Black_Friday_Notebook_1

September 16, 2019

0.1 Black Friday Notebook 1

[1]:
```
# IST 718 Big Data Analytics, Final Project: Black Friday
# Notebook Name: Black Friday Notebook #1
# Notebook Contents: EDA and K-Means Clustering models
# Group Members: Arielle Dortch, Lauren Foltz, Stephen Clark Washington III,␣
 ↪Inga Fleishhacker
# Written for Watson IBM Cloud using Python 3.5 with Spark 2.1

# Data is from https://www.kaggle.com/mehdidag/black-friday
```

[2]:
```
# Note: Some information below has been redacted

# @hidden_cell
# The project token is an authorization token that is used to access project␣
 ↪resources
# like data sources, connections, and used by platform APIs.
from project_lib import Project
project = Project(project_id='REDACTED', project_access_token='REDACTED')
pc = project.project_context
```

[3]:
```
# Import Libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns #for visualization
from matplotlib import pyplot as plt #for visualiztion
from sklearn.preprocessing import Imputer   #to fill in missing values
```

```
/usr/local/src/conda3_runtime/home/envs/DSX-
Python35-Spark/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning:
numpy.dtype size changed, may indicate binary incompatibility. Expected 96, got
88
  return f(*args, **kwds)
/usr/local/src/conda3_runtime/home/envs/DSX-
Python35-Spark/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning:
numpy.dtype size changed, may indicate binary incompatibility. Expected 96, got
88
  return f(*args, **kwds)
```

```
/usr/local/src/conda3_runtime/home/envs/DSX-
Python35-Spark/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning:
numpy.dtype size changed, may indicate binary incompatibility. Expected 96, got
88
  return f(*args, **kwds)
/usr/local/src/conda3_runtime/home/envs/DSX-
Python35-Spark/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning:
numpy.dtype size changed, may indicate binary incompatibility. Expected 96, got
88
  return f(*args, **kwds)
/usr/local/src/conda3_runtime/home/envs/DSX-
Python35-Spark/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning:
numpy.dtype size changed, may indicate binary incompatibility. Expected 96, got
88
  return f(*args, **kwds)
```

[4]:

```
# Import Libraries and Data

import os
import chardet #to find the encoding of the csv file
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# Note: Some information below has been redacted

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It␣
↪includes your credentials.
# You might want to remove those credentials before you share your notebook.
client_525f5d2245b6446693c23e44b7178e2a = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='REDACTED',
    ibm_auth_endpoint="REDACTED",
    config=Config(signature_version='REDACTED'),
    endpoint_url='REDACTED')

body = client_525f5d2245b6446693c23e44b7178e2a.
↪get_object(Bucket='finalproject-donotdelete-pr-pr6xudhaekii34',Key='BlackFriday.
↪csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__,␣
↪body )

# Store data as "data"
```

```
data = pd.read_csv(body)

# Check head of data
print('Head of Data:')
data.head()
```

```
Head of Data:
```

[4]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN	NaN	8370
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	14.0	15200
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN	NaN	1422
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	NaN	1057
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN	NaN	7969

[5]:
```
# Check the overview information about the data
data.info()
# We can see data came in as int, floats, and objects.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 537577 entries, 0 to 537576
Data columns (total 12 columns):
User_ID                     537577 non-null int64
Product_ID                  537577 non-null object
Gender                      537577 non-null object
Age                         537577 non-null object
Occupation                  537577 non-null int64
City_Category               537577 non-null object
Stay_In_Current_City_Years  537577 non-null object
Marital_Status              537577 non-null int64
Product_Category_1          537577 non-null int64
Product_Category_2          370591 non-null float64
Product_Category_3          164278 non-null float64
Purchase                    537577 non-null int64
```

```
dtypes: float64(2), int64(5), object(5)
memory usage: 49.2+ MB
```

[6]:
```
# Check the description of the data
print("Description of Data:")
data.describe()
```

```
Description of Data:
```

[6]:

	User_ID	Occupation	Marital_Status	Product_Category_1	\
count	5.375770e+05	537577.00000	537577.000000	537577.000000	
mean	1.002992e+06	8.08271	0.408797	5.295546	
std	1.714393e+03	6.52412	0.491612	3.750701	
min	1.000001e+06	0.00000	0.000000	1.000000	
25%	1.001495e+06	2.00000	0.000000	1.000000	
50%	1.003031e+06	7.00000	0.000000	5.000000	
75%	1.004417e+06	14.00000	1.000000	8.000000	
max	1.006040e+06	20.00000	1.000000	18.000000	

	Product_Category_2	Product_Category_3	Purchase
count	370591.000000	164278.000000	537577.000000
mean	9.842144	12.669840	9333.859853
std	5.087259	4.124341	4981.022133
min	2.000000	3.000000	185.000000
25%	5.000000	9.000000	5866.000000
50%	9.000000	14.000000	8062.000000
75%	15.000000	16.000000	12073.000000
max	18.000000	18.000000	23961.000000

[7]:
```
# Check for missing values
print('Missing Values:')
print()
data.isnull().sum()
```

```
Missing Values:
```

[7]:
```
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_1                0
Product_Category_2           166986
Product_Category_3           373299
```

```
Purchase                      0
dtype: int64
```

0.1.1 Data Change: Drop Product Category 2 and 3

[8]:
```
# Get percentage of missing values in Product_Category_2 and Product_Category_3
print('Percentage of Product_Category_2 with missing values: {:.2f}%'.
 ↪format(data['Product_Category_2'].isnull().sum()/len(data)*100))
print('Percentage of Product_Category_3 with missing values: {:.2f}%'.
 ↪format(data['Product_Category_3'].isnull().sum()/len(data)*100))
```

```
Percentage of Product_Category_2 with missing values: 31.06%
Percentage of Product_Category_3 with missing values: 69.44%
```

[9]:
```
# Drop both product category 2 and product category 3 (since many values are␣
 ↪missing)
data.drop(["Product_Category_2", "Product_Category_3"], axis = 1, inplace =␣
 ↪True)
```

0.1.2 Data Change: Create Aliased Data for Correlation Plot

[10]:
```
# Export to alias the gender, age, and city category to be numeric so that the␣
 ↪correlation matrix will run

# Note: Some information below has been redacted

# @hidden_cell
# The project token is an authorization token that is used to access project␣
 ↪resources
# like data sources, connections, and used by platform APIs.
from project_lib import Project
project = Project(project_id='REDACTED', project_access_token='REDACTED')
pc = project.project_context

project.save_data(data=data.to_csv(index=True),file_name='dataout.
 ↪csv',overwrite=True)
```

[10]:
```
{'asset_id': 'e7a83868-2037-4e7a-ba71-df050c40af8d',
 'bucket_name': 'finalproject-donotdelete-pr-pr6xudhaekii34',
 'file_name': 'dataout.csv',
 'message': 'File dataout.csv has been written successfully to the associated
OS'}
```

[11]:
```
# Import the aliased so that the correlation matrix will render correctly
```

```
body = client_525f5d2245b6446693c23e44b7178e2a.
 ↪get_object(Bucket='finalproject-donotdelete-pr-pr6xudhaekii34',Key='dataoutput2.
 ↪csv')['Body']
# Add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__,␣
 ↪body )

data = pd.read_csv(body)
# Check head of data
print('Head of Aliased Data:')
data.head()
```

```
Head of Aliased Data:
```

[11]:

	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_1	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	

	GenderRecode	AgeRecode	CityRecode
0	1	0	1
1	1	0	1
2	1	0	1
3	1	0	1
4	2	6	3

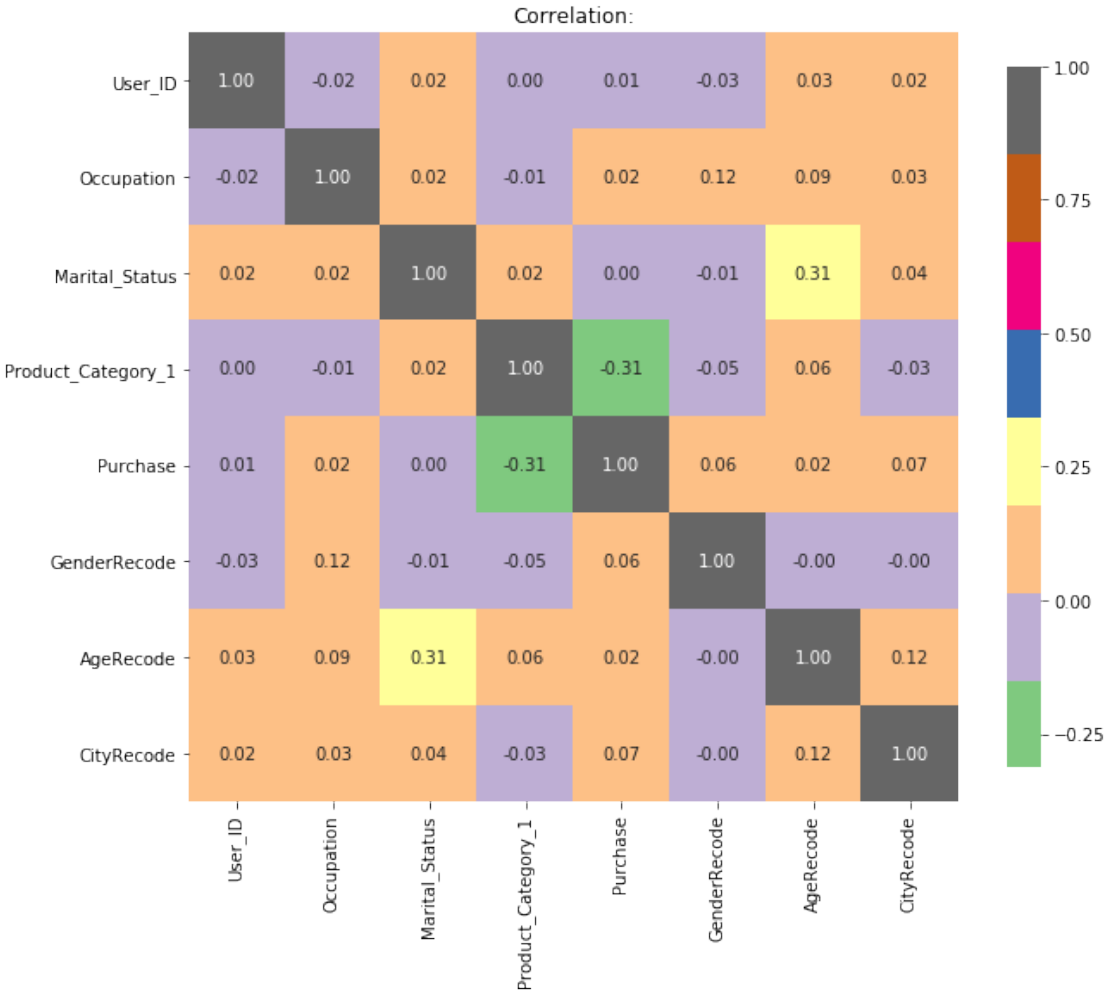
[12]:
```
# Check the data types
print('Data Types of Aliased Data:')
print(data.dtypes)
```

```
Data Types of Aliased Data:
User_ID                       int64
Product_ID                   object
Gender                       object
Age                          object
Occupation                    int64
City_Category                object
Stay_In_Current_City_Years   object
```

```
Marital_Status              int64
Product_Category_1          int64
Purchase                    int64
GenderRecode                int64
AgeRecode                   int64
CityRecode                  int64
dtype: object
```

0.2 Correlation Matrix

[13]:
```
# Show correlation plot
plt.figure(figsize=(10, 10))
sns.heatmap(data.corr(), annot=True, fmt='.2f', cmap='Accent', square=True,␣
 ↪cbar_kws={'shrink' : .75})
plt.title("Correlation:")
plt.show()
```



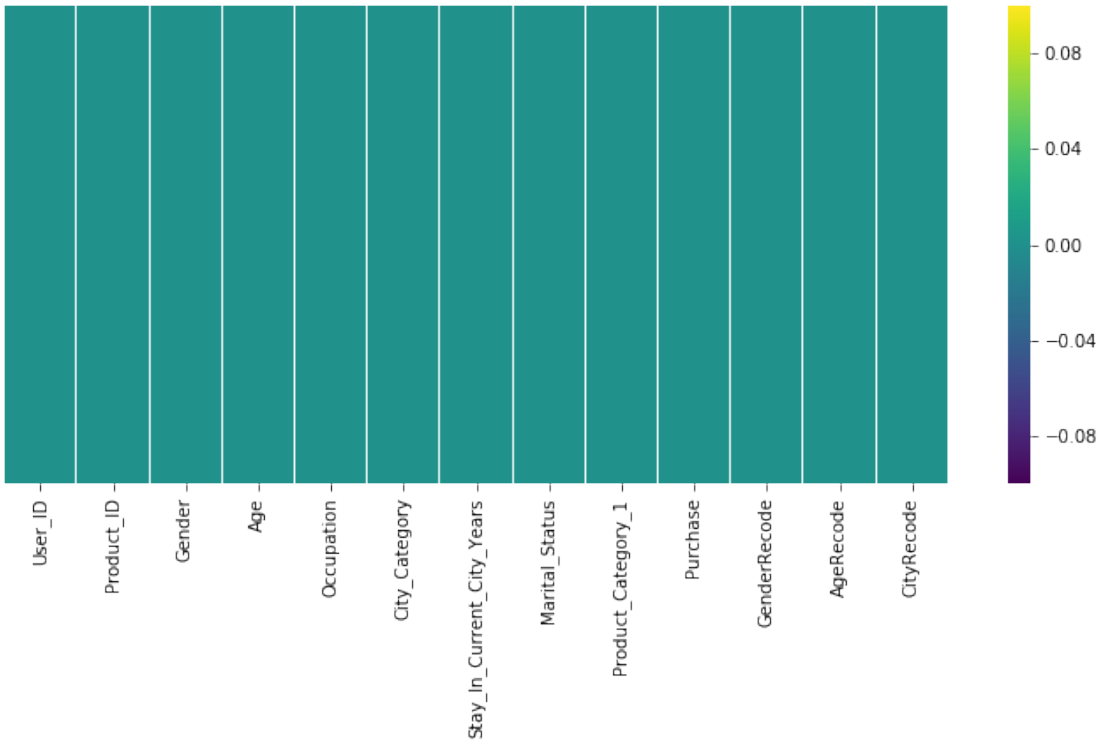
[14]:
```
# Check again for missing values
print('Missing Values of Aliased Data:')
print()
data.isnull().sum()
# There are no more missing values
```

```
Missing Values of Aliased Data:
```

[14]:
```
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_1          0
Purchase                    0
GenderRecode                0
AgeRecode                   0
CityRecode                  0
dtype: int64
```

[15]:
```
# Plot missing values
plt.figure(figsize=(12, 5))
sns.heatmap(data.isnull(), yticklabels=False, cmap='viridis')
```

[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb15d90a208>
```



[16]:
```
# Check for the number of unique values in each feature
print('Unique Values in Aliased Data:')
print()
data.nunique()
```

