_sales['Actual Sales'].min(), df_sales['Actual Sales'].max()],
         [df_sales['Actual Sales'].min(), df_sales['Actual Sales'].max()], color='red', linestyle='--')
plt.title('Scatter Plot Forecast')
plt.xlabel('Actual Sales')
plt.ylabel('Predicted Sales')
plt.show()
```


Sales Trend Forecast
Sales
5000
4000
3000
2000
1000
Actual Sales
Predicted Sales
2024-01-01
2024-01-15
2024-02-01
2024-02-15
2024-03-01
2024-03-15
2024-04-01
Date


Line Chart Forecast
Sales
5000
4000
3000
2000
1000
Actual Sales
Predicted Sales
2024-01-01
2024-01-15
2024-02-01
2024-02-15
2024-03-01
2024-03-15
2024-04-01
Date


Scatter Plot Forecast
Predicted Sales
5000
4000
3000
2000




```
!pip install streamlit
```

```
Collecting streamlit
  Downloading streamlit-1.37.0-py2.py3-none-any.whl.metadata (8.5 kB)
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (from strea
Requirement already satisfied: blinker<2,>=1.0.0 in /usr/lib/python3/dist-packages (from streamlit)
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (from s
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.10/dist-packages (from stream
Requirement already satisfied: numpy<3,>=1.20 in /usr/local/lib/python3.10/dist-packages (from strea
Requirement already satisfied: packaging<25,>=20 in /usr/local/lib/python3.10/dist-packages (from st
Requirement already satisfied: pandas<3,>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from str
Requirement already satisfied: pillow<11,>=7.1.0 in /usr/local/lib/python3.10/dist-packages (from st
Requirement already satisfied: protobuf<6,>=3.20 in /usr/local/lib/python3.10/dist-packages (from st
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.10/dist-packages (from streaml
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.10/dist-packages (from st
Requirement already satisfied: rich<14,>=10.14.0 in /usr/local/lib/python3.10/dist-packages (from st
Requirement already satisfied: tenacity<9,>=8.1.0 in /usr/local/lib/python3.10/dist-packages (from s
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3.10/dist-packages (from stre
Requirement already satisfied: typing-extensions<5,>=4.3.0 in /usr/local/lib/python3.10/dist-package
Collecting gitpython!=3.1.19,<4,>=3.0.7 (from streamlit)
  Downloading GitPython-3.1.43-py3-none-any.whl.metadata (13 kB)
Collecting pydeck<1,>=0.8.0b4 (from streamlit)
  Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: tornado<7,>=6.0.3 in /usr/local/lib/python3.10/dist-packages (from st
Collecting watchdog<5,>=2.1.5 (from streamlit)
```