Importing the packages

```
!pip install nbconvert

import kaggle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

download data set

```
!kaggle datasets download devarajv88/walmart-sales-dataset -f
walmart.csv
```

```
Dataset URL: https://www.kaggle.com/datasets/devarajv88/walmart-sales-
dataset
License(s): other
walmart.csv.zip: Skipping, found more recently modified local copy
(use --force to force download)
```

Extract the data

```
import zipfile
zip_ref = zipfile.ZipFile('walmart.csv.zip')
zip_ref.extractall() # extracting the data
zip_ref.close() # close the file
print("Data has been extracted successfully")
```

```
Data has been extracted successfully
```

Read the data and handle null values

```
df = pd.read_csv('walmart.csv')
df.head()
```

```
   User_ID Product_ID Gender   Age  Occupation City_Category  \
0  1000001  P00069042      F  0-17          10             A
1  1000001  P00248942      F  0-17          10             A
2  1000001  P00087842      F  0-17          10             A
3  1000001  P00085442      F  0-17          10             A
4  1000002  P00285442      M   55+          16             C

  Stay_In_Current_City_Years  Marital_Status  Product_Category
Purchase
0                          2               0                 3
8370
```

```
1                            2                  0                   1
15200
2                            2                  0                  12
1422
3                            2                  0                  12
1057
4                           4+                  0                   8
7969
```

```
# null values
```

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

```
User_ID                       0
Product_ID                    0
Gender                        0
Age                           0
Occupation                    0
City_Category                 0
Stay_In_Current_City_Years    0
Marital_Status                0
Product_Category              0
Purchase                      0
dtype: int64
```

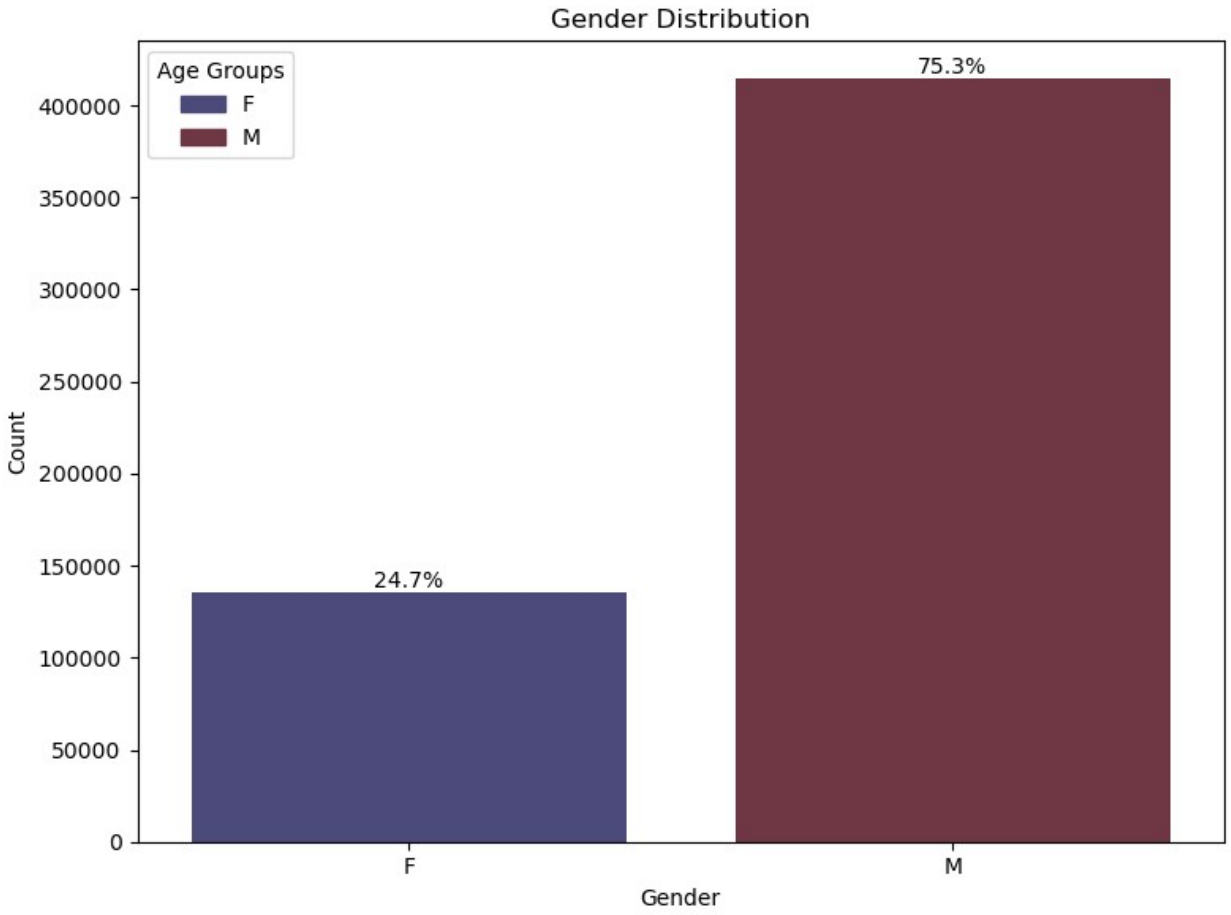
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 10 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   User_ID                     550068 non-null  int64
 1   Product_ID                  550068 non-null  object
 2   Gender                      550068 non-null  object
 3   Age                         550068 non-null  object
 4   Occupation                  550068 non-null  int64
 5   City_Category               550068 non-null  object
 6   Stay_In_Current_City_Years  550068 non-null  object
 7   Marital_Status              550068 non-null  int64
 8   Product_Category            550068 non-null  int64
 9   Purchase                    550068 non-null  int64
dtypes: int64(5), object(5)
memory usage: 42.0+ MB
```

