In [25]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Load the file and find the Central tendencies.

In [ ]:
```
# Load the data
df = pd.read_csv('Final.csv')

df['Perishable'] = df['Perishable'].astype(object)
df['Id'] = df['Id'].astype(object)
df['Store Nbr'] = df['Store Nbr'].astype(object)
df['Transferred'] = df['Transferred'].astype(object)
df['Class'] = df['Class'].astype(object)
df['Cluster'] = df['Cluster'].astype(object)
df['Onpromotion'] = df['Onpromotion'].astype(object)

df.drop('Unnamed: 0',axis=1,inplace=True)
```

In [26]:
```
:ategorical = ['Id','Type','Family','Locale','Onpromotion','Perishable'
```

In [27]:
```
relevant = ['Type','Locale','Dcoilwtico','Unit Sales','Transactions']
```

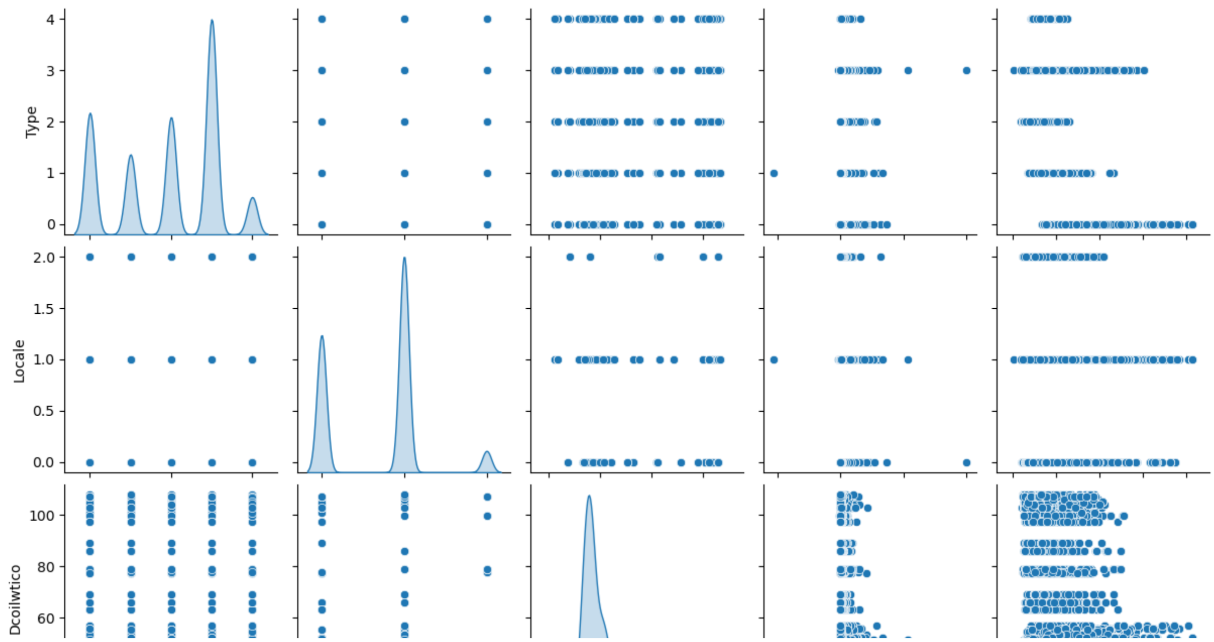
In [28]:
```
print(df.info())
print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 16 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   Date           100000 non-null  object 
 1   Id             100000 non-null  object 
 2   Type           100000 non-null  object 
 3   Family         100000 non-null  object 
 4   Locale         100000 non-null  object 
 5   Locale Name    100000 non-null  object 
 6   State          100000 non-null  object 
 7   Store Nbr      100000 non-null  object 
 8   Transferred    100000 non-null  object 
 9   Class          100000 non-null  object 
 10  Cluster        100000 non-null  object 
 11  Onpromotion    100000 non-null  object 
 12  Perishable     100000 non-null  object 
 13  Dcoilwtico     100000 non-null  float64
 14  Unit Sales     100000 non-null  float64
 15  Transactions   100000 non-null  int64  
dtypes: float64(2), int64(1), object(13)
memory usage: 12.2+ MB
None
          Dcoilwtico     Unit Sales   Transactions
count  100000.000000  100000.000000  100000.000000
mean       58.417996       8.554111    1929.809350
std        22.757038      19.173872    1136.648179
min        27.960000   -1053.000000      54.000000
25%        43.770000       2.000000    1151.000000
50%        46.720000       4.000000    1586.000000
75%        65.940000       9.000000    2468.000000
max       107.950000    2001.000000    8307.000000
```

Scatter Plot

In [35]:
```python
# Scatter plots
plt.figure(figsize=(12, 8))
sns.pairplot(df, diag_kind='kde')
plt.show()
```

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



CLT and Q-Q plots

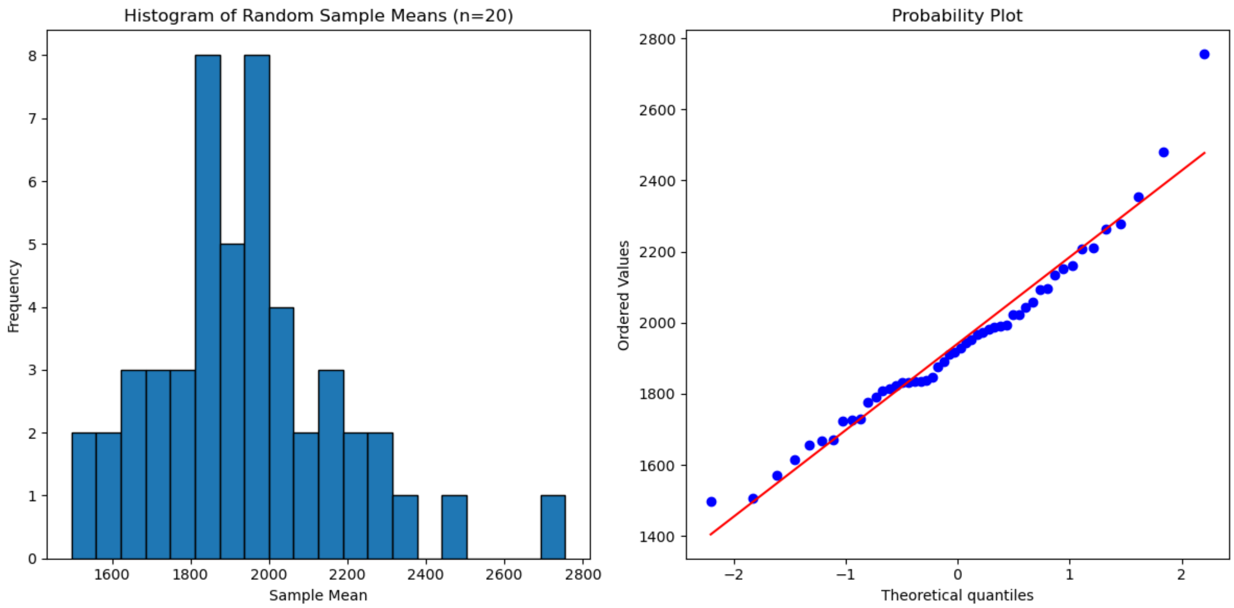
In [4]:
```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_norm = df.copy()
df_norm[['Unit Sales', 'Dcoilwtico', 'Transactions']] = scaler.fit_tra
```

In [47]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Load the data

# 1) Calculate the mean and standard deviation of the population
mu_p = df['Transactions'].mean()
sigma_p = df['Transactions'].std()
print(f'Population mean (μ_p): {mu_p:.4f}')
print(f'Population standard deviation (σ_p): {sigma_p:.4f}')
```

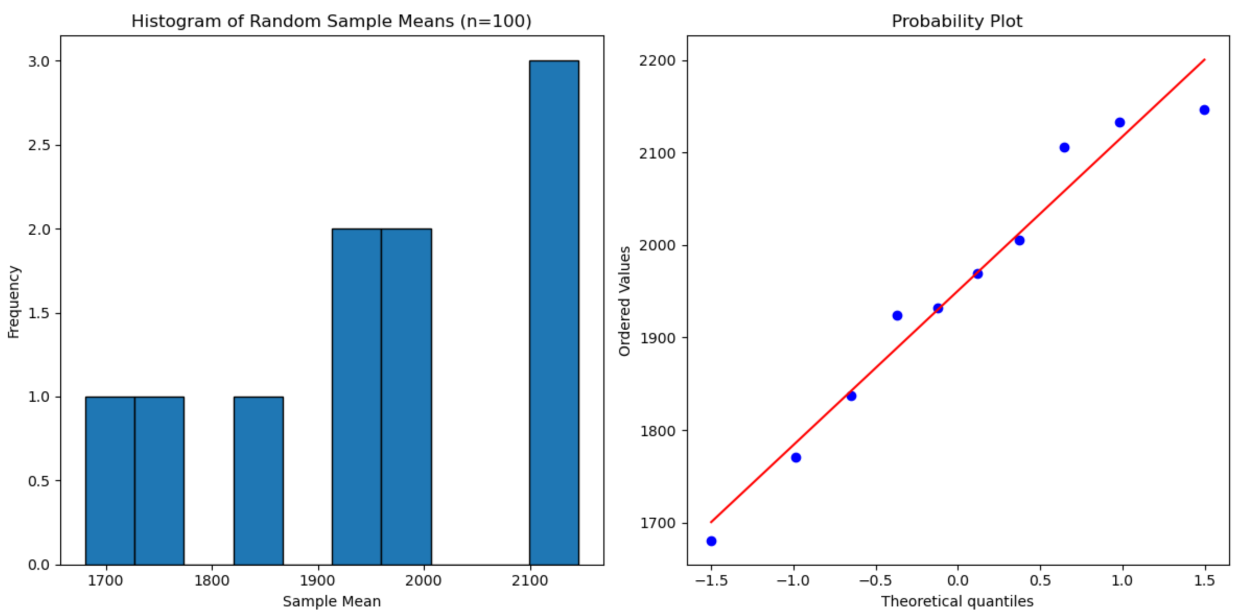
```
# Generate 50 samples of size 20 with replacement
sample_means = []
for _ in range(50):
    sample = df['Transactions'].sample(20, replace=True)
    sample_mean = sample.mean()
    sample_means.append(sample_mean)

# Calculate the mean and standard deviation of the sample means
mu_s = np.mean(sample_means)
sigma_s = np.std(sample_means)

# Draw a histogram and QQ-plot of the sample means (n=20)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.hist(sample_means, bins=20, edgecolor='black')
ax1.set_xlabel('Sample Mean')
ax1.set_ylabel('Frequency')
ax1.set_title('Histogram of Random Sample Means (n=20)')

ax2.set_title("QQ-plot of Random Sample Means (n=20)")
stats.probplot(sample_means, dist="norm", plot=ax2)

plt.tight_layout()
plt.show()

print(f'Sample mean (μ_s): {mu_s:.4f}')
print(f'Sample standard deviation (σ_s): {sigma_s:.4f}')
print(f'Expected sample standard deviation (σ_p / √n): {sigma_p / np.s

# Generate 10 samples of size 100 with replacement
sample_means_100 = []
for _ in range(10):
    sample = df['Transactions'].sample(100, replace=True)
    sample_mean = sample.mean()
    sample_means_100.append(sample_mean)

# Calculate the mean and standard deviation of the sample means
mu_x_100 = np.mean(sample_means_100)
sigma_x_100 = np.std(sample_means_100)

# Draw a histogram and QQ-plot of the sample means (n=100)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.hist(sample_means_100, bins=10, edgecolor='black')
ax1.set_xlabel('Sample Mean')
ax1.set_ylabel('Frequency')
ax1.set_title('Histogram of Random Sample Means (n=100)')

ax2.set_title("QQ-plot of Random Sample Means (n=100)")
```

```
stats.probplot(sample_means_100, dist="norm", plot=ax2)

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

```
Population mean (μ_p): 1929.8093
Population standard deviation (σ_p): 1136.6482
```




```
Sample mean (μ_s): 1941.0160
Sample standard deviation (σ_s): 240.0637
Expected sample standard deviation (σ_p / √n): 254.1623
```

