In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
df = pd.read_csv('retail_price.csv')
```

In [3]:

```
df
```

Out[3]:

	product_id	product_category_name	month_year	qty	total_price	freight_price	unit_price
0	bed1	bed_bath_table	01-05-2017	1	45.95	15.100000	45.950000
1	bed1	bed_bath_table	01-06-2017	3	137.85	12.933333	45.950000
2	bed1	bed_bath_table	01-07-2017	6	275.70	14.840000	45.950000
3	bed1	bed_bath_table	01-08-2017	4	183.80	14.287500	45.950000
4	bed1	bed_bath_table	01-09-2017	2	91.90	15.100000	45.950000
...	...	...	...	...	...	...	...
671	bed5	bed_bath_table	01-05-2017	1	215.00	8.760000	215.000000
672	bed5	bed_bath_table	01-06-2017	10	2090.00	21.322000	209.000000
673	bed5	bed_bath_table	01-07-2017	59	12095.00	22.195932	205.000000
674	bed5	bed_bath_table	01-08-2017	52	10375.00	19.412885	199.509804
675	bed5	bed_bath_table	01-09-2017	32	5222.36	24.324687	163.398710

676 rows × 30 columns

Exploratory Data Analysis

In [4]:

```
df.shape
```

Out[4]:

```
(676, 30)
```

In [5]:

```
df.columns
```

Out[5]:

```
Index(['product_id', 'product_category_name', 'month_year', 'qty',
       'total_price', 'freight_price', 'unit_price', 'product_name_lenght',
       'product_description_lenght', 'product_photos_qty', 'product_weight_g',
       'product_score', 'customers', 'weekday', 'weekend', 'holiday', 'month',
       'year', 's', 'volume', 'comp_1', 'ps1', 'fp1', 'comp_2', 'ps2', 'fp2',
       'comp_3', 'ps3', 'fp3', 'lag_price'],
      dtype='object')
```

In [6]:

```
df.duplicated().sum()
```

Out[6]:

```
0
```

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
product_id                    0
product_category_name         0
month_year                    0
qty                           0
total_price                   0
freight_price                 0
unit_price                    0
product_name_lenght           0
product_description_lenght    0
product_photos_qty            0
product_weight_g              0
product_score                 0
customers                     0
weekday                       0
weekend                       0
holiday                       0
month                         0
year                          0
s                             0
volume                        0
comp_1                        0
ps1                           0
fp1                           0
comp_2                        0
ps2                           0
fp2                           0
comp_3                        0
ps3                           0
fp3                           0
lag_price                     0
dtype: int64
```

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 676 entries, 0 to 675
Data columns (total 30 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   product_id                  676 non-null    object 
 1   product_category_name       676 non-null    object 
 2   month_year                  676 non-null    object 
 3   qty                         676 non-null    int64  
 4   total_price                 676 non-null    float64
 5   freight_price               676 non-null    float64
 6   unit_price                  676 non-null    float64
 7   product_name_lenght         676 non-null    int64  
 8   product_description_lenght  676 non-null    int64  
 9   product_photos_qty          676 non-null    int64  
 10  product_weight_g            676 non-null    int64  
 11  product_score               676 non-null    float64
 12  customers                   676 non-null    int64  
 13  weekday                     676 non-null    int64  
 14  weekend                     676 non-null    int64  
 15  holiday                     676 non-null    int64  
 16  month                       676 non-null    int64  
 17  year                        676 non-null    int64  
 18  s                           676 non-null    float64
 19  volume                      676 non-null    int64  
 20  comp_1                      676 non-null    float64
 21  ps1                         676 non-null    float64
 22  fp1                         676 non-null    float64
 23  comp_2                      676 non-null    float64
 24  ps2                         676 non-null    float64
 25  fp2                         676 non-null    float64
 26  comp_3                      676 non-null    float64
 27  ps3                         676 non-null    float64
 28  fp3                         676 non-null    float64
 29  lag_price                   676 non-null    float64
dtypes: float64(15), int64(12), object(3)
memory usage: 158.6+ KB
```

In [9]:

```
df.describe()
```

Out[9]:

	qty	total_price	freight_price	unit_price	product_name_lenght	product_descript
count	676.000000	676.000000	676.000000	676.000000	676.000000	
mean	14.495562	1422.708728	20.682270	106.496800	48.720414	
std	15.443421	1700.123100	10.081817	76.182972	9.420715	
min	1.000000	19.900000	0.000000	19.900000	29.000000	
25%	4.000000	333.700000	14.761912	53.900000	40.000000	
50%	10.000000	807.890000	17.518472	89.900000	51.000000	
75%	18.000000	1887.322500	22.713558	129.990000	57.000000	
max	122.000000	12095.000000	79.760000	364.000000	60.000000	3

8 rows × 27 columns

In [10]:

```
df.nunique()
```

Out[10]:

```
product_id                      52
product_category_name            9
month_year                      20
qty                             66
total_price                    573
freight_price                  653
unit_price                     280
product_name_lenght             24
product_description_lenght      46
product_photos_qty               7
product_weight_g                45
product_score                   11
customers                       94
weekday                          4
weekend                          3
holiday                          5
month                           12
year                             2
s                              450
volume                          40
comp_1                          88
ps1                              9
fp1                            179
comp_2                         123
ps2                             10
fp2                            242
comp_3                         105
ps3                              9
fp3                            229
lag_price                      307
dtype: int64
```

In [11]:

```
object_columns = df.select_dtypes(include='object').columns
print("Object Columns:")
print(object_columns)
print()

# Identify numerical types
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
print("Numerical Columns:")
print(numerical_columns)
```

```
Object Columns:
Index(['product_id', 'product_category_name', 'month_year'], dtype='object')

Numerical Columns:
Index(['qty', 'total_price', 'freight_price', 'unit_price',
       'product_name_lenght', 'product_description_lenght',
       'product_photos_qty', 'product_weight_g', 'product_score', 'customers',
       'weekday', 'weekend', 'holiday', 'month', 'year', 's', 'volume',
       'comp_1', 'ps1', 'fp1', 'comp_2', 'ps2', 'fp2', 'comp_3', 'ps3', 'fp3',
       'lag_price'],
      dtype='object')
```

In [12]:

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

In [13]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [14]:

```
for i in object_columns:
    print(i,':')
    print(df[i].unique())
    print('\n')
```

```
product_id :
['bed1' 'garden5' 'consoles1' 'garden7' 'health9' 'cool4' 'health3'
 'perfumery1' 'cool5' 'health8' 'garden4' 'computers5' 'garden10'
 'computers6' 'health6' 'garden6' 'health10' 'watches2' 'health1'
 'garden8' 'garden9' 'watches6' 'cool3' 'perfumery2' 'cool2' 'computers1'
 'consoles2' 'health5' 'watches8' 'furniture4' 'watches5' 'health7' 'bed3'
 'garden3' 'bed2' 'furniture3' 'watches4' 'watches3' 'furniture2'
 'garden2' 'furniture1' 'health2' 'garden1' 'cool1' 'computers4'
 'watches7' 'computers3' 'health4' 'watches1' 'computers2' 'bed4' 'bed5']


product_category_name :
['bed_bath_table' 'garden_tools' 'consoles_games' 'health_beauty'
 'cool_stuff' 'perfumery' 'computers_accessories' 'watches_gifts'
 'furniture_decor']


month_year :
['01-05-2017' '01-06-2017' '01-07-2017' '01-08-2017' '01-09-2017'
 '01-10-2017' '01-11-2017' '01-12-2017' '01-01-2018' '01-02-2018'
 '01-03-2018' '01-04-2018' '01-05-2018' '01-06-2018' '01-07-2018'
 '01-08-2018' '01-03-2017' '01-04-2017' '01-02-2017' '01-01-2017']
```

In [15]:

```
for i in object_columns:
    print(i,':')
    print(df[i].value_counts())
    print('\n')
```

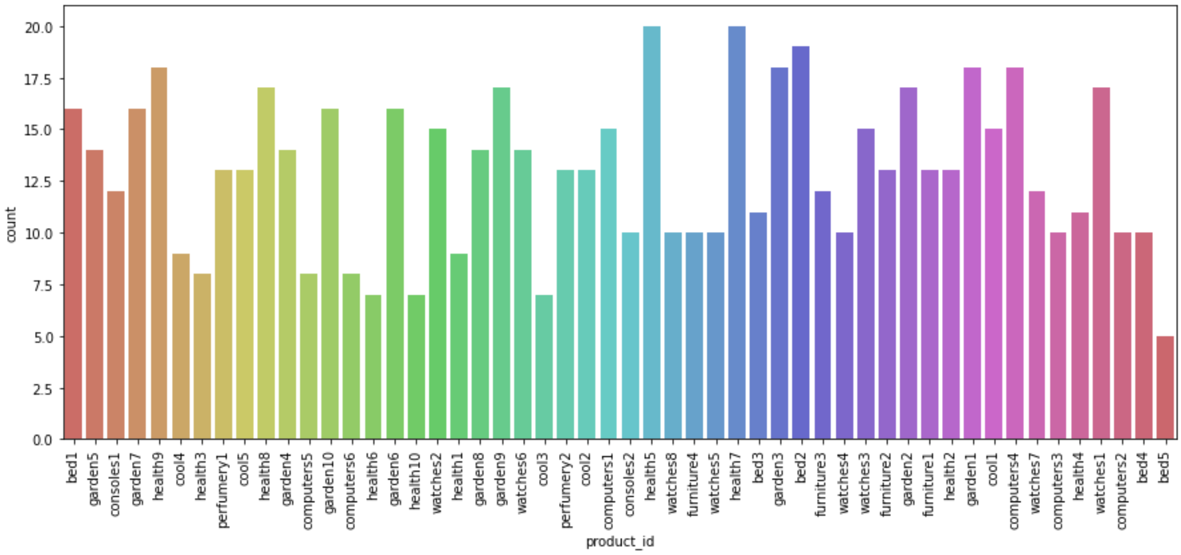
```
product_id :
health5       20
health7       20
bed2          19
garden1       18
health9       18
garden3       18
computers4    18
health8       17
watches1      17
garden9       17
garden2       17
garden7       16
garden10      16
garden6       16
bed1          16
computers1    15
cool1         15
watches3      15
watches2      15
garden5       14
garden4       14
garden8       14
watches6      14
perfumery2    13
cool2         13
furniture2    13
health2       13
furniture1    13
perfumery1    13
cool5         13
watches7      12
furniture3    12
consoles1     12
health4       11
bed3          11
computers3    10
computers2    10
bed4          10
consoles2     10
watches4      10
watches5      10
furniture4    10
watches8      10
health1        9
cool4          9
computers6     8
computers5     8
health3        8
cool3          7
health10       7
health6        7
bed5           5
Name: product_id, dtype: int64


product_category_name :
garden_tools             160
health_beauty            130
watches_gifts            103
computers_accessories     69
bed_bath_table            61
```