```
Unique Values in Aliased Data:
```

[16]:
```
User_ID                       5891
Product_ID                    3623
Gender                           2
Age                              7
Occupation                      21
City_Category                    3
Stay_In_Current_City_Years       5
Marital_Status                   2
Product_Category_1              18
Purchase                     17959
GenderRecode                     2
AgeRecode                        7
CityRecode                       3
dtype: int64
```

[17]:
```
# Calculate the total amount of purchases made
total_purchases = data.Purchase.values.sum()
```

[18]:
```
# Have a view of the data columns
print('Column Headings:')
print()
data.columns
```

```
Column Headings:
```

[18]:
```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category_1',
       'Purchase', 'GenderRecode', 'AgeRecode', 'CityRecode'],
      dtype='object')
```

0.2.1 Data Change: Customer Data (with Purchases Summed)

[19]:
```
# Add dataframe that has the purchased summed by user, gender, age, occupation,␣
 ↪city, and marital status.
# This will avoid double counting the same user id multiple times and will be␣
 ↪used for many of the charts.
# The old data set will come back after the initial demographic data discovery

body = client_525f5d2245b6446693c23e44b7178e2a.
 ↪get_object(Bucket='finalproject-donotdelete-pr-pr6xudhaekii34',Key='CustomerTotalPurchases.
 ↪csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__,␣
 ↪body )

data = pd.read_csv(body)
# Look at head of data
print("Head of Customer Data:")
data.head()
```

```
Head of Customer Data:
```

[19]:

	User_ID	Gender	Age	Occupation	City_Category	Marital_Status	Purchase
0	1000001	F	0-17	10	A	0	333481
1	1000008	M	26-35	12	C	1	796545
2	1000013	M	46-50	1	C	1	713927
3	1000015	M	26-35	7	A	0	1047124
4	1000047	M	18-25	4	C	1	131303

[20]:
```
# Check the data types
print('Data Types of Customer Data:')
```

```
print()
print(data.dtypes)
```

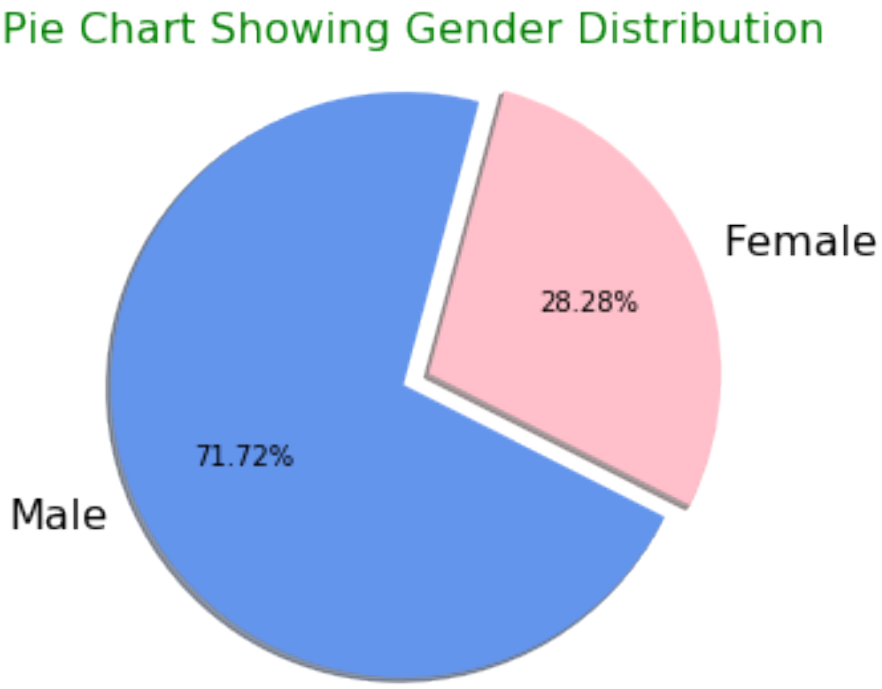
```
Data Types of Customer Data:

User_ID            int64
Gender            object
Age               object
Occupation         int64
City_Category     object
Marital_Status     int64
Purchase           int64
dtype: object
```

0.2.2 Gender Analysis of Customers

[21]:
```
# Calculate the number of both sexes in the customer group
num_of_males = len(data[data['Gender']=='M'])
num_of_females = len(data[data['Gender']=='F'])

# Calculate the percentage of both sexes in the customer group
pct_of_males = (num_of_males)/len(data)*100
pct_of_females = (num_of_females)/len(data)*100



# Print Analysis
print('Gender Analysis:')
print()

# Output the number of each gender in the customer group
print('Number of Males: {}'.format(num_of_males))
print('Number of Females: {}'.format(num_of_females))
print()

# Output the percentage of each gender in the customer group
print('Percentage of Males: {:.2f}%'.format(pct_of_males))
print('Percentage of Females: {:.2f}%'.format(pct_of_females))
```

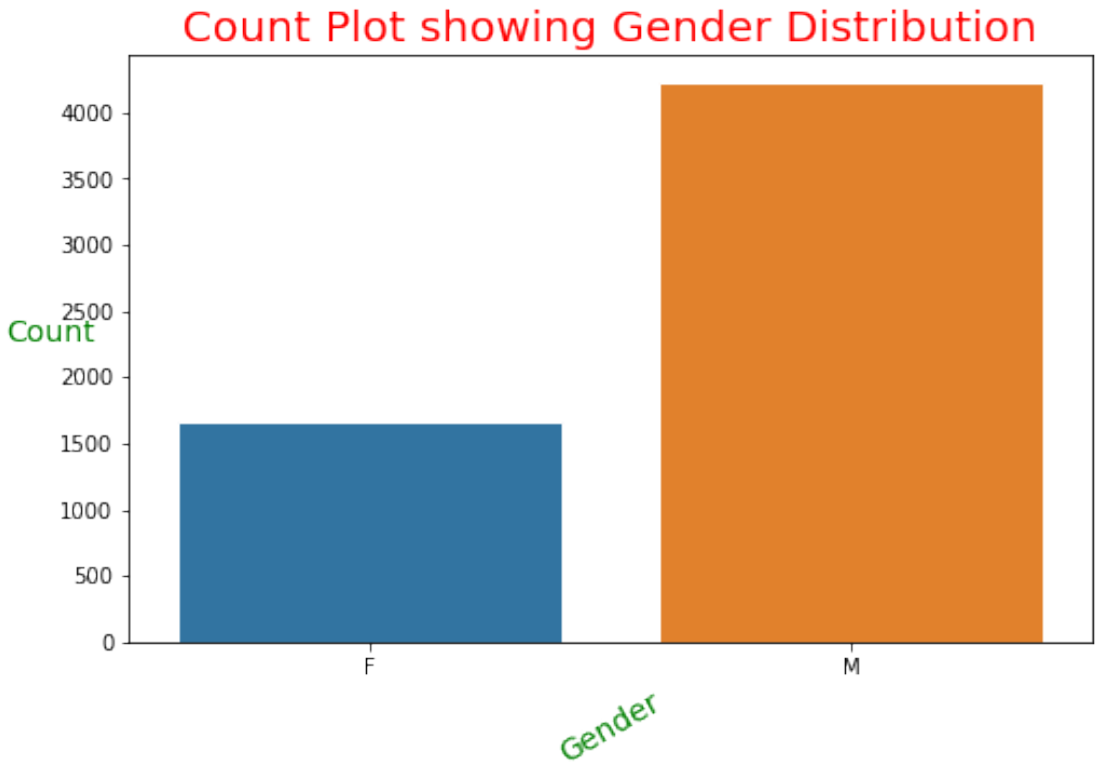
```
Gender Analysis:

Number of Males: 4225
Number of Females: 1666

Percentage of Males: 71.72%
Percentage of Females: 28.28%
```

[22]:
```
# Plot a pie chart to show the Gender distribution
plt.title('Pie Chart Showing Gender Distribution', fontdict={'size':16, 'color':
 ↪'green'})
patches, texts, autotexts = plt.pie(
    x = [pct_of_males, pct_of_females],
    labels = ['Male', 'Female'],
    shadow = True,
    radius=1.5,
    explode = [0.07, 0.07],
    startangle = 75,
    colors = ['cornflowerblue', 'pink'],
    autopct = '%.2f%%',
)
for text in texts:
    text.set_color('black')
    text.set_size(16)
for autotext in autotexts:
    autotext.set_color('black')

plt.axis('equal')
plt.tight_layout()
```



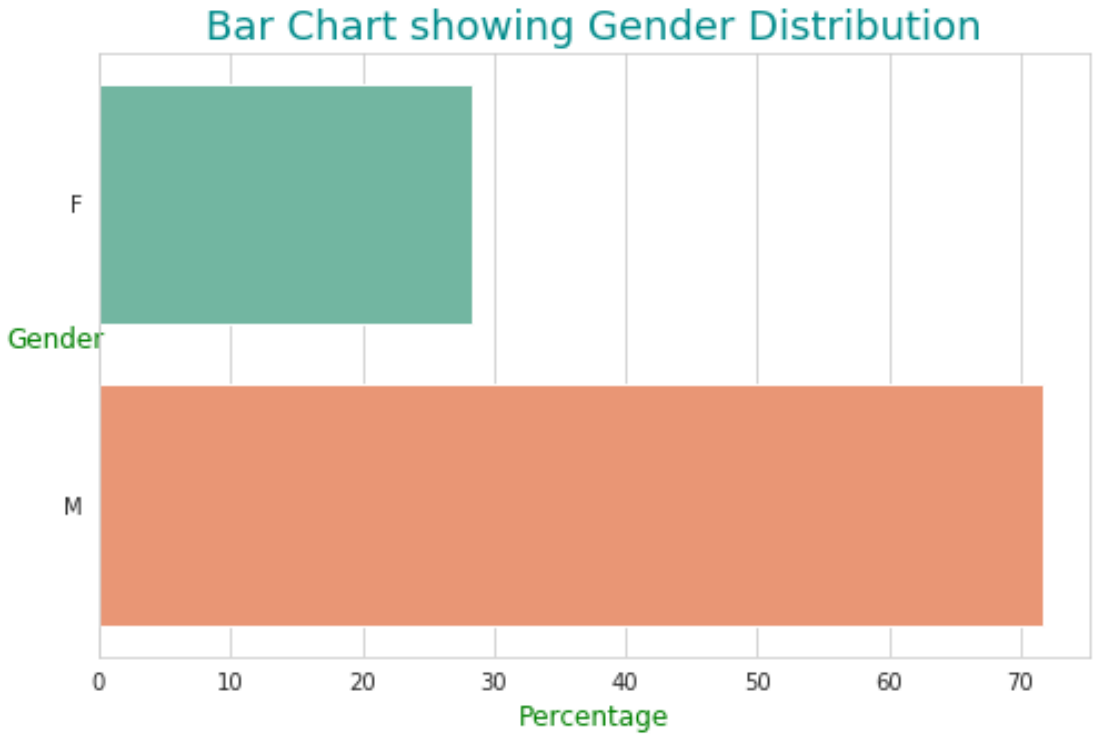
[23]:
```
# Plot a countplot to show the Gender distribution
plt.figure(figsize=(8, 5))
plt.title('Count Plot showing Gender Distribution', fontdict={'size' : 20,␣
↪'color' : 'red'})
sns.set_style('whitegrid')
sns.countplot(x='Gender', data=data)
plt.xlabel('Gender', fontdict={'size' : 14, 'color' : 'green'}, rotation=30)
plt.ylabel('Count', fontdict={'size' : 14, 'color' : 'green'}, rotation=0)
```

[23]:
```
Text(0,0.5,'Count')
```



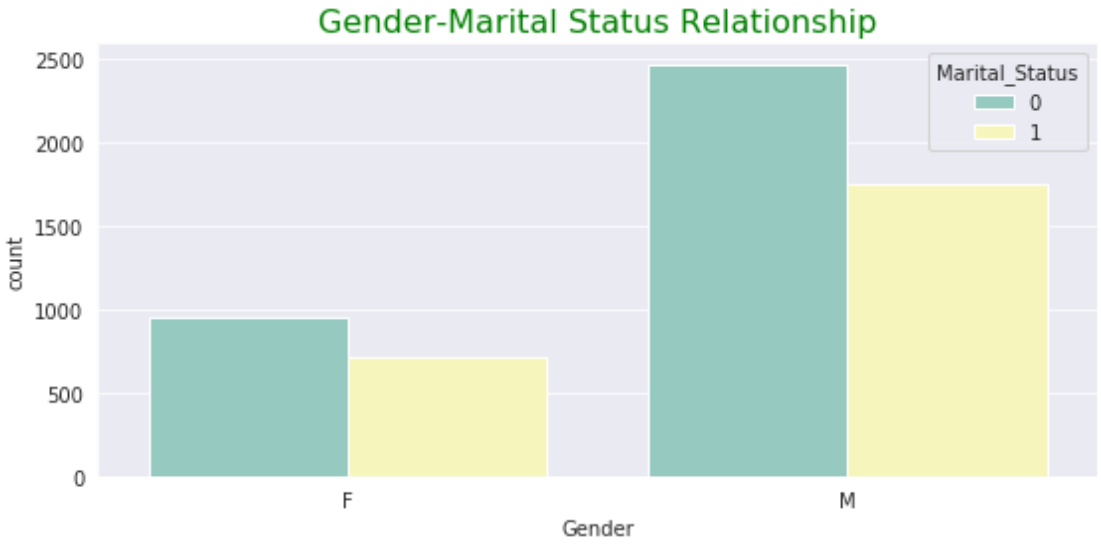
[24]:
```
# Horizontal bar chart showing the distribution of Gender by percentage
plt.figure(figsize=(8, 5))
plt.title('Bar Chart showing Gender Distribution', fontdict={'size' : 18,␣
↪'color' : 'darkcyan'})
sns.set_style('darkgrid')
sns.barplot(x=[pct_of_females, pct_of_males], y=['F', 'M'],␣
↪orient='horizontal', palette='Set2')
plt.ylabel('Gender', fontdict={'size' : 12, 'color' : 'green'}, rotation=0)
plt.xlabel('Percentage', fontdict={'size' : 12, 'color' : 'green'}, rotation=0)
```

[24]:
```
Text(0.5,0,'Percentage')
```



[25]:
```
# Check out the relationship between the gender of the customers and their␣
 ↪marital status
plt.figure(figsize=(9, 4))# changed from 9,5
plt.title('Gender-Marital Status Relationship', fontdict={'size' : 16, 'color' :
 ↪ 'green'})
sns.set_style('darkgrid')
sns.countplot(x='Gender', hue='Marital_Status', data=data, palette="Set3")
```

[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb15c12a4a8>



[26]:
```
print('Finding: From the plots, most customers are male and most are single.')
```

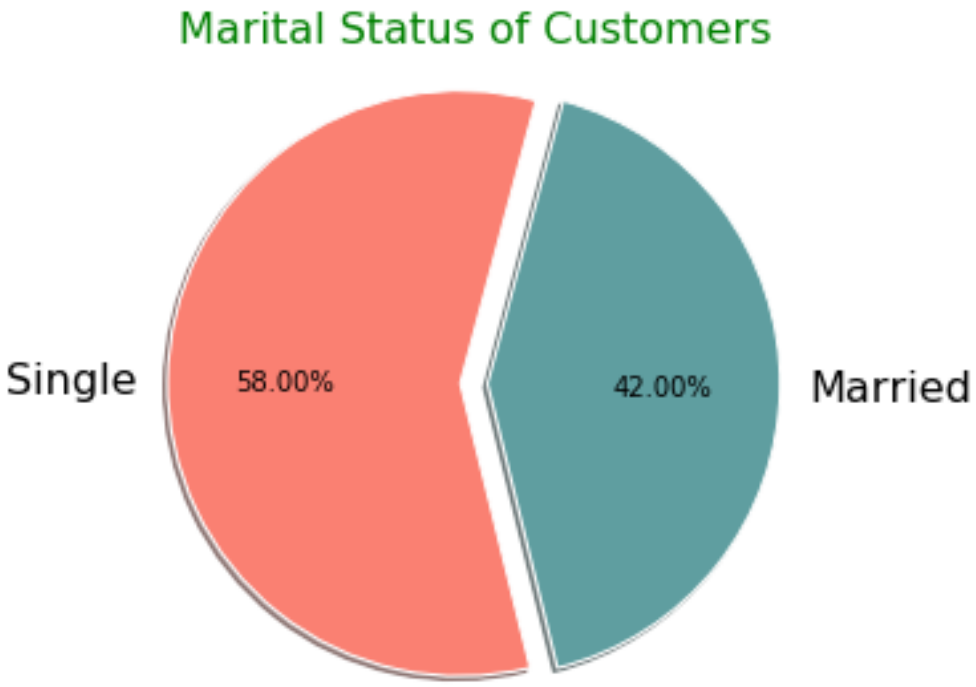
```
Finding: From the plots, most customers are male and most are single.
```

0.2.3 Marital Status Analysis of Customers

[27]:
```
# Calculate the number of each marital status in the customer group
num_of_single = len(data[data['Marital_Status']==0])
num_of_married = len(data[data['Marital_Status']==1])