Exploratory Data Analysis

```
import matplotlib.patches as mpatches
```

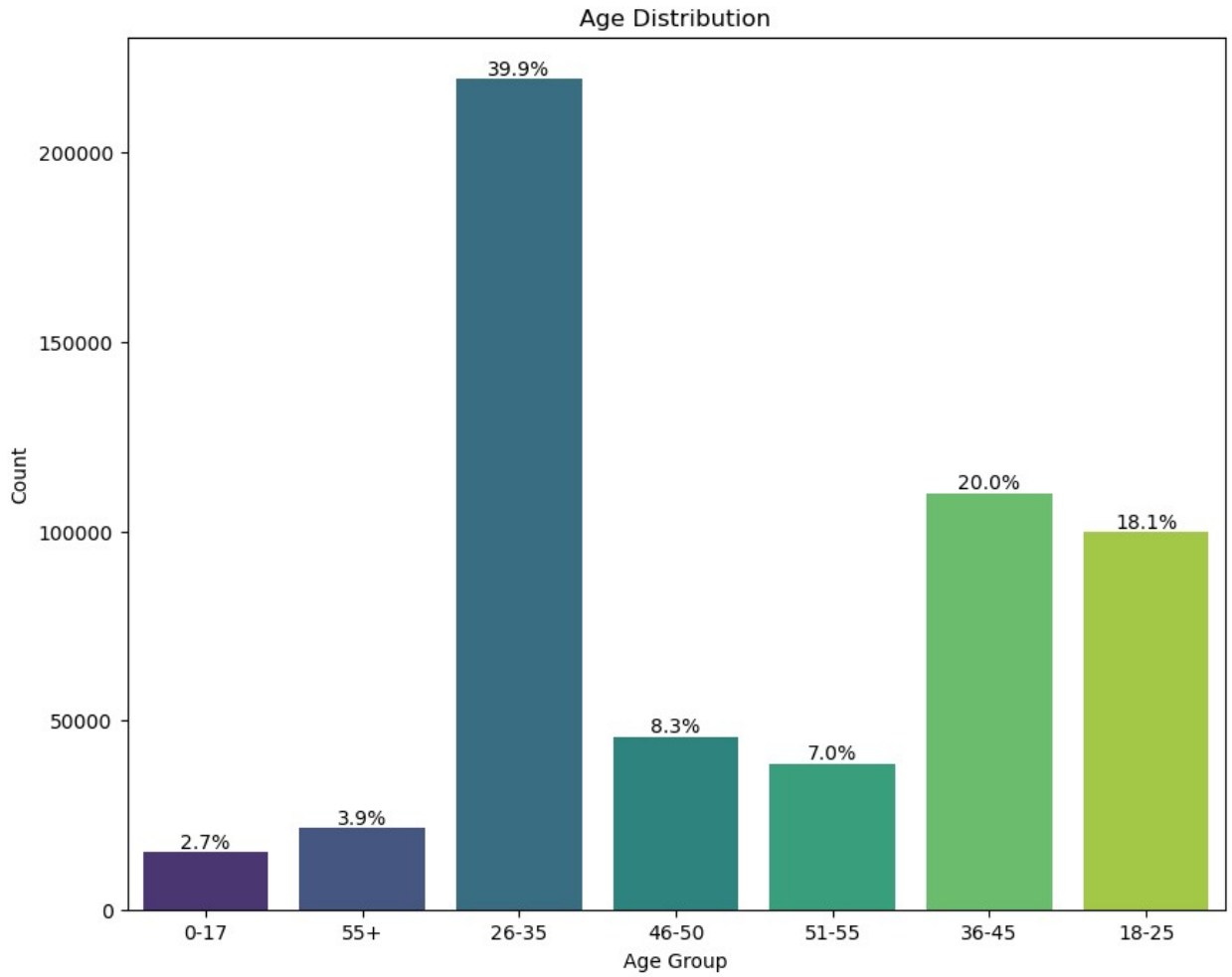
```
# Gender Distribution

plt.figure(figsize = (8,6))
ax = sns.countplot(x='Gender', data = df, palette = 'icefire')

#calculate percentage

total = len(df['Gender'])
for p in ax.patches:
    percentage = f'{100 * p.get_height()/total:.1f}%'
    x = p.get_x() + p.get_width()/2
    y = p.get_height()
    ax.annotate(percentage, (x,y), ha = 'center', va = 'bottom')

# setting title and labels

ax.set_title('Gender Distribution')
ax.set_xlabel('Gender')
ax.set_ylabel('Count')

#creating legend
color = [p.get_facecolor() for p in ax.patches]
gender_groups = df['Gender'].unique()
gender_groups.sort()
patches = [mpatches.Patch(color = color[i], label = age_group) for i,
age_group in enumerate(gender_groups)]
ax.legend(handles = patches, title = 'Age Groups')

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




```
#Age Distribution

plt.figure(figsize = (10,8))
ax = sns.countplot(x = 'Age', data = df, palette = 'viridis')

#Calculate percentages

total = len(df['Age'])
for p in ax.patches:
    percentage = f'{100* p.get_height() / total:.1f}%'
    x = p.get_x() + p.get_width() /2
    y = p.get_height()
    ax.annotate(percentage, (x,y), ha = 'center', va = 'bottom')

#Setting title and labels
ax.set_title('Age Distribution')
ax.set_xlabel('Age Group')
ax.set_ylabel('Count')

#Creating legend
colors = [p.get_facecolor() for p in ax.patches]
age_groups = df['Age'].unique()
```

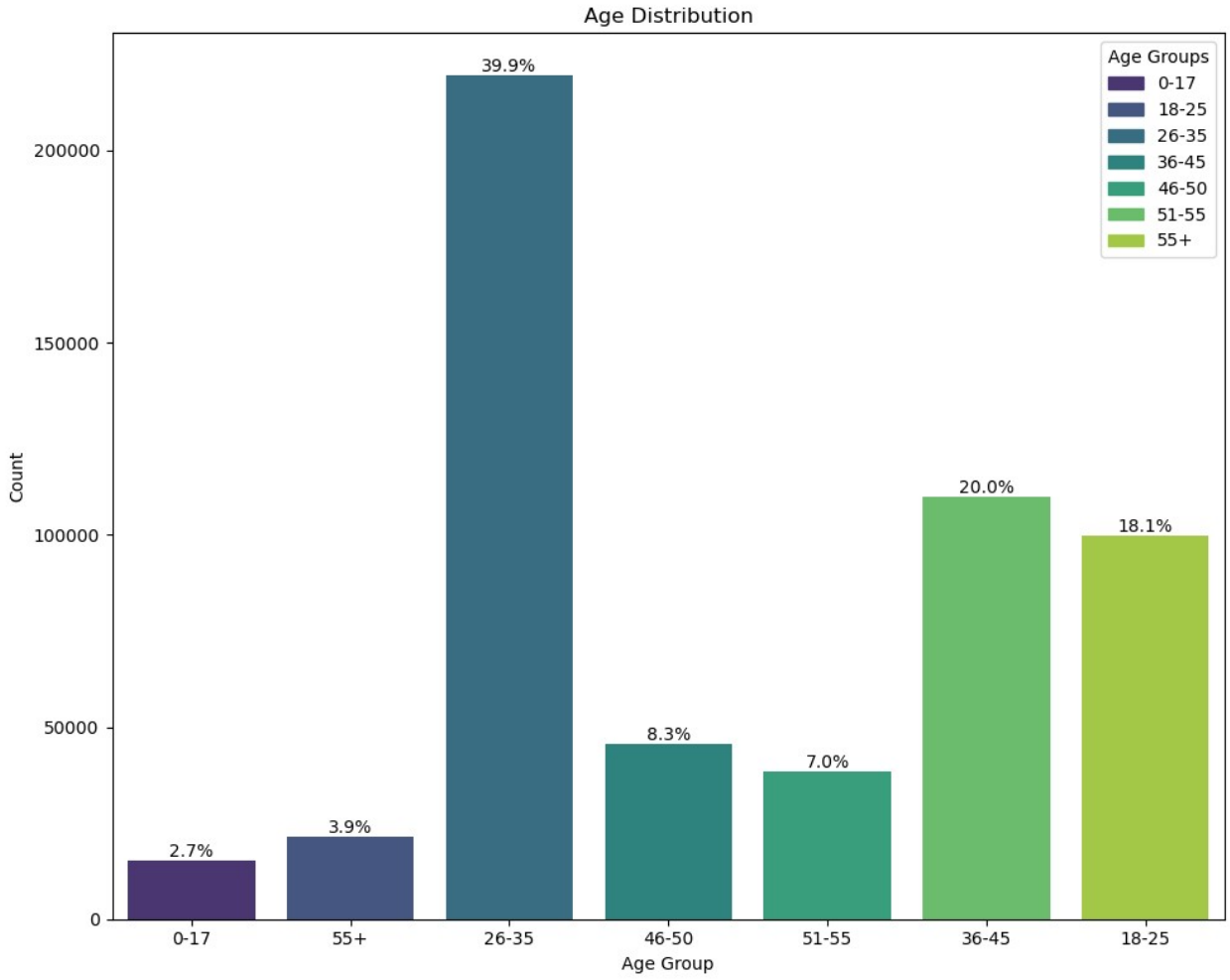
```
age_groups.sort()
patches = [mpatches.Patch(color = color[i], label = age_group) for i,
age_group in enumerate(age_groups)]
ax.legend(handles = patches, title = 'Age Groups')