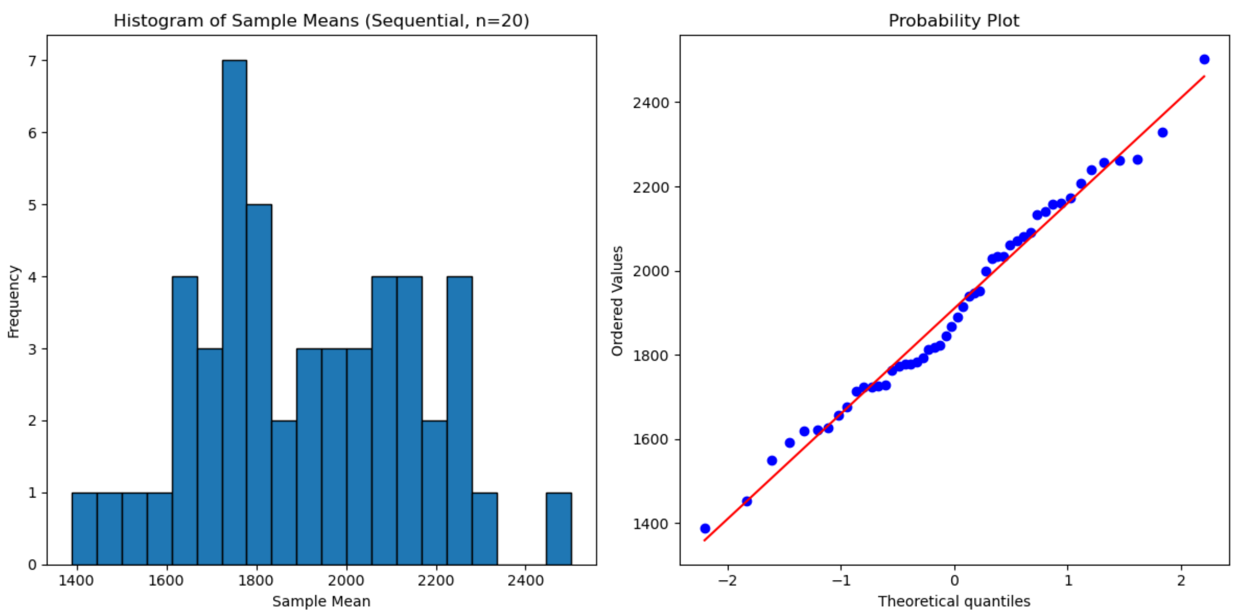


In [48]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

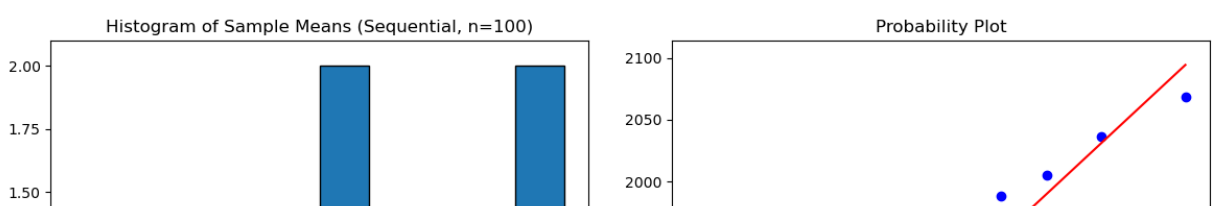
```
import scipy.stats as stats

# Load the data

mu_p = df['Transactions'].mean()
sigma_p = df['Transactions'].std()
print(f'Population mean (μ_p): {mu_p:.4f}')
print(f'Population standard deviation (σ_p): {sigma_p:.4f}')

# Generate 50 sequential samples of size 20
sample_means = []
for i in range(50):
    start = i * 20
    end = start + 20
    sample = df['Transactions'].iloc[start:end]
    sample_mean = sample.mean()
    sample_means.append(sample_mean)

# Calculate the mean and standard deviation of the sample means
mu_s = np.mean(sample_means)
sigma_s = np.std(sample_means)

# Draw a histogram and QQ-plot of the sample means (sequential, n=20)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.hist(sample_means, bins=20, edgecolor='black')
ax1.set_xlabel('Sample Mean')
ax1.set_ylabel('Frequency')
ax1.set_title('Histogram of Sample Means (Sequential, n=20)')

ax2.set_title("QQ-plot of Sample Means (Sequential, n=20)")
stats.probplot(sample_means, dist="norm", plot=ax2)

plt.tight_layout()
plt.show()

print(f'Sample mean (μ_s): {mu_s:.4f}')
print(f'Sample standard deviation (σ_s): {sigma_s:.4f}')
print(f'Expected sample standard deviation (σ_p / √n): {sigma_p / np.s

# Generate 10 sequential samples of size 100
sample_means_100 = []
for i in range(10):
    start = i * 100
    end = start + 100
    sample = df['Transactions'].iloc[start:end]
    sample_mean = sample.mean()
    sample_means_100.append(sample_mean)

# Calculate the mean and standard deviation of the sample means
```

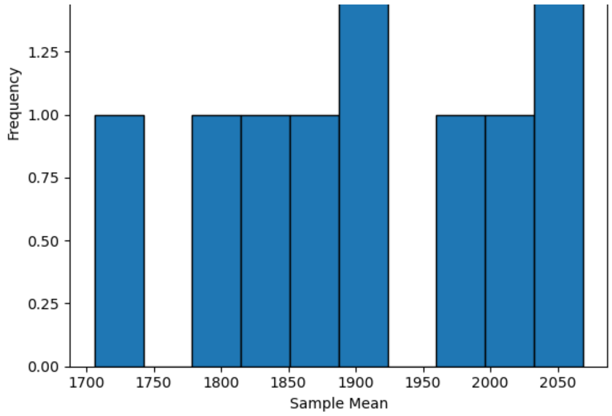
```
mu_x_100 = np.mean(sample_means_100)
sigma_x_100 = np.std(sample_means_100)

# Draw a histogram and QQ-plot of the sample means (sequential, n=100)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.hist(sample_means_100, bins=10, edgecolor='black')
ax1.set_xlabel('Sample Mean')
ax1.set_ylabel('Frequency')
ax1.set_title('Histogram of Sample Means (Sequential, n=100)')

ax2.set_title("QQ-plot of Sample Means (Sequential, n=100)")
stats.probplot(sample_means_100, dist="norm", plot=ax2)

plt.tight_layout()
plt.show()

print(f'Sample mean (μ_s): {mu_x:.4f}')
print(f'Sample standard deviation (σ_s): {sigma_x:.4f}')
print(f'Expected sample standard deviation (σ_p / √n): {sigma_p / np.s
```

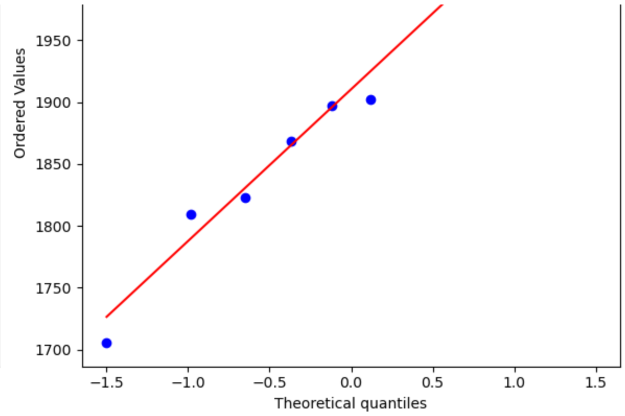
```
Population mean (μ_p): 1929.8093
Population standard deviation (σ_p): 1136.6482
```




```
Sample mean (μ_s): 1910.4170
Sample standard deviation (σ_s): 242.7158
Expected sample standard deviation (σ_p / √n): 254.1623
```








```
Sample mean (μ_s): 1885.2730
Sample standard deviation (σ_s): 222.1012
Expected sample standard deviation (σ_p / √n): 113.6648
```

In [ ]:

ProjectFile_Harsh

April 30, 2024

[31]:
```
import time
import pandas as pd
import numpy as np
import dask.dataframe as dd
import matplotlib.pyplot as plt
from functools import wraps
from sklearn import linear_model
import scipy.stats as stats
from scipy.stats import shapiro
from scipy.stats import ttest_1samp
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import shapiro, kstest
from scipy.stats import probplot
from sklearn.preprocessing import power_transform
import seaborn as sns
```

[21]:
```
# file_name = "train_combined_14_n_16.csv"
# file_name = "random_sampled.csv"
file_name = "Final.csv"
df = pd.read_csv(file_name)
list_of_column_names = list(df.columns)
```

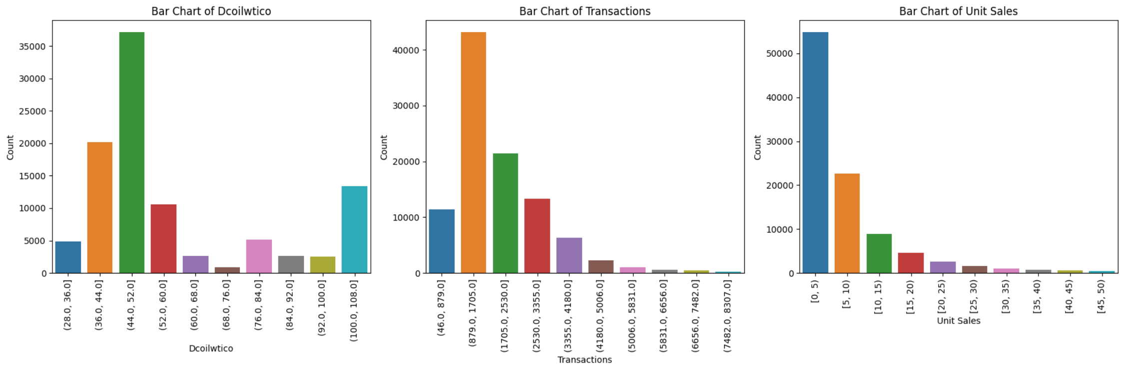
[22]:
```
df
```

[22]:
```
       Unnamed: 0       Date        Id Type                   Family  \
0          784051  11/6/2014  34644254    B                GROCERY I
1          493024  5/24/2016  80164020    C                GROCERY I
2          421524  8/24/2016  88922696    B                GROCERY I
3          171897   4/1/2016  75014142    E  PLAYERS AND ELECTRONICS
4          106636  4/18/2016  76613817    D                 CLEANING
…            …         …        …  …                      …
99995      592299  12/5/2016  98973041    C                 CLEANING
99996      645833  6/20/2014  25677077    B                 CLEANING
99997      329344   5/3/2016  78149260    D                 CLEANING
99998       26631  12/1/2014  36394322    C                    DAIRY
99999      265427  4/20/2016  76869186    E            PERSONAL CARE
```

```
          Locale                     Locale Name      State  Store Nbr  \
0       Regional  Santo Domingo de los Tsachilas   Los Rios         31
1       National                         Ecuador     Manabi         54
2          Local                          Ambato  Pichincha          9
3       Regional                        Cotopaxi     Guayas         28
4       National                         Ecuador  Pichincha          1
…            …                               …          …          …
99995      Local                           Quito  Pichincha         17
99996   National                         Ecuador  Pichincha         18
99997   National                         Ecuador     Manabi         53
99998   National                         Ecuador  Pichincha         17
99999   National                         Ecuador     Guayas         36

       Transferred  Class  Cluster  Onpromotion  Perishable  Dcoilwtico  \
0            False   1022       10        False           0       77.87
1             True   1034        3         True           0       48.04
2            False   1056        6        False           0       46.29
3            False   5446       10        False           0       35.36
4            False   3038       13        False           0       39.74
…              …     …       …            …           …          …
99995        False   3032       12        False           0       51.72
99996        False   3034       16        False           0      107.95
99997        False   3024       13         True           0       43.65
99998        False   2116       12        False           1       68.98
99999        False   4114       10        False           0       42.72

       Unit Sales  Transactions
0            14.0          1268
1             2.0           816
2             2.0          1744
3             2.0          1238
4             5.0          2133
…             …            …
99995         2.0          1518
99996         1.0          1290
99997         2.0          1505
99998         1.0          1443
99999        11.0          1271

[100000 rows x 17 columns]
```