# Calculate the percentage of both sexes in the customer group
pct_of_single = (num_of_single)/len(data)*100
pct_of_married = (num_of_married)/len(data)*100



# Print Analysis
print('Marital Status Analysis:')
print()

# Output the number of each gender in the customer group
print('Number of Single: {}'.format(num_of_single))
print('Number of Married: {}'.format(num_of_married))
print()

# Output the percentage of each gender in the customer group
print('Percentage of Single: {:.2f}%'.format(pct_of_single))
print('Percentage of Married: {:.2f}%'.format(pct_of_married))
```

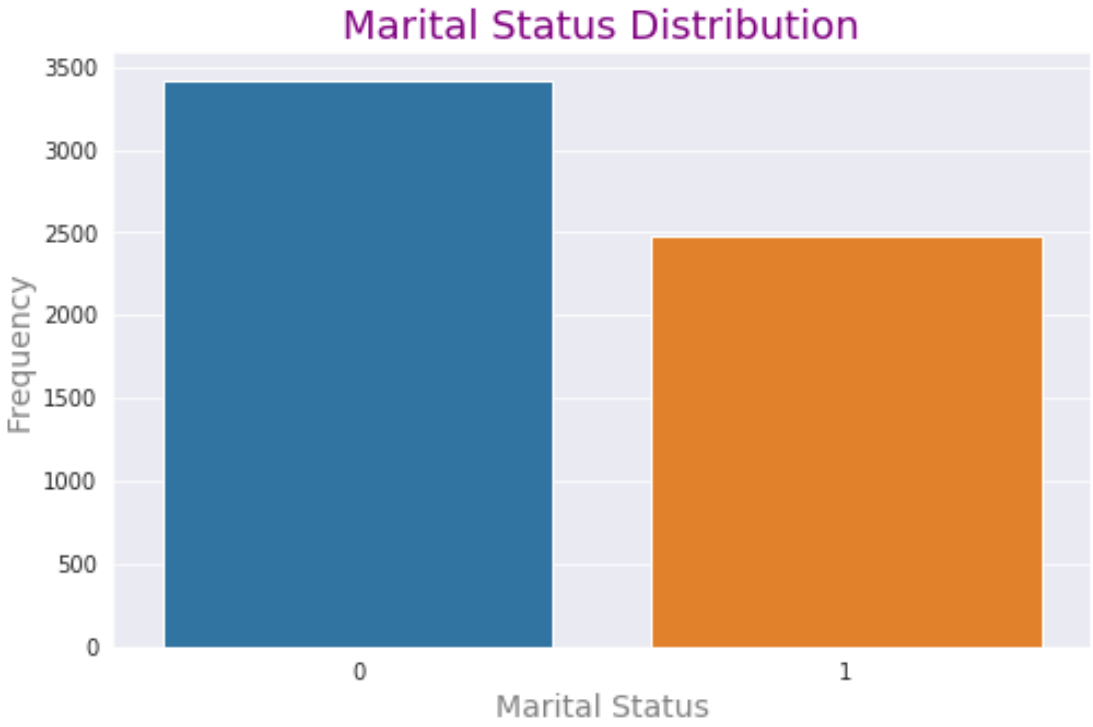
```
Marital Status Analysis:

Number of Single: 3417
Number of Married: 2474

Percentage of Single: 58.00%
Percentage of Married: 42.00%
```

[28]:
```python
# Plot a pie chart to show the Marital Status Distribution
plt.title('Marital Status of Customers', fontdict={'size':16, 'color':'green'})
patches, texts, autotexts = plt.pie(
    x = [pct_of_single, pct_of_married],
    labels = ['Single', 'Married'],
    shadow = True,
    radius=1.5,
    explode = [0.07, 0.07],
    startangle = 75,
    colors = ['salmon', 'cadetblue'],
    autopct = '%.2f%%',
)
for text in texts:
    text.set_color('black')
    text.set_size(16)
for autotext in autotexts:
    autotext.set_color('black')

plt.axis('equal')
plt.tight_layout()
```



[29]:
```
# countplot for marital status distribution
plt.figure(figsize=(8, 5))
plt.title('Marital Status Distribution', fontdict={'size' : 18, 'color' :␣
↪'purple'})
sns.countplot(x='Marital_Status', data=data)
plt.xlabel('Marital Status', fontdict={'size' : 14, 'color' : 'gray'})
plt.ylabel('Frequency', fontdict={'size' : 14, 'color' : 'gray'})
```

[29]:
```
Text(0,0.5,'Frequency')
```



0.2.4 Age Analysis of Customers

[30]:
```
# Age Analysis
# The unique values in the age feature
print('Age Ranges: ', data['Age'].unique())

# Fit the unique age values in a list
age_range = list(data['Age'].unique())
```

```
Age Ranges:  ['0-17' '26-35' '46-50' '18-25' '51-55' '36-45' '55+']
```

[477]:
```
# Calculating the minimum and maximum age groups
print('The mininum age range is {}'.format(data['Age'].min()))
print('The maximum age range is {}'.format(data['Age'].max()))
```

```
The mininum age range is 0-17
The maximum age range is 55+
```

[31]:
```
# Convert the binned ages to a category
under = data[data['Age']==age_range[0]]   #0-17
youngadult = data[data['Age']==age_range[3]]   #18-25
adult = data[data['Age']==age_range[1]]   #26-35
middleaged = data[data['Age']==age_range[5]]   #36-45
```

```
olderadult = data[data['Age']==age_range[2]]   #46-50
senior = data[data['Age']==age_range[4]]   #51-55
old = data[data['Age']==age_range[6]]   #55+
```

[32]:
```
# Calculating the number of customers in each age group
num_of_under = len(under)
num_of_youngadult = len(youngadult)
num_of_adult = len(adult)
num_of_middleaged = len(middleaged)
num_of_olderadult = len(olderadult)
num_of_senior = len(senior)
num_of_old = len(old)
```

[33]:
```
# Print Customer Info
print('Number of under18 Customers: {}'.format(num_of_under))
print('Number of YoungAdult Customers: {}'.format(num_of_youngadult))
print('Number of Adult Customers: {}'.format(num_of_adult))
print('Number of MiddleAged Customers: {}'.format(num_of_middleaged))
print('Number of Older Adult Customers: {}'.format(num_of_olderadult))
print('Number of Seniors: {}'.format(num_of_senior))
print('Number of Old Customers: {}'.format(num_of_old))
```

```
Number of under18 Customers: 218
Number of YoungAdult Customers: 1069
Number of Adult Customers: 2053
Number of MiddleAged Customers: 1167
Number of Older Adult Customers: 531
Number of Seniors: 481
Number of Old Customers: 372
```

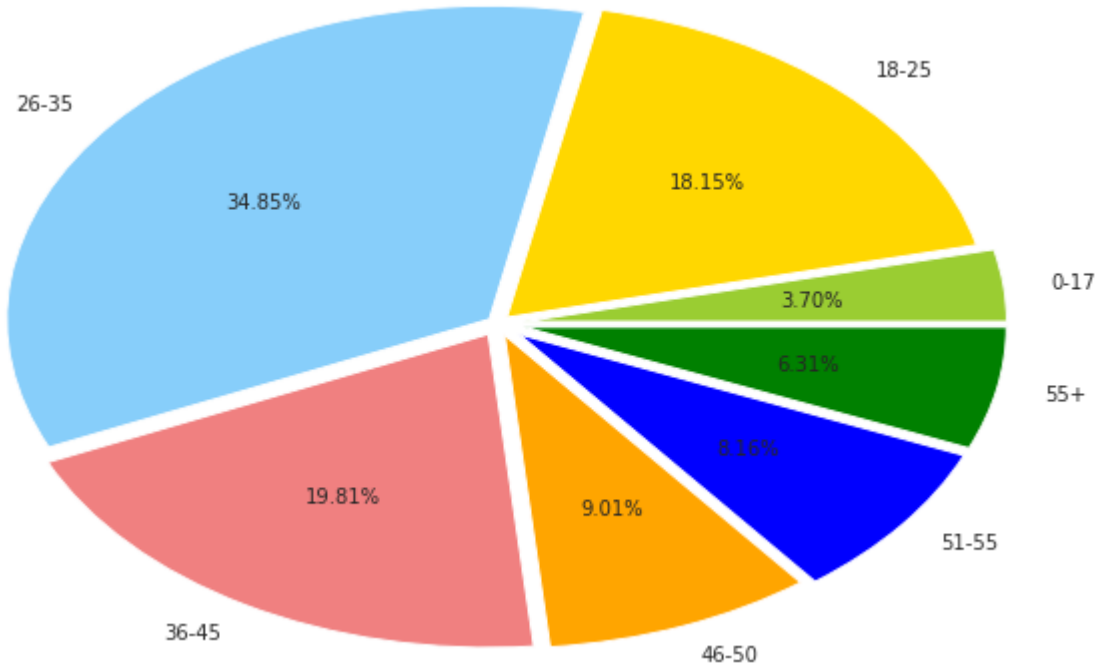
[34]:
```
# Calculating the percentage of of customers that come from each age group
pct_of_under = num_of_under/len(data)*100
pct_of_youngadult = num_of_youngadult/len(data)*100
pct_of_adult = num_of_adult/len(data)*100
pct_of_middleaged = num_of_middleaged/len(data)*100
pct_of_olderadult = num_of_olderadult/len(data)*100
pct_of_senior = num_of_senior/len(data)*100
pct_of_old = num_of_old/len(data)*100
```

[35]:
```
# Print Customer Info in percentages
print('Percentage of Customers Under 18: {:.2f}%'.format(pct_of_under))
print('Percentage of Young Adult Customers: {:.2f}%'.format(pct_of_youngadult))
print('Percentage of Adult Customers: {:.2f}%'.format(pct_of_adult))
print('Percentage of Middle Aged Customers: {:.2f}%'.format(pct_of_middleaged))
print('Percentage of Older Adult Customers: {:.2f}%'.format(pct_of_olderadult))
print('Percentage of Senior Customers: {:.2f}%'.format(pct_of_senior))
print('Percentage of the Old Customers: {:.2f}%'.format(pct_of_old))
```

```
Percentage of Customers Under 18: 3.70%
Percentage of Young Adult Customers: 18.15%
Percentage of Adult Customers: 34.85%
Percentage of Middle Aged Customers: 19.81%
Percentage of Older Adult Customers: 9.01%
Percentage of Senior Customers: 8.16%
Percentage of the Old Customers: 6.31%
```

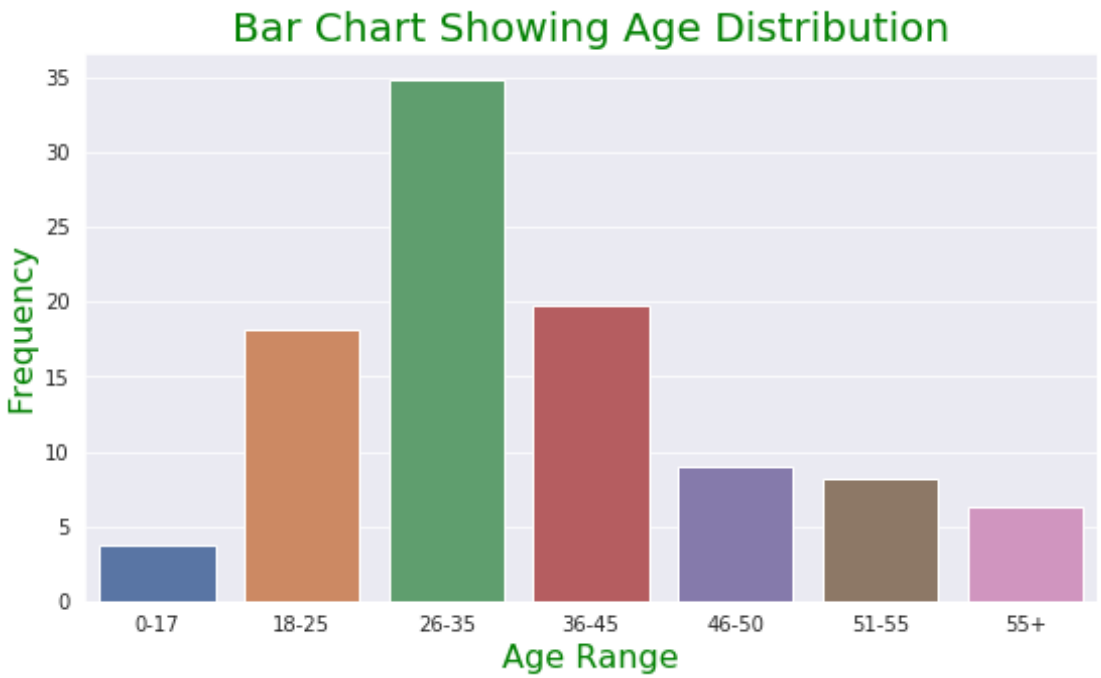
[36]:
```
# Pie chart for the age distribution
plt.pie(
    x = [pct_of_under, pct_of_youngadult, pct_of_adult, pct_of_middleaged,␣
 ↪pct_of_olderadult, pct_of_senior, pct_of_old],
    labels = ['0-17', '18-25', '26-35', '36-45', '46-50', '51-55', '55+'],
    colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', 'orange',␣
 ↪'blue', 'green'],
    autopct = '%.2f%%',
    radius = 1.8,
    explode = [0.1, 0.05, 0.04, 0.07, 0.07, 0.1, 0.1],
)
```

[36]:
```
([<matplotlib.patches.Wedge at 0x7fb15679ddd8>,
  <matplotlib.patches.Wedge at 0x7fb1567a45c0>,
  <matplotlib.patches.Wedge at 0x7fb1567a4d68>,
  <matplotlib.patches.Wedge at 0x7fb1567a8550>,
  <matplotlib.patches.Wedge at 0x7fb1567a8cf8>,
  <matplotlib.patches.Wedge at 0x7fb1567b04e0>,
  <matplotlib.patches.Wedge at 0x7fb1567b0c88>],
 [Text(2.06596,0.241269,'0-17'),
  Text(1.41053,1.4599,'18-25'),
  Text(-1.5782,1.26083,'26-35'),
  Text(-1.03225,-1.77115,'36-45'),
  Text(0.756262,-1.90541,'46-50'),
  Text(1.65172,-1.26421,'51-55'),
  Text(2.0392,-0.409934,'55+')],
 [Text(1.17203,0.136874,'3.70%'),
  Text(0.785171,0.812654,'18.15%'),
  Text(-0.875039,0.699075,'34.85%'),
  Text(-0.579067,-0.99357,'19.81%'),
  Text(0.424244,-1.06889,'9.01%'),
  Text(0.937034,-0.717194,'8.16%'),
  Text(1.15686,-0.232559,'6.31%')])
```



[37]:
```
# Bar chart for the age distribution
#pct_of_under, pct_of_youngadult, pct_of_adult, pct_of_middleaged,␣
 ↪pct_of_olderadult, pct_of_senior, pct_of_old
plt.figure(figsize=(9, 5))#changed from 9,6
plt.title('Bar Chart Showing Age Distribution', color='green', fontsize=20)
sns.barplot(
    x = ['0-17', '18-25', '26-35', '36-45', '46-50', '51-55', '55+'],
    y = [pct_of_under, pct_of_youngadult, pct_of_adult, pct_of_middleaged,␣
 ↪pct_of_olderadult, pct_of_senior, pct_of_old],
    palette="deep"
)
plt.xlabel('Age Range', color='green', fontsize=16)
plt.ylabel('Frequency', color='green', fontsize=16)
```

[37]:
```
Text(0,0.5,'Frequency')
```



0.2.5 Occupation Analysis of Customers

[38]:
```
# Occupations
print('Unique Values for Occupation:')
print()
data['Occupation'].unique()
```