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

```
---------------------------------------------------------------------------
-----
IndexError                                Traceback (most recent call
last)
Cell In[17], line 24
     22 age_groups = df['Age'].unique()
     23 age_groups.sort()
---> 24 patches = [mpatches.Patch(color = color[i], label = age_group)
for i, age_group in enumerate(age_groups)]
     25 ax.legend(handles = patches, title = 'Age Groups')
     27 plt.tight_layout()

Cell In[17], line 24, in <listcomp>(.0)
     22 age_groups = df['Age'].unique()
     23 age_groups.sort()
---> 24 patches = [mpatches.Patch(color = color[i], label = age_group)
for i, age_group in enumerate(age_groups)]
     25 ax.legend(handles = patches, title = 'Age Groups')
     27 plt.tight_layout()

IndexError: list index out of range
```



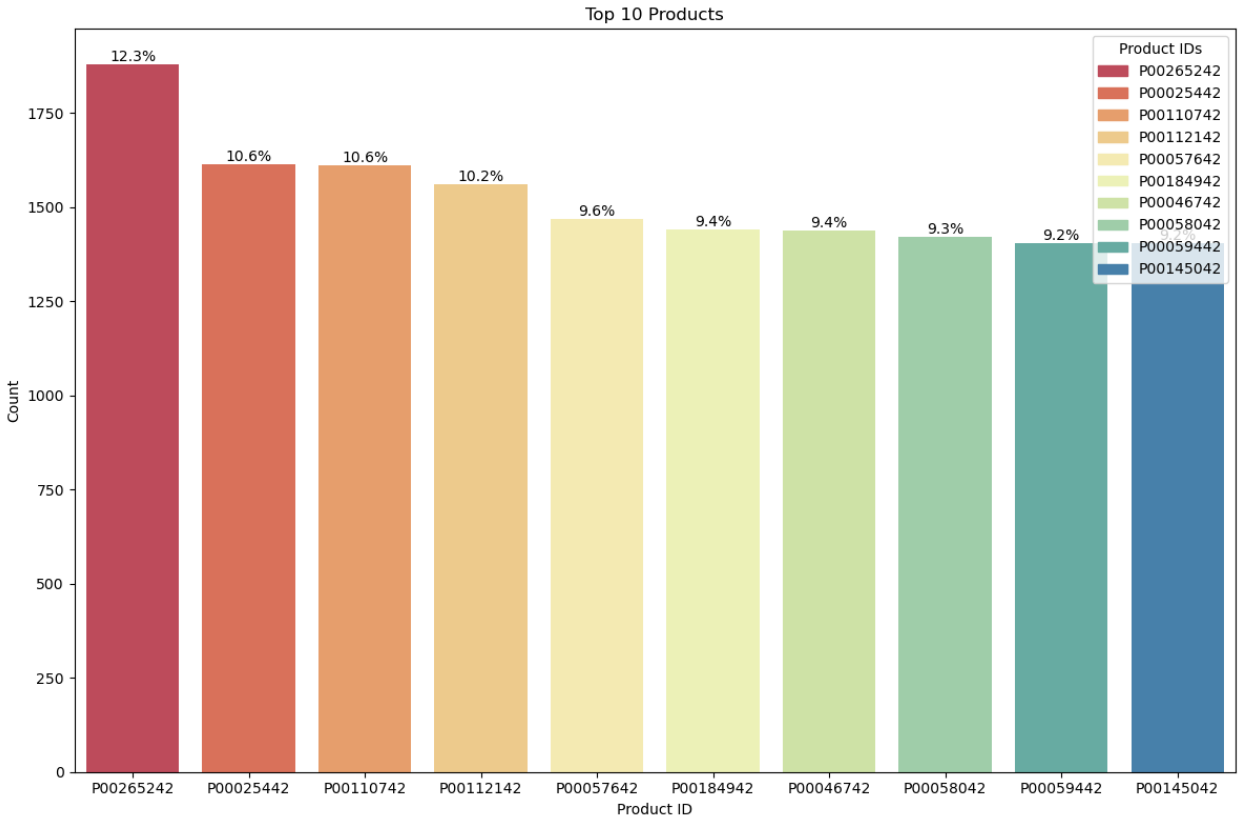

```
# Age Distribution
plt.figure(figsize=(10, 8))
ax = sns.countplot(x='Age', data=df, palette='viridis')

# Calculate percentages
total = len(df['Age'])
for p in ax.patches:
    percentage = f'{100 * p.get_height() / total:.1f}%'
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='bottom')

# Setting title and labels
ax.set_title('Age Distribution')
ax.set_xlabel('Age Group')
ax.set_ylabel('Count')

# Creating legend
colors = [p.get_facecolor() for p in ax.patches]
age_groups = df['Age'].unique()
```

```
age_groups.sort()
patches = [mpatches.Patch(color=colors[i], label=age_group) for i,
age_group in enumerate(age_groups)]
ax.legend(handles=patches, title='Age Groups')

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




```
#top 10 products

top_product_ids = df['Product_ID'].value_counts().head(10).index
df_top_products = df[df['Product_ID'].isin(top_product_ids)]

# Plotting with seaborn
plt.figure(figsize=(12, 8))
ax = sns.countplot(x='Product_ID', data=df_top_products,
palette='Spectral', order=top_product_ids)

# Calculate percentages
```

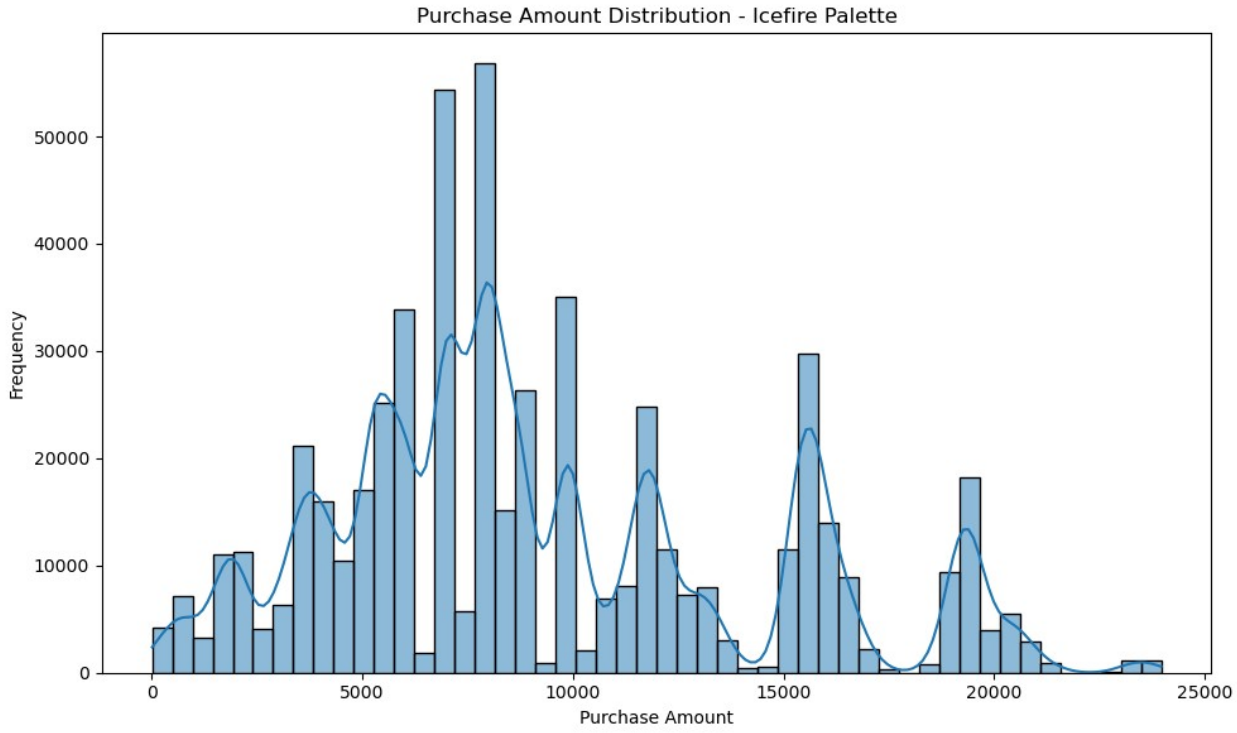
```
total = len(df_top_products)
for p in ax.patches:
    percentage = f'{100 * p.get_height() / total:.1f}%'
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='bottom')

