# V2_full_osemn

November 5, 2025

## 1 Activity: Full OSEMN

### 1.1 Introduction

In this assignment, you will work on a data analysis project. This project will let you practice the skills you have learned in this course and write real code in Python.

You will perform the following steps of the OSEMN framework:
- Section 1.2 - Section 1.3 - Section 1.5

[1]:
```
# We'll import the libraries you'll likely use for this activity
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Data
df = pd.read_csv('transactions-pet_store.csv')
df_orig = df.copy()
```

### 1.2 Scrub

You will scrub the data. It's important that you follow the directions as stated. Doing more or less than what is asked might lead to not getting full points for the question.

---

If while you're working on the scrubbing phase you need to reset the DataFrame, you can restart the kernel (in the toolbar: "Kernel" > "Restart").

**Question 1** Remove all rows that have are missing either the `Product_Name` or the `Product_Category`. Assign the cleaned DataFrame to the variable `df` (overwriting the original DataFrame.).

[2]:
```
# Remove rows with missing Product_Name or Product_Category
df = df.dropna(subset=['Product_Name', 'Product_Category'])
```

[3]:
```
# Question 1 Grading Checks

assert df.shape[0] <= 2874, 'Did you remove all the rows with missing values␣
↪for the columns Product_Name & Product_Category?'
assert df.shape[0] >= 2700, 'Did you remove too many the rows with missing␣
↪values?'
assert len(df.columns) == 10, 'Make sure you do not drop any columns.'
```

**Question 2** Find any clearly "incorrect" values in the `Price` column and "clean" the DataFrame to address those values.

Ensure you make the changes to the DataFrame assigned to the variable `df`.

[4]:
```
# Clean incorrect (extreme) Price values
lower = df.Price.quantile(0.001)
upper = df.Price.quantile(0.999)
df = df[(df.Price >= lower) & (df.Price <= upper)]
```

[5]:
```
# Question 2 Grading Checks

assert (df.Price < df.Price.quantile(0.0001)).sum() == 0, 'Check for very small␣
↪values'
assert (df.Price > df.Price.quantile(0.999)).sum() == 0, 'Check for very large␣
↪values'
```

**Question 3** After you've done the cleaning above, remove any column that has more than `500` missing values.

Ensure you make the changes to the DataFrame assigned to the variable `df`.

[6]:
```
# Remove columns with more than 500 missing values
df = df.dropna(axis=1, thresh=len(df) - 500)
```

[7]:
```
# Question 3 Grading Checks

assert len(df.columns) < 10, 'You should have dropped 1 or more columns (with␣
↪more than 500 missing values)'
```

**Question 4** Address the other missing values. You can replace the values or remvove them, but whatever method you decide to clean the DataFrame, you should no longer have any missing values.

Ensure you make the changes to the DataFrame assigned to the variable `df`.

[8]:
```
# Fill or remove remaining missing values
df = df.dropna()
```

[9]:
```
# Question 4 Grading Checks

assert df.Customer_ID.isna().sum() == 0, 'Did you address all the missing␣
↪values?'
```

### 1.3 Explore

You will explore the data. It's important that you follow the directions as stated. Doing more or less than what is asked might lead to not getting full points for the question.

You may use either exploratory statistics or exploratory visualizations to help answer these questions.

---

Note that the DataFrame loaded for this section (in the below cell) is different from the data you used in the Section 1.2 section.

If while you're working on the scrubbing phase you need to reset the DataFrame, you can restart the kernel (in the toolbar: "Kernel" > "Restart").

[10]:
```
df = pd.read_csv('transactions-pet_store-clean.csv')
```

**Question 5** Create a `Subtotal` column by multiplying the `Price` and `Quantity` values. This represents how much was spent for a given transaction (row).