```
Unique Values for Occupation:
```

[38]:
```
array([10, 12,  1,  7,  4,  0, 19, 17, 20, 14, 13,  2,  3, 11,  5, 18,  6,
        9, 16,  8, 15])
```

[39]:
```
# Print Occupation Counts
print('Occupation Counts:')
print()
data['Occupation'].value_counts()
```

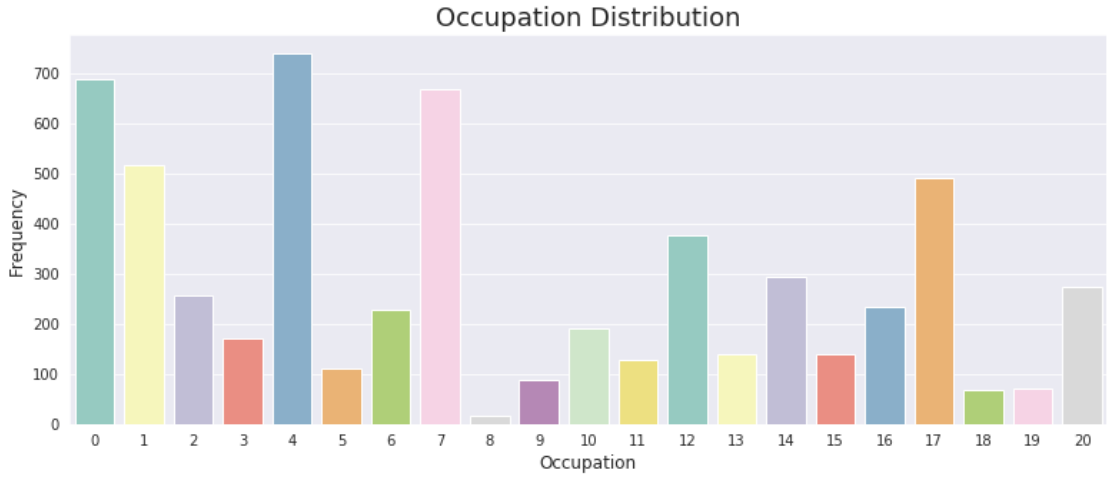
```
Occupation Counts:
```

[39]:
```
4     740
0     688
7     669
1     517
```

```
17    491
12    376
14    294
20    273
2     256
16    235
6     228
10    192
3     170
15    140
13    140
11    128
5     111
9      88
19     71
18     67
8      17
Name: Occupation, dtype: int64
```

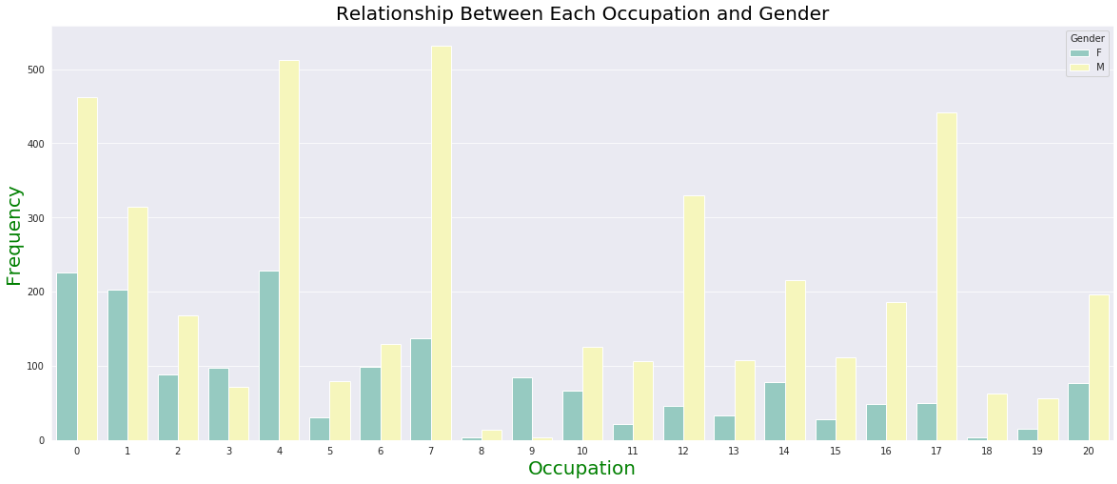
[40]:
```
# Show Occupation Distribution
plt.figure(figsize=(13, 5))
plt.title('Occupation Distribution', size=18)
sns.barplot(data['Occupation'].value_counts().index, data['Occupation'].
 ↪value_counts().values, palette="Set3")
plt.xlabel('Occupation', size=12)
plt.ylabel('Frequency', size=12)
```

[40]:
```
Text(0,0.5,'Frequency')
```



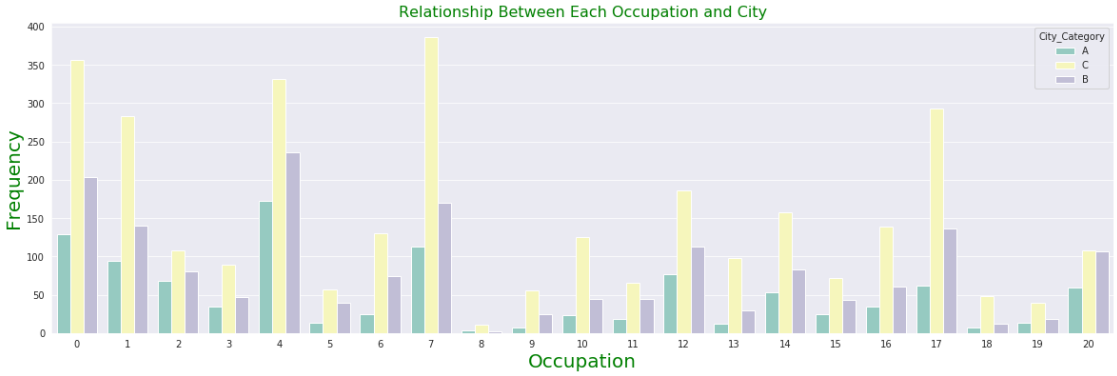
[41]:
```
# Plot the relationship between Occupation and Gender
plt.figure(figsize=(20, 8))
plt.title('Relationship Between Each Occupation and Gender', fontdict={'size' :␣
↪20, 'color':'black'})
sns.countplot(x='Occupation', hue='Gender', data=data,  palette="Set3")
plt.xlabel('Occupation', fontdict={'size' : 20, 'color' : 'green'})
plt.ylabel('Frequency', fontdict={'size' : 20, 'color':'green'})
```

[41]: Text(0,0.5,'Frequency')



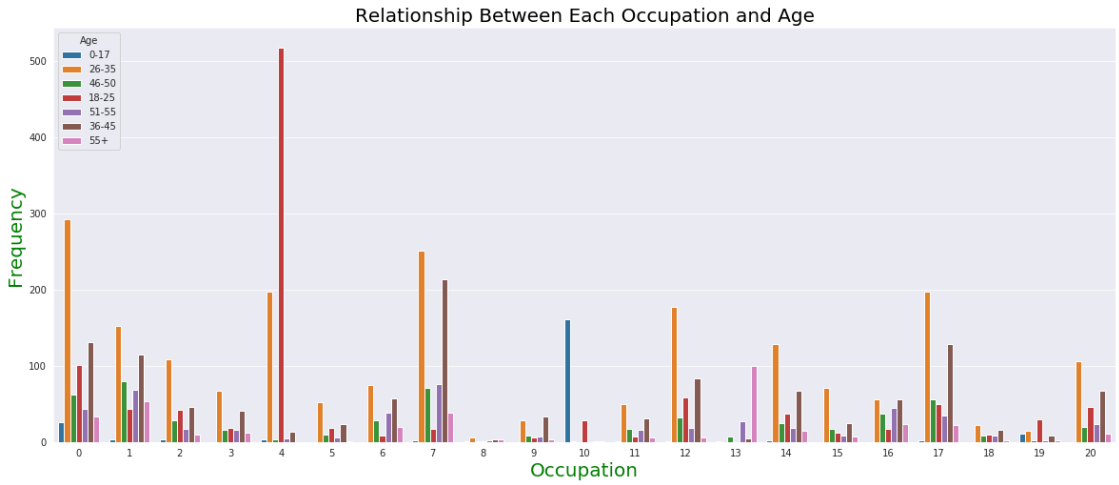
[42]:
```
#Plot the Relationship between Occupation and City
plt.figure(figsize=(20, 6))#changed from 20,8
plt.title('Relationship Between Each Occupation and City', fontdict={'size' :␣
↪16, 'color':'green'})
sns.countplot(x='Occupation', hue='City_Category', data=data,  palette="Set3")
plt.xlabel('Occupation', fontdict={'size' : 20, 'color' : 'green'})
plt.ylabel('Frequency', fontdict={'size' : 20, 'color':'green'})
```

[42]: Text(0,0.5,'Frequency')



[43]:
```
#Plot relationship between Occupation and Age
plt.figure(figsize=(20, 8))
plt.title('Relationship Between Each Occupation and Age', fontdict={'size' :␣
↪20, 'color':'black'})
sns.countplot(x='Occupation', hue='Age', data=data)
plt.xlabel('Occupation', fontdict={'size' : 20, 'color' : 'green'})
plt.ylabel('Frequency', fontdict={'size' : 20, 'color':'green'})
```

[43]:
```
Text(0,0.5,'Frequency')
```



[44]:
```
print('Finding: Many customers ages 18-25 have occupation 4')
print('Finding: Many customers ages 0-17 have occupation 10')
```

```
Finding: Many customers ages 18-25 have occupation 4
Finding: Many customers ages 0-17 have occupation 10
```

0.2.6 Marital Status Analysis by Summed Purchases

[45]:
```
# total sum of purchases made by single and married people
single_purchase = data[data.Marital_Status==0]['Purchase'].values.sum()
married_purchase = data[data.Marital_Status==1]['Purchase'].values.sum()

#percentage of total sum of purchases made by single and married people
pct_of_single_purchase = single_purchase/total_purchases*100
pct_of_married_purchase = married_purchase/total_purchases*100
```

[46]:
```
print('Summed Purchases by Marital Status:')
print('Percentage of Single Purchases: {:.2f}%'.format(pct_of_single_purchase))
```

```
print('Percentage of Married Purchases: {:.2f}%'.
 ↪format(pct_of_married_purchase))
```

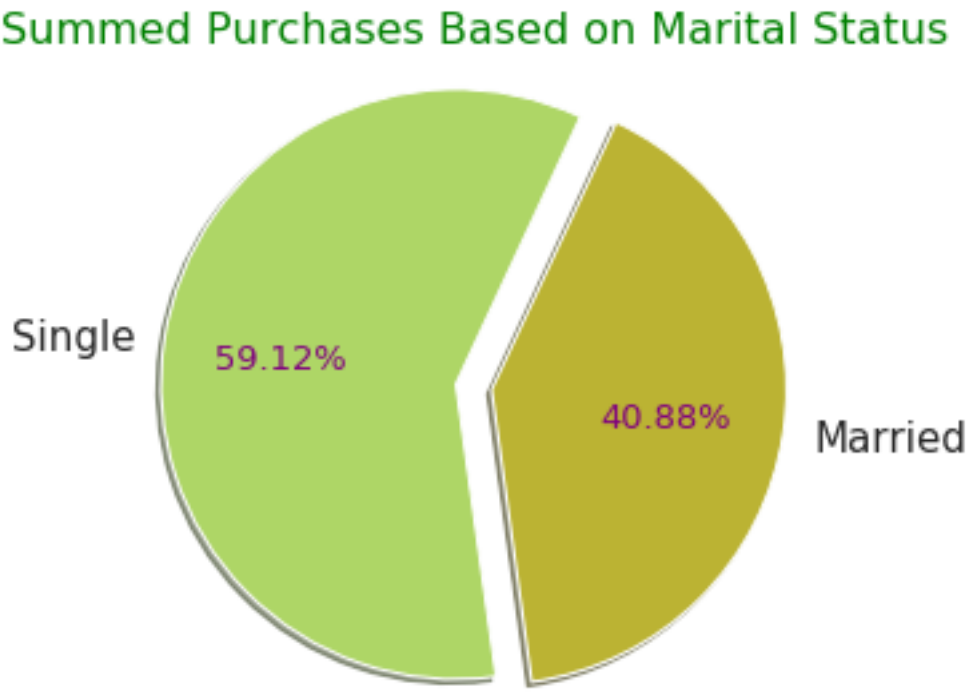
```
Summed Purchases by Marital Status:
Percentage of Single Purchases: 59.12%
Percentage of Married Purchases: 40.88%
```

[47]:
```
print('Finding: More money was spent by single customers.')
```