# Setting title and labels
ax.set_title('Top 10 Products')
ax.set_xlabel('Product ID')
ax.set_ylabel('Count')

# Creating legend
colors = [p.get_facecolor() for p in ax.patches]
patches = [mpatches.Patch(color=colors[i], label=product_id) for i,
product_id in enumerate(top_product_ids)]
ax.legend(handles=patches, title='Product IDs')

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




```
# Using the "icefire" color palette
plt.figure(figsize=(10, 6))
sns.histplot(df['Purchase'], bins=50, kde=True, palette='icefire')
```

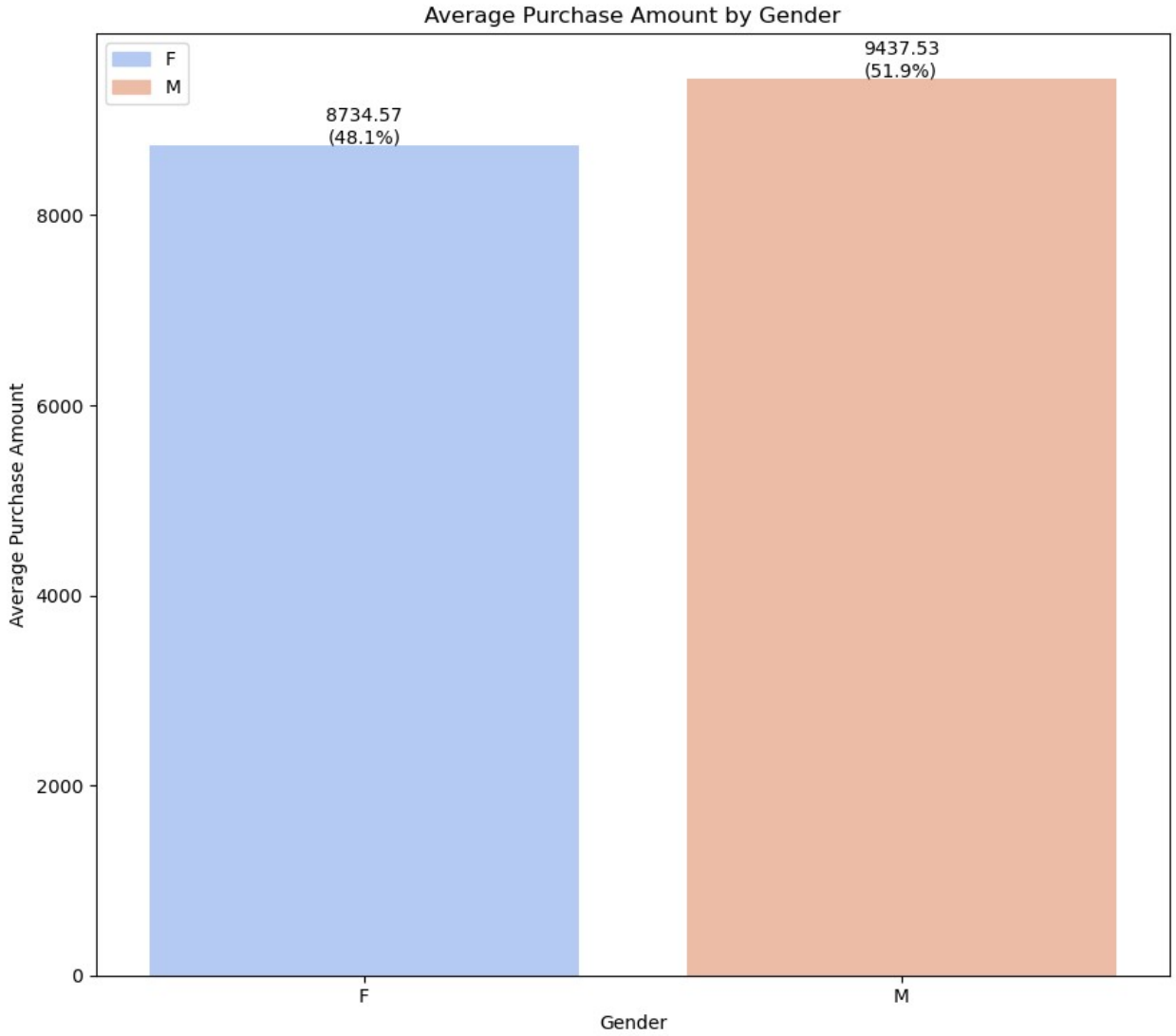
```
plt.title('Purchase Amount Distribution - Icefire Palette')
plt.xlabel('Purchase Amount')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

```
C:\Users\PRASANNA KUMAR\AppData\Local\Temp\
ipykernel_324\1032617419.py:3: UserWarning: Ignoring `palette` because
no `hue` variable has been assigned.
  sns.histplot(df['Purchase'], bins=50, kde=True, palette='icefire')
C:\Users\PRASANNA KUMAR\anaconda3\Lib\site-packages\seaborn\
_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