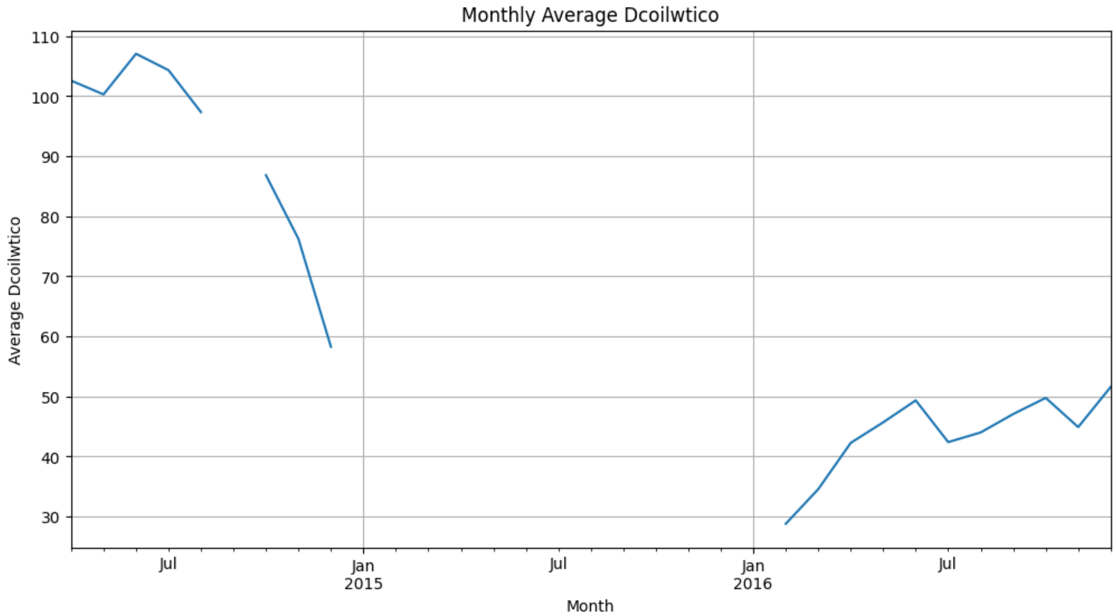
0.1 Bar Graph

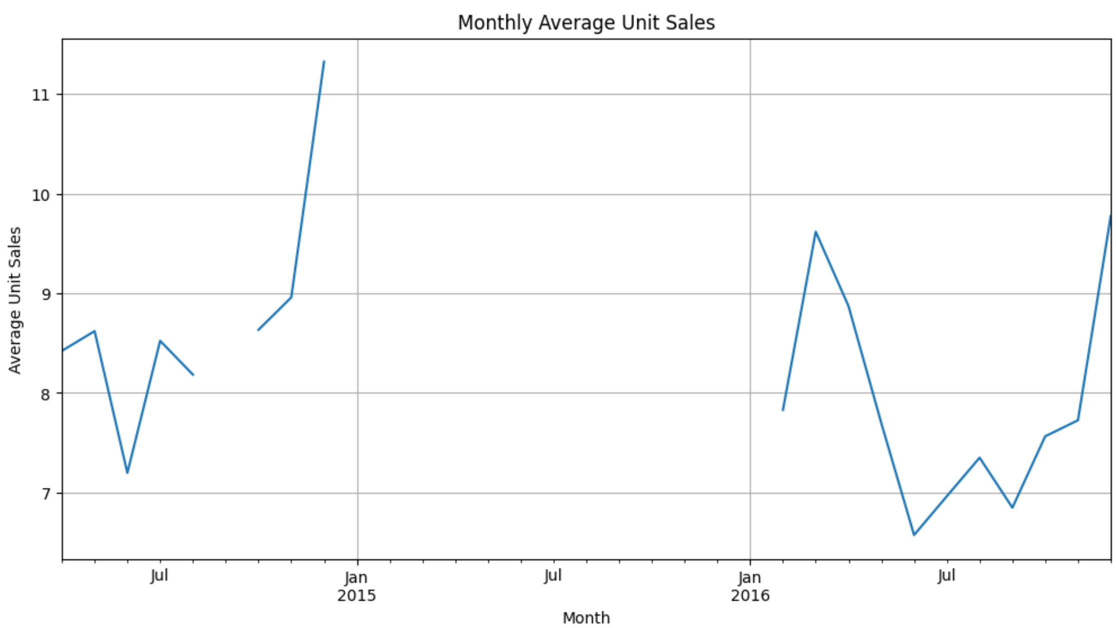
[32]:
```
data = pd.read_csv('Final.csv')

# Convert 'Date' column to datetime type
data['Date'] = pd.to_datetime(data['Date'])

# Define bin sizes for Dcoilwtico and Transactions
bin_sizes = {
    'Dcoilwtico': 10,
    'Transactions': 10
}

# Create bins for Dcoilwtico and Transactions
for col in ['Dcoilwtico', 'Transactions']:
    data[f"{col}_binned"] = pd.cut(data[col], bins=bin_sizes[col], precision=0)

# Adjust bin ranges for 'Unit Sales' to better focus on the common range of␣
↪sales
bin_ranges = pd.interval_range(start=0, end=50, freq=5, closed='left')

# Re-bin the 'Unit Sales' with the new focused range
data['Unit Sales_binned'] = pd.cut(data['Unit Sales'], bins=bin_ranges)

# Create subplots
fig, axs = plt.subplots(1, 3, figsize=(18, 6))
axs = axs.flatten()

# Plotting the binned data
for i, col in enumerate(['Dcoilwtico', 'Transactions', 'Unit Sales']):
    # Extract and set x-axis labels
    labels = [str(interval) for interval in data[f"{col}_binned"].cat.categories]
    sns.countplot(data=data, x=f"{col}_binned", ax=axs[i])
    axs[i].set_xticklabels(labels)
    axs[i].set_xlabel(col)  # Set x-axis label to just the column name
    axs[i].set_title(f'Bar Chart of {col}')
    axs[i].set_ylabel('Count')
    axs[i].tick_params(axis='x', rotation=90)  # Rotate x labels for better␣
↪readability

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

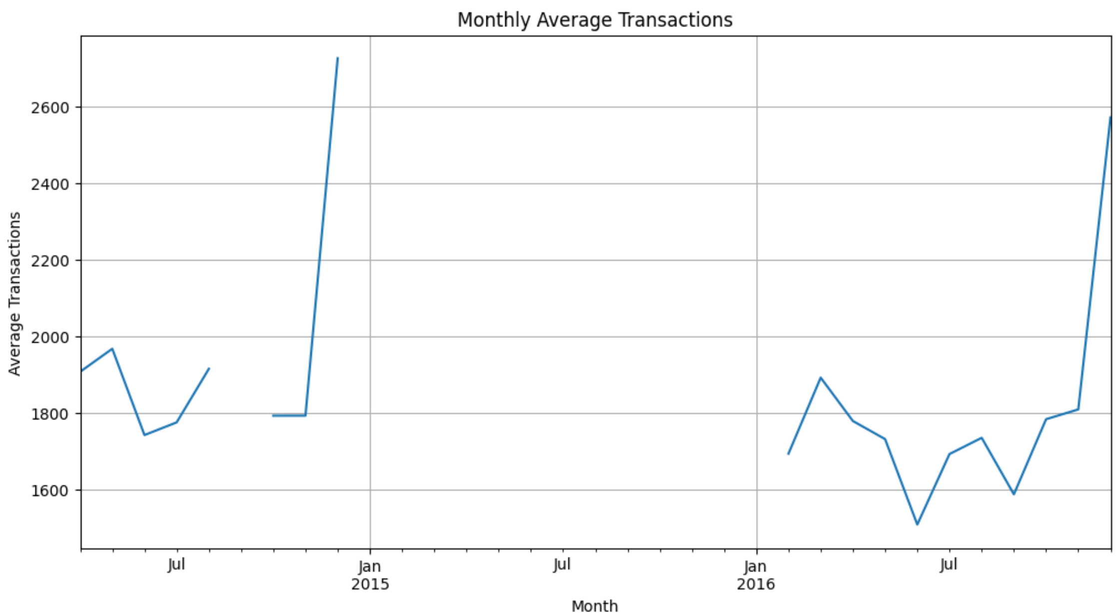


0.2 Time Series Plot

[23]:
```python
df['Date'] = pd.to_datetime(df['Date'])

df.set_index('Date', inplace=True)

unit_sale = df['Unit Sales'].resample('M').mean()
dcoil = df['Dcoilwtico'].resample('M').mean()
transactions = df['Transactions'].resample('M').mean()

plt.figure(figsize=(12, 6))
dcoil.plot(title='Monthly Average Dcoilwtico')
plt.xlabel('Month')
plt.ylabel('Average Dcoilwtico')
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
unit_sale.plot(title='Monthly Average Unit Sales')
plt.xlabel('Month')
plt.ylabel('Average Unit Sales')
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
transactions.plot(title='Monthly Average Transactions')
plt.xlabel('Month')
plt.ylabel('Average Transactions')
plt.grid(True)
plt.show()
```


Monthly Average Dcoilwtico
110
100
90
80
70
60
50
40
30
Average Dcoilwtico
Jul
Jan
2015
Jul
Jan
2016
Jul
Month


Monthly Average Unit Sales
11
10
9
8
7
Average Unit Sales
Jul
Jan
2015
Jul
Jan
2016
Jul
Month



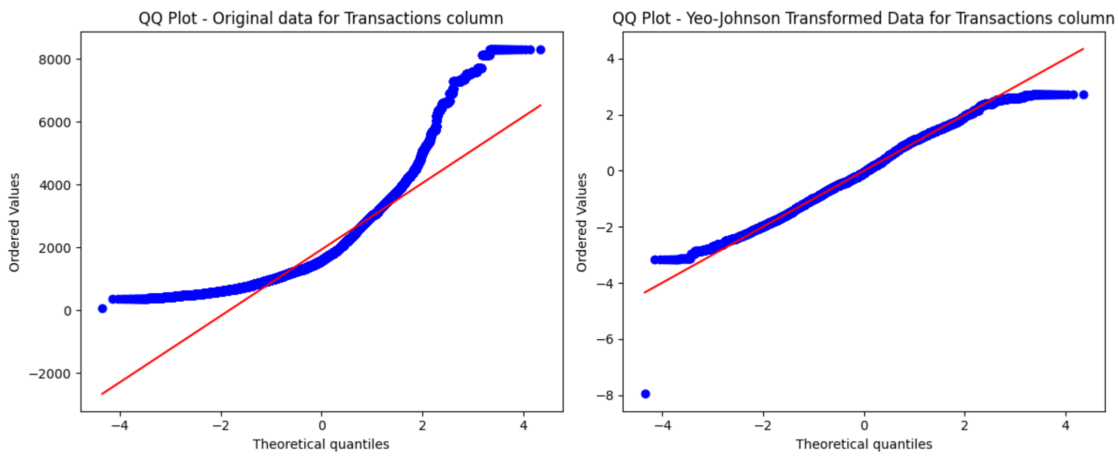
0.3 Q-Q Plot

[24]:
```
original_data = df['Transactions']



yj_transformed = power_transform(original_data.values.reshape(-1, 1),␣
↪method='yeo-johnson')
yj_transformed = pd.Series(yj_transformed.flatten())

print('Original data for Transactions column:')
_, p_value_original = shapiro(original_data)
print(f'Shapiro-Wilk p-value: {p_value_original:.4f}')

_, p_value_ks_original = kstest(original_data, 'norm')
print(f'Kolmogorov-Smirnov p-value: {p_value_ks_original:.4f}')

print('\nYeo-Johnson transformed data for Transactions column:')
_, p_value_yj = shapiro(yj_transformed)
print(f'Shapiro-Wilk p-value: {p_value_yj:.4f}')

_, p_value_ks_yj = kstest(yj_transformed, 'norm')
print(f'Kolmogorov-Smirnov p-value: {p_value_ks_yj:.4f}')

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

probplot(original_data, plot=ax1)
```

```
ax1.set_title('QQ Plot - Original data for Transactions column')

probplot(yj_transformed, plot=ax2)
ax2.set_title('QQ Plot - Yeo-Johnson Transformed Data for Transactions column')

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

```
Original data for Transactions column:
Shapiro-Wilk p-value: 0.0000
Kolmogorov-Smirnov p-value: 0.0000

Yeo-Johnson transformed data for Transactions column:
Shapiro-Wilk p-value: 0.0000
Kolmogorov-Smirnov p-value: 0.0000
```

```
/Users/harshshah/anaconda3/lib/python3.11/site-
packages/scipy/stats/_morestats.py:1882: UserWarning: p-value may not be
accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

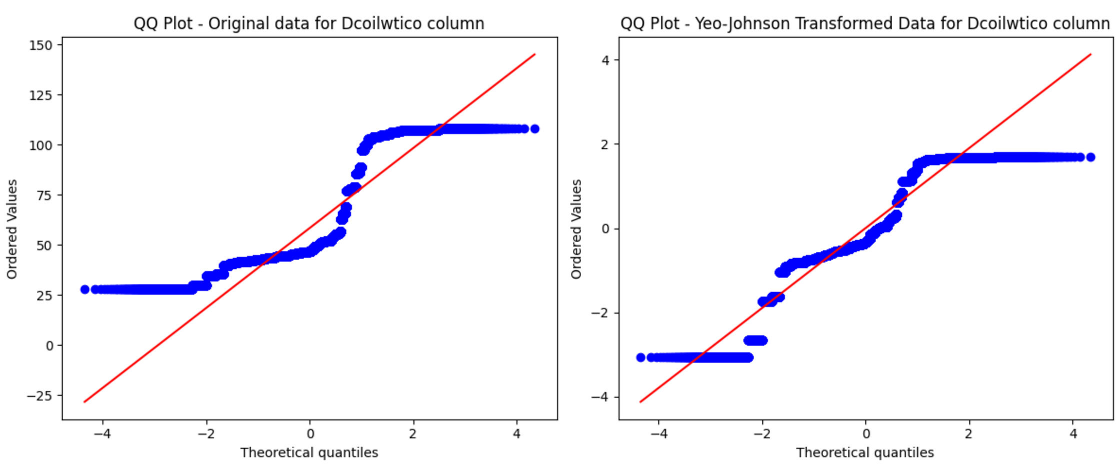


[25]:
```
original_data = df['Dcoilwtico']


yj_transformed = power_transform(original_data.values.reshape(-1, 1),
  method='yeo-johnson')
yj_transformed = pd.Series(yj_transformed.flatten())

print('Original data for Dcoilwtico column:')
_, p_value_original = shapiro(original_data)
print(f'Shapiro-Wilk p-value: {p_value_original:.4f}')
```

```
_, p_value_ks_original = kstest(original_data, 'norm')
print(f'Kolmogorov-Smirnov p-value: {p_value_ks_original:.4f}')

print('\nYeo-Johnson transformed data for Dcoilwtico column:')
_, p_value_yj = shapiro(yj_transformed)
print(f'Shapiro-Wilk p-value: {p_value_yj:.4f}')

_, p_value_ks_yj = kstest(yj_transformed, 'norm')
print(f'Kolmogorov-Smirnov p-value: {p_value_ks_yj:.4f}')

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

probplot(original_data, plot=ax1)
ax1.set_title('QQ Plot - Original data for Dcoilwtico column')

probplot(yj_transformed, plot=ax2)
ax2.set_title('QQ Plot - Yeo-Johnson Transformed Data for Dcoilwtico column')

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

```
Original data for Dcoilwtico column:
Shapiro-Wilk p-value: 0.0000
Kolmogorov-Smirnov p-value: 0.0000

Yeo-Johnson transformed data for Dcoilwtico column:
Shapiro-Wilk p-value: 0.0000
Kolmogorov-Smirnov p-value: 0.0000
```

```
/Users/harshshah/anaconda3/lib/python3.11/site-
packages/scipy/stats/_morestats.py:1882: UserWarning: p-value may not be
accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

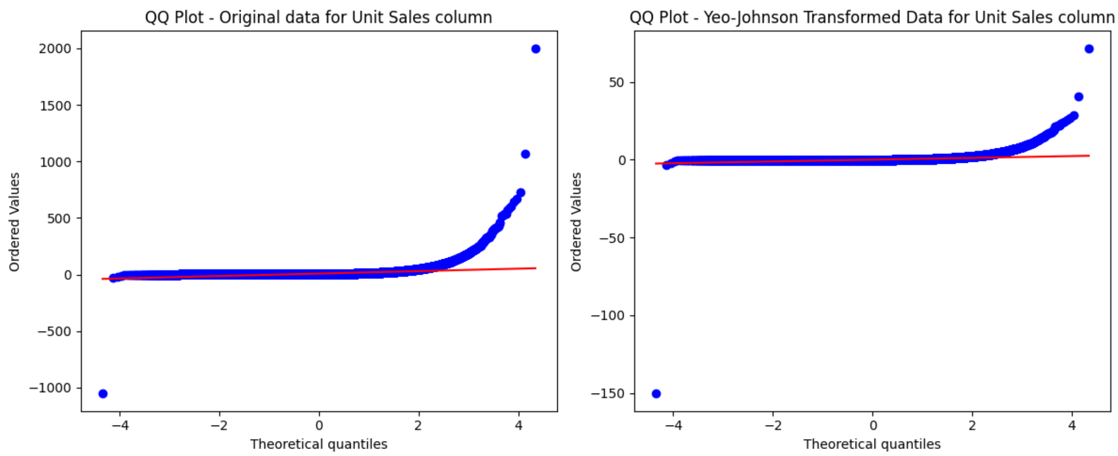


[26]:

```
original_data = df['Unit Sales']


yj_transformed = power_transform(original_data.values.reshape(-1, 1),␣
 ↪method='yeo-johnson')
yj_transformed = pd.Series(yj_transformed.flatten())

print('Original data for Unit Sales column:')
_, p_value_original = shapiro(original_data)
print(f'Shapiro-Wilk p-value: {p_value_original:.4f}')

_, p_value_ks_original = kstest(original_data, 'norm')
print(f'Kolmogorov-Smirnov p-value: {p_value_ks_original:.4f}')

print('\nYeo-Johnson transformed data for Unit Sales column:')
_, p_value_yj = shapiro(yj_transformed)
print(f'Shapiro-Wilk p-value: {p_value_yj:.4f}')

_, p_value_ks_yj = kstest(yj_transformed, 'norm')
print(f'Kolmogorov-Smirnov p-value: {p_value_ks_yj:.4f}')

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

probplot(original_data, plot=ax1)
ax1.set_title('QQ Plot - Original data for Unit Sales column')

probplot(yj_transformed, plot=ax2)
ax2.set_title('QQ Plot - Yeo-Johnson Transformed Data for Unit Sales column')

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

```
Original data for Unit Sales column:
Shapiro-Wilk p-value: 0.0000
Kolmogorov-Smirnov p-value: 0.0000

Yeo-Johnson transformed data for Unit Sales column:
Shapiro-Wilk p-value: 0.0000
Kolmogorov-Smirnov p-value: 0.0000
```

```
/Users/harshshah/anaconda3/lib/python3.11/site-
packages/scipy/stats/_morestats.py:1882: UserWarning: p-value may not be
accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```



0.4 CI for CLT

[18]:
```
from scipy.stats import norm

transactions = df['Transactions']
df = pd.DataFrame(transactions.sample(n=1000, replace=True))

mu_x = df['Transactions'].mean()
std_x = df['Transactions'].std()

sample_size_100 = 100
sample_100 = df['Transactions'].sample(n=sample_size_100, replace=True)

# Calculating the sample mean and standard deviation
mean_100 = sample_100.mean()
std_100 = sample_100.std()

# Calculating the margin of error for 95% confidence
z_score = norm.ppf(0.975)  # For a two-tailed test
margin_of_error_100 = z_score * (std_100 / np.sqrt(sample_size_100))

# Constructing the confidence interval
conf_interval_100 = (mean_100 - margin_of_error_100, mean_100 +
 ↪margin_of_error_100)

print(f"95% Confidence Interval for Transaction sample size 100:
 ↪{conf_interval_100}")
print(f"Sample population mean for Transaction sample size 100: {mean_100}")
print(f"True population mean for Transaction sample size 100: {mu_x}")
print(f"Sample Standard Deviation for Transaction sample size 100: {std_100}")
print(f"True Standard Deviation for Transaction sample size 100: {std_x}")
```

```
95% Confidence Interval for Transaction sample size 100: (1529.2739435183519,
1872.746056481648)
Sample population mean for Transaction sample size 100: 1701.01
True population mean for Transaction sample size 100: 1903.549
Sample Standard Deviation for Transaction sample size 100: 876.2204705610928
True Standard Deviation for Transaction sample size 100: 1058.587175629578
```

[19]:
```
sample_size_20 = 20
sample_20 = df['Transactions'].sample(n=sample_size_20, replace=True)

mean_20 = sample_20.mean()
std_20 = sample_20.std()

margin_of_error_20 = z_score * (std_20 / np.sqrt(sample_size_20))

conf_interval_20 = (mean_20 - margin_of_error_20, mean_20 + margin_of_error_20)

print(f"95% Confidence Interval for Transaction sample size 20:␣
↪{conf_interval_20}")
print(f"Sample population mean for Transaction sample size 20: {mean_20}")
print(f"True population mean for Transaction sample size 100: {mu_x}")
print(f"Standard Deviation for Transaction sample size 20: {std_20}")
print(f"True Standard Deviation for Transaction sample size 100: {std_x}")
```

```
95% Confidence Interval for Transaction sample size 20: (1069.138077909557,
2381.961922090443)
Sample population mean for Transaction sample size 20: 1725.55
True population mean for Transaction sample size 100: 1903.549
Standard Deviation for Transaction sample size 20: 1497.7639289427773
True Standard Deviation for Transaction sample size 100: 1058.587175629578
```

0.5 Hypothesis Testing

[29]:
```
from scipy.stats import ttest_1samp, chi2
import numpy as np

samples_100 = [np.random.choice(df['Transactions'], size=100, replace=True) for␣
↪_ in range(10)]

samples_20 = [np.random.choice(df['Transactions'], size=20, replace=True) for _␣
↪in range(50)]

sample_100 = samples_100[0]
sample_20 = samples_20[0]
# 1)
t_statistic, p_value = ttest_1samp(sample_100, 1900)
print(f"Test 1: Sample size 100: t-statistic = {t_statistic}, p-value =␣
↪{p_value}")
```

```
# 2)
t_statistic, p_value = ttest_1samp(sample_20, 1900)
print(f"Test 2: Sample size 20: t-statistic = {t_statistic}, p-value = 
 ↪{p_value}")

# 3)
sample_mean = np.mean(sample_20)
sample_std = np.std(sample_20, ddof=1)
n = len(sample_20)
chi_square_statistic = (n - 1) * sample_std**2 / 1100**2
p_value_two_tailed = chi2.sf(chi_square_statistic, n-1) * 2
print(f"Test 3: Chi-square statistic = {chi_square_statistic}, p-value = 
 ↪{p_value_two_tailed}")

# 4)
p_value_one_tailed = chi2.sf(chi_square_statistic, n-1)
print(f"Test 4: Chi-square statistic = {chi_square_statistic}, p-value = 
 ↪{p_value_one_tailed}")
```

```
Test 1: Sample size 100: t-statistic = 0.5566546216626699, p-value =
0.5790195976480433
Test 2: Sample size 20: t-statistic = 0.3257668960177601, p-value =
0.7481611868996239
Test 3: Chi-square statistic = 15.986706611570249, p-value = 1.3163295368537122
Test 4: Chi-square statistic = 15.986706611570249, p-value = 0.6581647684268561
```

[ ]:

```

```

Final Model Evaluation and comaparison (FIN)

April 30, 2024

[1]:
```python
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

0.0.1 Loading file and type casting

[2]:
```python
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Load the dataset
df = pd.read_csv('Final.csv')

df['Perishable'] = df['Perishable'].astype(object)
df['Id'] = df['Id'].astype(object)
df['Store Nbr'] = df['Store Nbr'].astype(object)
df['Transferred'] = df['Transferred'].astype(object)
df['Class'] = df['Class'].astype(object)
df['Cluster'] = df['Cluster'].astype(object)
df['Onpromotion'] = df['Onpromotion'].astype(object)
```

[3]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 17 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Unnamed: 0    100000 non-null  int64
 1   Date          100000 non-null  object
 2   Id            100000 non-null  object
 3   Type          100000 non-null  object
 4   Family        100000 non-null  object
 5   Locale        100000 non-null  object
 6   Locale Name   100000 non-null  object
 7   State         100000 non-null  object
 8   Store Nbr     100000 non-null  object
 9   Transferred   100000 non-null  object
 10  Class         100000 non-null  object
```

```
 11  Cluster       100000 non-null  object 
 12  Onpromotion   100000 non-null  object 
 13  Perishable    100000 non-null  object 
 14  Dcoilwtico    100000 non-null  float64
 15  Unit Sales    100000 non-null  float64
 16  Transactions  100000 non-null  int64  
dtypes: float64(2), int64(2), object(13)
memory usage: 13.0+ MB
```

[4]:
```
df.head()
```

[4]:
```
   Unnamed: 0       Date        Id Type                       Family    Locale  \
0      784051  11/6/2014  34644254    B                    GROCERY I  Regional   
1      493024  5/24/2016  80164020    C                    GROCERY I  National   
2      421524  8/24/2016  88922696    B                    GROCERY I     Local   
3      171897   4/1/2016  75014142    E  PLAYERS AND ELECTRONICS  Regional   
4      106636  4/18/2016  76613817    D                     CLEANING  National   

                      Locale Name      State Store Nbr Transferred Class  \
0  Santo Domingo de los Tsachilas   Los Rios        31       False  1022   
1                         Ecuador     Manabi        54        True  1034   
2                          Ambato  Pichincha         9       False  1056   
3                        Cotopaxi     Guayas        28       False  5446   
4                         Ecuador  Pichincha         1       False  3038   

  Cluster Onpromotion Perishable  Dcoilwtico  Unit Sales  Transactions  
0      10       False          0       77.87        14.0          1268  
1       3        True          0       48.04         2.0           816  
2       6       False          0       46.29         2.0          1744  
3      10       False          0       35.36         2.0          1238  
4      13       False          0       39.74         5.0          2133  
```

0.0.2 To use label encoding to convert all categorical features for generating feature importances

[5]:
```
categorical = ['Id','Type','Family','Locale','Onpromotion','Perishable','Locale␣
↪Name','State','Transferred','Class','Cluster']
```

[6]:
```
from sklearn.preprocessing import LabelEncoder

for col in categorical:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
```

[7]:
```
df.head()
```

[7]:
```
   Unnamed: 0       Date     Id  Type  Family  Locale  Locale Name  State  \
0      784051  11/6/2014  19875     1      12       2           23      9
1      493024  5/24/2016  68882     2      12       1            4     10
2      421524  8/24/2016  77456     1      12       0            0     12
3      171897   4/1/2016  37854     4      27       2            2      6
4      106636  4/18/2016  40799     3       7       1            4     12

  Store Nbr  Transferred  Class  Cluster  Onpromotion  Perishable  Dcoilwtico  \
0        31            0     11        9            0           0       77.87
1        54            1     21        2            1           0       48.04
2         9            0     35        5            0           0       46.29
3        28            0    264        9            0           0       35.36
4         1            0    223       12            0           0       39.74

   Unit Sales  Transactions
0        14.0          1268
1         2.0           816
2         2.0          1744
3         2.0          1238
4         5.0          2133
```

[8]:
```python
df.drop('Unnamed: 0',axis=1,inplace=True)
```

[9]:
```python
df.head()
```

[9]:
```
        Date     Id  Type  Family  Locale  Locale Name  State  Store Nbr  \
0  11/6/2014  19875     1      12       2           23      9         31
1  5/24/2016  68882     2      12       1            4     10         54
2  8/24/2016  77456     1      12       0            0     12          9
3   4/1/2016  37854     4      27       2            2      6         28
4  4/18/2016  40799     3       7       1            4     12          1

   Transferred  Class  Cluster  Onpromotion  Perishable  Dcoilwtico  \
0            0     11        9            0           0       77.87
1            1     21        2            1           0       48.04
2            0     35        5            0           0       46.29
3            0    264        9            0           0       35.36
4            0    223       12            0           0       39.74

   Unit Sales  Transactions
0        14.0          1268
1         2.0           816
2         2.0          1744
3         2.0          1238
4         5.0          2133
```

0.0.3 Renaming columns for OLS reports

[10]:
```
# Replace column name in-place
df.rename(columns={'Unit Sales': 'unit_sales'}, inplace=True)
```

[11]:
```
df.rename(columns={'Locale Name': 'locale_name'}, inplace=True)
```

[12]:
```
df.rename(columns={'Store Nbr': 'stor_nbr'}, inplace=True)
```

[13]:
```
df.head()
```

[13]:
```
        Date     Id  Type  Family  Locale  locale_name  State stor_nbr  \
0  11/6/2014  19875     1      12       2           23      9       31
1  5/24/2016  68882     2      12       1            4     10       54
2  8/24/2016  77456     1      12       0            0     12        9
3   4/1/2016  37854     4      27       2            2      6       28
4  4/18/2016  40799     3       7       1            4     12        1

   Transferred  Class  Cluster  Onpromotion  Perishable  Dcoilwtico  \
0            0     11        9            0           0       77.87
1            1     21        2            1           0       48.04
2            0     35        5            0           0       46.29
3            0    264        9            0           0       35.36
4            0    223       12            0           0       39.74

   unit_sales  Transactions
0        14.0          1268
1         2.0           816
2         2.0          1744
3         2.0          1238
4         5.0          2133
```

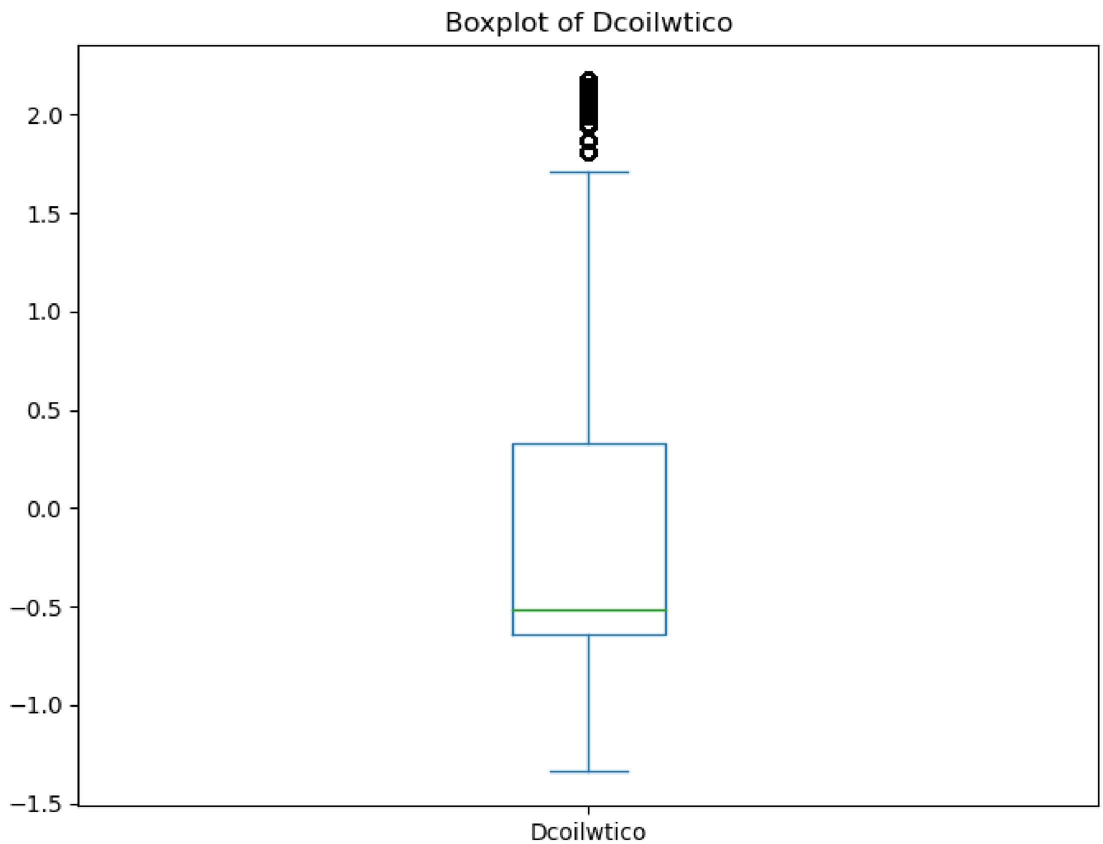
0.0.4 Standard scaling of continuous variables

[14]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
features = ['Dcoilwtico','unit_sales','Transactions']

for i in features:
    df[i] =  scaler.fit_transform(df[i].values.reshape(-1,1))
```