```
cool_stuff                57
furniture_decor           48
perfumery                 26
consoles_games            22
Name: product_category_name, dtype: int64


month_year :
01-03-2018    50
01-02-2018    49
01-01-2018    48
01-04-2018    48
01-11-2017    44
01-12-2017    44
01-10-2017    43
01-06-2018    42
01-05-2018    40
01-07-2018    40
01-08-2018    38
01-08-2017    37
01-09-2017    36
01-07-2017    33
01-06-2017    25
01-05-2017    20
01-04-2017    15
01-03-2017    13
01-02-2017     9
01-01-2017     2
Name: month_year, dtype: int64
```

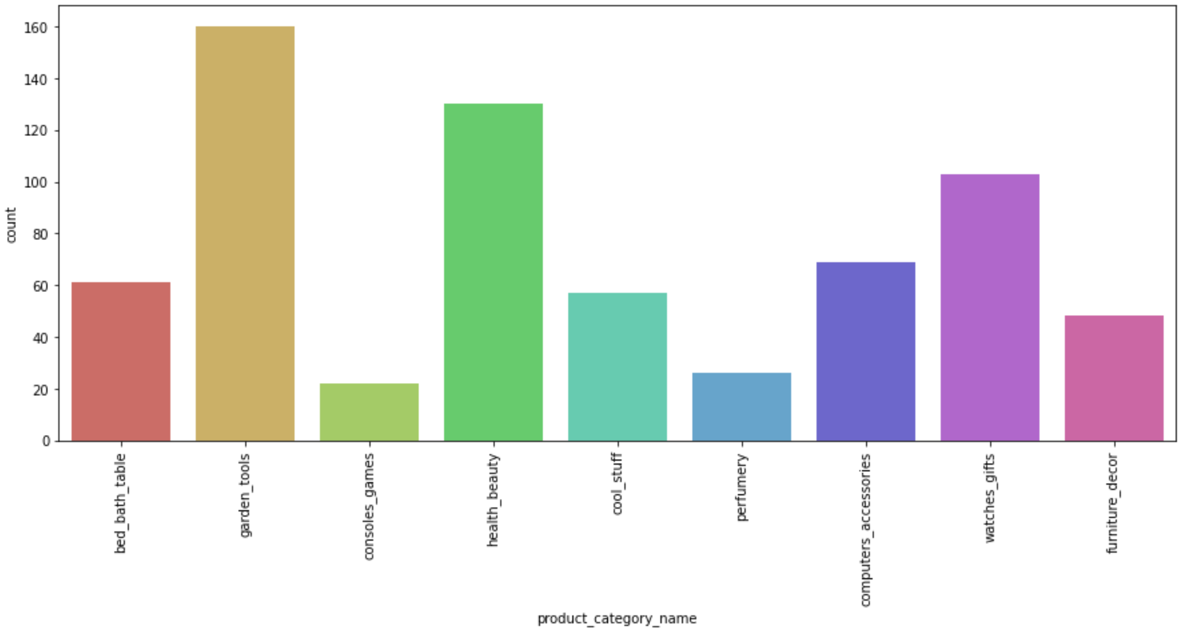
In [16]:

```
for i in object_columns:
    print(i,':')
    plt.figure(figsize=(15,6))
    sns.countplot(df[i], data = df, palette = 'hls')
    plt.xticks(rotation = 90)
    plt.show()
    print('\n')
```

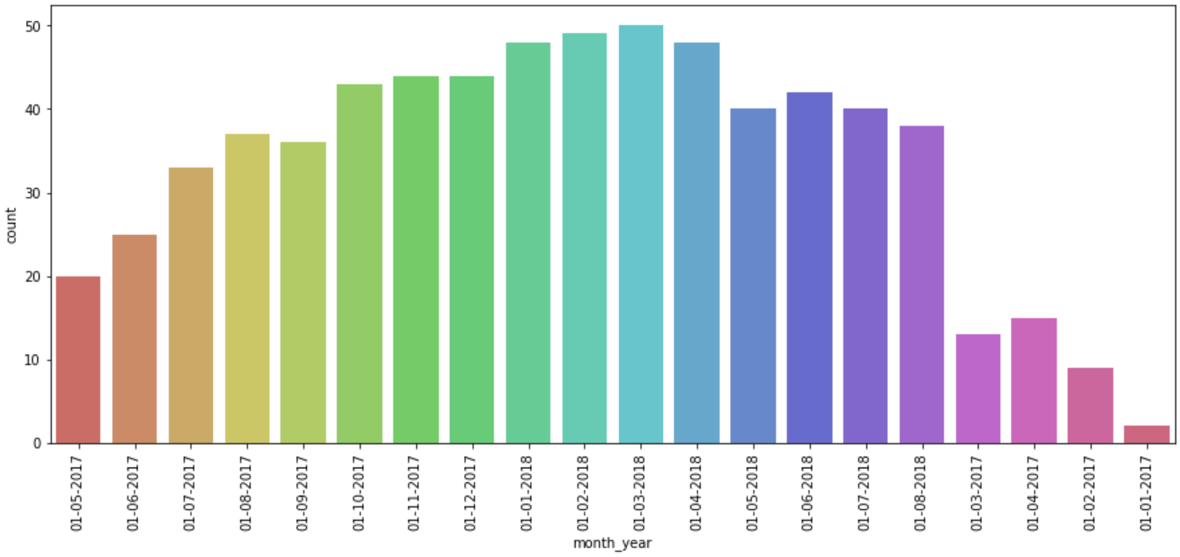
```
product_id :
```



```
product_category_name :
```



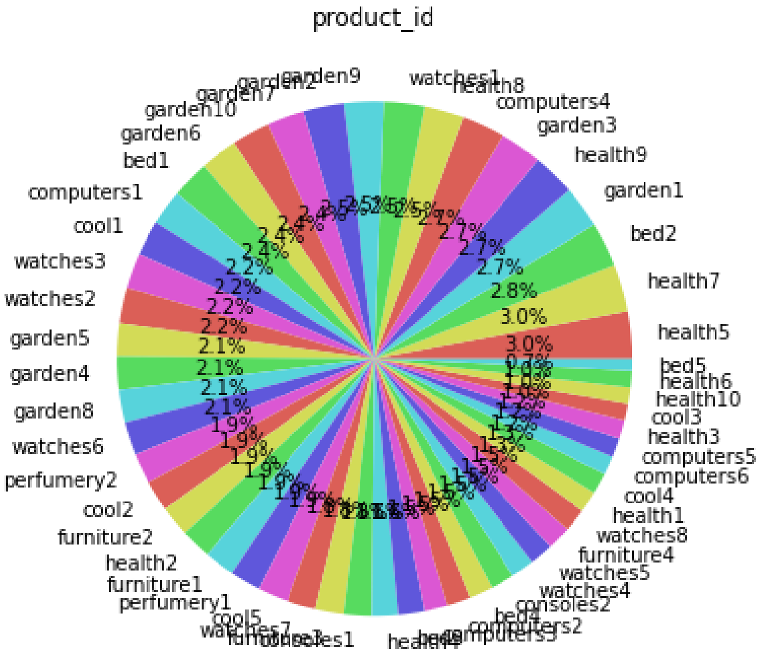
```
month_year :
```


count
0
10
20
30
40
50
01-05-2017
01-06-2017
01-07-2017
01-08-2017
01-09-2017
01-10-2017
01-11-2017
01-12-2017
01-01-2018
01-02-2018
01-03-2018
01-04-2018
01-05-2018
01-06-2018
01-07-2018
01-08-2018
01-03-2017
01-04-2017
01-02-2017
01-01-2017
month_year

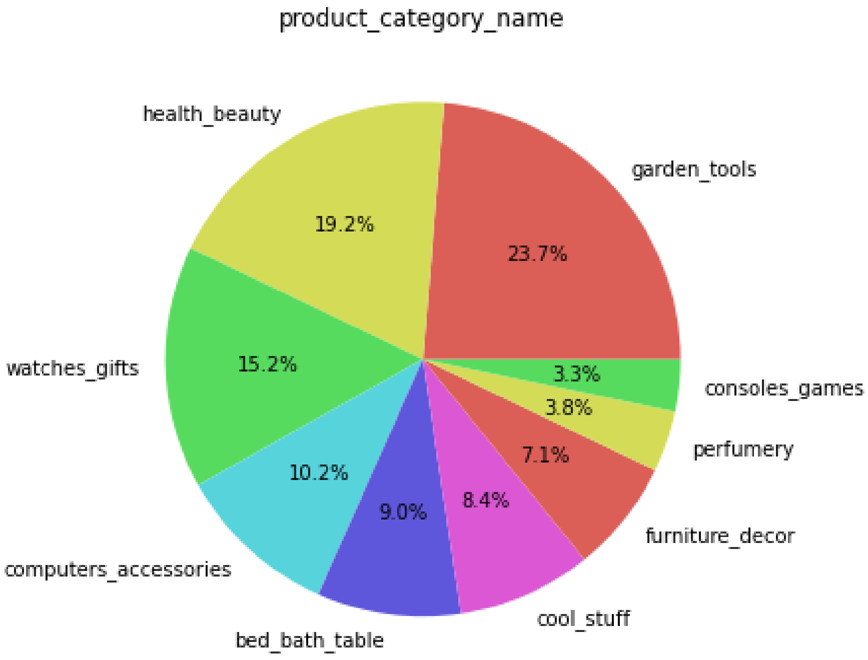
In [17]:

```
for i in object_columns:
    print(i, ':')
    plt.figure(figsize=(15, 6))
    counts = df[i].value_counts()
    plt.pie(counts, labels=counts.index, autopct='%1.1f%%', colors=sns.color_palette('hls'))
    plt.title(i)
    plt.show()
    print('\n')
```

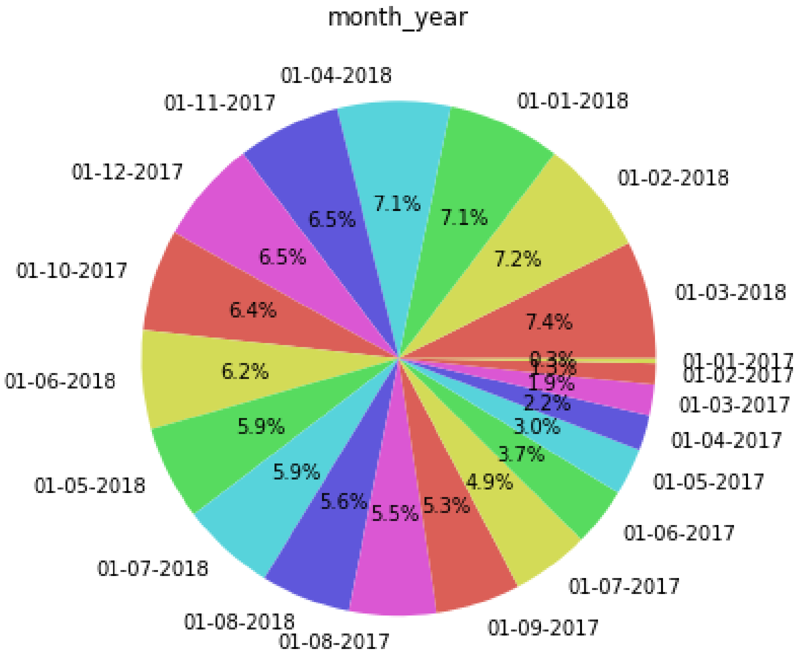
```
product_id :
```



```
product_category_name :
```



```
month_year :
```



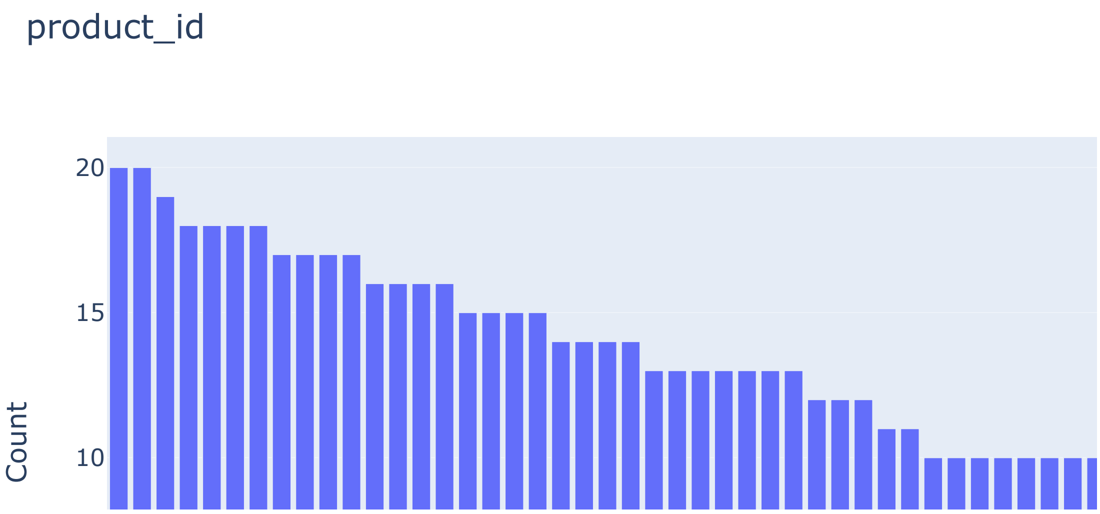
In [18]:

```python
import plotly.express as px
import plotly.graph_objects as go
```

In [19]:

```
for i in object_columns:
    print(i, ':')
    fig = go.Figure(data=[go.Bar(x=df[i].value_counts().index, y=df[i].value_counts())])
    fig.update_layout(
        title=i,
        xaxis_title="Categories",
        yaxis_title="Count"
    )
    fig.show()
    print('\n')
```

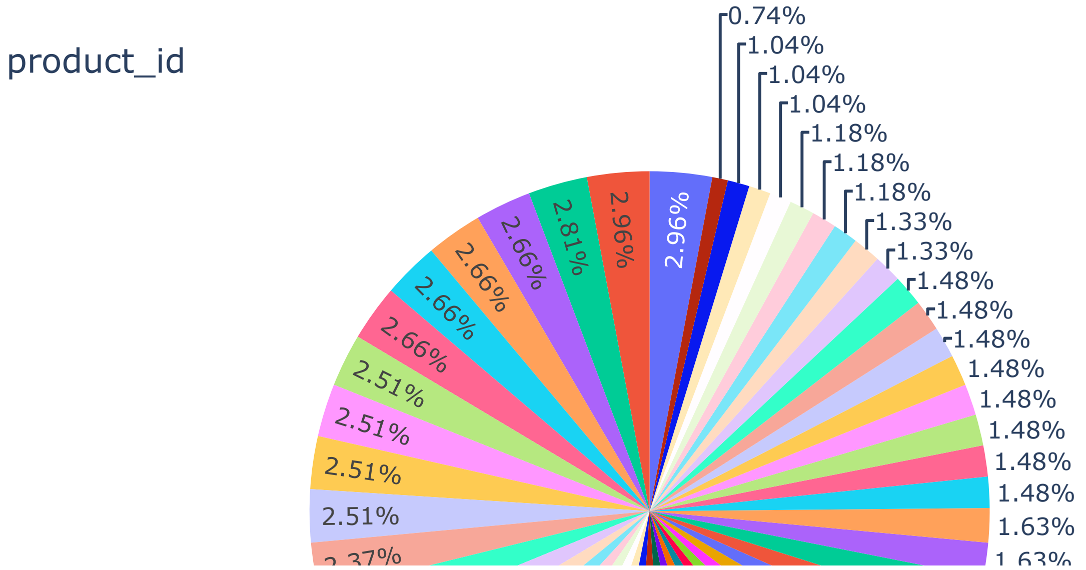
```
product_id :
```



In [20]:

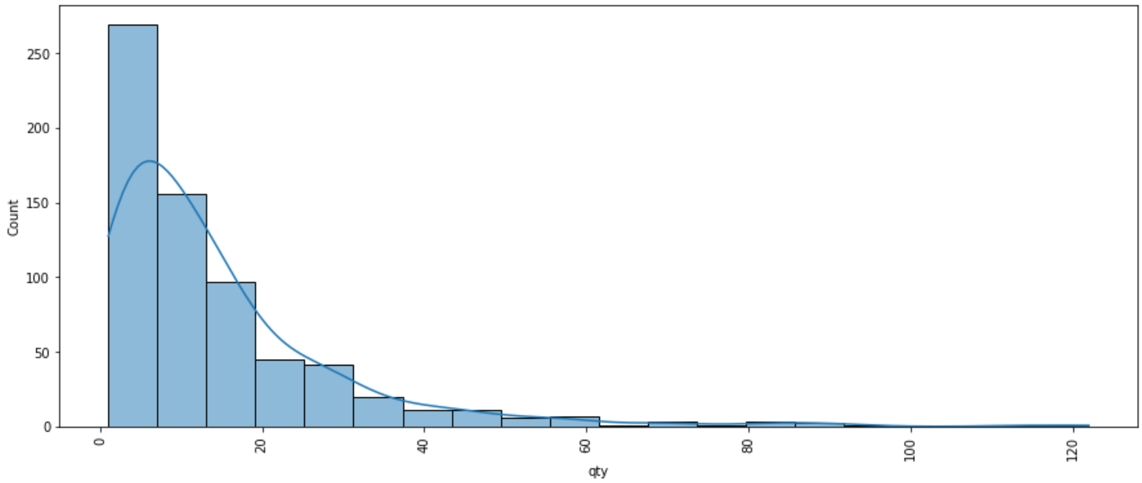
```
for i in object_columns:
    print(i, ':')
    counts = df[i].value_counts()
    fig = go.Figure(data=[go.Pie(labels=counts.index, values=counts)])
    fig.update_layout(title=i)
    fig.show()
    print('\n')
```