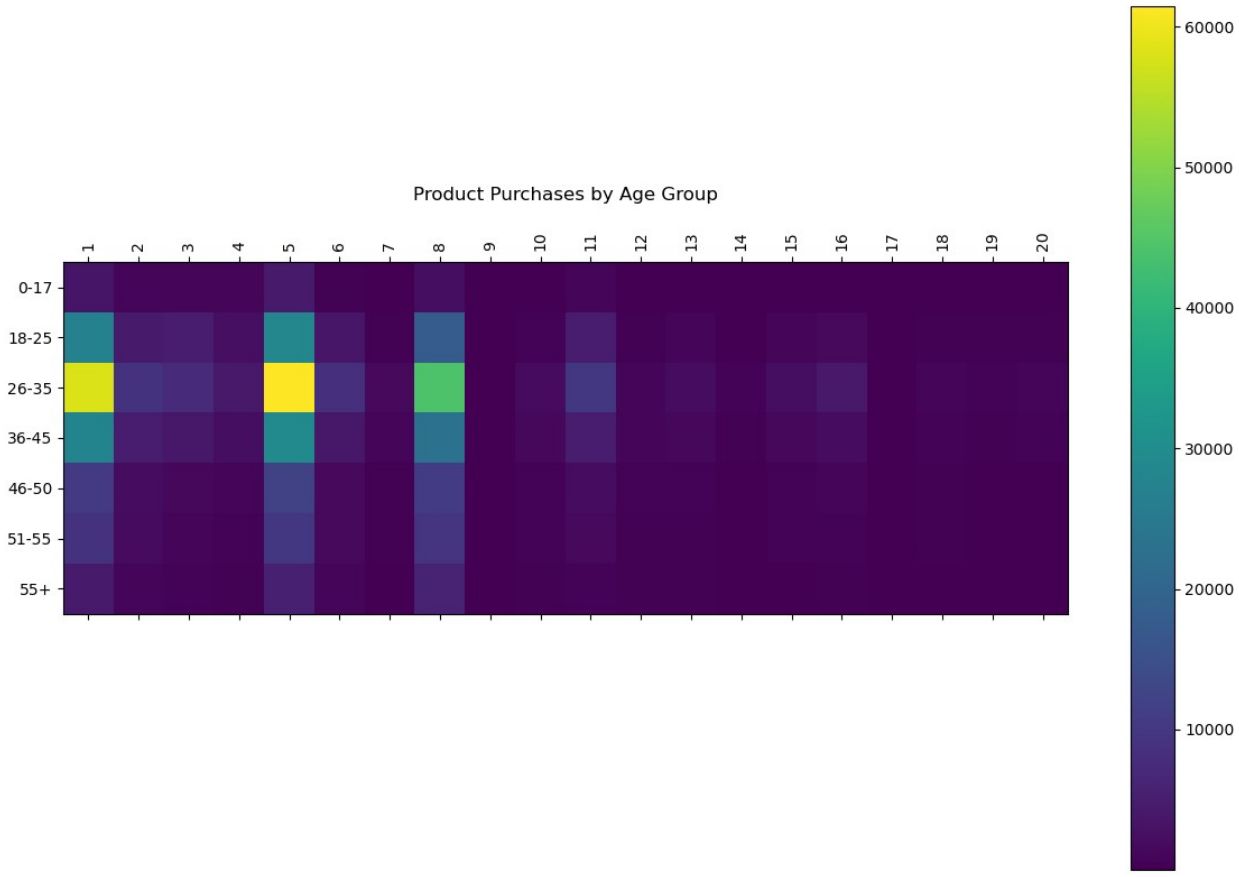

```
# Calculate average purchase amount by gender
gender_purchase_avg = df.groupby('Gender')['Purchase'].mean()
gender_purchase_avg_df = gender_purchase_avg.reset_index()
gender_purchase_avg_df.columns = ['Gender', 'Average Purchase Amount']

# Calculate percentage
total_avg_purchase = gender_purchase_avg_df['Average Purchase
Amount'].sum()
gender_purchase_avg_df['Percentage'] = gender_purchase_avg_df['Average
Purchase Amount'] / total_avg_purchase * 100
```

```
# Plotting with seaborn
plt.figure(figsize=(9, 8))
ax = sns.barplot(x='Gender', y='Average Purchase Amount',
data=gender_purchase_avg_df, palette='coolwarm')

# Annotate bars with percentage
for p in ax.patches:
    ax.annotate(f"{p.get_height():.2f}\n({p.get_height() /
total_avg_purchase * 100:.1f}%)",
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=10, color='black',
xytext=(0, 10),
                textcoords='offset points')

# Setting title and labels
ax.set_title('Average Purchase Amount by Gender')
ax.set_xlabel('Gender')
ax.set_ylabel('Average Purchase Amount')

# Adding legend manually
colors = [p.get_facecolor() for p in ax.patches]
gender_groups = df['Gender'].unique()
gender_groups.sort()
patches = [mpatches.Patch(color=colors[i], label=age_group) for i,
age_group in enumerate(gender_groups)]
ax.legend(handles=patches)

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

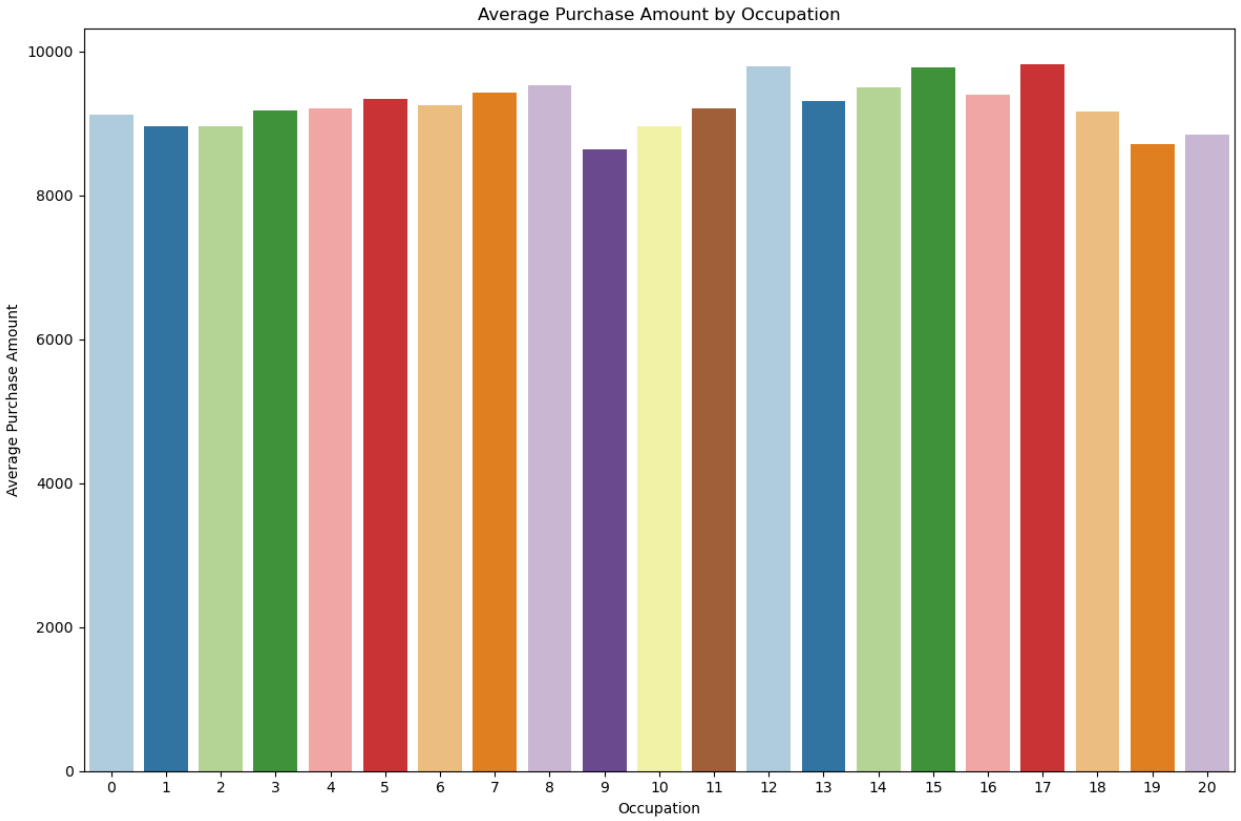



```
# Calculate the frequency of product purchases across different age
groups
age_product_freq = df.groupby(['Age',
'Product_Category']).size().unstack().fillna(0)

print("Frequency of Product Purchases by Age Group:")
print(age_product_freq)

# Plot heatmap of product purchases by age group
fig, ax = plt.subplots(figsize=(12, 8))
cax = ax.matshow(age_product_freq, cmap='viridis')
fig.colorbar(cax)
ax.set_xticks(range(len(age_product_freq.columns)))
ax.set_xticklabels(age_product_freq.columns, rotation=90)
ax.set_yticks(range(len(age_product_freq.index)))
ax.set_yticklabels(age_product_freq.index)
```

```
ax.set_title('Product Purchases by Age Group', pad=20)
plt.tight_layout()
plt.show()

Frequency of Product Purchases by Age Group:
Product_Category     1     2     3     4      5     6     7      8
9   \
Age

0-17                3585   805  1200   758   4330   399    53   2258
16
18-25              26962  4428  4710  2463  28522  3749   481  17911
63
26-35              58249  8928  7662  4192  61473  8485  1651  44256
154
36-45              27648  4912  3854  2354  29377  3899   809  23296
107
46-50              10474  2105  1376   990  11971  1622   327  10656
33
51-55               9049  1781   924   678   9893  1450   266   9340
29
55+                 4411   905   487   318   5367   862   134   6208
8