```
Finding: More money was spent by single customers.
```

[48]:
```
# Plot Marital Status by summed purchase
plt.title('Summed Purchases Based on Marital Status', fontdict={'size' : 16,␣
 ↪'color' : 'green'})
patches, texts, autotexts = plt.pie(
    x=[pct_of_single_purchase, pct_of_married_purchase],
    labels=['Single', 'Married'],
    radius = 1.3,
    colors = ['#aed666', '#bbb333'],
    autopct= '%.2f%%',
    startangle=65,
    shadow=True,
    explode = [0.08, 0.08]

)

for text in texts:
    text.set_size(15)
for autotext in autotexts:
    autotext.set_size(13)
    autotext.set_color('purple')

plt.tight_layout()
plt.axis('equal')
```

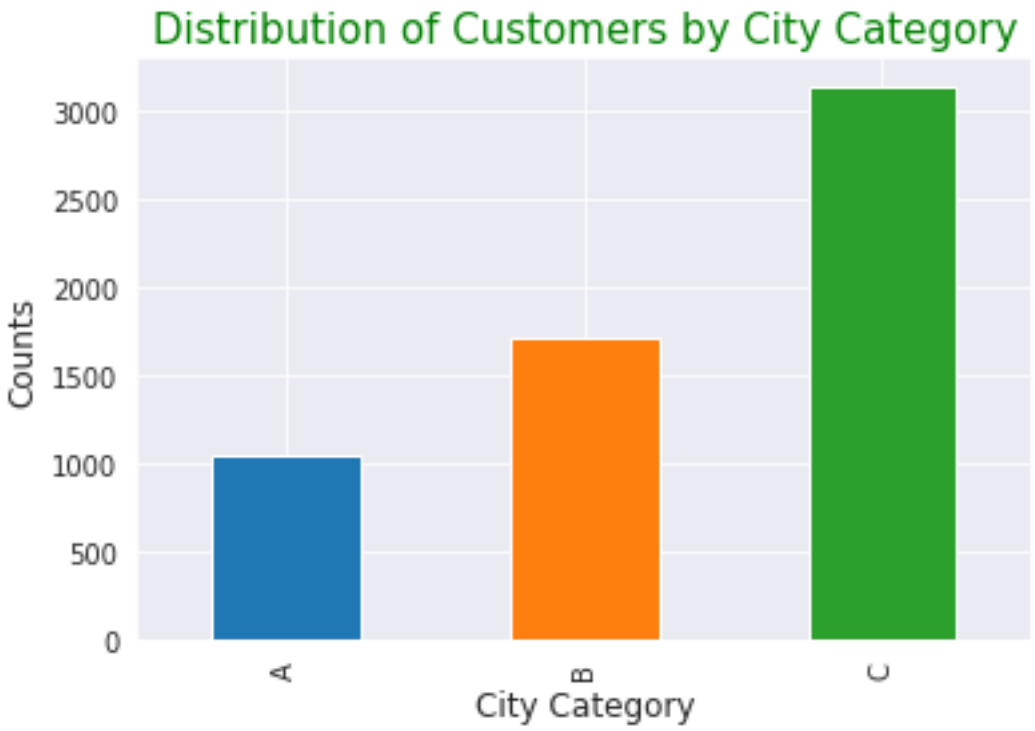
[48]:
```
(-1.5347628270910967,
 1.5464952703543695,
 -1.4344876090819103,
 1.4481196249284183)
```



0.2.7 City Category of Customers

[49]:
```
# Customers by City Category
plt.figure(figsize=(6, 4))
plt.title("Distribution of Customer by City Category")
plt.title('Distribution of Customers by City Category', fontdict={'size' : 15,␣
↪'color':'green'})
data['User_ID'].groupby(data['City_Category']).nunique().plot(kind='bar')
plt.xlabel('City Category', size=12)
plt.ylabel('Counts', size=12)
```

[49]:
```
Text(0,0.5,'Counts')
```

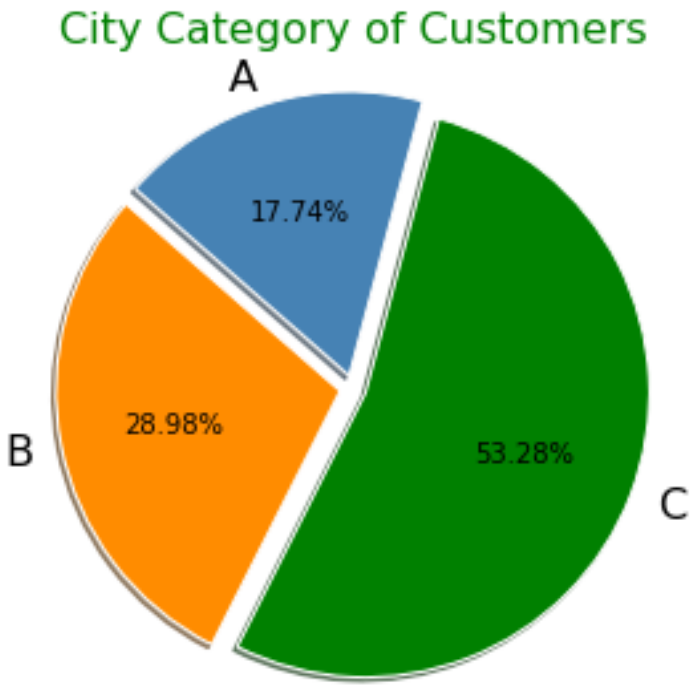


[50]:
```
# Calculate the number of customers from each city category
num_of_A = len(data[data['City_Category']=='A'])
num_of_B = len(data[data['City_Category']=='B'])
num_of_C = len(data[data['City_Category']=='C'])

# Calculate the percentage of customers from each city category
pct_of_A = (num_of_A)/len(data)*100
pct_of_B = (num_of_B)/len(data)*100
pct_of_C = (num_of_C)/len(data)*100

# Print Analysis
print('Customers based on City Category:')
print()

# Output the number of each gender in the customer group
print('Customers from A: {}'.format(num_of_A))
print('Customers from B: {}'.format(num_of_B))
print('Customers from C: {}'.format(num_of_C))

print()

# Output the percentage of each gender in the customer group
print('Percentage of Customers from A: {:.2f}%'.format(pct_of_A))
```

```
print('Percentage of Customers from B: {:.2f}%'.format(pct_of_B))
print('Percentage of Customers from C: {:.2f}%'.format(pct_of_C))
```

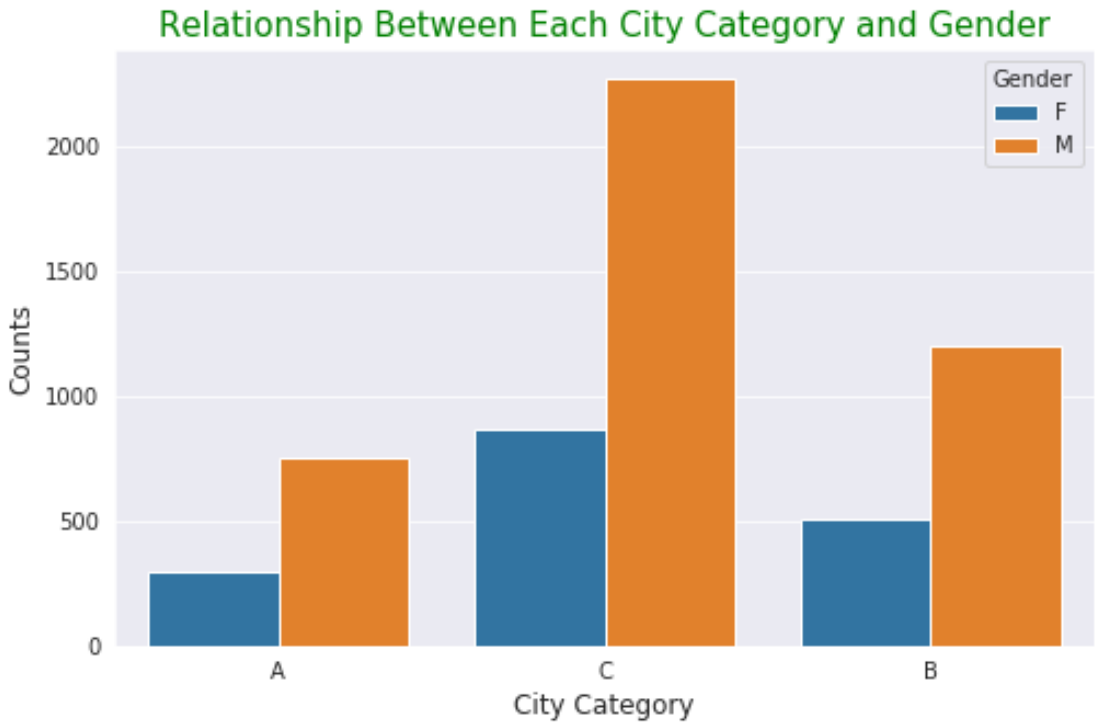
```
Customers based on City Category:

Customers from A: 1045
Customers from B: 1707
Customers from C: 3139

Percentage of Customers from A: 17.74%
Percentage of Customers from B: 28.98%
Percentage of Customers from C: 53.28%
```

[51]:
```
# Plot a pie chart to show the City Category Distribution
plt.title('City Category of Customers', fontdict={'size':16, 'color':'green'})
patches, texts, autotexts = plt.pie(
    x = [pct_of_A, pct_of_B, pct_of_C],
    labels = ['A', 'B', 'C'],
    shadow = True,
    radius=1.5,
    explode = [0.07, 0.07, 0.07],
    startangle = 75,
    colors = ['steelblue', 'darkorange', 'green'],
    autopct = '%.2f%%',
)
for text in texts:
    text.set_color('black')
    text.set_size(16)
for autotext in autotexts:
    autotext.set_color('black')

plt.axis('equal')
plt.tight_layout()
```



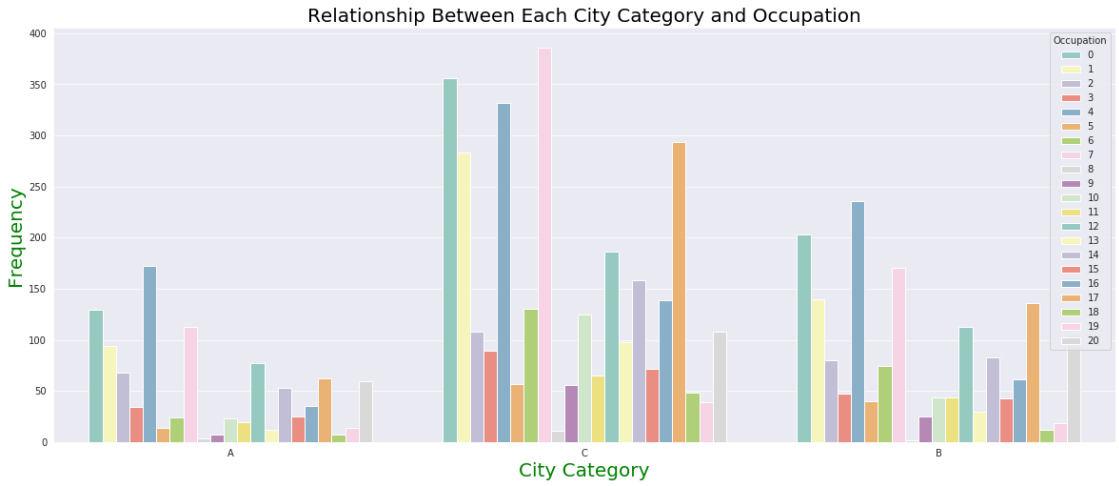
[52]:
```
# Plot city category distribution of customers based on gender
plt.figure(figsize=(8, 5))
plt.title('Relationship Between Each City Category and Gender',␣
 ↪fontdict={'size' : 15, 'color' : 'green'})
sns.countplot(x='City_Category', hue='Gender', data=data)
plt.xlabel('City Category', size=12)
plt.ylabel('Counts', size=12)
```

[52]:
```
Text(0,0.5,'Counts')
```



[53]:
```
# Plot the relationship between City and Occupation
plt.figure(figsize=(20, 8))
plt.title('Relationship Between Each City Category and Occupation',␣
 ↪fontdict={'size' : 20, 'color':'black'})
sns.countplot(x='City_Category', hue='Occupation', data=data, palette='Set3')
plt.xlabel('City Category', fontdict={'size' : 20, 'color' : 'green'})
plt.ylabel('Frequency', fontdict={'size' : 20, 'color':'green'})
```

[53]: Text(0,0.5,'Frequency')



0.2.8 City Category Summed by Purchases

[54]:
```
# Sum Purchases made by customers from city categories A, B, C
A_purchase = data[data.City_Category=='A']['Purchase'].values.sum()
B_purchase = data[data.City_Category=='B']['Purchase'].values.sum()
C_purchase = data[data.City_Category=='C']['Purchase'].values.sum()
```

[55]:
```
# Percentage of total sum of purchases made by customers from city categories␣
↪A,B,C
pct_of_A = A_purchase/total_purchases*100
pct_of_B = B_purchase/total_purchases*100
pct_of_C = C_purchase/total_purchases*100
```

[56]:
```
print('Summed Purchases by City Category:')
print('Percentage from A: {:.2f}%'.format(pct_of_A))
print('Percentage from B: {:.2f}%'.format(pct_of_B))
print('Percentage from C: {:.2f}%'.format(pct_of_C))
```

```
Summed Purchases by City Category:
Percentage from A: 25.82%
Percentage from B: 41.52%
Percentage from C: 32.66%
```

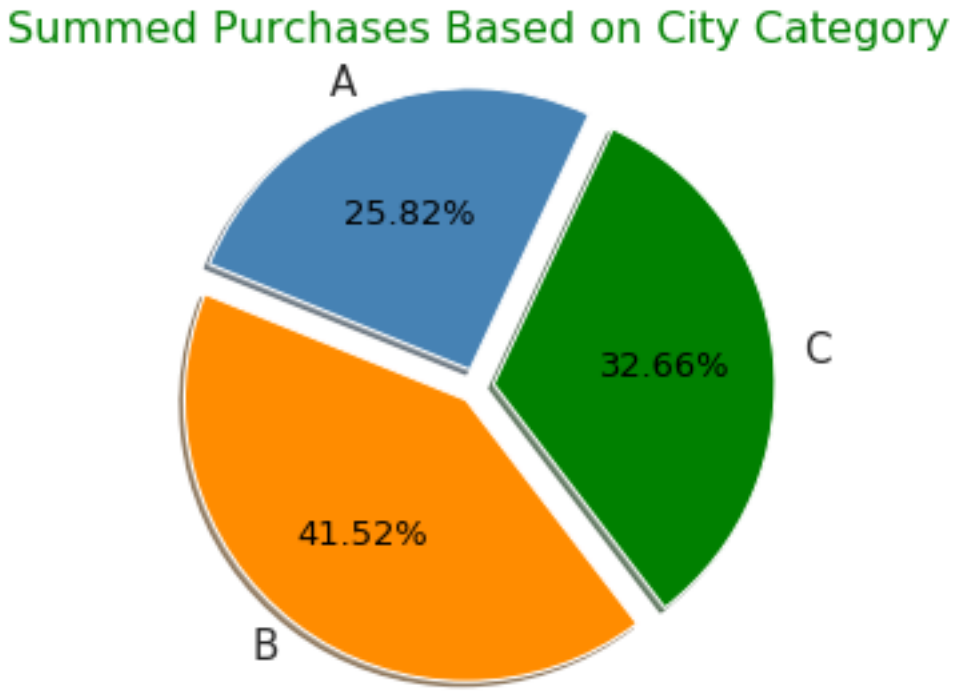
[57]:
```
print('Finding: More money was spent by customers from City Category B.')
```