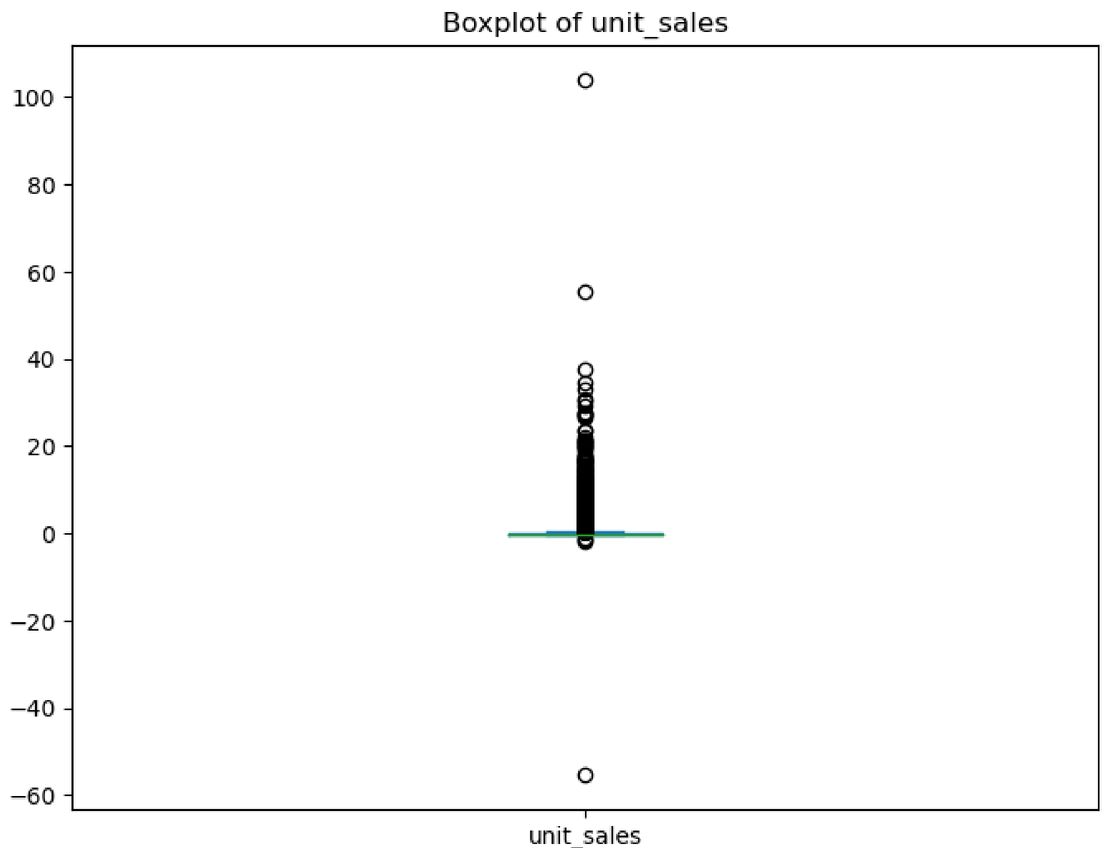
[15]:
```
df_copy = df.copy()
```

0.0.5 Outlier Treatment

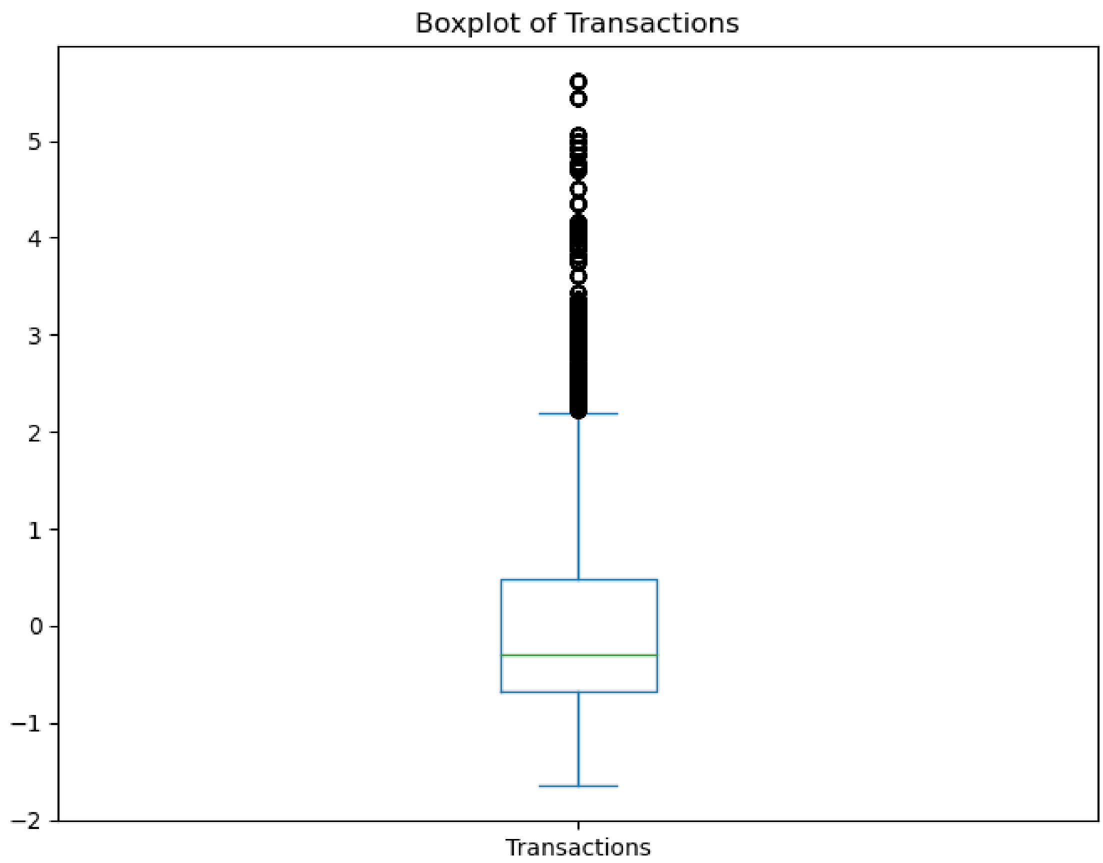
[16]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a DataFrame 'df' with columns 'feature1', 'feature2', ...,␣
↪'featureN'
for column in features:
    # Create a boxplot for the current feature
    plt.figure(figsize=(8, 6))
    df[column].plot(kind='box')
    plt.title(f'Boxplot of {column}')
    plt.show()

    # Detect outliers using the IQR method
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]

    # Print the number of outliers
    print(f'Number of outliers in {column}: {len(outliers)}')

    # Treat the outliers (e.g., remove, replace with median, or apply␣
↪winsorization)
    if len(outliers) > 0:
        # Option 1: Remove the outliers
        #df = df[~df[column].isin(outliers.index)]

        # Option 2: Replace the outliers with the median
        #df.loc[outliers.index, column] = df[column].median()

        # Option 3: Apply winsorization
        df[column] = df[column].clip(lower_bound, upper_bound)
```




```
Number of outliers in Dcoilwtico: 14693
```




```
Number of outliers in unit_sales: 9153
```




```
Number of outliers in Transactions: 3288
```

0.0.6 OLS summary for feature significance

[17]:
```
# Fit the linear regression model
model = ols('Transactions ~ Type + Locale + Dcoilwtico + unit_sales', data=df).
 ↪fit()

# Print the summary of the model
print(model.summary())

# Check the p-values of the features
pvalues = model.pvalues
print('P-values of the features:')
print(pvalues)

# Evaluate the relevance of the features
significance_level = 0.05  # Set the desired significance level (e.g., 5%)

print('Relevant features:')
```

```python
for feature, p_value in pvalues.items():
    if p_value < significance_level:
        print(f'{feature} is relevant (p-value = {p_value:.4f})')
    else:
        print(f'{feature} is not relevant (p-value = {p_value:.4f})')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Transactions   R-squared:                       0.314
Model:                            OLS   Adj. R-squared:                  0.314
Method:                 Least Squares   F-statistic:                 1.147e+04
Date:                Mon, 29 Apr 2024   Prob (F-statistic):               0.00
Time:                        03:52:17   Log-Likelihood:            -1.1009e+05
No. Observations:              100000   AIC:                         2.202e+05
Df Residuals:                   99995   BIC:                         2.202e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.6579      0.005    130.407      0.000       0.648       0.668
Type          -0.3409      0.002   -186.714      0.000      -0.344      -0.337
Locale         0.0503      0.004     12.404      0.000       0.042       0.058
Dcoilwtico    -0.0101      0.003     -4.016      0.000      -0.015      -0.005
unit_sales     0.5671      0.008     74.475      0.000       0.552       0.582
==============================================================================
Omnibus:                     5507.781   Durbin-Watson:                   2.008
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             6466.548
Skew:                           0.620   Prob(JB):                         0.00
Kurtosis:                       3.114   Cond. No.                         8.50
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
P-values of the features:
Intercept     0.000000e+00
Type          0.000000e+00
Locale        2.646640e-35
Dcoilwtico    5.928063e-05
unit_sales    0.000000e+00
dtype: float64
Relevant features:
Intercept is relevant (p-value = 0.0000)
Type is relevant (p-value = 0.0000)
Locale is relevant (p-value = 0.0000)
Dcoilwtico is relevant (p-value = 0.0001)
unit_sales is relevant (p-value = 0.0000)
```

0.0.7 Split the dataset into features (X) and target (y)

[18]:
```
# Split the dataset into features (X) and target (y)
X = df[['Type','Dcoilwtico','unit_sales','Locale']]
y = df['Transactions']
```

0.0.8 Linear Regression

[19]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split


# Split the dataset into features (X) and target (y)
X = df[['Type','Dcoilwtico','unit_sales','Locale']]
y = df['Transactions']

# 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)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
Root Mean Squared Error: 0.73
R-squared: 0.31
```

0.0.9 ANOVA shows interaction has significance

[20]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('Transactions ~ Type + Locale + Type:Locale', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                   sum_sq       df             F        PR(>F)
Type         21247.117933      1.0  38027.143772  0.000000e+00
Locale         117.973380      1.0    211.143493  8.653382e-48
Type:Locale      7.813008      1.0     13.983374  1.845372e-04
Residual     55871.322273  99996.0           NaN           NaN
```

0.0.10 OLS Summary including interaction between the categorical features

[21]:
```
# Fit the linear regression model
model = ols('Transactions ~ Type + Locale + Type:Locale + Dcoilwtico +␣
↪unit_sales', data=df).fit()

# Print the summary of the model
print(model.summary())

# Check the p-values of the features
pvalues = model.pvalues
print('P-values of the features:')
print(pvalues)

# Evaluate the relevance of the features
significance_level = 0.05  # Set the desired significance level (e.g., 5%)

print('Relevant features:')
for feature, p_value in pvalues.items():
    if p_value < significance_level:
        print(f'{feature} is relevant (p-value = {p_value:.4f})')
    else:
        print(f'{feature} is not relevant (p-value = {p_value:.4f})')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Transactions   R-squared:                       0.315
Model:                            OLS   Adj. R-squared:                  0.315
Method:                 Least Squares   F-statistic:                     9178.
```

```
Date:                  Mon, 29 Apr 2024   Prob (F-statistic):               0.00
Time:                          03:52:22   Log-Likelihood:            -1.1009e+05
No. Observations:                100000   AIC:                         2.202e+05
Df Residuals:                     99994   BIC:                         2.202e+05
Df Model:                             5
Covariance Type:              nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept       0.6416      0.007     97.501      0.000       0.629       0.654
Type           -0.3325      0.003   -117.139      0.000      -0.338      -0.327
Locale          0.0742      0.007     10.041      0.000       0.060       0.089
Type:Locale    -0.0122      0.003     -3.863      0.000      -0.018      -0.006
Dcoilwtico     -0.0100      0.003     -3.985      0.000      -0.015      -0.005
unit_sales      0.5671      0.008     74.482      0.000       0.552       0.582
==============================================================================
Omnibus:                       5504.718   Durbin-Watson:                   2.008
Prob(Omnibus):                    0.000   Jarque-Bera (JB):             6462.166
Skew:                             0.620   Prob(JB):                         0.00
Kurtosis:                         3.116   Cond. No.                         13.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
P-values of the features:
Intercept      0.000000e+00
Type           0.000000e+00
Locale         1.033275e-23
Type:Locale    1.121472e-04
Dcoilwtico     6.739046e-05
unit_sales     0.000000e+00
dtype: float64
Relevant features:
Intercept is relevant (p-value = 0.0000)
Type is relevant (p-value = 0.0000)
Locale is relevant (p-value = 0.0000)
Type:Locale is relevant (p-value = 0.0001)
Dcoilwtico is relevant (p-value = 0.0001)
unit_sales is relevant (p-value = 0.0000)
```

0.0.11 Decision Tree Regressor

[22]:
```python
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Split the dataset into features (X) and target (y)
X = df_copy[['Type','Dcoilwtico','unit_sales','Locale']]
y = df_copy['Transactions']