Product_Category    10    11    12    13   14    15    16   17    18
19   20
Age

0-17                111   740   125   112   39   160   229    6    27
59   90
18-25               603  4597   439   756  230  1024  1598   41   339
275  469
26-35              1787  9874  1096  2096  564  2372  4118  127  1042
563  898
36-45              1235  4953   994  1250  312  1395  1955  135   702
320  506
46-50               520  2104   520   551  149   602   879   95   351
149  227
51-55               519  1458   433   483  154   508   672  107   423
134  200
55+                 350   561   340   301   75   229   377   67   241
103  160
```



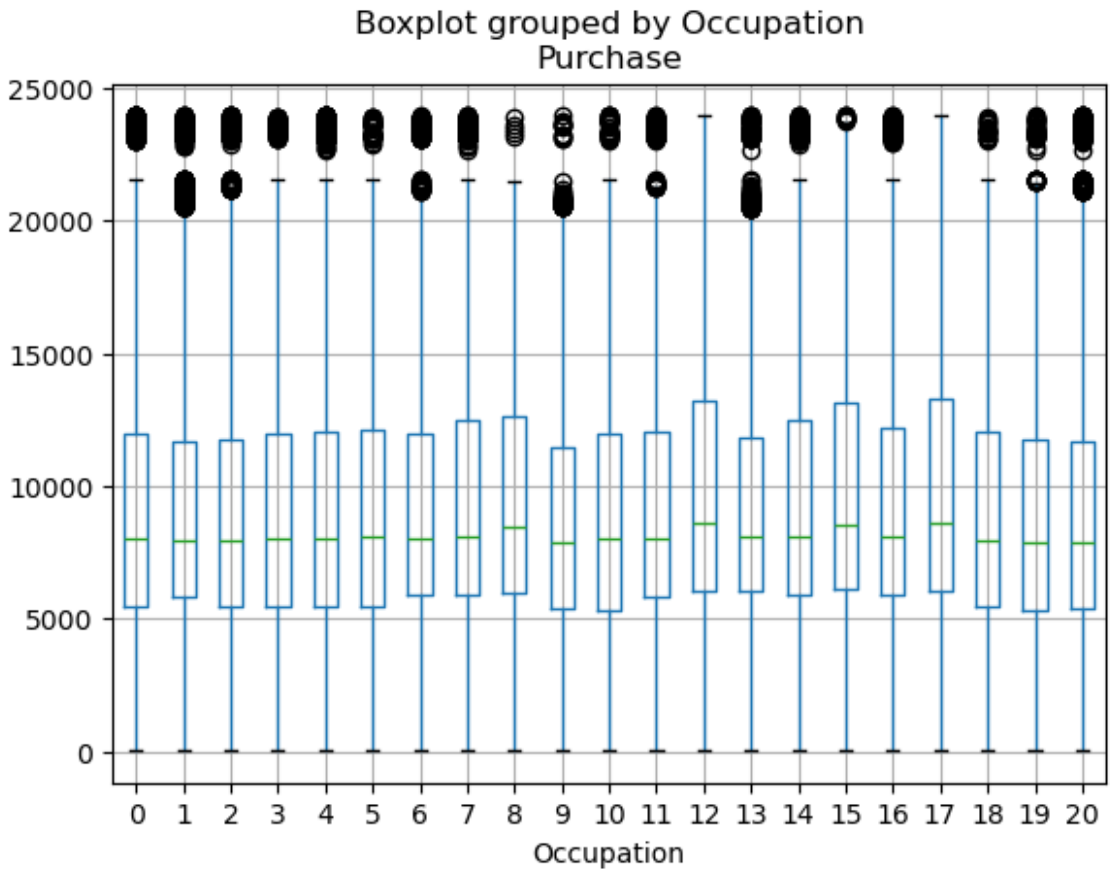

```
# Calculate average purchase amount by occupation
occupation_purchase_avg = df.groupby('Occupation')['Purchase'].mean()
occupation_purchase_avg_df = occupation_purchase_avg.reset_index()
occupation_purchase_avg_df.columns = ['Occupation', 'Average Purchase
Amount']

# Plotting with seaborn
plt.figure(figsize=(12, 8))
ax = sns.barplot(x='Occupation', y='Average Purchase Amount',
data=occupation_purchase_avg_df, palette='Paired')

# Setting title and labels
ax.set_title('Average Purchase Amount by Occupation')
ax.set_xlabel('Occupation')
ax.set_ylabel('Average Purchase Amount')