[11]:
```
# Create Subtotal column
df['Subtotal'] = df['Price'] * df['Quantity']
```

[12]:
```
# Question 5 Grading Checks

assert 'Subtotal' in df.columns, ''
```

**Question 6** Determine most common category (`Product_Category`) purchases (number of total items) for both `Product_Line` categories. Assign the (string) name of these categories to their respective variables `common_category_cat` & `common_category_dog`.

[17]:
```
df['Product_Line'].unique()
common_category_dog = (
    df[df['Product_Line'].str.contains('dog', case=False, na=False)]
    ['Product_Category']
    .mode()
    .iloc[0]
)

common_category_cat = (
    df[df['Product_Line'].str.contains('cat', case=False, na=False)]
```

```
    ['Product_Category']
    .mode()
    .iloc[0]
)
```

[18]:
```
# Question 6 Grading Checks

assert isinstance(common_category_dog, str), 'Ensure you assign the name of the␣
 ↪category (string) to the variable common_category_dog'
assert isinstance(common_category_cat, str), 'Ensure you assign the name of the␣
 ↪category (string) to the variable common_category_cat'
```

**Question 7** Determine which categories (`Product_Category`), by `Product_Line` have the ***median*** highest `Price`. Assign the (string) name of these categories to their respective variables `priciest_category_cat` & `priciest_category_dog`.

[19]:
```
# Category with highest median Price per Product_Line
priciest_category_dog = (
    df[df['Product_Line'] == 'dog']
    .groupby('Product_Category')['Price']
    .median()
    .idxmax()
)

priciest_category_cat = (
    df[df['Product_Line'] == 'cat']
    .groupby('Product_Category')['Price']
    .median()
    .idxmax()
)
```

[20]:
```
# Question 7 Grading Checks

assert isinstance(priciest_category_dog, str), 'Ensure you assign the name of␣
 ↪the category (string) to the variable priciest_category_dog'
assert isinstance(priciest_category_cat, str), 'Ensure you assign the name of␣
 ↪the category (string) to the variable priciest_category_cat'
```

## 1.4 Modeling

This is the point of the framework where we'd work on modeling with our data. However, in this activity, we're going to move straight to interpretting.

## 1.5 Interpret

You will interpret the data based on what you found so far. It's important that you follow the directions as stated. Doing more or less than what is asked might lead to not getting full points for the question.

---

Note that the DataFrame loaded for this section (in the below cell) is the same as the data you used in the Section 1.3 section.

If while you're working on the scrubbing phase you need to reset the DataFrame, you can restart the kernel (in the toolbar: "Kernel" > "Restart").

**Question 8** You want to emphasize to your stakeholders that the total number of product categories sold differ between the two `Product_Line` categories (`'cat'` & `'dog'`).

Create a ***horizontal* bar plot** that has `Product_Category` on the y-axis and the total number of that category sold (using the `Quantity`) by each `Product_Line` category. Also **change the axis labels** to something meaningful and add a title.

You will likely want to use Seaborn. Make sure you set the result to the variable `ax` like the following:

```
ax = # code to create a bar plot
```