# 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)

# Create a Regression Tree model
model = DecisionTreeRegressor(random_state=42)

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
Root Mean Squared Error: 0.61
R-squared: 0.63
```

0.0.12 HPO for Decision Tree Regressor

[23]:

```
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

X = df_copy[['Type','Dcoilwtico','unit_sales','Locale']]
y = df_copy['Transactions']


# 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)

# Define the parameter grid for GridSearchCV
param_grid = {
```

```
    'max_depth': [3, 5, 7, 9, 11],
    'min_samples_split': [2, 4, 6, 8, 10],
    'min_samples_leaf': [1, 2, 3, 4, 5]
}

# Create the Decision Tree Regressor model
model = DecisionTreeRegressor(random_state=42)

# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,␣
 ↪scoring='neg_mean_squared_error', n_jobs=-1)

# Fit the GridSearchCV to the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters and the best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Evaluate the best model on the test data
y_pred = best_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)

print(f"Best Hyperparameters: {best_params}")
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
Best Hyperparameters: {'max_depth': 9, 'min_samples_leaf': 5,
'min_samples_split': 2}
Root Mean Squared Error: 0.55
R-squared: 0.69
```

0.0.13 Best fit polynomial Regression at n=5

[24]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Split the dataset into features (X) and target (y)
X = df[['Type','Dcoilwtico','unit_sales','Locale']]
y = df['Transactions']
```

```
# 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)

# Create a Polynomial Features transformer
poly = PolynomialFeatures(degree=5)  # Specify the degree of the polynomial

# Transform the training and testing data
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the transformed training data
model.fit(X_train_poly, y_train)

# Make predictions on the transformed test data
y_pred = model.predict(X_test_poly)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
Root Mean Squared Error: 0.56
R-squared: 0.59
```

[327]:

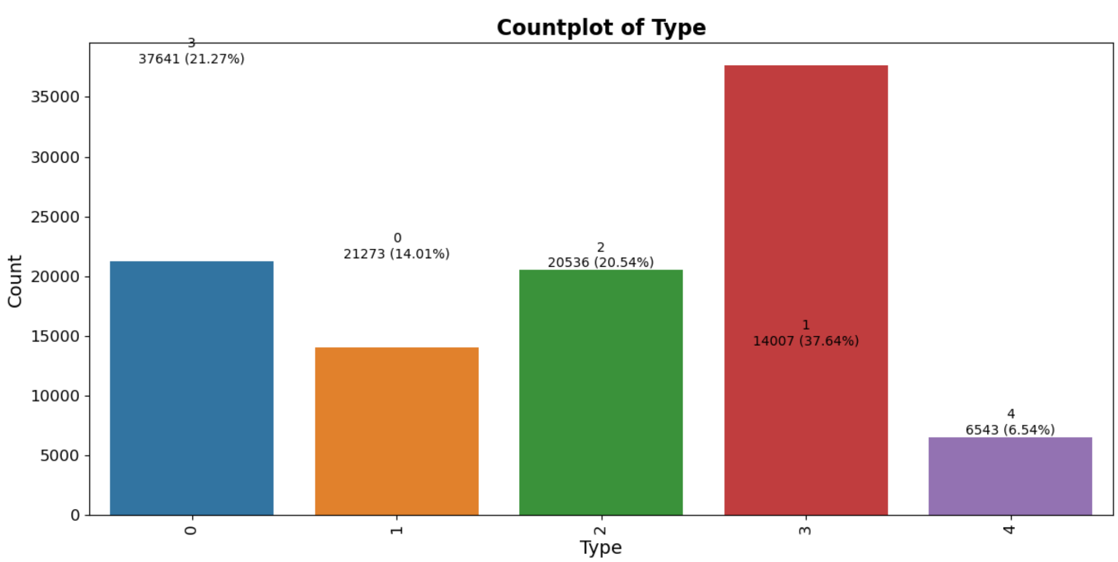
```
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Split the dataset into features (X) and target (y)
X = df.drop(['Transactions','Date','Id'], axis=1)
y = df['Transactions']

# 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)

# Create a Regression Tree model
model = DecisionTreeRegressor(random_state=42)
```

```python
# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")

# Get feature importances
feature_importances = model.feature_importances_

# Sort the feature importances in descending order
sorted_importances = sorted(zip(X.columns, feature_importances), key=lambda x:␣
 ↪x[1], reverse=True)

# Print the feature importances in descending order
print("Feature Importances:")
for feature, importance in sorted_importances:
    print(f"{feature}: {importance:.2f}")
```

```
Root Mean Squared Error: 0.01
R-squared: 1.00
Feature Importances:
Type: 0.45
stor_nbr: 0.24
Dcoilwtico: 0.22
Cluster: 0.06
locale_name: 0.01
State: 0.01
Locale: 0.00
Transferred: 0.00
unit_sales: 0.00
Onpromotion: 0.00
Class: 0.00
Family: 0.00
Perishable: 0.00
```

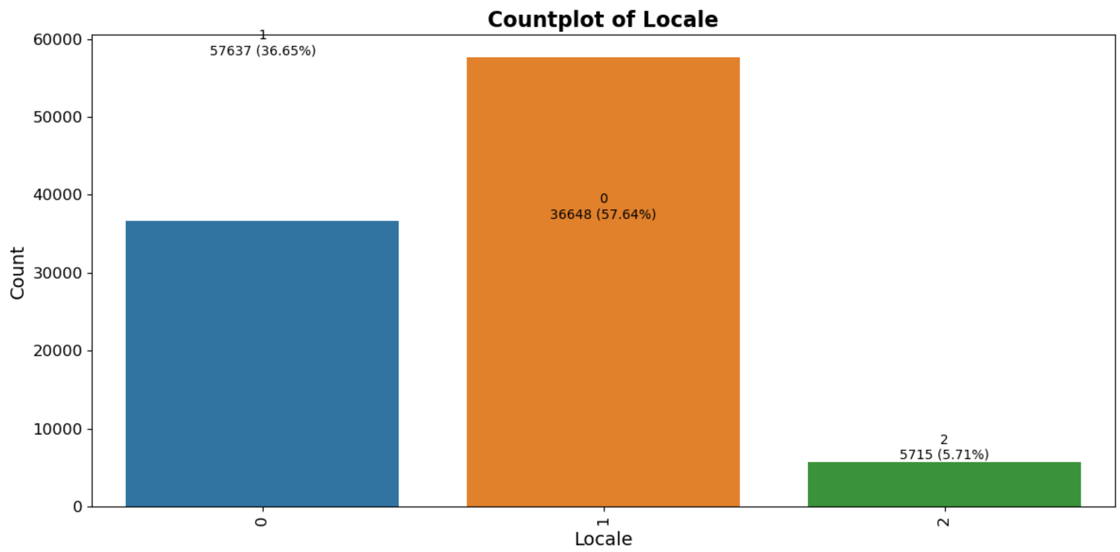
[144]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# Select the feature you want to create a countplot for
feature = 'Type'

# Calculate the counts and percentages for each unique value of the feature
feature_counts = df[feature].value_counts()
feature_percentages = (feature_counts / feature_counts.sum()) * 100

# Create the countplot
fig, ax = plt.subplots(figsize=(12, 6))
sns.countplot(x=feature, data=df, ax=ax)

# Add the count and percentage labels to the bars
for i, (label, count) in enumerate(feature_counts.items()):
    percentage = feature_percentages[i]
    ax.text(i, count + 0.5, f"{label}\n{count} ({percentage:.2f}%)",␣
 ↪ha='center', va='bottom', fontsize=10)

# Set the title, axis labels, and tick label rotation
ax.set_title(f"Countplot of {feature}", fontsize=16, fontweight='bold')
ax.set_xlabel(feature, fontsize=14)
ax.set_ylabel("Count", fontsize=14)
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)

# Adjust the layout and display the plot
plt.tight_layout()
plt.show()
```



[145]:

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


# Select the feature you want to create a countplot for
feature = 'Locale'

# Calculate the counts and percentages for each unique value of the feature
feature_counts = df[feature].value_counts()
feature_percentages = (feature_counts / feature_counts.sum()) * 100

# Create the countplot
fig, ax = plt.subplots(figsize=(12, 6))
sns.countplot(x=feature, data=df, ax=ax)

# Add the count and percentage labels to the bars
for i, (label, count) in enumerate(feature_counts.items()):
    percentage = feature_percentages[i]
    ax.text(i, count + 0.5, f"{label}\n{count} ({percentage:.2f}%)",␣
 ↪ha='center', va='bottom', fontsize=10)

# Set the title, axis labels, and tick label rotation
ax.set_title(f"Countplot of {feature}", fontsize=16, fontweight='bold')
ax.set_xlabel(feature, fontsize=14)
ax.set_ylabel("Count", fontsize=14)
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)

# Adjust the layout and display the plot
plt.tight_layout()
plt.show()
```



0.0.14 One hot Encoding

[25]:
```
import pandas as pd

# Select the categorical feature you want to one-hot encode
categorical_feature = 'Type'

# Perform one-hot encoding
one_hot_encoded = pd.get_dummies(df[categorical_feature],␣
↪prefix=categorical_feature)

# Add the one-hot encoded features to the original DataFrame
df = pd.concat([df, one_hot_encoded], axis=1)

# Drop the original categorical feature (optional)
df = df.drop(categorical_feature, axis=1)

# Display the updated DataFrame
df.head()
```

[25]:

	Date	Id	Family	Locale	locale_name	State	stor_nbr	Transferred	\
0	11/6/2014	19875	12	2	23	9	31	0	
1	5/24/2016	68882	12	1	4	10	54	1	
2	8/24/2016	77456	12	0	0	12	9	0	
3	4/1/2016	37854	27	2	2	6	28	0	
4	4/18/2016	40799	7	1	4	12	1	0	

```
   Class  Cluster  Onpromotion  Perishable  Dcoilwtico  unit_sales  \
```

	Class	Cluster	Onpromotion	Perishable	Dcoilwtico	unit_sales
0	11	9	0	0	0.854773	0.284028
1	21	2	1	0	-0.456037	-0.341827
2	35	5	0	0	-0.532937	-0.341827
3	264	9	0	0	-1.013230	-0.341827
4	223	12	0	0	-0.820761	-0.185363

	Transactions	Type_0	Type_1	Type_2	Type_3	Type_4
0	-0.582249	0	1	0	0	0
1	-0.979912	0	0	1	0	0
2	-0.163472	0	1	0	0	0
3	-0.608643	0	0	0	0	1
4	0.178764	0	0	0	1	0

[52]:
```python
import pandas as pd

# Select the categorical feature you want to one-hot encode
categorical_feature = 'Type'

# Perform one-hot encoding
one_hot_encoded = pd.get_dummies(df_copy[categorical_feature],␣
 ↪prefix=categorical_feature)

# Add the one-hot encoded features to the original DataFrame
df_copy = pd.concat([df_copy, one_hot_encoded], axis=1)

# Drop the original categorical feature (optional)
df_copy = df_copy.drop(categorical_feature, axis=1)

# Display the updated DataFrame
df_copy.head()
```

[52]:

	Date	Id	Family	Locale	locale_name	State	stor_nbr	Transferred \
0	11/6/2014	19875	12	2	23	9	31	0
1	5/24/2016	68882	12	1	4	10	54	1
2	8/24/2016	77456	12	0	0	12	9	0
3	4/1/2016	37854	27	2	2	6	28	0
4	4/18/2016	40799	7	1	4	12	1	0

	Class	Cluster	Onpromotion	Perishable	Dcoilwtico	unit_sales \
0	11	9	0	0	0.854773	0.284028
1	21	2	1	0	-0.456037	-0.341827
2	35	5	0	0	-0.532937	-0.341827
3	264	9	0	0	-1.013230	-0.341827
4	223	12	0	0	-0.820761	-0.185363

	Transactions	Type_0	Type_1	Type_2	Type_3	Type_4
0	-0.582249	0	1	0	0	0

```
1     -0.979912       0       0       1       0       0
2     -0.163472       0       1       0       0       0
3     -0.608643       0       0       0       0       1
4      0.178764       0       0       0       1       0
```

[26]:

```python
import pandas as pd

# Select the categorical feature you want to one-hot encode
categorical_feature = 'Locale'

# Perform one-hot encoding
one_hot_encoded = pd.get_dummies(df[categorical_feature],␣
 ↪prefix=categorical_feature)

# Add the one-hot encoded features to the original DataFrame
df = pd.concat([df, one_hot_encoded], axis=1)

# Drop the original categorical feature (optional)
df = df.drop(categorical_feature, axis=1)

# Display the updated DataFrame
df.head()
```

[26]:

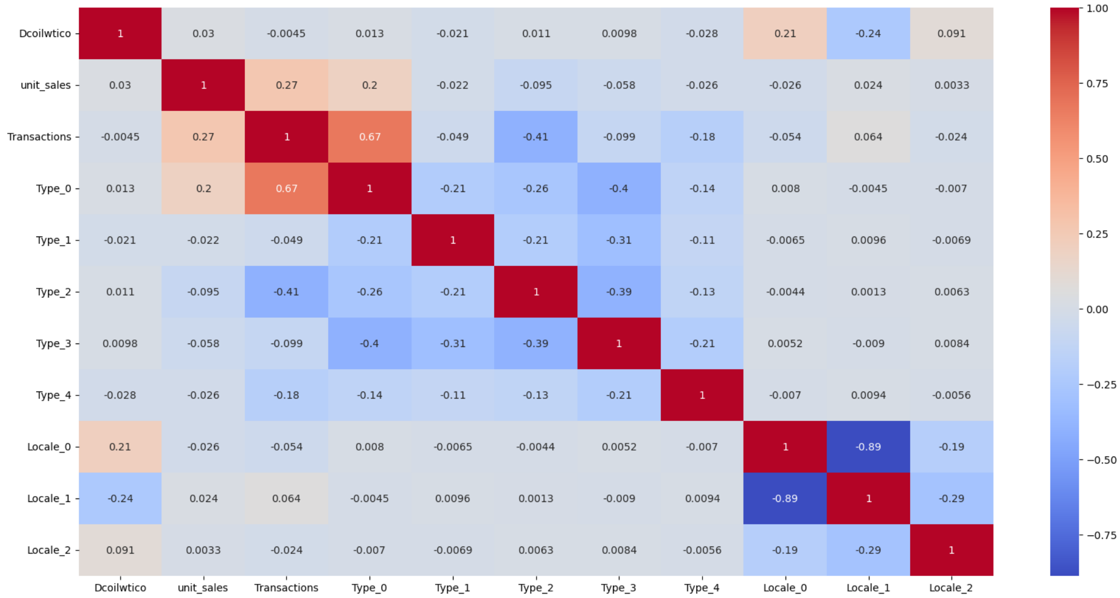
```
        Date     Id  Family  locale_name  State  stor_nbr  Transferred  Class  \
0  11/6/2014  19875      12           23      9        31            0     11
1  5/24/2016  68882      12            4     10        54            1     21
2  8/24/2016  77456      12            0     12         9            0     35
3   4/1/2016  37854      27            2      6        28            0    264
4  4/18/2016  40799       7            4     12         1            0    223

   Cluster  Onpromotion  …  unit_sales  Transactions  Type_0  Type_1  \
0        9            0  …    0.284028     -0.582249       0       1
1        2            1  …   -0.341827     -0.979912       0       0
2        5            0  …   -0.341827     -0.163472       0       1
3        9            0  …   -0.341827     -0.608643       0       0
4       12            0  …   -0.185363      0.178764       0       0

   Type_2  Type_3  Type_4  Locale_0  Locale_1  Locale_2
0       0       0       0         0         0         1
1       1       0       0         0         1         0
2       0       0       0         1         0         0
3       0       0       1         0         0         1
4       0       1       0         0         1         0

[5 rows x 22 columns]
```

[53]:
```
import pandas as pd

# Select the categorical feature you want to one-hot encode
categorical_feature = 'Locale'

# Perform one-hot encoding
one_hot_encoded = pd.get_dummies(df_copy[categorical_feature],␣
↪prefix=categorical_feature)

# Add the one-hot encoded features to the original DataFrame
df_copy = pd.concat([df_copy, one_hot_encoded], axis=1)

# Drop the original categorical feature (optional)
df_copy = df_copy.drop(categorical_feature, axis=1)

# Display the updated DataFrame
df_copy.head()
```

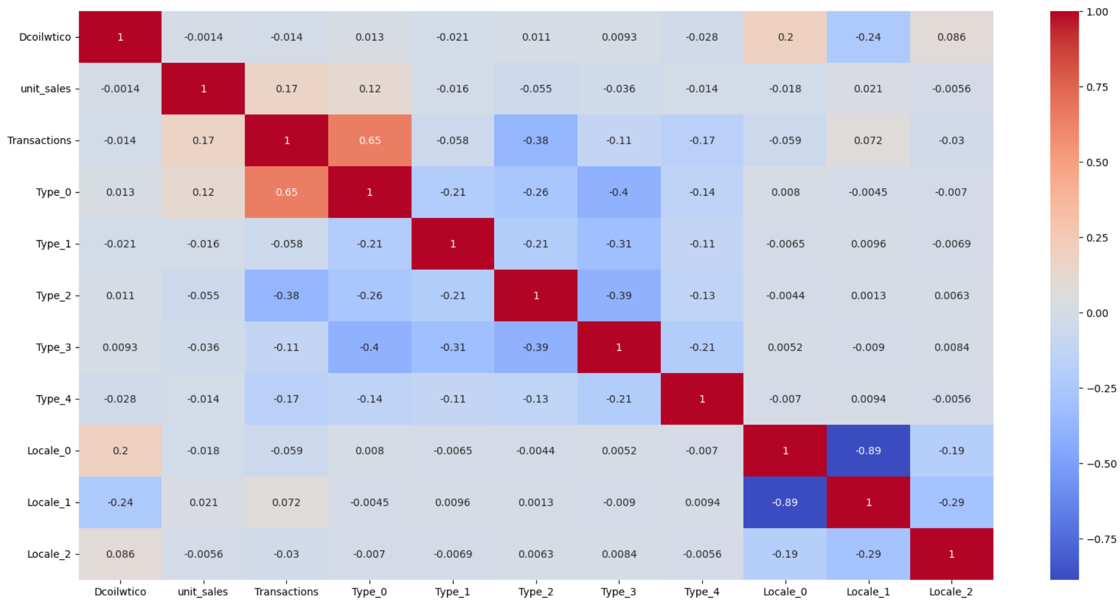
[53]:
```
        Date     Id  Family  locale_name  State  stor_nbr  Transferred  Class  \
0  11/6/2014  19875      12           23      9        31            0     11
1  5/24/2016  68882      12            4     10        54            1     21
2  8/24/2016  77456      12            0     12         9            0     35
3   4/1/2016  37854      27            2      6        28            0    264
4  4/18/2016  40799       7            4     12         1            0    223

   Cluster  Onpromotion  …  unit_sales  Transactions  Type_0  Type_1  \
0        9            0  …    0.284028     -0.582249       0       1
1        2            1  …   -0.341827     -0.979912       0       0
2        5            0  …   -0.341827     -0.163472       0       1
3        9            0  …   -0.341827     -0.608643       0       0
4       12            0  …   -0.185363      0.178764       0       0

   Type_2  Type_3  Type_4  Locale_0  Locale_1  Locale_2
0       0       0       0         0         0         1
1       1       0       0         0         1         0
2       0       0       0         1         0         0
3       0       0       1         0         0         1
4       0       1       0         0         1         0

[5 rows x 22 columns]
```

[63]:
```
import seaborn as sns
plt.figure(figsize=(20, 10))
# Calculate the correlation matrix
corr_matrix = df.corr()

# Create a heatmap using seaborn
```

```
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')

# Display the heatmap
plt.show()
```



[64]:
```
import seaborn as sns
plt.figure(figsize=(20, 10))
# Calculate the correlation matrix
corr_matrix = df_copy.corr()

# Create a heatmap using seaborn
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')

# Display the heatmap
plt.show()
```



0.0.15 backup

[27]:
```
relevant =␣
↪['Type_0','Type_1','Type_2','Type_3','Type_4','Locale_0','Locale_1','Locale_2','Dcoilwtico'
```

0.0.16 Changes only for model purposes; backup in df_copy

[28]:
```
irrelevant =␣
↪['Id','Date','locale_name','Family','State','stor_nbr','Transferred','Class','Cluster','Per
```

[29]:
```
df = df.drop(irrelevant,axis = 1)
```

[54]:
```
df_copy = df_copy.drop(irrelevant,axis=1)
```

[30]:
```
df.columns
```

[30]:
```
Index(['Dcoilwtico', 'unit_sales', 'Transactions', 'Type_0', 'Type_1',
       'Type_2', 'Type_3', 'Type_4', 'Locale_0', 'Locale_1', 'Locale_2'],
      dtype='object')
```

[31]:
```
df.isnull().sum()
```

[31]:
```
Dcoilwtico      0
unit_sales      0
Transactions    0
Type_0          0
Type_1          0
```

```
Type_2          0
Type_3          0
Type_4          0
Locale_0        0
Locale_1        0
Locale_2        0
dtype: int64
```

[32]:
```
df.shape
```

[32]:
```
(100000, 11)
```

[33]:
```
df.columns
```

[33]:
```
Index(['Dcoilwtico', 'unit_sales', 'Transactions', 'Type_0', 'Type_1',
       'Type_2', 'Type_3', 'Type_4', 'Locale_0', 'Locale_1', 'Locale_2'],
      dtype='object')
```

0.0.17 Again ANOVA checking for interaction on OHE

[34]:
```
df.shape
```

[34]:
```
(100000, 11)
```

[35]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_0 + Locale_0 + Type_0:Locale_0
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                       sum_sq       df             F         PR(>F)
Type_0           34653.400535      1.0  81923.588403   0.000000e+00
Locale_0           270.663216      1.0    639.870881  9.910007e-141
Type_0:Locale_0     49.183116      1.0    116.273074   4.286318e-27
Residual         42297.969455  99996.0           NaN            NaN
```

[36]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

```
# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_0 + Locale_1 + Type_0:Locale_1
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                       sum_sq       df             F         PR(>F)
Type_0           34636.921069      1.0  82035.414988   0.000000e+00
Locale_1           343.308210      1.0    813.104358  3.969238e-178
Type_0:Locale_1     54.284036      1.0    128.568396   8.790148e-30
Residual         42220.223542  99996.0           NaN            NaN
```

[37]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_0 + Locale_2 + Type_0:Locale_2
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                       sum_sq       df             F        PR(>F)
Type_0           34591.335859      1.0  81219.112592  0.000000e+00
Locale_2            28.011979      1.0     65.771039  5.121421e-16
Type_0:Locale_2      1.364829      1.0      3.204566  7.343601e-02
Residual         42588.438979  99996.0           NaN           NaN
```

[38]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_1 + Locale_0 + Type_1:Locale_0
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
```

```
print(anova_table)
```

```
                        sum_sq       df           F        PR(>F)
Type_1              187.252677      1.0  243.770217  6.880189e-55
Locale_0            226.794128      1.0  295.246265  4.452237e-66
Type_1:Locale_0       1.132971      1.0    1.474930  2.245723e-01
Residual          76812.167458  99996.0         NaN           NaN
```

[39]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_1 + Locale_1 + Type_1:Locale_1
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                        sum_sq       df           F        PR(>F)
Type_1              189.240307      1.0  246.655325  1.622226e-55
Locale_1            317.906217      1.0  414.358138  6.342898e-92
Type_1:Locale_1       2.686096      1.0    3.501050  6.133284e-02
Residual          76719.502244  99996.0         NaN           NaN
```

[40]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_1 + Locale_2 + Type_1:Locale_2
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                        sum_sq       df           F        PR(>F)
Type_1              185.847455      1.0  241.369478  2.289703e-54
Locale_2             44.802344      1.0   58.187068  2.404353e-14
Type_1:Locale_2       1.290399      1.0    1.675906  1.954724e-01
Residual          76994.001814  99996.0         NaN           NaN
```

[41]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_2 + Locale_0 + Type_2:Locale_0
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                      sum_sq       df             F        PR(>F)
Type_2          12774.848615      1.0  19892.912228  0.000000e+00
Locale_0          239.346097      1.0    372.708205  6.820244e-83
Type_2:Locale_0    10.181519      1.0     15.854596  6.844853e-05
Residual        64215.522972  99996.0           NaN           NaN
```

[42]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_2 + Locale_1 + Type_2:Locale_1
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                      sum_sq       df             F         PR(>F)
Type_2          12765.010018      1.0  19903.234108   0.000000e+00
Locale_1          318.631959      1.0    496.811712  8.706964e-110
Type_2:Locale_1    13.628206      1.0     21.249132   4.037907e-06
Residual        64132.790422  99996.0           NaN            NaN
```

[43]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_2 + Locale_2 + Type_2:Locale_2
            '''
```

```
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                      sum_sq       df             F        PR(>F)
Type_2          12750.718370      1.0  19789.631578  0.000000e+00
Locale_2           34.629290      1.0     53.746062  2.298654e-13
Type_2:Locale_2     1.691079      1.0      2.624623  1.052207e-01
Residual        64428.730219  99996.0           NaN           NaN
```

[44]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_3 + Locale_0 + Type_3:Locale_0
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                      sum_sq       df           F         PR(>F)
Type_3            754.262415      1.0  989.209957  4.542562e-216
Locale_0          219.918032      1.0  288.420983   1.353277e-64
Type_3:Locale_0     0.369524      1.0    0.484628   4.863353e-01
Residual        76245.921168  99996.0         NaN            NaN
```

[45]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_3 + Locale_1 +  Type_3:Locale_1
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                      sum_sq       df           F         PR(>F)
Type_3            749.811567      1.0  984.462941  4.775967e-215
Locale_1          304.591644      1.0  399.912723   8.589873e-89
```

```
Type_3:Locale_1      0.134147      1.0    0.176129   6.747225e-01
Residual         76161.482932  99996.0         NaN            NaN
```

[46]:
```python
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_3 + Locale_2 +  Type_3:Locale_2
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                      sum_sq       df           F         PR(>F)
Type_3            755.478918      1.0  988.480964  6.519404e-216
Locale_2           40.547974      1.0   53.053632   3.269415e-13
Type_3:Locale_2     0.446109      1.0    0.583696   4.448689e-01
Residual        76425.214641  99996.0         NaN            NaN
```

[47]:
```python
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_4 + Locale_0 +  Type_4:Locale_0
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                      sum_sq       df            F        PR(>F)
Type_4           2541.611563      1.0  3413.305125  0.000000e+00
Locale_0          234.754431      1.0   315.267885  1.995529e-70
Type_4:Locale_0     0.028376      1.0     0.038108  8.452263e-01
Residual        74458.913167  99996.0          NaN           NaN
```

