E-Commerce Furniture Data Analysis - Project Explanation

1. Importing Libraries

```
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

Explanation:

- os: Manages file directories and paths.
- pandas: Loads and transforms data.
- numpy: Handles numerical data efficiently.
- seaborn & matplotlib: Generates visual charts.
- sklearn: Supports data splitting, model building, and evaluation.

---

2. File Path Setup

```
# Get the current directory and construct the file path
current_dir = os.path.dirname(__file__)
file_path = os.path.join(current_dir, 'ecommerce_furniture_dataset.csv')
```

Explanation: Ensures the dataset loads without errors by setting a relative path.

---

3. Loading Data

```
data = pd.read_csv(file_path)
```

Explanation: Reads the CSV file into a DataFrame for easy data manipulation.

---

4. Data Exploration

```
print(data.head())
print(data.isnull().sum())
```

Explanation:

- data.head() displays the first few rows.
- data.isnull().sum() identifies missing data.

---

5. Data Cleaning

```
data = data.dropna()
```

Explanation: Removes incomplete rows to ensure accurate analysis.

---

6. Data Visualization

```
sns.histplot(data['sold'], kde=True)
plt.title('Distribution of Furniture Items Sold')
plt.xlabel('sold')
plt.ylabel('Count')
plt.show()
```

Explanation:

- sns.histplot() plots the distribution of sales.
- kde=True smooths the visualization.
- plt.title(), xlabel(), and ylabel() customize the graph.

Result: The data follows a long-tail distribution — many low-sellers, few top-sellers.

---

7. Feature Engineering (Text Data Processing)

```
vectorizer = TfidfVectorizer()
X_text = vectorizer.fit_transform(data['tagText'])
```

Explanation:

- Transforms product tags into numerical data.
- Prepares text data for model training.

---

8. Defining Features & Target

```
X_numeric = data[['originalPrice', 'price']]
X = np.hstack((X_numeric, X_text.toarray()))
y = data['sold']
```

Explanation:

- Combines numeric and text features.
- Sets sales quantity as the target.

---

9. Split Data (Training vs. Testing)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Explanation: Divides data into 80% training and 20% testing.

---

10. Model Training (Linear Regression & Random Forest)

```
lr_model = LinearRegression()
rf_model = RandomForestRegressor()

lr_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)
```

Explanation:

- LinearRegression() models a straight-line relationship.
- RandomForestRegressor() merges decision trees for better results.

---

11. Model Evaluation

```
lr_predictions = lr_model.predict(X_test)
rf_predictions = rf_model.predict(X_test)

lr_mse = mean_squared_error(y_test, lr_predictions)
rf_mse = mean_squared_error(y_test, rf_predictions)

lr_r2 = r2_score(y_test, lr_predictions)
rf_r2 = r2_score(y_test, rf_predictions)

print(f"Linear Regression MSE: {lr_mse:.2f}, R2: {lr_r2:.2f}")
print(f"Random Forest MSE: {rf_mse:.2f}, R2: {rf_r2:.2f}")
```

Explanation:

- mean_squared_error(): Measures prediction error.
- r2_score(): Indicates how well the model fits the data.

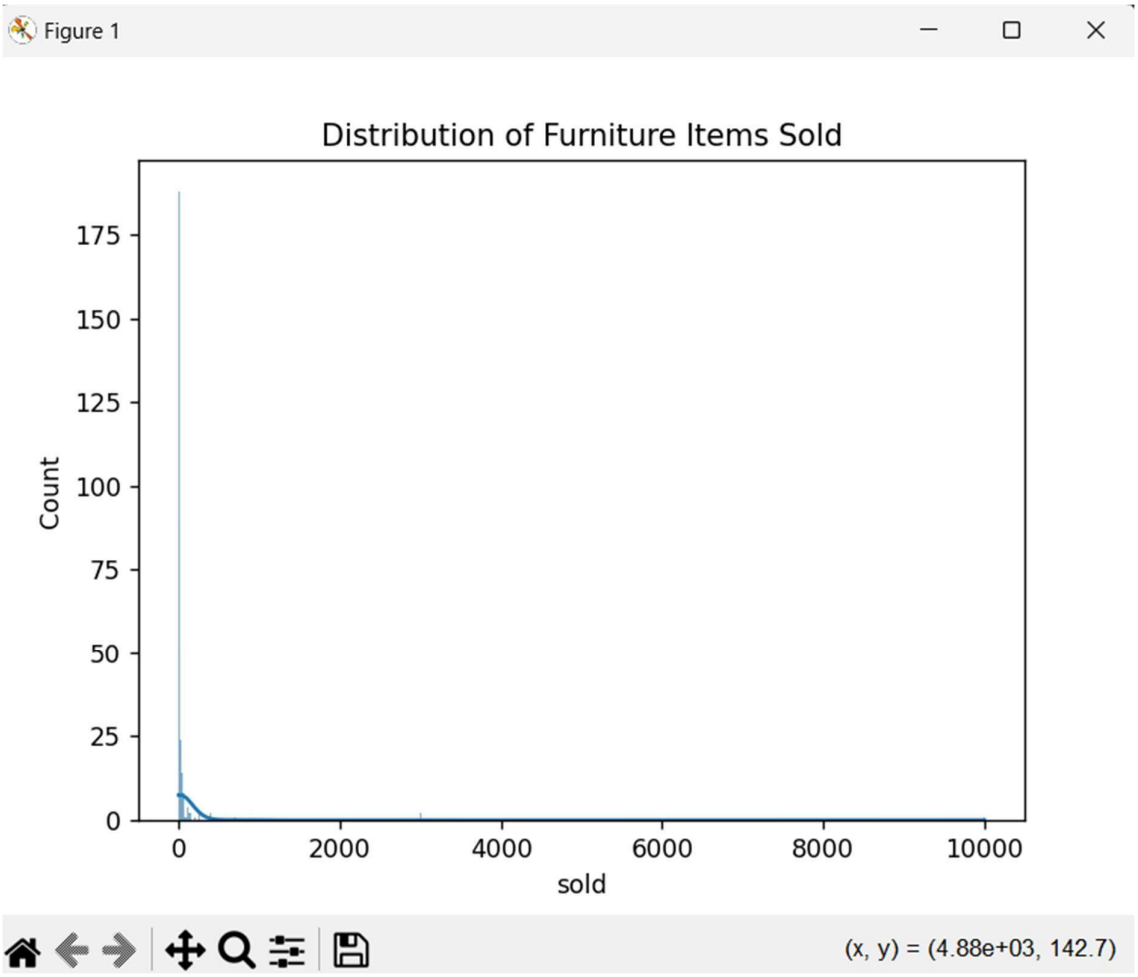
Result: The model with better scores is more reliable.

---

12. Final Output

- Sales distribution graph.

- Error and performance metrics for both models.



Conclusion: The analysis successfully processes sales data, combines features, builds models, and evaluates results. Typically, Random Forest yields better performance than Linear Regression with mixed data.