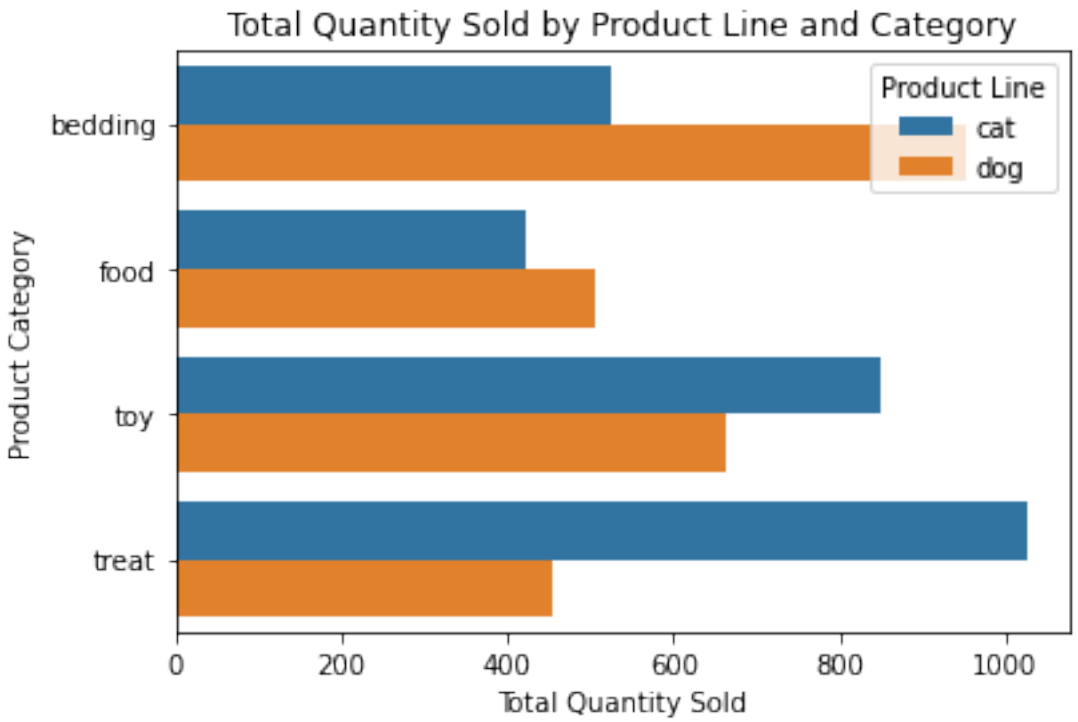
[21]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Summarize total Quantity sold by Product_Line and Product_Category
summary = df.groupby(['Product_Line', 'Product_Category'])['Quantity'].sum().
 ↪reset_index()

# Create the horizontal bar plot
ax = sns.barplot(
    data=summary,
    x='Quantity',           # numerical axis
    y='Product_Category',   # category axis (horizontal bars)
    hue='Product_Line',     # color-coded by product line (cat/dog)
    orient='h'
)

# Add labels and title
ax.set_xlabel('Total Quantity Sold')
ax.set_ylabel('Product Category')
ax.set_title('Total Quantity Sold by Product Line and Category')
plt.legend(title='Product Line')
```

[21]:
```
<matplotlib.legend.Legend at 0x7ddee54c66d0>
```




[22]:
```
# Question 8 Grading Checks

assert isinstance(ax, plt.Axes), 'Did you assign the plot result to the␣
↪variable ax?'
```

**Question 9** Based on the plot from Section 1.5, what would you conclude for your stakeholders about what products they should sell? What would be the considerations and/or caveats you'd communicate to your stakeholders?

Write at least a couple sentences of your thoughts in a string assigned to the variable `answer_to_9`.

The cell below should look something like this:

```
answer_to_9 = '''
I think that based on the visualization that ****.
Therefore I would communicate with the stakeholders that ****
'''
```

[23]:
```
# Your code here
answer_to_9 = '''
Based on the visualization, it appears that certain product categories within␣
↪the Dog product line have higher total sales quantities than those in the␣
↪Cat product line.
```

```
This suggests that Dog-related products might be more popular or in higher␣
 ↪demand overall.
However, before making any business decisions, I would advise stakeholders to␣
 ↪also consider factors like profit margins, seasonal effects, and stock␣
 ↪availability.
Focusing on the best-performing categories from each line could help optimize␣
 ↪sales while maintaining a balanced product offering.
'''

print(len(answer_to_9))
```

```
559
```

[24]:
```
# Question 9 Grading Checks

assert isinstance(answer_to_9, str), 'Make sure you create a string for your␣
 ↪answer.'
```

**Question 10** The plot you created for Section 1.5 is good but could be modified to emphasize which products are important for the business.

Create an explanatory visualization that emphasizes the insight you about the product category. This would be a visualization you'd share with the business stakeholders.

Make sure you set the result to the variable `ax` like the following:

```
ax = # code to create explanatory visualization
```

[25]:
```
# Create an explanatory bar plot emphasizing key product categories

import seaborn as sns
import matplotlib.pyplot as plt

# Summarize total Quantity sold by Product_Line and Product_Category
summary = df.groupby(['Product_Line', 'Product_Category'])['Quantity'].sum().
 ↪reset_index()

# Sort categories by total quantity for better visual flow
summary = summary.sort_values('Quantity', ascending=True)

# Create the horizontal bar plot
ax = sns.barplot(
    data=summary,
    x='Quantity',
    y='Product_Category',
    hue='Product_Line',
    orient='h',
    palette='muted'
```

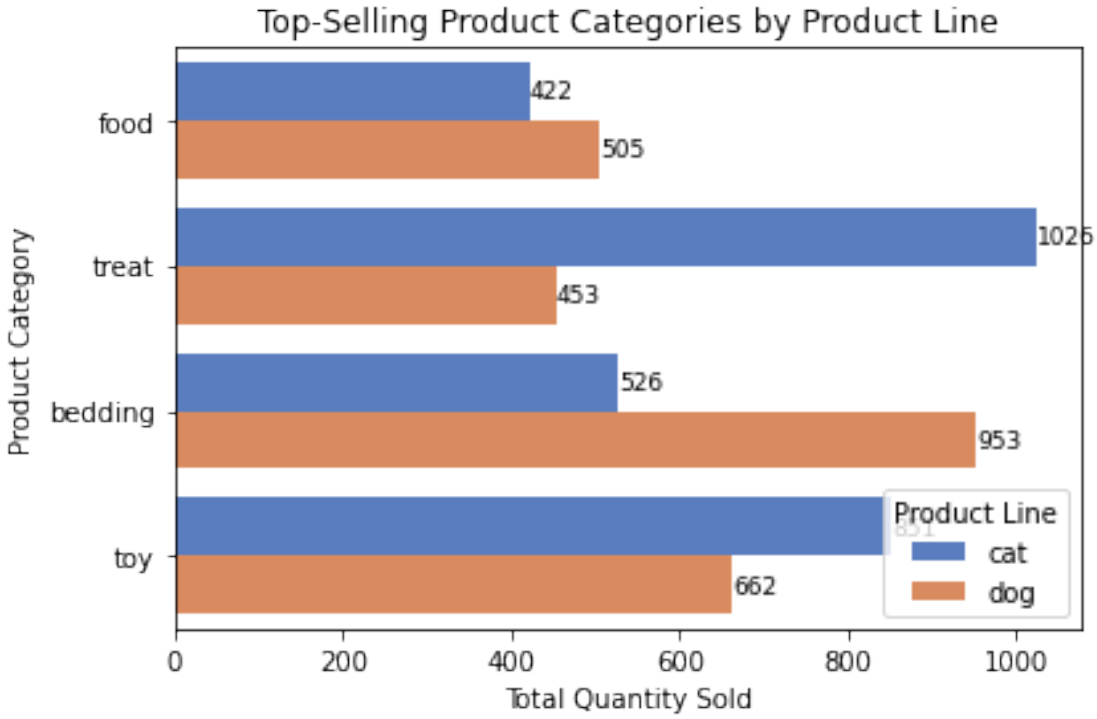
```
)

# Add labels and title
ax.set_xlabel('Total Quantity Sold')
ax.set_ylabel('Product Category')
ax.set_title('Top-Selling Product Categories by Product Line')

# Add annotations to highlight important insights
for container in ax.containers:
    ax.bar_label(container, fmt='%.0f', label_type='edge', fontsize=9)

# Move legend to a more visible position
plt.legend(title='Product Line', loc='lower right')
plt.tight_layout()# Your code here

# your code here
```




[26]:
```
# Question 10 Grading Checks

assert isinstance(ax, plt.Axes), 'Did you assign the plot result to the␣
↪variable ax?'
```

[ ]:
```

```