[48]:
```python
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
```

```
model = ols('''
            Transactions ~ Type_4 + Locale_1 +  Type_4:Locale_1
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                      sum_sq       df            F        PR(>F)
Type_4           2547.985484      1.0  3426.270289  0.000000e+00
Locale_1          330.252811      1.0   444.090205  2.284188e-98
Type_4:Locale_1     0.262504      1.0     0.352989  5.524276e-01
Residual        74363.180659  99996.0          NaN           NaN
```

[49]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Perform two-way ANOVA
model = ols('''
            Transactions ~ Type_4 + Locale_2 +  Type_4:Locale_2
            '''
            , data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Print the ANOVA table
print(anova_table)
```

```
                      sum_sq       df            F        PR(>F)
Type_4           2534.773573      1.0  3395.590385  0.000000e+00
Locale_2           47.329880      1.0    63.403251  1.701940e-15
Type_4:Locale_2     0.372458      1.0     0.498946  4.799652e-01
Residual        74645.993636  99996.0          NaN           NaN
```

Removing all unnecessary interactions and only using significant interactions in model.

0.0.18 Again Linear model ols

[50]:
```
 # Fit the linear regression model
model = ols('''
            Transactions ~ Type_0 + Type_1 + Type_2 + Type_3 + Type_4 +
            Locale_0 + Locale_1 + Locale_2 +
            Type_0:Locale_0 + Type_0:Locale_1 +
            Type_2:Locale_0 + Type_2:Locale_1 +
            Dcoilwtico + unit_sales
            ''', data=df).fit()
```

```python
# Print the summary of the model
print(model.summary())

# Check the p-values of the features
pvalues = model.pvalues
print('P-values of the features:')
print(pvalues)

# Evaluate the relevance of the features
significance_level = 0.05  # Set the desired significance level (e.g., 5%)

print('Relevant features:')
for feature, p_value in pvalues.items():
    if p_value < significance_level:
        print(f'{feature} is relevant (p-value = {p_value:.4f})')
    else:
        print(f'{feature} is not relevant (p-value = {p_value:.4f})')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Transactions   R-squared:                       0.545
Model:                            OLS   Adj. R-squared:                  0.545
Method:                 Least Squares   F-statistic:                     9999.
Date:                Mon, 29 Apr 2024   Prob (F-statistic):               0.00
Time:                        03:56:50   Log-Likelihood:                -89549.
No. Observations:              100000   AIC:                         1.791e+05
Df Residuals:                   99987   BIC:                         1.792e+05
Df Model:                          12
Covariance Type:            nonrobust
================================================================================
===
                      coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
---
Intercept          -0.0637      0.004    -18.140      0.000      -0.071
-0.057
Type_0              1.1198      0.016     69.738      0.000       1.088
1.151
Type_1             -0.0339      0.007     -4.707      0.000      -0.048
-0.020
Type_2             -0.5898      0.016    -37.753      0.000      -0.620
-0.559
Type_3             -0.0354      0.006     -5.524      0.000      -0.048
-0.023
Type_4             -0.5244      0.008    -62.355      0.000      -0.541
-0.508
Locale_0           -0.0510      0.005    -10.467      0.000      -0.061
```

```
-0.041
Locale_1            0.0485      0.005     10.545      0.000       0.039
0.058
Locale_2           -0.0612      0.007     -8.818      0.000      -0.075
-0.048
Type_0:Locale_0    -0.0260      0.022     -1.196      0.232      -0.069
0.017
Type_0:Locale_1     0.0786      0.021      3.704      0.000       0.037
0.120
Type_2:Locale_0    -0.0013      0.021     -0.061      0.952      -0.043
0.040
Type_2:Locale_1    -0.0402      0.021     -1.938      0.053      -0.081
0.000
Dcoilwtico         -0.0007      0.002     -0.351      0.726      -0.005
0.003
unit_sales          0.3846      0.006     61.450      0.000       0.372
0.397
==============================================================================
Omnibus:                     5382.300   Durbin-Watson:                   2.003
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7593.831
Skew:                           0.496   Prob(JB):                         0.00
Kurtosis:                       3.916   Cond. No.                     2.40e+15
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 3.13e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
P-values of the features:
Intercept           2.024545e-73
Type_0              0.000000e+00
Type_1              2.514774e-06
Type_2             1.044414e-309
Type_3              3.314024e-08
Type_4              0.000000e+00
Locale_0            1.265120e-25
Locale_1            5.518517e-26
Locale_2            1.180431e-18
Type_0:Locale_0     2.317815e-01
Type_0:Locale_1     2.122412e-04
Type_2:Locale_0     9.516561e-01
Type_2:Locale_1     5.257151e-02
Dcoilwtico          7.258214e-01
unit_sales          0.000000e+00
dtype: float64
Relevant features:
Intercept is relevant (p-value = 0.0000)
```

```
Type_0 is relevant (p-value = 0.0000)
Type_1 is relevant (p-value = 0.0000)
Type_2 is relevant (p-value = 0.0000)
Type_3 is relevant (p-value = 0.0000)
Type_4 is relevant (p-value = 0.0000)
Locale_0 is relevant (p-value = 0.0000)
Locale_1 is relevant (p-value = 0.0000)
Locale_2 is relevant (p-value = 0.0000)
Type_0:Locale_0 is not relevant (p-value = 0.2318)
Type_0:Locale_1 is relevant (p-value = 0.0002)
Type_2:Locale_0 is not relevant (p-value = 0.9517)
Type_2:Locale_1 is not relevant (p-value = 0.0526)
Dcoilwtico is not relevant (p-value = 0.7258)
unit_sales is relevant (p-value = 0.0000)
```

[51]:

```python
 # Fit the linear regression model
model = ols('''
            Transactions ~ Type_0 + Type_1 + Type_2 + Type_3 + Type_4 +
            Locale_0 + Locale_1 + Locale_2 +
            Dcoilwtico + unit_sales
            ''', data=df).fit()

# Print the summary of the model
print(model.summary())

# Check the p-values of the features
pvalues = model.pvalues
print('P-values of the features:')
print(pvalues)

# Evaluate the relevance of the features
significance_level = 0.05  # Set the desired significance level (e.g., 5%)

print('Relevant features:')
for feature, p_value in pvalues.items():
    if p_value < significance_level:
        print(f'{feature} is relevant (p-value = {p_value:.4f})')
    else:
        print(f'{feature} is not relevant (p-value = {p_value:.4f})')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Transactions   R-squared:                       0.545
Model:                            OLS   Adj. R-squared:                  0.545
Method:                 Least Squares   F-statistic:                 1.495e+04
Date:                Mon, 29 Apr 2024   Prob (F-statistic):               0.00
Time:                        03:56:51   Log-Likelihood:                -89630.
No. Observations:              100000   AIC:                         1.793e+05
```

```
Df Residuals:                   99991   BIC:                         1.794e+05
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.0633      0.002    -29.815      0.000      -0.067      -0.059
Type_0         1.1529      0.004    297.943      0.000       1.145       1.161
Type_1        -0.0362      0.004     -8.066      0.000      -0.045      -0.027
Type_2        -0.6157      0.004   -158.282      0.000      -0.623      -0.608
Type_3        -0.0376      0.003    -11.933      0.000      -0.044      -0.031
Type_4        -0.5268      0.006    -84.424      0.000      -0.539      -0.515
Locale_0      -0.0580      0.003    -18.287      0.000      -0.064      -0.052
Locale_1       0.0560      0.003     19.050      0.000       0.050       0.062
Locale_2      -0.0613      0.006    -10.364      0.000      -0.073      -0.050
Dcoilwtico    -0.0008      0.002     -0.401      0.689      -0.005       0.003
unit_sales     0.3847      0.006     61.425      0.000       0.372       0.397
==============================================================================
Omnibus:                     5326.145   Durbin-Watson:                   2.003
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7437.558
Skew:                           0.496   Prob(JB):                         0.00
Kurtosis:                       3.896   Cond. No.                     2.30e+15
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 3.32e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
P-values of the features:
Intercept     1.780062e-194
Type_0         0.000000e+00
Type_1         7.354787e-16
Type_2         0.000000e+00
Type_3         8.350200e-33
Type_4         0.000000e+00
Locale_0       1.385764e-74
Locale_1       9.194345e-81
Locale_2       3.708340e-25
Dcoilwtico     6.887891e-01
unit_sales     0.000000e+00
dtype: float64
Relevant features:
Intercept is relevant (p-value = 0.0000)
Type_0 is relevant (p-value = 0.0000)
Type_1 is relevant (p-value = 0.0000)
Type_2 is relevant (p-value = 0.0000)
Type_3 is relevant (p-value = 0.0000)
```

```
Type_4 is relevant (p-value = 0.0000)
Locale_0 is relevant (p-value = 0.0000)
Locale_1 is relevant (p-value = 0.0000)
Locale_2 is relevant (p-value = 0.0000)
Dcoilwtico is not relevant (p-value = 0.6888)
unit_sales is relevant (p-value = 0.0000)
```

[55]:

```
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

X = df_copy[relevant]
y = df_copy['Transactions']

# 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)

# Define the parameter grid for GridSearchCV
param_grid = {
    'max_depth': [3, 5, 7, 9, 11],
    'min_samples_split': [2, 4, 6, 8, 10],
    'min_samples_leaf': [1, 2, 3, 4, 5]
}

# Create the Decision Tree Regressor model
model = DecisionTreeRegressor(random_state=42)

# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,␣
↪scoring='neg_mean_squared_error', n_jobs=-1)

# Fit the GridSearchCV to the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters and the best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Evaluate the best model on the test data
y_pred = best_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)

print(f"Best Hyperparameters: {best_params}")
```

```
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
Best Hyperparameters: {'max_depth': 9, 'min_samples_leaf': 5,
'min_samples_split': 2}
Root Mean Squared Error: 0.55
R-squared: 0.69
```

[56]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

X = df[relevant]
y = df['Transactions']

# 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)

# Create a Polynomial Features transformer
poly = PolynomialFeatures(degree=5)  # Specify the degree of the polynomial

# Transform the training and testing data
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the transformed training data
model.fit(X_train_poly, y_train)

# Make predictions on the transformed test data
y_pred = model.predict(X_test_poly)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
Root Mean Squared Error: 0.56
R-squared: 0.59
```

[57]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score

# Fit a linear regression model
linear_model = LinearRegression()
linear_model.fit(X, y)
y_linear_pred = linear_model.predict(X)
linear_r2 = r2_score(y, y_linear_pred)

# Fit a polynomial regression model (degree=2)
poly_model = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
poly_model.fit(X, y)
y_poly_pred = poly_model.predict(X)
poly_r2 = r2_score(y, y_poly_pred)

# Compare the R-squared values
print(f'Linear Regression R-squared: {linear_r2:.4f}')
print(f'Polynomial Regression R-squared: {poly_r2:.4f}')

# Determine the better fit
if poly_r2 > linear_r2:
    print('Polynomial regression provides a better fit to the data.')
else:
    print('Linear regression provides a better fit to the data.')
```

```
Linear Regression R-squared: 0.5446
Polynomial Regression R-squared: 0.5647
Polynomial regression provides a better fit to the data.
```

[58]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score


# Fit a linear regression model
linear_model = LinearRegression()
linear_model.fit(X, y)
y_linear_pred = linear_model.predict(X)
linear_r2 = r2_score(y, y_linear_pred)

# Fit a polynomial regression model (degree=2)
```

```python
poly_model = make_pipeline(PolynomialFeatures(degree=3), LinearRegression())
poly_model.fit(X, y)
y_poly_pred = poly_model.predict(X)
poly_r2 = r2_score(y, y_poly_pred)

# Compare the R-squared values
print(f'Linear Regression R-squared: {linear_r2:.4f}')
print(f'Polynomial Regression R-squared: {poly_r2:.4f}')

# Determine the better fit
if poly_r2 > linear_r2:
    print('Polynomial regression provides a better fit to the data.')
else:
    print('Linear regression provides a better fit to the data.')
```

```
Linear Regression R-squared: 0.5446
Polynomial Regression R-squared: 0.5712
Polynomial regression provides a better fit to the data.
```

[59]:

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score

# Fit a linear regression model
linear_model = LinearRegression()
linear_model.fit(X, y)
y_linear_pred = linear_model.predict(X)
linear_r2 = r2_score(y, y_linear_pred)

# Fit a polynomial regression model (degree=2)
poly_model = make_pipeline(PolynomialFeatures(degree=4), LinearRegression())
poly_model.fit(X, y)
y_poly_pred = poly_model.predict(X)
poly_r2 = r2_score(y, y_poly_pred)

# Compare the R-squared values
print(f'Linear Regression R-squared: {linear_r2:.4f}')
print(f'Polynomial Regression R-squared: {poly_r2:.4f}')

# Determine the better fit
if poly_r2 > linear_r2:
    print('Polynomial regression provides a better fit to the data.')
else:
    print('Linear regression provides a better fit to the data.')
```

```
Linear Regression R-squared: 0.5446
Polynomial Regression R-squared: 0.5925
Polynomial regression provides a better fit to the data.
```

[60]:

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score


# Fit a linear regression model
linear_model = LinearRegression()
linear_model.fit(X, y)
y_linear_pred = linear_model.predict(X)
linear_r2 = r2_score(y, y_linear_pred)

# Fit a polynomial regression model (degree=2)
poly_model = make_pipeline(PolynomialFeatures(degree=5), LinearRegression())
poly_model.fit(X, y)
y_poly_pred = poly_model.predict(X)
poly_r2 = r2_score(y, y_poly_pred)

# Compare the R-squared values
print(f'Linear Regression R-squared: {linear_r2:.4f}')
print(f'Polynomial Regression R-squared: {poly_r2:.4f}')

# Determine the better fit
if poly_r2 > linear_r2:
    print('Polynomial regression provides a better fit to the data.')
else:
    print('Linear regression provides a better fit to the data.')
```

```
Linear Regression R-squared: 0.5446
Polynomial Regression R-squared: 0.6006
Polynomial regression provides a better fit to the data.
```

[ ]:

```python

```