```
Finding: More money was spent by customers from City Category B.
```

[58]:
```
plt.title('Summed Purchases Based on City Category', fontdict={'size' : 16,␣
↪'color' : 'green'})
patches, texts, autotexts = plt.pie(
    x=[pct_of_A, pct_of_B, pct_of_C],
    labels=['A', 'B','C'],
    radius = 1.3,
    colors = ['steelblue', 'darkorange', 'green'],
    autopct= '%.2f%%',
    startangle=65,
    shadow=True,
    explode = [0.08, 0.08, .08]

)

for text in texts:
    text.set_size(15)
for autotext in autotexts:
```

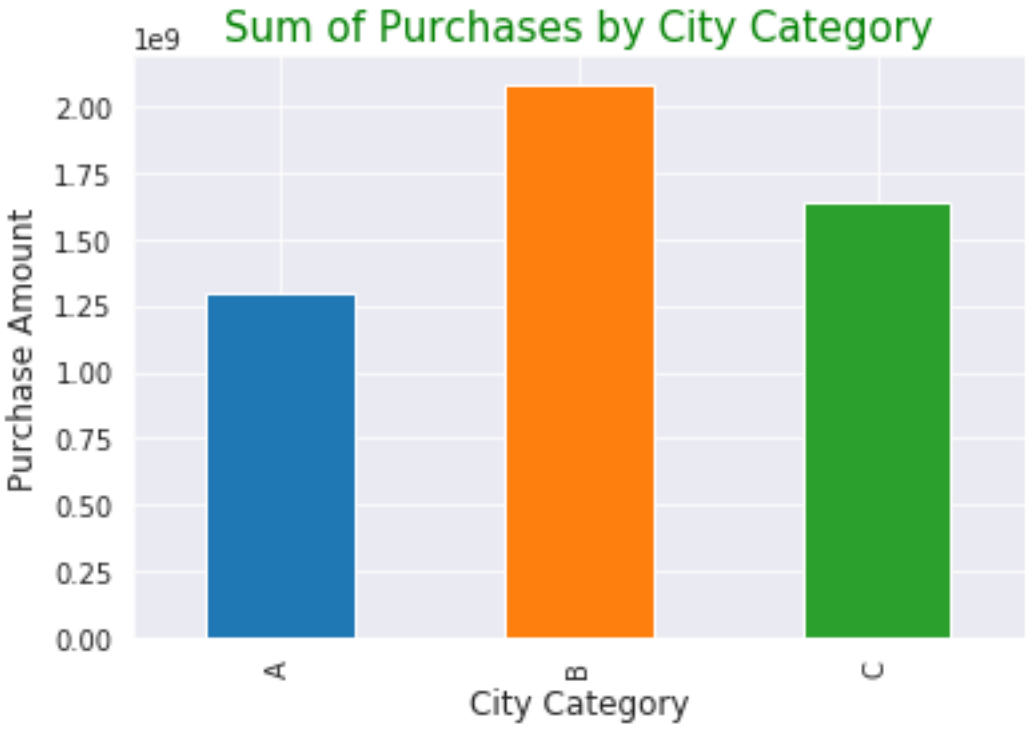
```
    autotext.set_size(13)
    autotext.set_color('black')

plt.tight_layout()
plt.axis('equal')
```

[58]: (-1.516287388222629, 1.534774607661738, -1.5011398494436086, 1.516408661558438)



[59]:
```
# Show Bar Chart of Summed Purchases based on City Category
plt.figure(figsize=(6, 4))
var5 = data.groupby('City_Category').Purchase.sum()
plt.title('Sum of Purchases by City Category', fontdict={'size' : 15, 'color':
 ↪'green'})
var5.plot(kind='bar')
plt.xlabel('City Category', size=12)
plt.ylabel('Purchase Amount', size=12)
```

[59]: Text(0,0.5,'Purchase Amount')



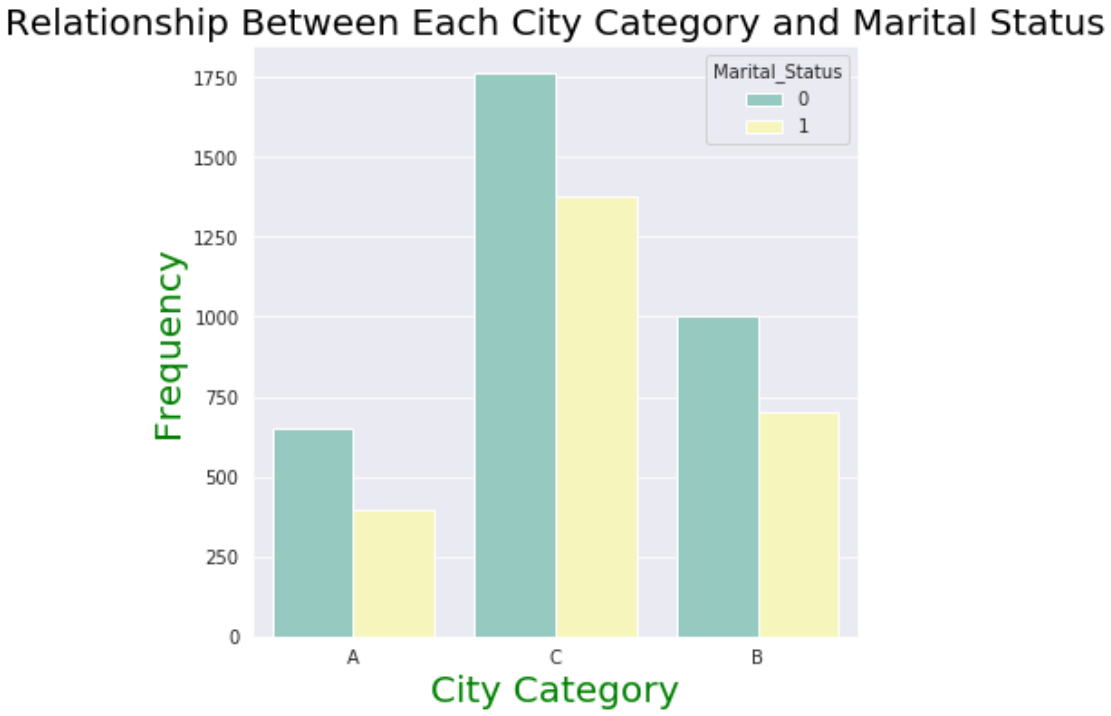
[60]:
```
print('Finding: More customers are from City Category C, but more money was␣
 ↪spent by customers from City Category B.')
```

Finding: More customers are from City Category C, but more money was spent by customers from City Category B.

[61]:
```
## What's going on with City Category C? Why aren't they spending more?

# We looked at city category by occupation
# We looked at city category by gender

# We can still look by age and by marital status
```

[62]:
```
# Plot the relationship between City and Marital Status

plt.figure(figsize=(6, 6))
plt.title('Relationship Between Each City Category and Marital Status',␣
 ↪fontdict={'size' : 20, 'color':'black'})
sns.countplot(x='City_Category', hue='Marital_Status', data=data,␣
 ↪palette='Set3')
plt.xlabel('City Category', fontdict={'size' : 20, 'color' : 'green'})
plt.ylabel('Frequency', fontdict={'size' : 20, 'color':'green'})
```

[62]: Text(0,0.5,'Frequency')



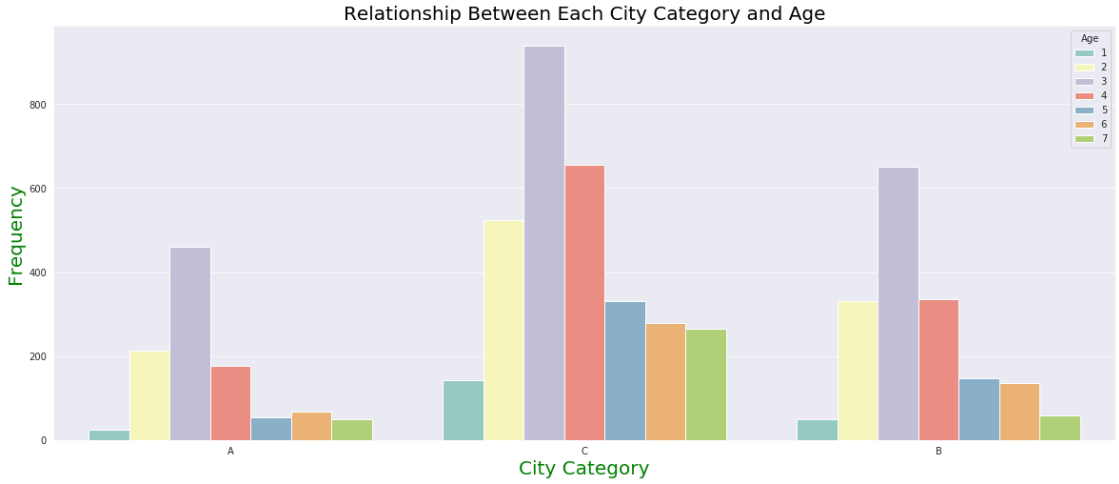
[63]:
```
## Look at age, but convert age so it's in numerical order

# Convert values to numbers: Ages
# Create function
def map_age(age):
    if age == '0-17':
        return 1
    elif age == '18-25':
        return 2
    elif age == '26-35':
        return 3
    elif age == '36-45':
        return 4
    elif age == '46-50':
        return 5
    elif age == '51-55':
        return 6
    else:
        return 7

# Apply function
data['Age'] = data['Age'].apply(map_age)
```

[64]:
```
# Check
print('Age Ranges: ', data['Age'].unique())
age_range = list(data['Age'].unique())
```

```
Age Ranges:  [1 3 5 2 6 4 7]
```

[65]:
```
# Plot the relationship between City and Age

plt.figure(figsize=(20, 8))
plt.title('Relationship Between Each City Category and Age', fontdict={'size' :␣
↪20, 'color':'black'})
sns.countplot(x='City_Category', hue='Age', data=data, palette='Set3')
plt.xlabel('City Category', fontdict={'size' : 20, 'color' : 'green'})
plt.ylabel('Frequency', fontdict={'size' : 20, 'color':'green'})
```

[65]:
```
Text(0,0.5,'Frequency')
```



[66]:
```
# Key for Ages
# 1 = 0-17
# 2 = 18-25
# 3 = 26-35
# 4 = 36-45
# 5 = 46-50
# 6 = 51-55
# 7 = 55+
```

[69]:
```
print('Finding: City Category C has a larger proportion of 55+ customers than␣
↪City Category B.')
print('Finding: City Category C seems to skew older in general.')
```

```
Finding: City Category C has a larger proportion of 55+ customers than City
```

Category B.
Finding: City Category C seems to skew older in general.

0.3 Data Change: Look at Purchases, not Customers

[70]:
```
#Bring the original data set back that has the product information so that␣
 ↪additional analysis can be done at the purchase level
body = client_525f5d2245b6446693c23e44b7178e2a.
 ↪get_object(Bucket='finalproject-donotdelete-pr-pr6xudhaekii34',Key='dataoutput2.
 ↪csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__,␣
 ↪body )

data = pd.read_csv(body)
print("Head of Purchase Data:")
print()
data.head()
```

Head of Purchase Data:

[70]:

	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_1	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	

	GenderRecode	AgeRecode	CityRecode
0	1	0	1
1	1	0	1
2	1	0	1
3	1	0	1
4	2	6	3

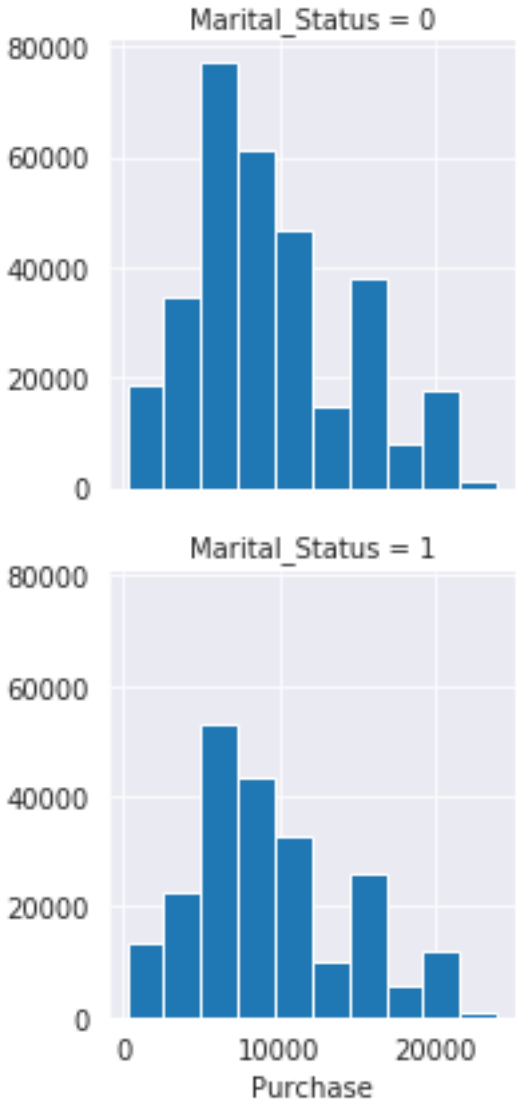
[71]:
```
# Check the data types
print('Data Types of Purchase Data:')
print(data.dtypes)
```

Data Types of Purchase Data:

```
User_ID                      int64
Product_ID                  object
Gender                      object
Age                         object
Occupation                   int64
City_Category               object
Stay_In_Current_City_Years  object
Marital_Status               int64
Product_Category_1           int64
Purchase                     int64
GenderRecode                 int64
AgeRecode                    int64
CityRecode                   int64
dtype: object
```

[72]:
```
# Show BoxPlot of Purchases

# Create a subset to better visualize
data2=data.loc[ : , ['Purchase'] ]

# Visualize
sns.boxplot(data=data2)
plt.title("Distribution of Purchases")
plt.show()
```

[73]:
```
# Purchases based on customer marital status
g = sns.FacetGrid(data, 'Marital_Status')
g = g.map(plt.hist, 'Purchase', bins=10)
```



[74]:
```
# Print the city categories in the data set
print("City Categories:")
print()
data['City_Category'].unique()
```