```
product_id :
```



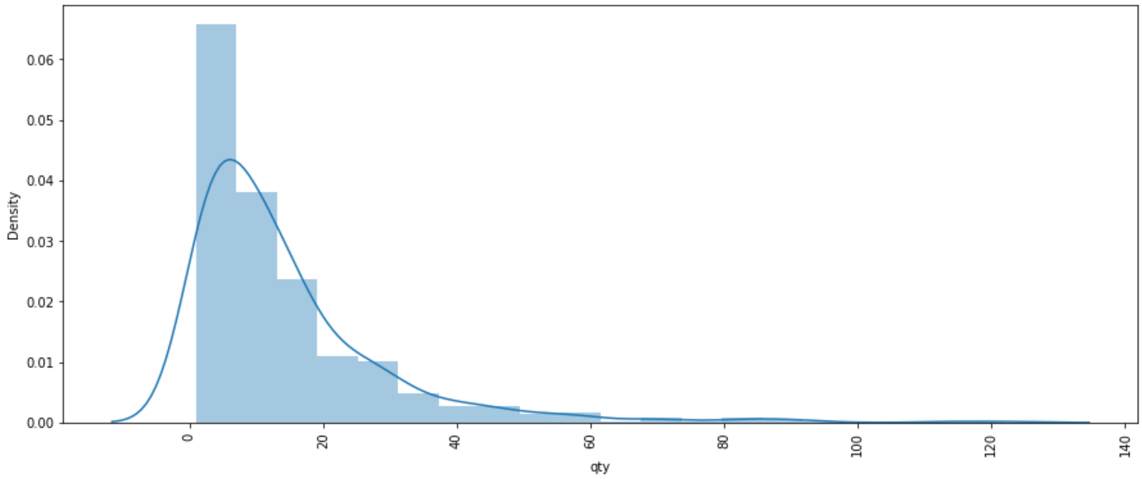
In [21]:

```
for i in numerical_columns:
    plt.figure(figsize=(15,6))
    sns.histplot(df[i], kde = True, bins = 20, palette = 'hls')
    plt.xticks(rotation = 90)
    plt.show()
```



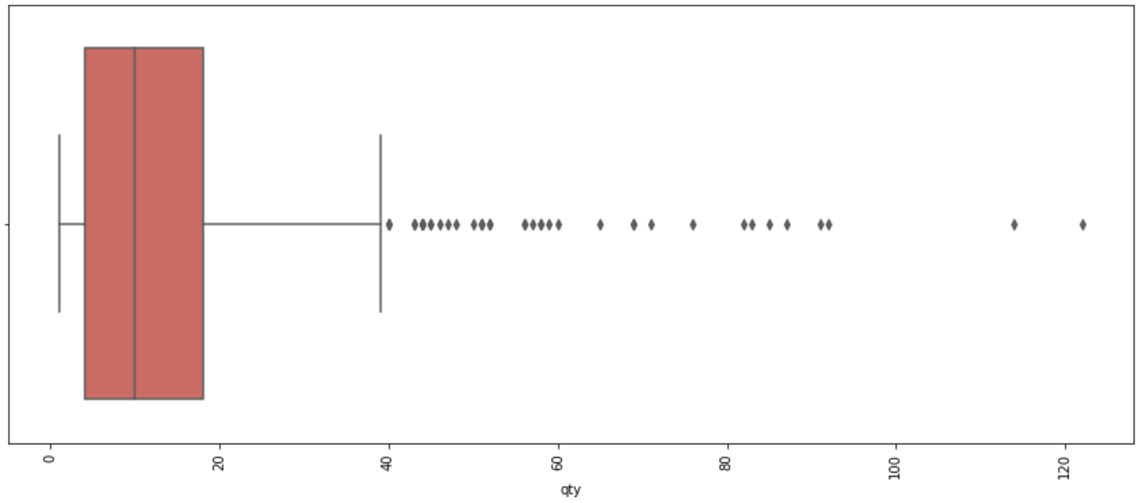
In [22]:

```
for i in numerical_columns:
    plt.figure(figsize=(15,6))
    sns.distplot(df[i], kde = True, bins = 20)
    plt.xticks(rotation = 90)
    plt.show()
```



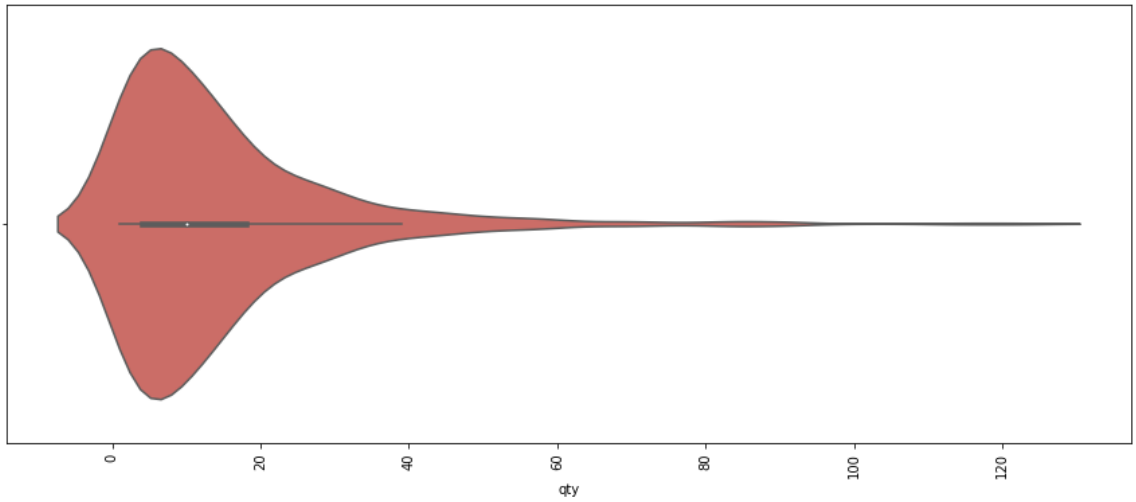
In [23]:

```
for i in numerical_columns:
    plt.figure(figsize=(15,6))
    sns.boxplot(df[i], data = df, palette = 'hls')
    plt.xticks(rotation = 90)
    plt.show()
```



In [24]:

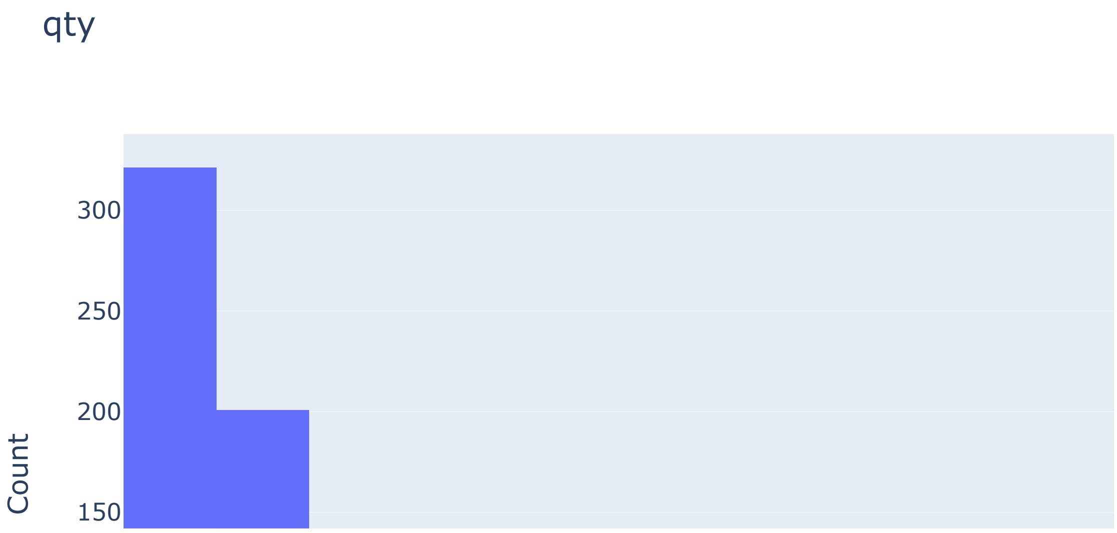
```python
for i in numerical_columns:
    plt.figure(figsize=(15,6))
    sns.violinplot(df[i], data = df, palette = 'hls')
    plt.xticks(rotation = 90)
    plt.show()
```



In [25]:

```python
for i in numerical_columns:
    print(i, ':')
    fig = go.Figure(data=[go.Histogram(x=df[i], nbinsx=20)])
    fig.update_layout(
        title=i,
        xaxis_title="Value",
        yaxis_title="Count"
    )
    fig.show()
    print('\n')
```

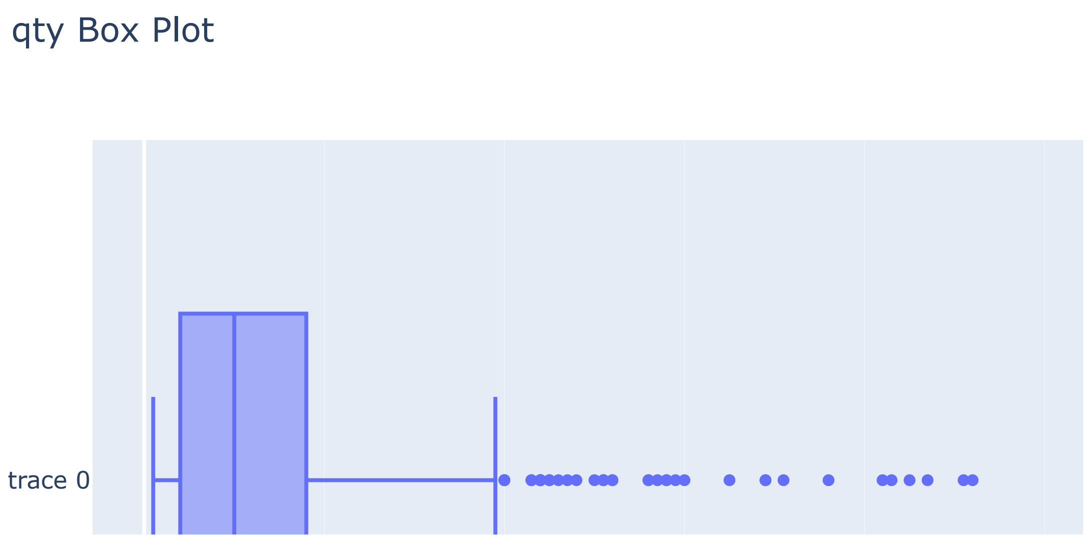
```
qty :
```



In [26]:

```
for i in numerical_columns:
    print(i, ':')
    fig = go.Figure(data=[go.Box(x=df[i])])
    fig.update_layout(
        title=i + ' Box Plot',
        xaxis_title="Value"
    )
    fig.show()
    print('\n')
```

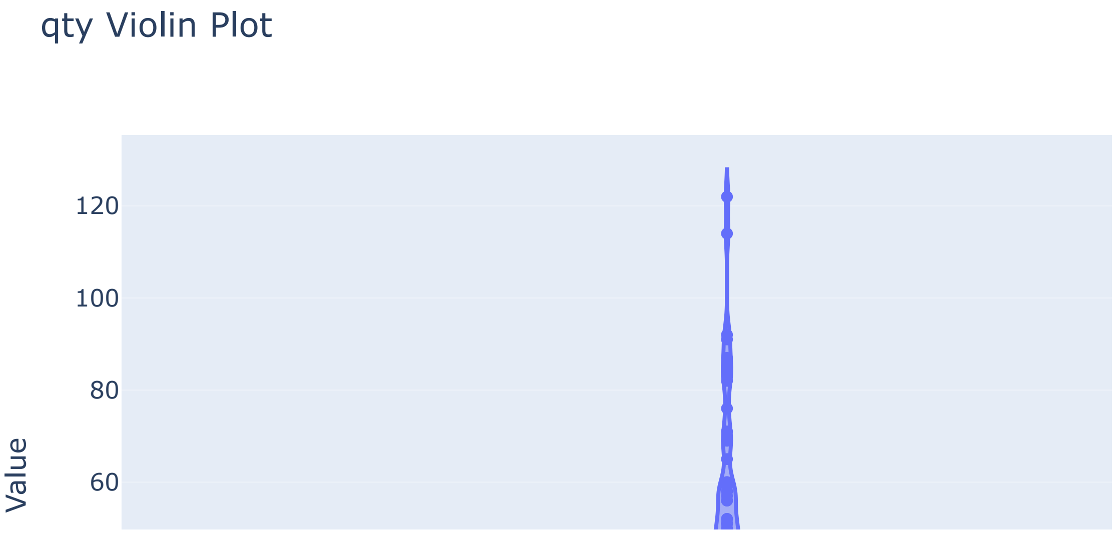
```
qty :
```



In [27]:

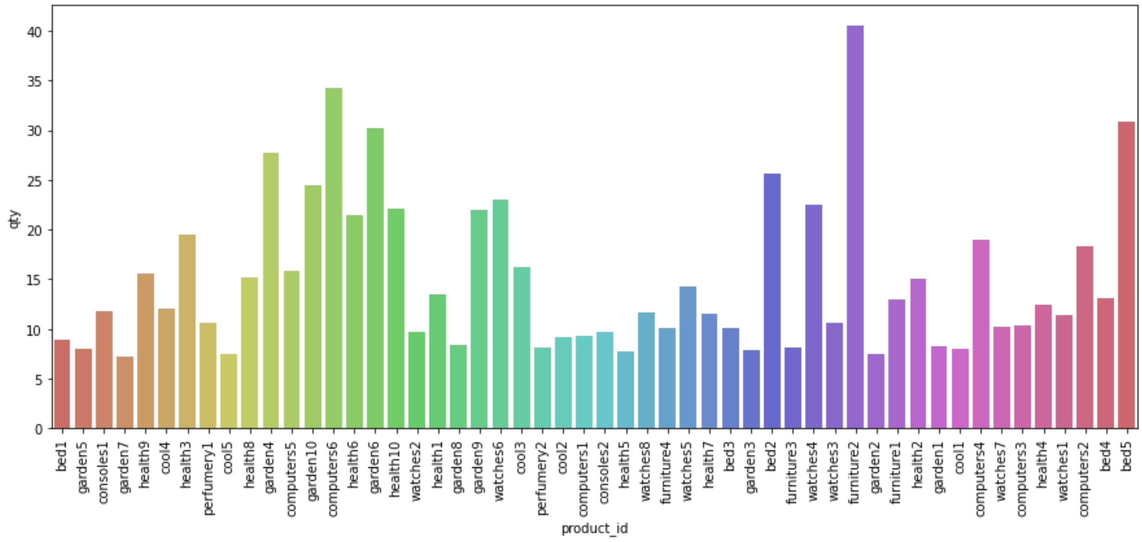
```
for i in numerical_columns:
    print(i, ':')
    fig = go.Figure(data=[go.Violin(y=df[i])])
    fig.update_layout(
        title=i + ' Violin Plot',
        yaxis_title="Value"
    )
    fig.show()
    print('\n')
```

```
qty :
```



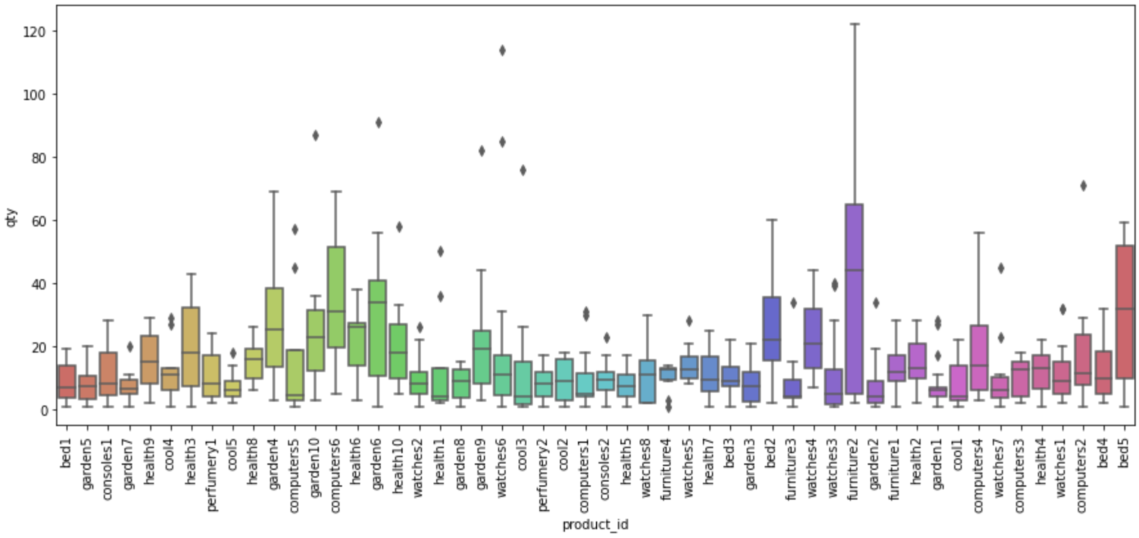
In [28]:

```
for i in numerical_columns:
    for j in object_columns:
        plt.figure(figsize=(15,6))
        sns.barplot(x = df[j], y = df[i], data = df, ci = None, palette = 'hls')
        plt.xticks(rotation = 90)
        plt.show()
```



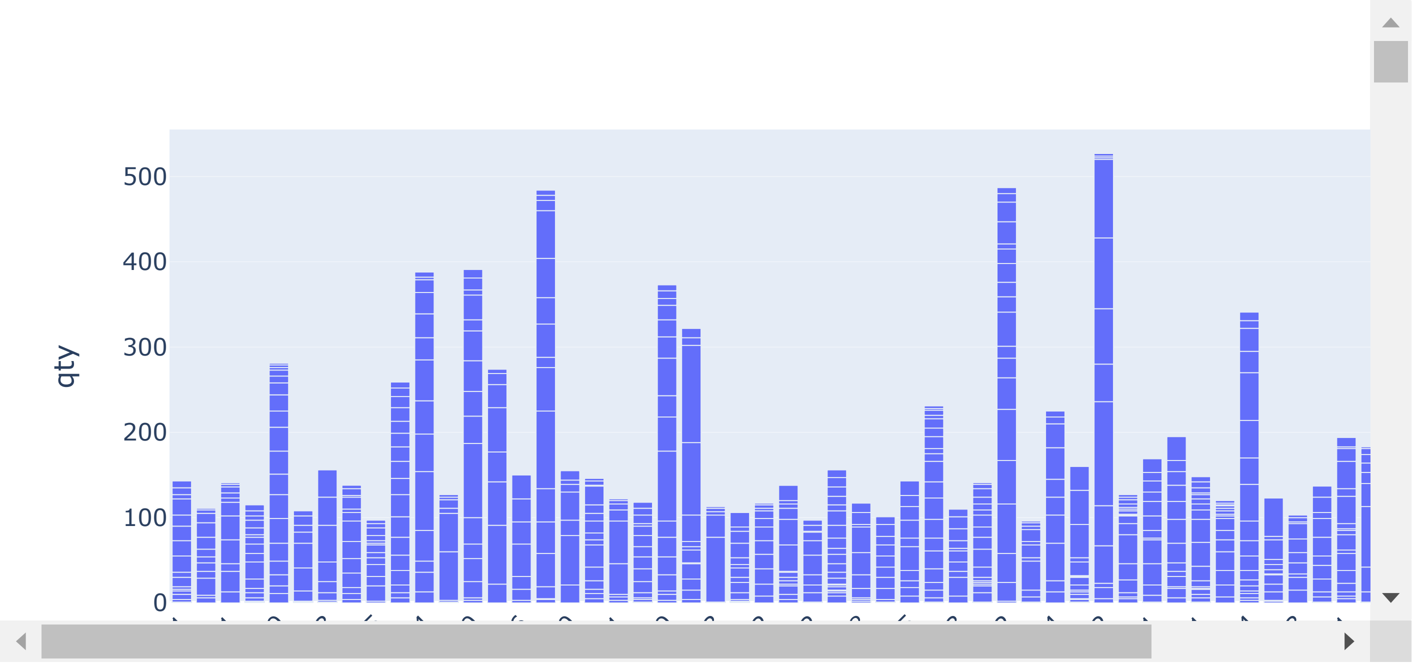
In [29]:

```
for i in numerical_columns:
    for j in object_columns:
        plt.figure(figsize=(15,6))
        sns.boxplot(x = df[j], y = df[i], data = df, palette = 'hls')
        plt.xticks(rotation = 90)
        plt.show()
```



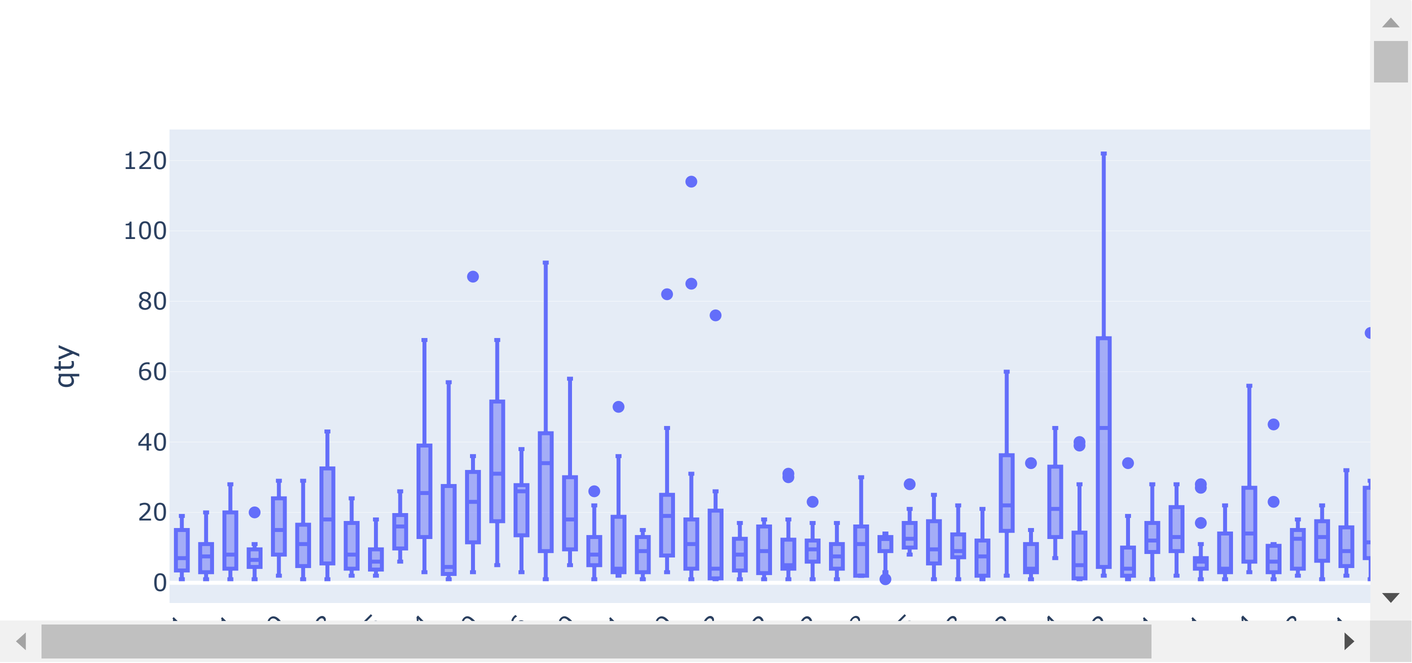
In [30]:

```
for i in numerical_columns:
    # Iterate over the object columns
    for j in object_columns:
        # Create a bar plot using Plotly
        fig = px.bar(df, x=j, y=i)

        # Set the figure size
        fig.update_layout(width=800, height=400)

        # Rotate x-axis labels if necessary
        fig.update_layout(xaxis_tickangle=-45)

        # Show the plot
        fig.show()
```



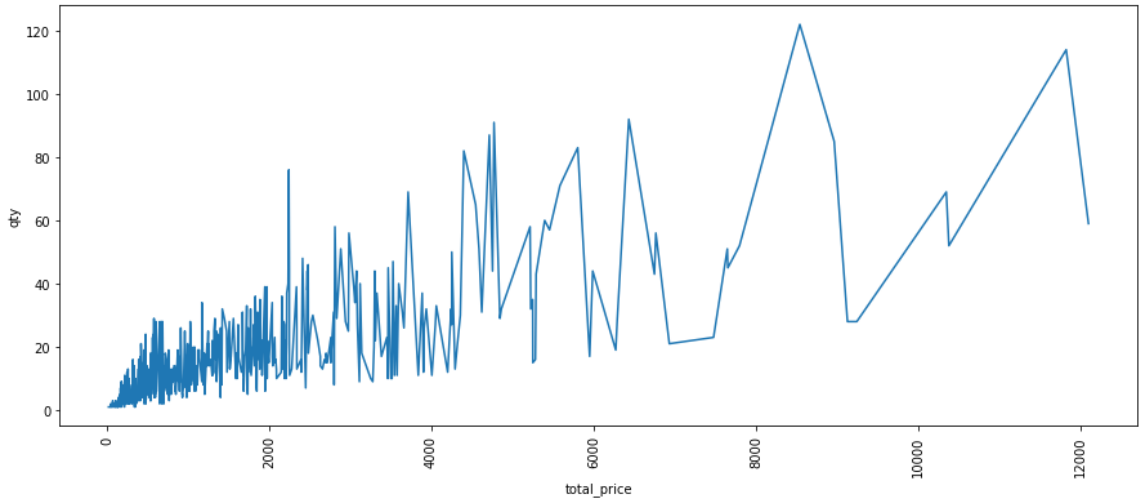
In [31]:

```
for i in numerical_columns:
    # Iterate over the object columns
    for j in object_columns:
        # Create a box plot using Plotly
        fig = px.box(df, x=j, y=i)

        # Set the figure size
        fig.update_layout(width=800, height=400)

        # Rotate x-axis labels if necessary
        fig.update_layout(xaxis_tickangle=-45)

        # Show the plot
        fig.show()
```



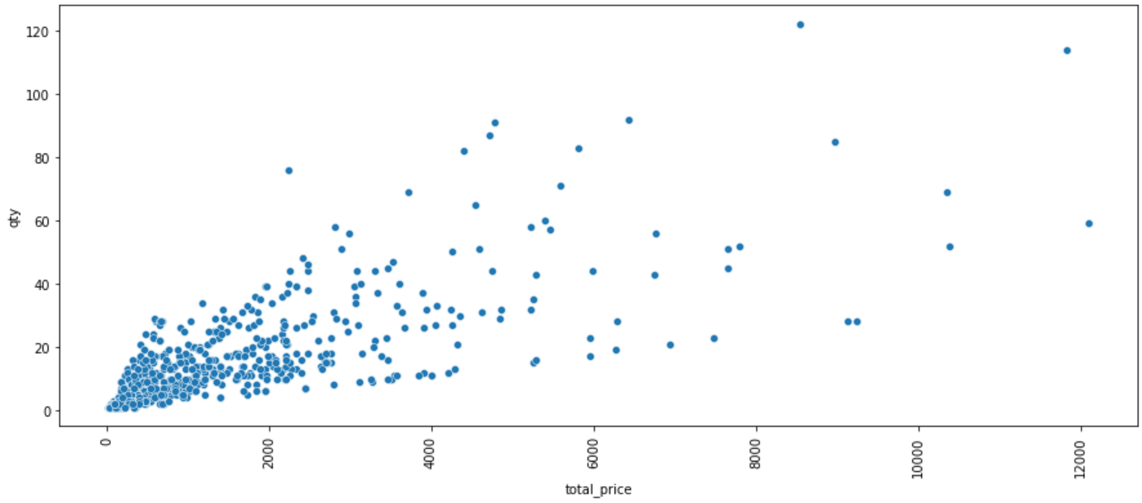
In [32]:

```
for i in numerical_columns:
    for j in numerical_columns:
        if i != j:
            plt.figure(figsize=(15,6))
            sns.lineplot(x = df[j], y = df[i], data = df, ci = None, palette = 'hls')
            plt.xticks(rotation = 90)
            plt.show()
```



In [33]:

```
for i in numerical_columns:
    for j in numerical_columns:
        if i != j:
            plt.figure(figsize=(15,6))
            sns.scatterplot(x = df[j], y = df[i], data = df, ci = None, palette = 'hls')
            plt.xticks(rotation = 90)
            plt.show()
```



Calculate Revenue and Profit:

In [34]:

```
df['revenue'] = df['qty'] * df['total_price']
df['profit'] = df['revenue'] - df['freight_price']
```

Calculate Margin:

In [35]:

```
df['margin'] = (df['profit'] / df['revenue']) * 100
```

Price Ratios:

In [36]:

```
df['price_ratio_1'] = df['unit_price'] / df['comp_1']
df['price_ratio_2'] = df['unit_price'] / df['comp_2']
df['price_ratio_3'] = df['unit_price'] / df['comp_3']
```

Price Differences:

In [37]:

```
df['price_diff_1'] = df['unit_price'] - df['comp_1']
df['price_diff_2'] = df['unit_price'] - df['comp_2']
df['price_diff_3'] = df['unit_price'] - df['comp_3']
```

In [38]:

```
df.head()
```

Out[38]:

	product_id	product_category_name	month_year	qty	total_price	freight_price	unit_price	proc
0	bed1	bed_bath_table	01-05-2017	1	45.95	15.100000	45.95	
1	bed1	bed_bath_table	01-06-2017	3	137.85	12.933333	45.95	
2	bed1	bed_bath_table	01-07-2017	6	275.70	14.840000	45.95	
3	bed1	bed_bath_table	01-08-2017	4	183.80	14.287500	45.95	
4	bed1	bed_bath_table	01-09-2017	2	91.90	15.100000	45.95	

5 rows × 39 columns

Market Demand Indicators:

In [39]:

```
df['customer_score_ratio'] = df['customers'] / df['product_score']
df['customer_photo_ratio'] = df['customers'] / df['product_photos_qty']
df['description_length_ratio'] = df['product_description_lenght'] / df['product_name_lenght'
```

In [40]:

```
df.head()
```

Out[40]:

	product_id	product_category_name	month_year	qty	total_price	freight_price	unit_price	pro
0	bed1	bed_bath_table	01-05-2017	1	45.95	15.100000	45.95	
1	bed1	bed_bath_table	01-06-2017	3	137.85	12.933333	45.95	
2	bed1	bed_bath_table	01-07-2017	6	275.70	14.840000	45.95	
3	bed1	bed_bath_table	01-08-2017	4	183.80	14.287500	45.95	
4	bed1	bed_bath_table	01-09-2017	2	91.90	15.100000	45.95	

5 rows × 42 columns

Time-related Features:

In [41]:

```
df['month_year'] = pd.to_datetime(df['month_year'])
df['month'] = df['month_year'].dt.month
df['year'] = df['month_year'].dt.year
df['is_weekend'] = df['weekday'].apply(lambda x: 1 if x >= 5 else 0)
df['is_holiday'] = df['holiday']
```

In [42]:

```
df.head()
```

Out[42]:

	product_id	product_category_name	month_year	qty	total_price	freight_price	unit_price	proc
0	bed1	bed_bath_table	2017-01-05	1	45.95	15.100000	45.95	
1	bed1	bed_bath_table	2017-01-06	3	137.85	12.933333	45.95	
2	bed1	bed_bath_table	2017-01-07	6	275.70	14.840000	45.95	
3	bed1	bed_bath_table	2017-01-08	4	183.80	14.287500	45.95	
4	bed1	bed_bath_table	2017-01-09	2	91.90	15.100000	45.95	

5 rows × 44 columns

Lagged Price:

In [43]:

```
df['lag_price'] = df.groupby('product_id')['total_price'].shift(1)
```

Handling Categorical Variables:

In [44]:

```
# One-hot encoding example
df_encoded = pd.get_dummies(df, columns=['product_category_name', 'weekday'])
```

Scaling Numeric Features:

In [45]:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
numeric_features = ['qty', 'total_price', 'freight_price', 'unit_price', 'product_name_lenght
                    'product_description_lenght', 'product_photos_qty', 'product_weight_g',
                    'product_score', 'customers', 'volume', 'comp_1', 'ps1', 'fp1', 'comp_2'
                    'ps2', 'fp2', 'comp_3', 'ps3', 'fp3', 'lag_price']

df_scaled = df_encoded.copy()
df_scaled[numeric_features] = scaler.fit_transform(df_encoded[numeric_features])
```

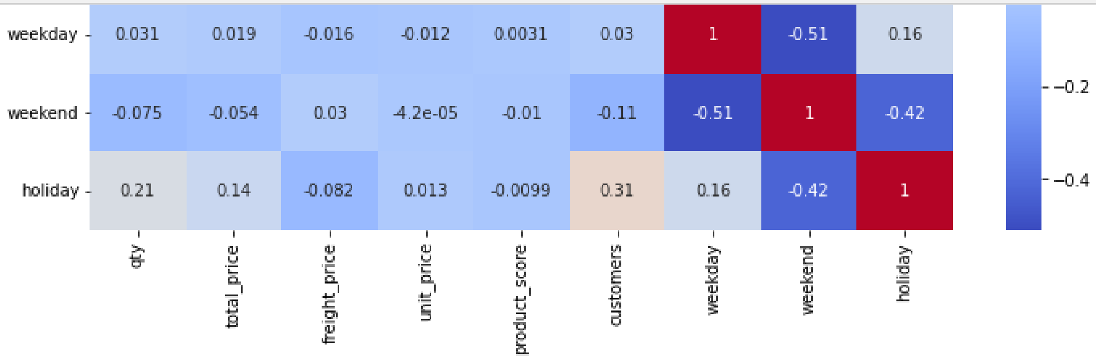
Splitting the Dataset:

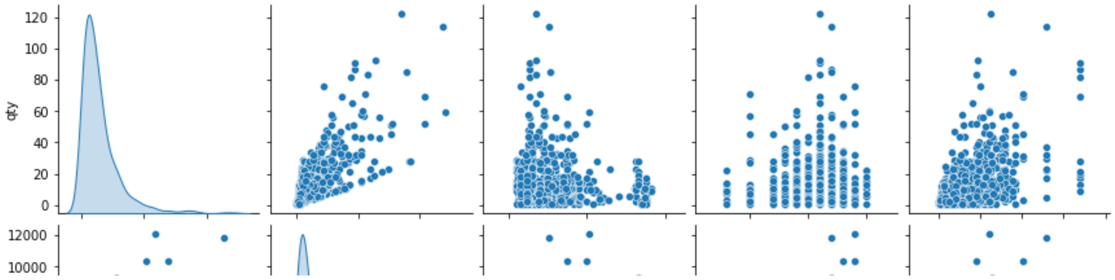
In [46]:

```
from sklearn.model_selection import train_test_split
exclude_columns = ['month_year']
X = df_scaled.drop(['total_price', 'product_id'] + exclude_columns, axis=1)
y = df_scaled['total_price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [47]:

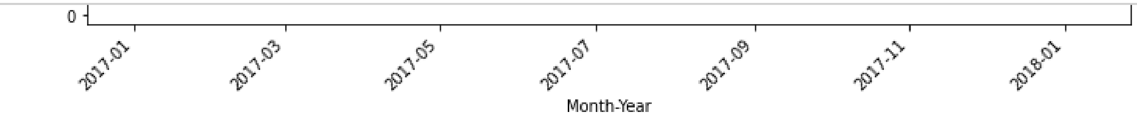
```
subset_cols = ['qty', 'total_price', 'freight_price', 'unit_price', 'product_score', 'custome
subset_df = df[subset_cols]

corr_matrix = subset_df.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

sns.pairplot(subset_df, vars=['qty', 'total_price', 'unit_price', 'product_score', 'customers
plt.show()

plt.figure(figsize=(12, 8))
sns.boxplot(x='product_category_name', y='total_price', data=df)
plt.title('Product Category vs. Total Price')
plt.xticks(rotation=90)
plt.show()

plt.figure(figsize=(10, 6))
sns.barplot(x='weekday', y='total_price', data=df)
plt.title('Weekday vs. Total Price')
plt.show()

plt.figure(figsize=(8, 6))
sns.countplot(x='holiday', data=df, hue='total_price')
plt.title('Holiday vs. Total Price')
plt.show()
```

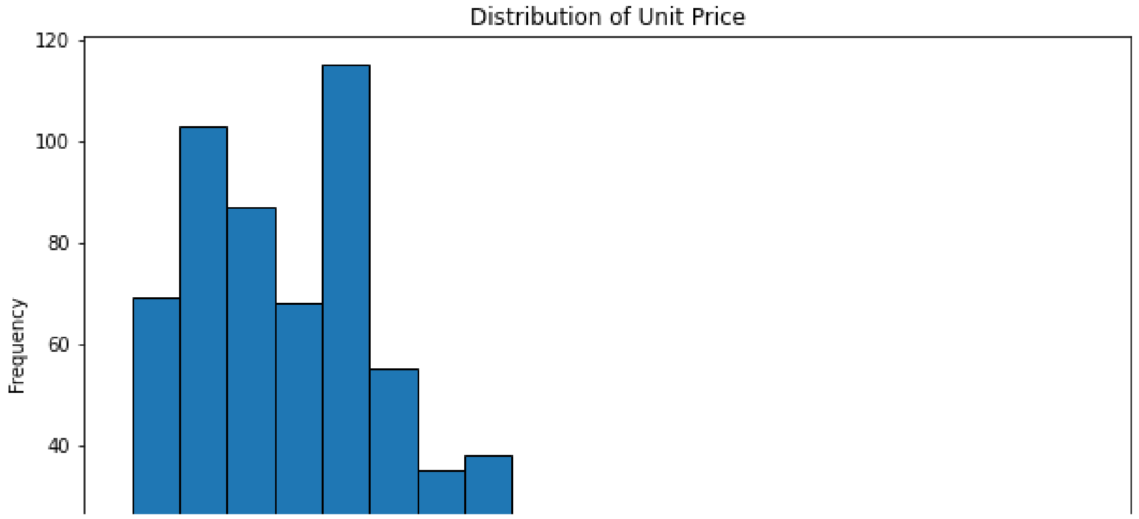




In [48]:

```
plt.figure(figsize=(12, 8))
df.groupby('product_category_name')['total_price'].mean().sort_values().plot(kind='bar')
plt.title('Product Category vs. Average Total Price')
plt.xlabel('Product Category')
plt.ylabel('Average Total Price')
plt.xticks(rotation=90)
plt.show()


plt.figure(figsize=(12, 8))
df.groupby('month_year')['total_price'].sum().plot(kind='line')
plt.title('Total Price Trend over Time')
plt.xlabel('Month-Year')
plt.ylabel('Total Price')
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(10, 6))
plt.hist(df['unit_price'], bins=20, edgecolor='k')
plt.title('Distribution of Unit Price')
plt.xlabel('Unit Price')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(df['product_weight_g'], df['total_price'])
plt.title('Product Weight vs. Total Price')
plt.xlabel('Product Weight (grams)')
plt.ylabel('Total Price')
plt.show()
```





In [49]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

model = LinearRegression()
```

In [50]:

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

In [51]:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
```

In [52]:

```
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)
```

In [53]:

```
model.fit(X_train_imputed, y_train)
```

Out[53]:

```
▾ LinearRegression
LinearRegression()
```

In [54]:

```
y_pred = model.predict(X_test_imputed)
```

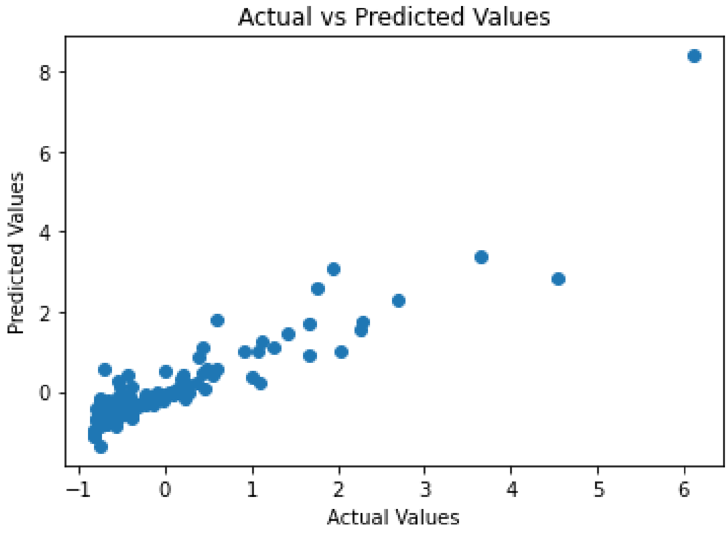
In [55]:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Calculate mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calculate root mean squared error (RMSE)
rmse = mean_squared_error(y_test, y_pred, squared=False)

# Calculate R-squared score
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared Score:", r2)
```

```
Mean Squared Error (MSE): 0.17492913184386694
Root Mean Squared Error (RMSE): 0.41824530104218377
R-squared Score: 0.8401763893992591
```

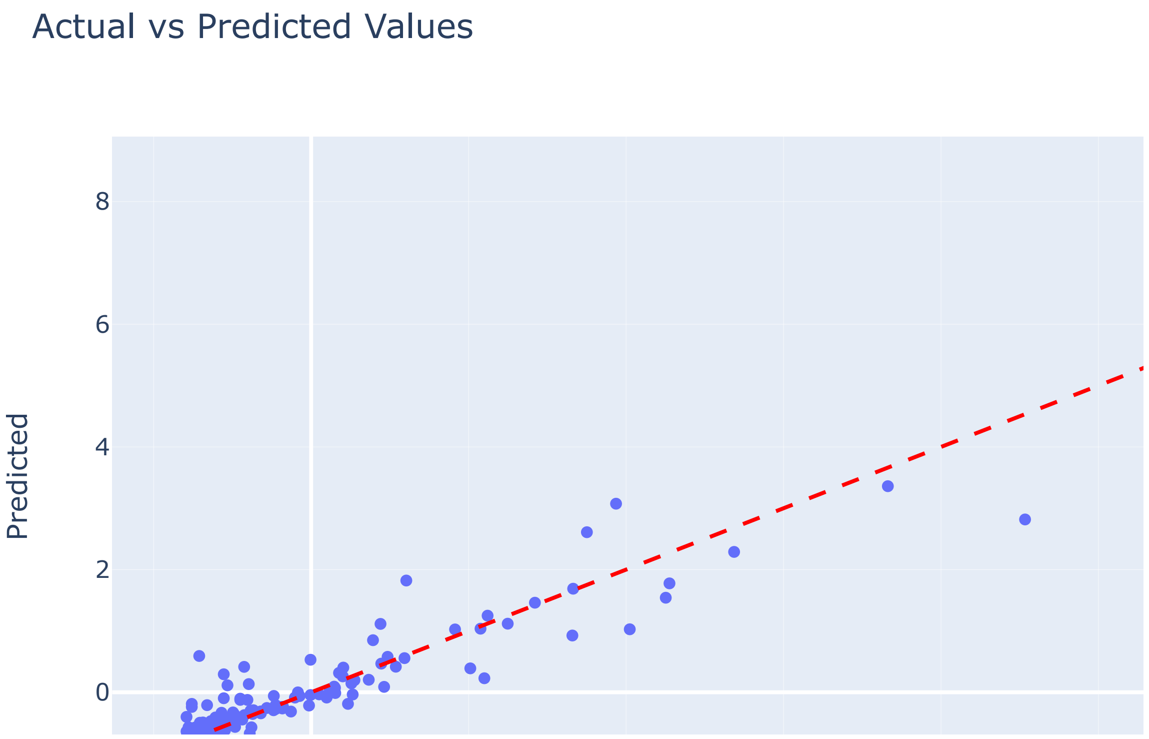
In [56]:

```
# Plotting the predicted values against the actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()
```



In [57]:

```
# Create a DataFrame with actual and predicted values
df_plot = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Create a scatter plot
fig = px.scatter(df_plot, x='Actual', y='Predicted', title='Actual vs Predicted Values')

# Add a diagonal line for reference
fig.add_shape(type='line', x0=df_plot['Actual'].min(), y0=df_plot['Actual'].min(),
              x1=df_plot['Actual'].max(), y1=df_plot['Actual'].max(),
              line=dict(color='red', dash='dash'))

# Show the plot
fig.show()
```



In [58]:

```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()
```

In [59]:

```
model.fit(X_train_imputed, y_train)
```

Out[59]:

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

In [60]:

```
y_pred = model.predict(X_test_imputed)
```

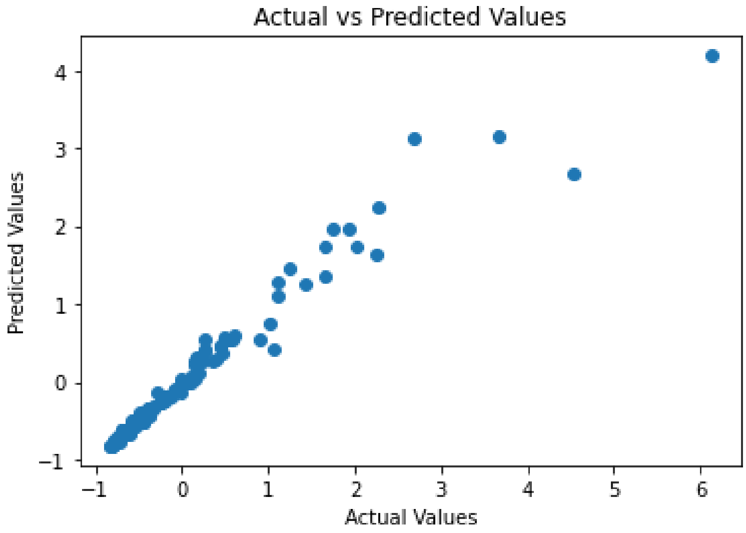
In [61]:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Calculate mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calculate root mean squared error (RMSE)
rmse = mean_squared_error(y_test, y_pred, squared=False)

# Calculate R-squared score
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared Score:", r2)
```

```
Mean Squared Error (MSE): 0.06924014365516341
Root Mean Squared Error (RMSE): 0.26313521933630135
R-squared Score: 0.9367388973989801
```

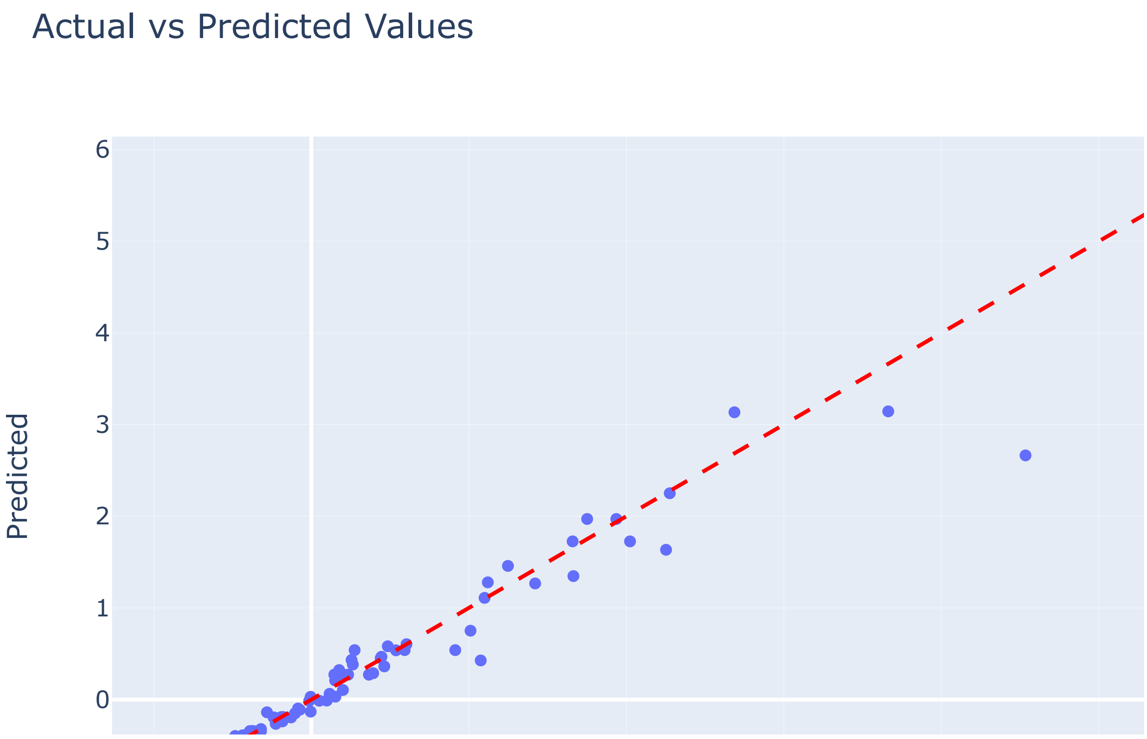
In [62]:

```
# Plotting the predicted values against the actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()
```



In [63]:

```
# Create a DataFrame with actual and predicted values
df_plot = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Create a scatter plot
fig = px.scatter(df_plot, x='Actual', y='Predicted', title='Actual vs Predicted Values')

# Add a diagonal line for reference
fig.add_shape(type='line', x0=df_plot['Actual'].min(), y0=df_plot['Actual'].min(),
              x1=df_plot['Actual'].max(), y1=df_plot['Actual'].max(),
              line=dict(color='red', dash='dash'))

# Show the plot
fig.show()
```


Actual vs Predicted Values
Predicted