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

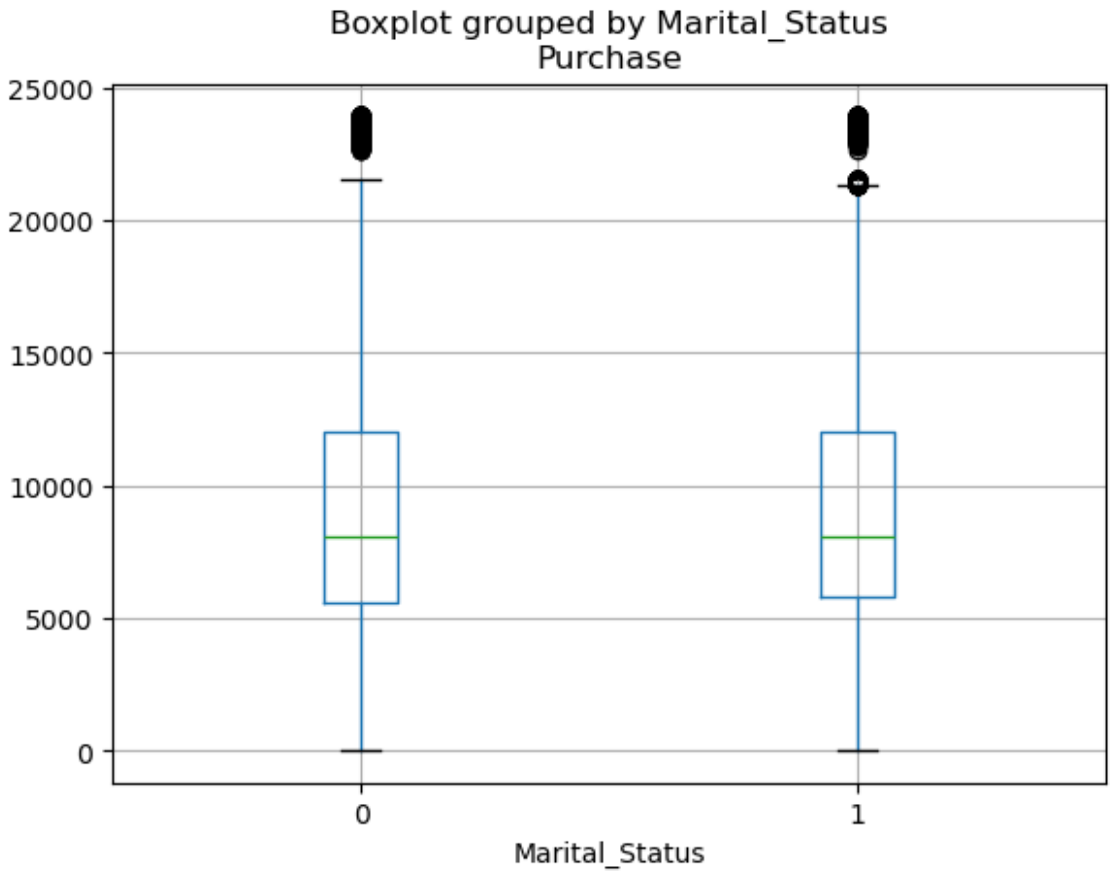


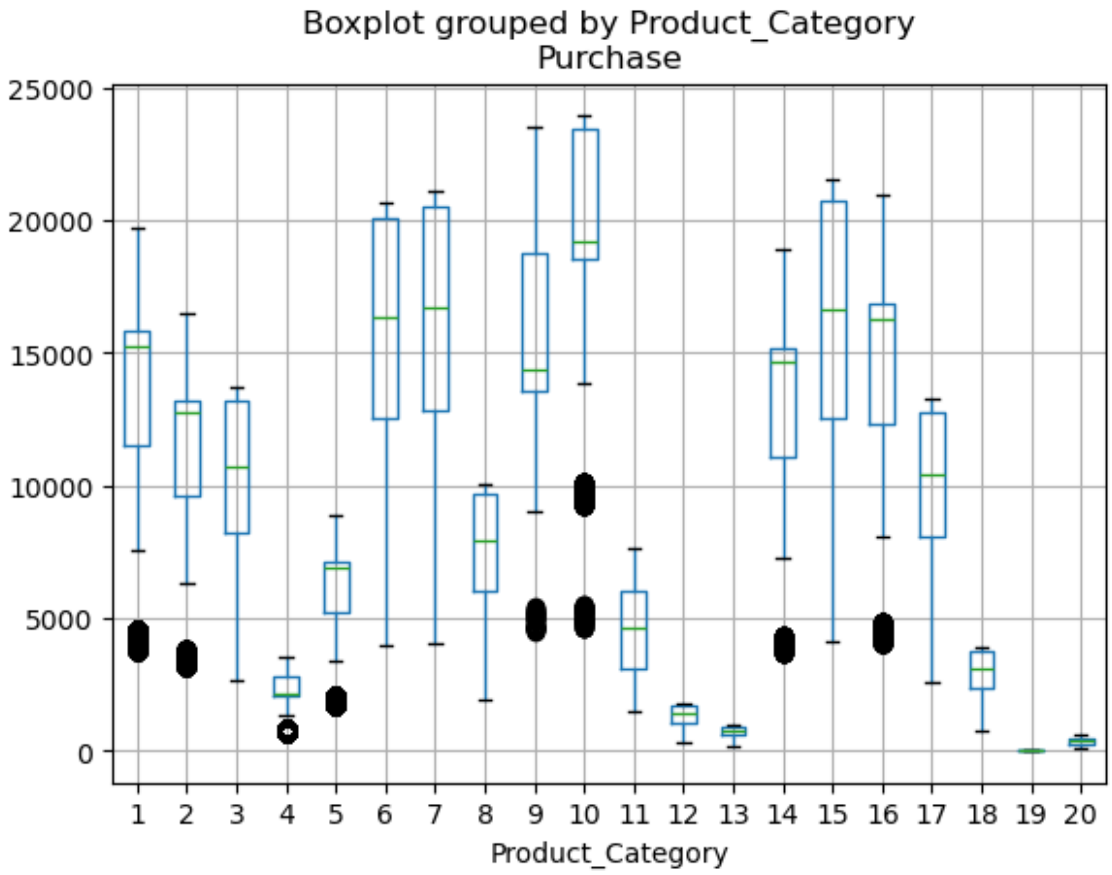
Understanding the Relationship of the Columns

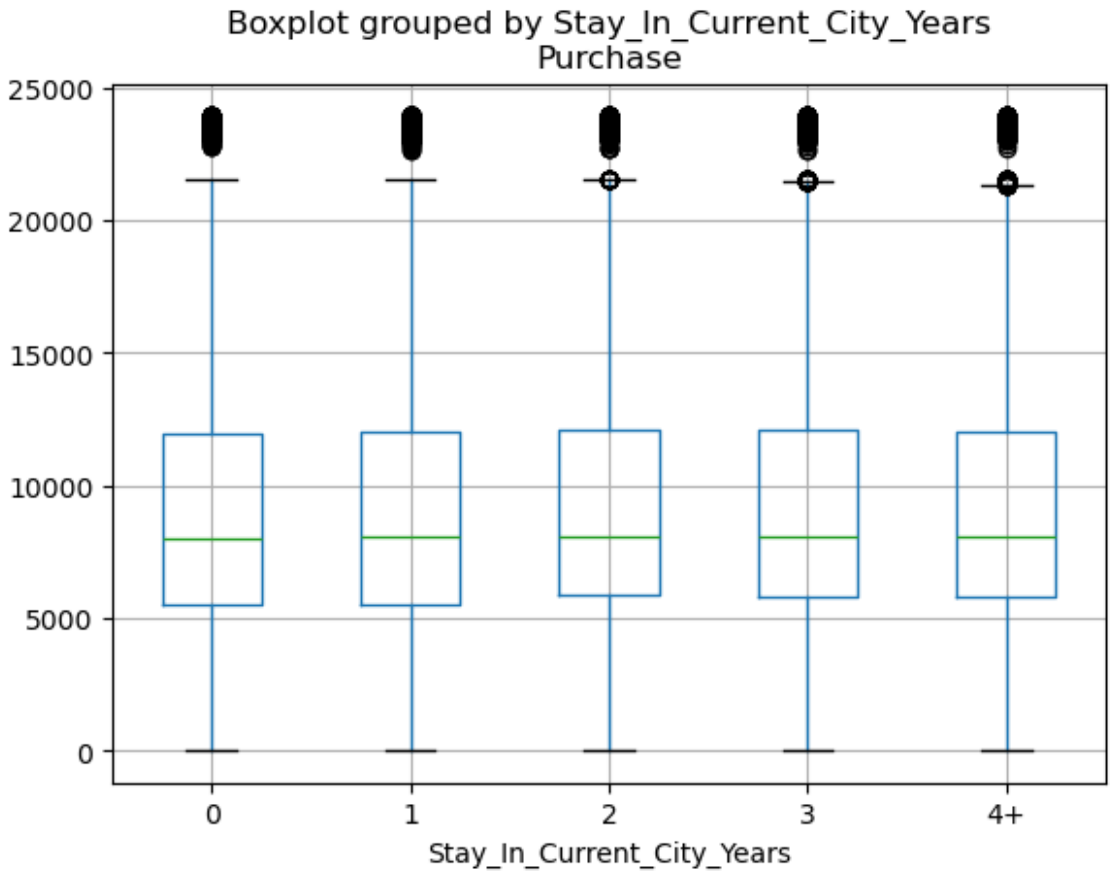
```
df.boxplot("Purchase", by = "Occupation")
df.boxplot("Purchase", by = "Marital_Status")
df.boxplot("Purchase", by = "Product_Category")
df.boxplot("Purchase", by = "Stay_In_Current_City_Years")
df.boxplot("Purchase", by = "Gender")
```

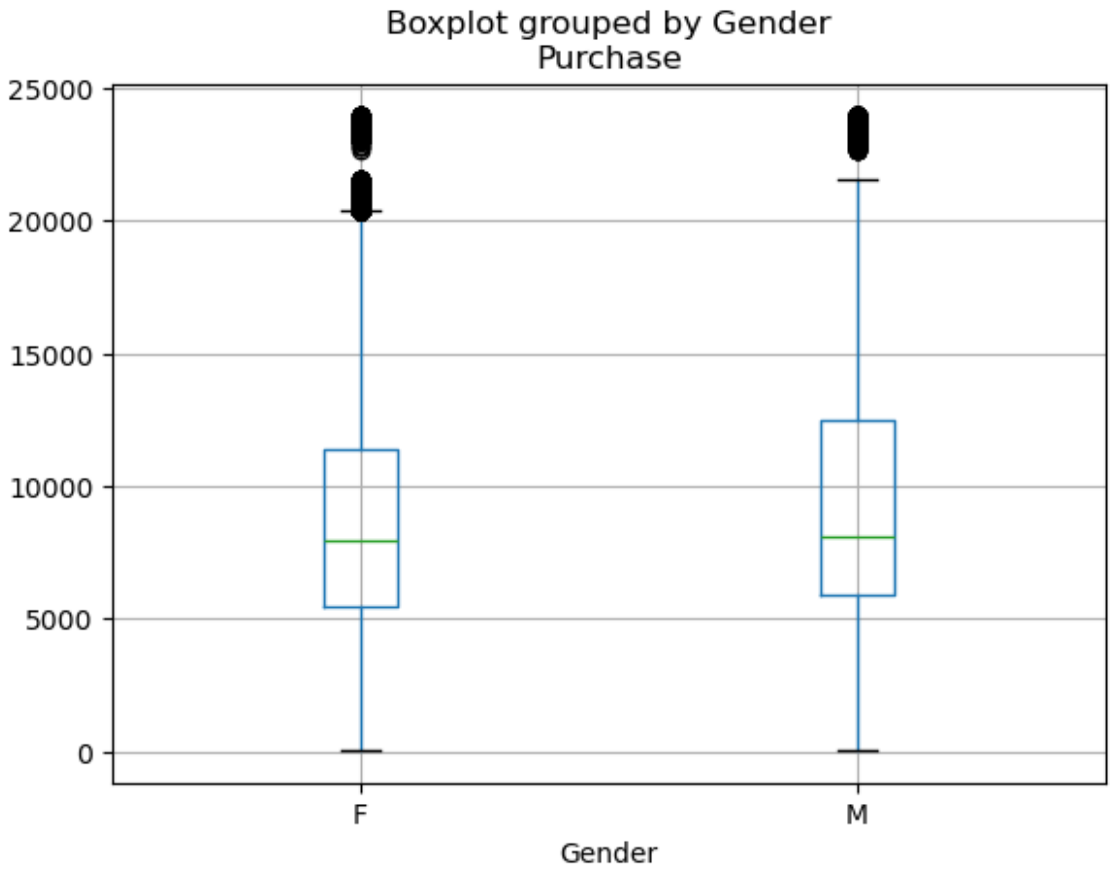
```
<Axes: title={'center': 'Purchase'}, xlabel='Gender'>
```


Boxplot grouped by Occupation
Purchase
25000
20000
15000
10000
5000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Occupation


Boxplot grouped by Marital_Status
Purchase
25000
20000
15000
10000
5000
0
0
1
Marital_Status


Boxplot grouped by Product_Category
Purchase
25000
20000
15000
10000
5000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Product_Category


Boxplot grouped by Stay_In_Current_City_Years
Purchase
25000
20000
15000
10000
5000
0
0
1
2
3
4+
Stay_In_Current_City_Years



Creating Pipeline and Encoding

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn import set_config
from copy import deepcopy

num_features = ['Purchase']
cat_features = ['Gender',
                'Age',
                'Occupation',
                'City_Category',
                'Stay_In_Current_City_Years',
                'Marital_Status',
                'Product_Category']

num_features = pd.DataFrame(df['Purchase'])
num_features
```

```
        Purchase
0           8370
1          15200
2           1422
3           1057
4           7969
...          ...
550063       368
550064       371
550065       137
550066       365
550067       490

[550068 rows x 1 columns]
```

```
cat_features = pd.DataFrame(df[['Gender',
                'Age',
                'Occupation',
                'City_Category',
                'Stay_In_Current_City_Years',
                'Marital_Status',
                'Product_Category']])
cat_features
```

```
       Gender    Age  Occupation City_Category
Stay_In_Current_City_Years  \
0            F   0-17          10             A
2
1            F   0-17          10             A
2
2            F   0-17          10             A
2
3            F   0-17          10             A
2
4            M    55+          16             C
4+
...        ...    ...         ...           ...
...
550063       M  51-55          13             B
1
550064       F  26-35           1             C
3
550065       F  26-35          15             B
4+
550066       F    55+           1             C
2
550067       F  46-50           0             B
4+

        Marital_Status  Product_Category
```

```
0                    0                  3
1                    0                  1
2                    0                 12
3                    0                 12
4                    0                  8
...                ...                ...
550063               1                 20
550064               0                 20
550065               1                 20
550066               0                 20
550067               1                 20

[550068 rows x 7 columns]
```

```
X = cat_features
y = num_features
```

```
X.dtypes
```

```
Gender                        object
Age                           object
Occupation                     int64
City_Category                 object
Stay_In_Current_City_Years    object
Marital_Status                 int64
Product_Category               int64
dtype: object
```

```
y.dtypes
```

```
Purchase    int64
dtype: object
```

```
X
```

```
       Gender    Age  Occupation City_Category
Stay_In_Current_City_Years  \
0           F   0-17          10             A
2
1           F   0-17          10             A
2
2           F   0-17          10             A
2
3           F   0-17          10             A
2
4           M    55+          16             C
4+
...       ...    ...         ...           ...
...
550063      M  51-55          13             B
1
```

```
550064      F  26-35           1              C
3
550065      F  26-35          15              B
4+
550066      F    55+           1              C
2
550067      F  46-50           0              B
4+

        Marital_Status  Product_Category
0                    0                 3
1                    0                 1
2                    0                12
3                    0                12
4                    0                 8
...                ...               ...
550063               1                20
550064               0                20
550065               1                20
550066               0                20
550067               1                20

[550068 rows x 7 columns]
```

```python
class FeatureEncoder(BaseEstimator, TransformerMixin):

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        #Age Numeric
        age_dct = {"0-17": 0, "18-25": 1, "26-35": 2, "36-45": 3, "46-
50": 4, "51-55": 5, "55+": 6}
        X = X.copy()  # Avoid SettingWithCopyWarning
        X["Age"] = X["Age"].map(age_dct)

        #City_Category One Hot Encoding
        ohe = OneHotEncoder()
        matrix = ohe.fit_transform(X[["City_Category"]]).toarray()

        column_names = ["City_Cat_A", "City_Cat_B", "City_Cat_C"]

        for i in range(len(matrix.T)):
            X[column_names[i]] = matrix.T[i]

        X = X.drop(columns = ["City_Category"], axis=1)

        #Occupation One Hot Encoding
        ohe = OneHotEncoder()
        matrix = ohe.fit_transform(X[["Occupation"]]).toarray()
```

```
        column_names = ["Occupation_" + str(i) for i in range(21)]

        for i in range(len(matrix.T)):
            X[column_names[i]] = matrix.T[i]

        X = X.drop(columns = ["Occupation"], axis=1)

        #Product_Category One Hot Encoding
        ohe = OneHotEncoder()
        matrix = ohe.fit_transform(X[["Product_Category"]]).toarray()

        column_names = ["Product_Category_" + str(i) for i in
range(20)]

        for i in range(len(matrix.T)):
            X[column_names[i]] = matrix.T[i]

        X = X.drop(columns = ["Product_Category"], axis=1)

        return X

# Define the pipeline
prepipe = Pipeline([
    ('preprocessor', FeatureEncoder()),
    ('encoder', OneHotEncoder())
])

# Fit the pipeline
X= prepipe.fit_transform(X)

X = pd.DataFrame(X.toarray())
X.head()
```

```
   0    1    2    3    4    5    6    7    8    9    ...  94   95   96
97   \
0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  1.0  0.0
1.0  0.0
1  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  1.0  0.0
1.0  0.0
2  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  1.0  0.0
1.0  0.0
3  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  1.0  0.0
1.0  0.0
4  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  ...  1.0  0.0
1.0  0.0

   98   99   100  101  102  103
0  1.0  0.0  1.0  0.0  1.0  0.0
1  1.0  0.0  1.0  0.0  1.0  0.0
```

```
2  1.0  0.0  1.0  0.0  1.0  0.0
3  1.0  0.0  1.0  0.0  1.0  0.0
4  1.0  0.0  1.0  0.0  1.0  0.0

[5 rows x 104 columns]
```

```
scaler = RobustScaler().fit(y)
y = scaler.transform(y)
y = pd.DataFrame(y)
y
```

```
                0
0       0.051838
1       1.147970
2      -1.063232
3      -1.121810
4      -0.012518
...          ...
550063 -1.232386
550064 -1.231905
550065 -1.269459
550066 -1.232868
550067 -1.212807

[550068 rows x 1 columns]
```