```
City Categories:
```

[74]:
```
array(['A', 'C', 'B'], dtype=object)
```

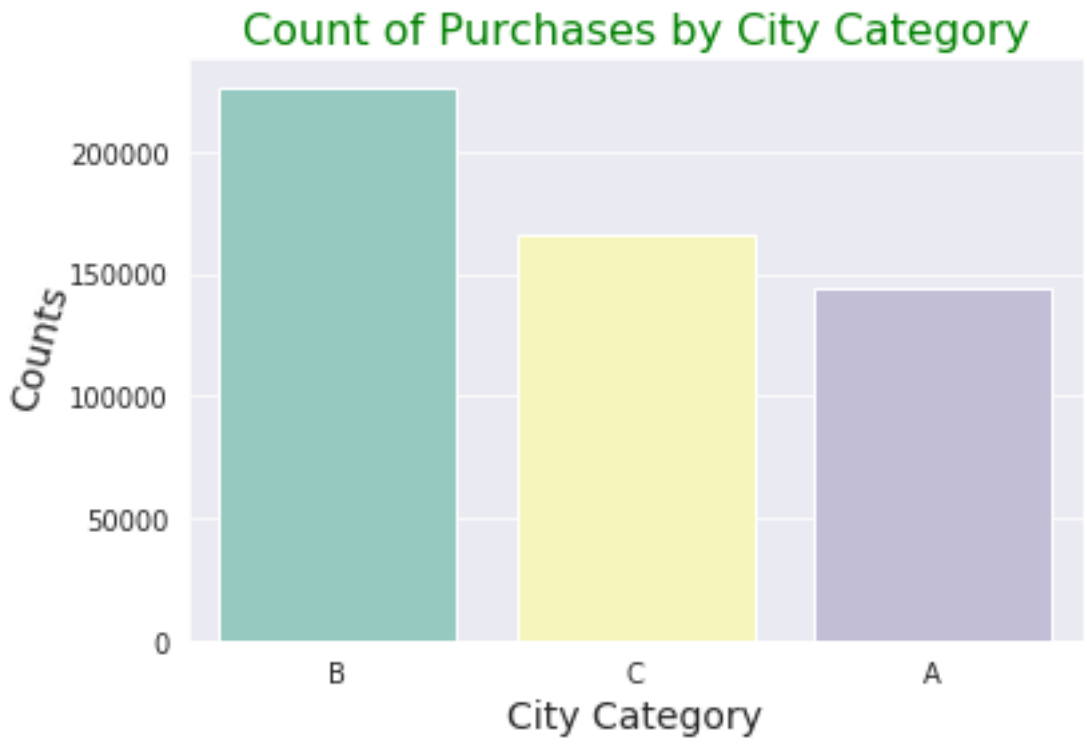
[75]:
```
city_category_indices = data['City_Category'].value_counts().index    #city␣
 ↪category
city_category_values = data['City_Category'].value_counts().values    #number of␣
 ↪purchases from each city
```

[76]:
```
print('Count of Purchases from City B : {}'.format(city_category_values[0]))
print('Count of Purchases from City C : {}'.format(city_category_values[1]))
print('Count of Purchases from City A : {}'.format(city_category_values[2]))
```

```
Count of Purchases from City B : 226493
Count of Purchases from City C : 166446
Count of Purchases from City A : 144638
```

[77]:
```
# City category counts with a bar chart (not summed purchases)
plt.title('Count of Purchases by City Category', fontdict={'size' : 16, 'color'␣
 ↪: 'green'})
sns.barplot(x=city_category_indices, y=city_category_values,  palette="Set3")
plt.xlabel('City Category', size=14)
plt.ylabel('Counts', size=14, rotation=75)
```

[77]:
```
Text(0,0.5,'Counts')
```

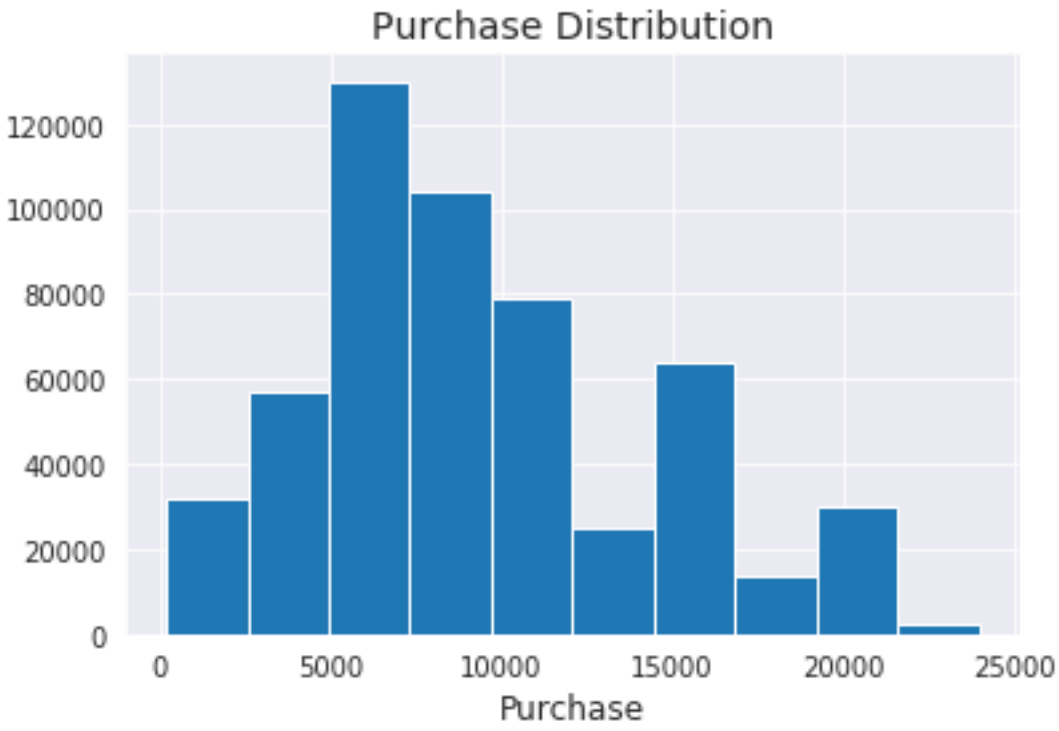


[78]:
```
print('Finding: Most purchases are from city category B.')
```

Finding: Most purchases are from city category B.

[79]:
```
# Histogram showing the distribution of Purchases
plt.title('Purchase Distribution', size=14)
data['Purchase'].hist()
plt.xlabel('Purchase', size=12)
```

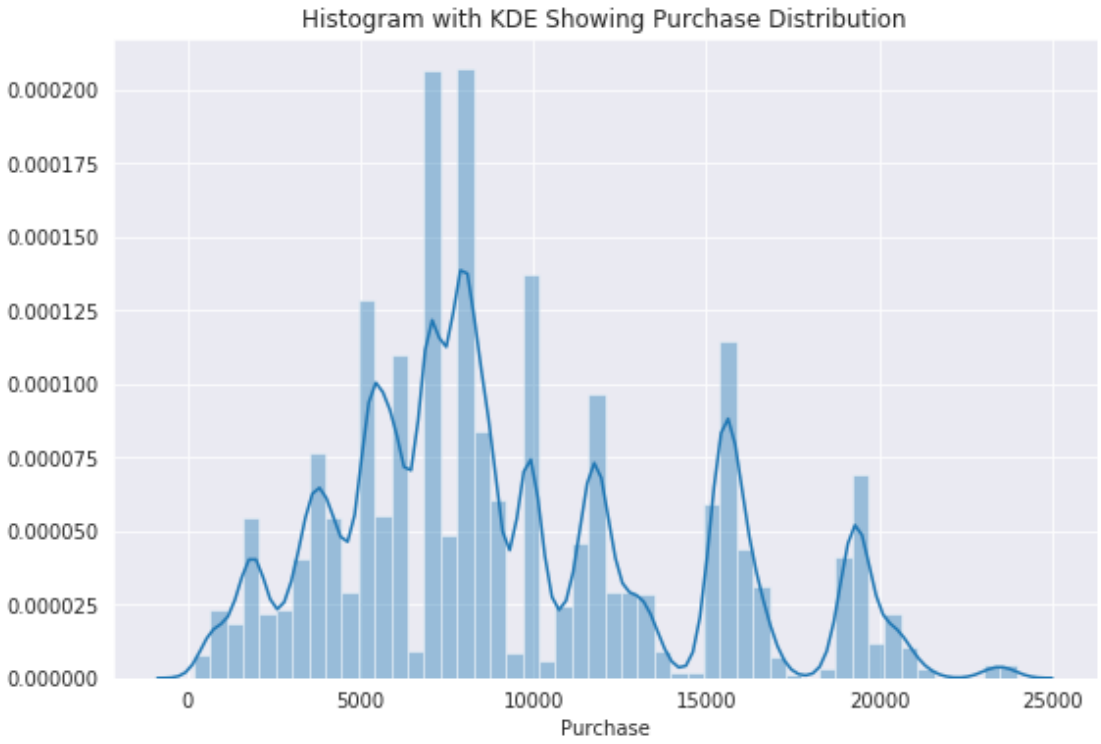
[79]: Text(0.5,0,'Purchase')



[80]:
```
# Show Histogram with KDE Showing Purchase Distribution
plt.figure(figsize=(9, 6))
plt.title('Histogram with KDE Showing Purchase Distribution')
sns.distplot(data['Purchase'])
```

```
/usr/local/src/conda3_runtime/home/envs/DSX-Python35-Spark/lib/python3.5/site-
packages/scipy/stats/stats.py:1706: FutureWarning: Using a non-tuple sequence
for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of
`arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different
result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

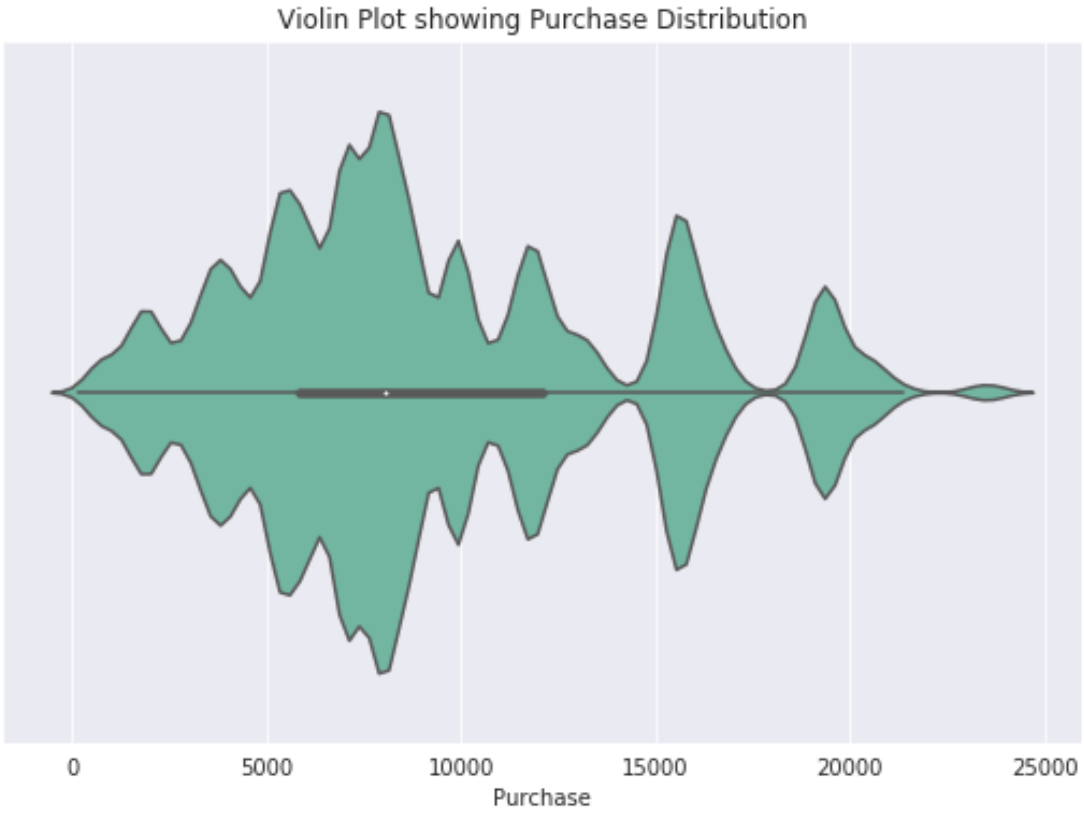
[80]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb10bbbcba8>



[81]:
```
# Make a violin plot showing the distribution:
plt.figure(figsize=(9, 6))
plt.title('Violin Plot showing Purchase Distribution')
sns.violinplot(x='Purchase', data=data, palette='Set2')
```

```
/usr/local/src/conda3_runtime/home/envs/DSX-Python35-Spark/lib/python3.5/site-
packages/scipy/stats/stats.py:1706: FutureWarning: Using a non-tuple sequence
for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of
`arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different
result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

[81]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb10bba7d30>
```



[82]:
```
# Get the mean, standard deviation, min/max
print("Descriptive Statistics:")
print()
print("Mean value: ",data['Purchase'].mean(), "\nStandard deviation:",␣
 ↪data['Purchase'].std(),
      "\nMaximun value ", data['Purchase'].max(), "\nMinimun value" ,␣
 ↪data['Purchase'].min())
```

```
Descriptive Statistics:

Mean value:  9333.859852635065
Standard deviation: 4981.022132656483
Maximun value  23961
Minimun value 185
```

0.4 Create New DataFrames

[83]:
```
# Create a new data frame called 'df_prodcat1'
df_prodcat1 = data.groupby(['Product_Category_1',␣
 ↪'Gender','Age','Occupation','City_Category','Marital_Status'])['Purchase'].
 ↪sum()
```

[84]:
```
# Export the data

# @hidden_cell
# The project token is an authorization token that is used to access project␣
 ↪resources like data sources, connections, and used by platform APIs.
from project_lib import Project
project = Project(project_id='4595243a-5466-4f19-80ec-e1ae2abb13fc',␣
 ↪project_access_token='p-7ec26b4fc07c0d14a20db0050aa1997ba5b65235')
pc = project.project_context

project.save_data(data=df_prodcat1.to_csv(index=True),file_name='prodcat1.
 ↪csv',overwrite=True)
```

[84]:
```
{'asset_id': 'f8284e1f-b6e9-49b2-8170-eab8e0d7c811',
 'bucket_name': 'finalproject-donotdelete-pr-pr6xudhaekii34',
 'file_name': 'prodcat1.csv',
 'message': 'File prodcat1.csv has been written successfully to the associated
OS'}
```

[85]:
```
# Create a new data frame called 'df_cust1'
df_cust1 = data.groupby(['User_ID',␣
 ↪'Gender','Age','Occupation','City_Category','Marital_Status'])['Purchase'].
 ↪sum()
```

[86]:
```
# This imports the product category data which sums purchases
body = client_525f5d2245b6446693c23e44b7178e2a.
 ↪get_object(Bucket='finalproject-donotdelete-pr-pr6xudhaekii34',Key='prodcat1.
 ↪csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__,␣
 ↪body )

productcategory = pd.read_csv(body)
# Look at head of data
productcategory.head()
```

[86]:

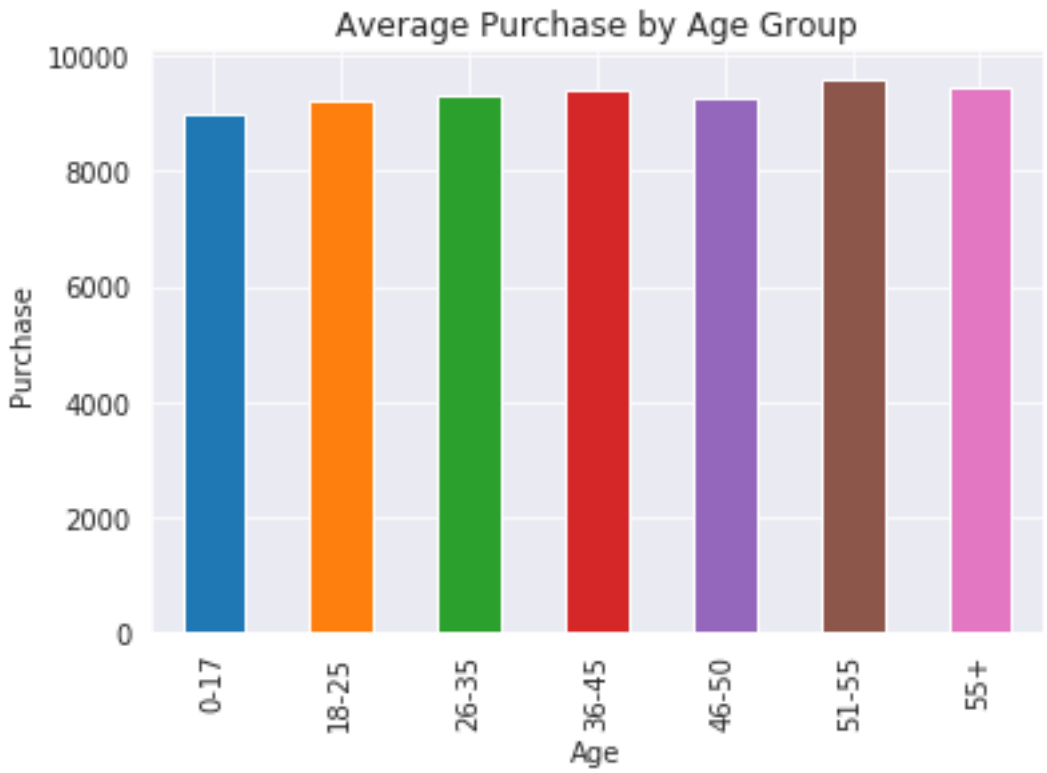
	1	F	0-17	0	A	0.1	12113
0	1	F	0-17	0	B	0	1042281
1	1	F	0-17	0	C	0	364058
2	1	F	0-17	1	B	0	583109
3	1	F	0-17	2	A	0	50236
4	1	F	0-17	2	C	0	50376

[87]:
```
# Show Purchase distribution by gender
sns.boxplot(x="Gender", y="Purchase", hue="Gender", data=data, palette="PRGn")
plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")
plt.title("Purchase Distribution by Gender")
```

```
plt.show()
```

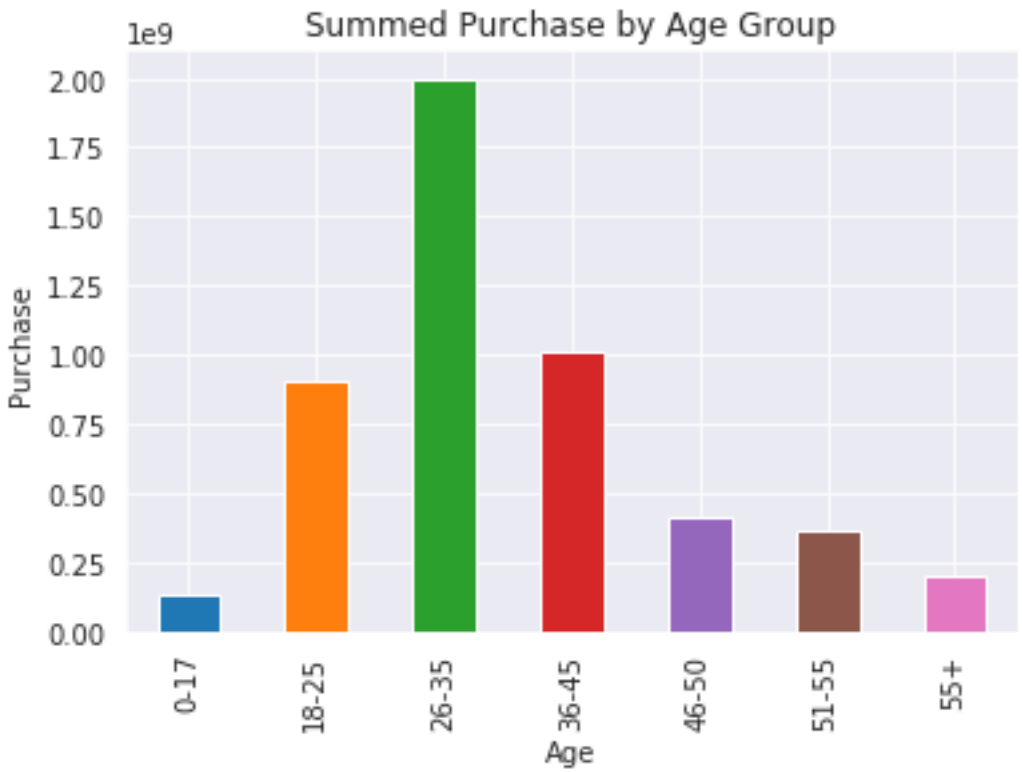
[88]:
```
# Show Average Purchase by Age
var = data.groupby('Age').Purchase.mean() #grouped mean of sales at Age level
fig = plt.figure()

ax1 = fig.add_subplot(1,1,1)
ax1.set_xlabel('Age'),

ax1.set_ylabel('Purchase')

ax1.set_title("Average Purchase by Age Group")
var.plot(kind='bar')
```

[88]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb10b182f98>
```



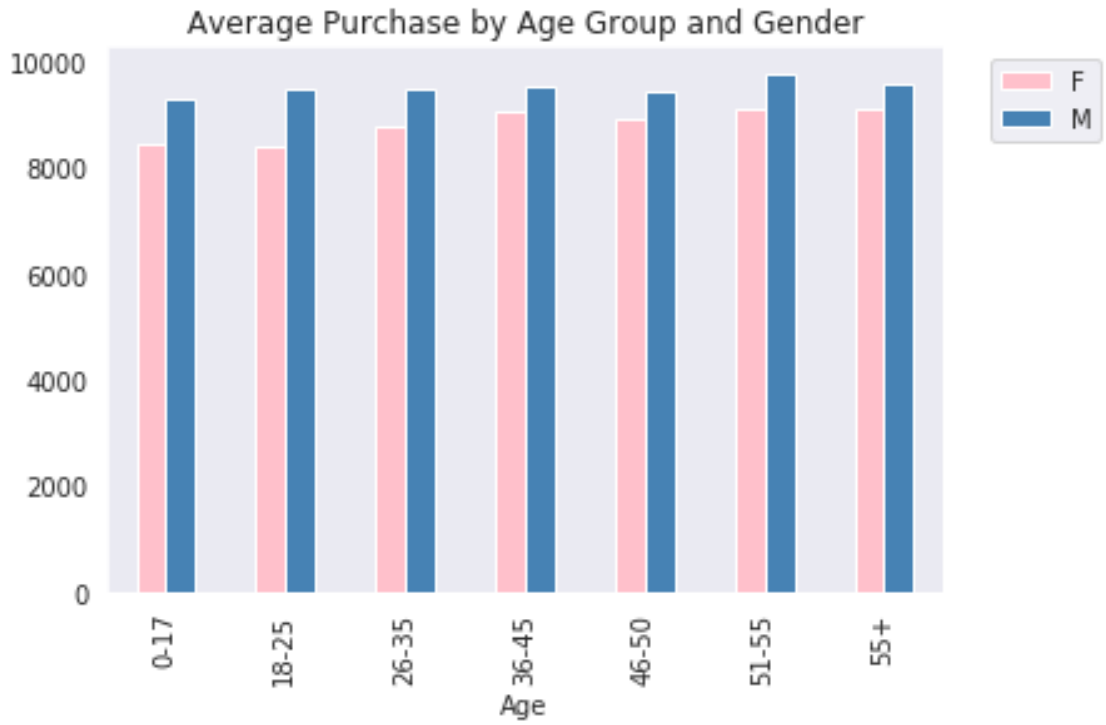
[89]:
```
# Show Summed Purchase by Age
var = data.groupby('Age').Purchase.sum()
fig = plt.figure()

ax1 = fig.add_subplot(1,1,1)
ax1.set_xlabel('Age'),

ax1.set_ylabel('Purchase')

ax1.set_title("Summed Purchase by Age Group")
var.plot(kind='bar')
```

[89]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb10b1abbe0>



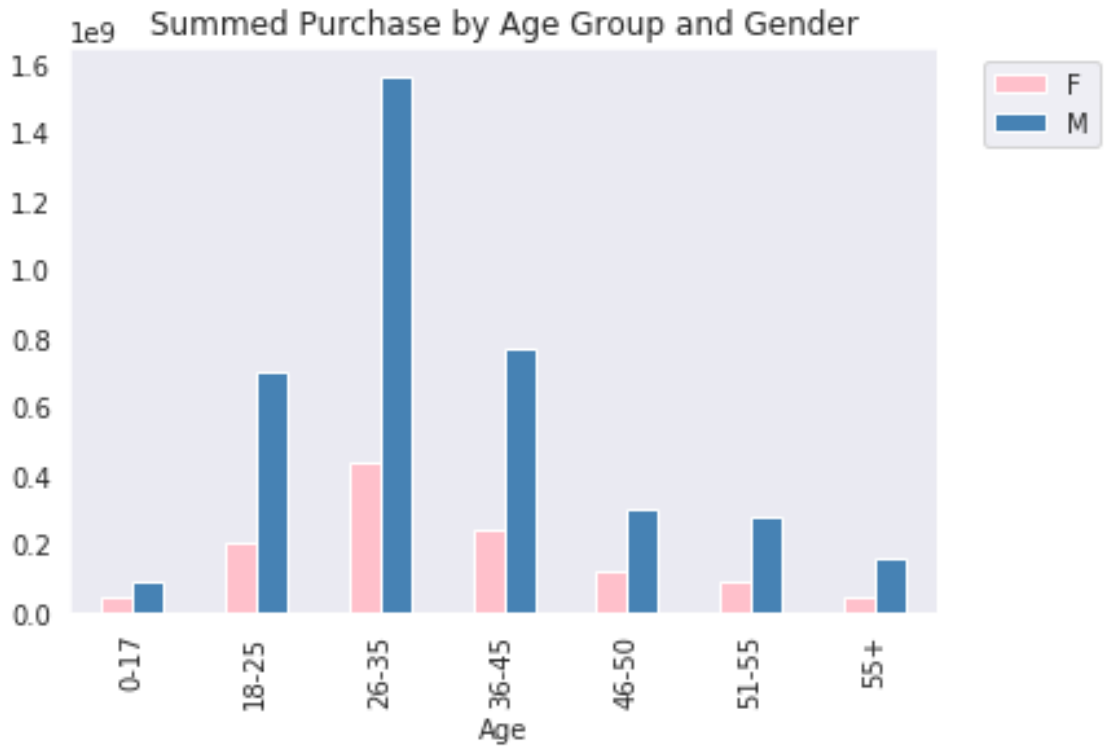
[90]:
```
# Show Average Purchase by Age Group and Gender
var = data.groupby(['Age','Gender']).Purchase.mean()#grouped mean of sales at␣
 ↪Age & Gender level
var.unstack().plot(kind='bar',stacked=False,  legend=True,␣
 ↪color=['pink','steelblue'], grid=False)
plt.title("Average Purchase by Age Group and Gender")
plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")

#males age 51-55 are spending the highest amount of money on average
```

[90]: <matplotlib.legend.Legend at 0x7fb10b10c240>



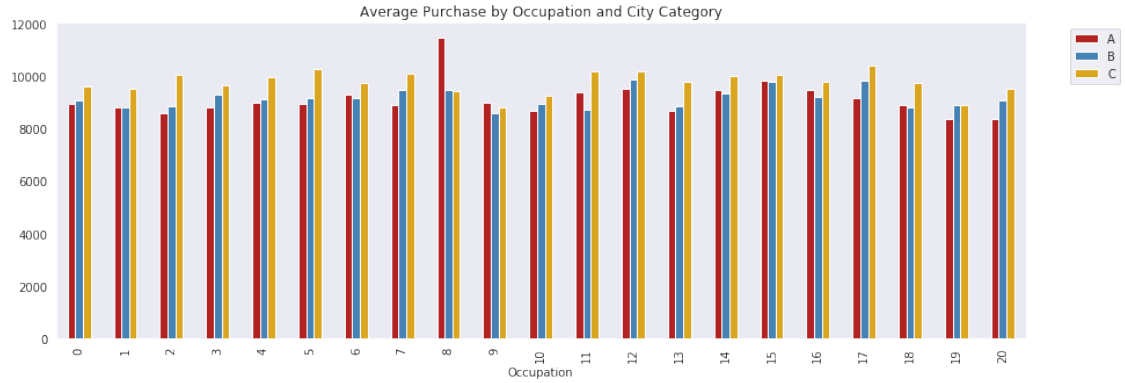
[91]:
```
# Show Summed Purchase by Age Group and Gender
var = data.groupby(['Age','Gender']).Purchase.sum()
var.unstack().plot(kind='bar',stacked=False,  legend=True,␣
 ↪color=['pink','steelblue'], grid=False)
plt.title("Summed Purchase by Age Group and Gender")
plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")
```

[91]:
```
<matplotlib.legend.Legend at 0x7fb10b1900f0>
```



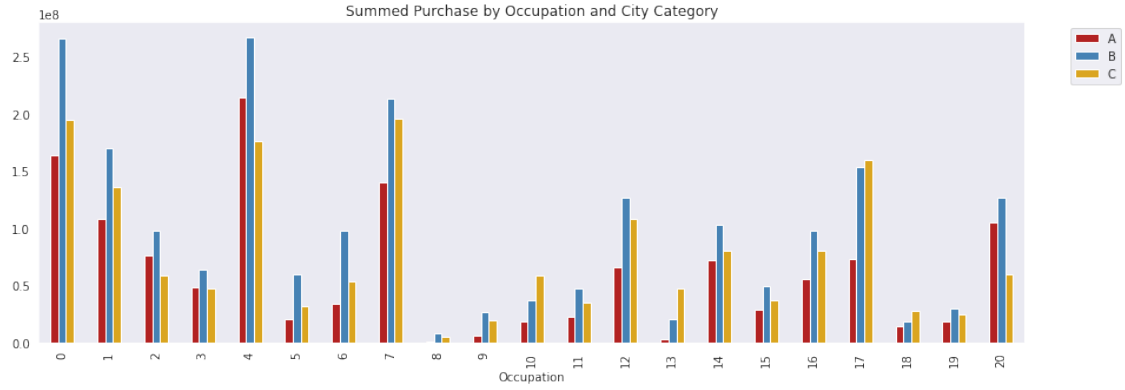
[92]:
```
# Show Average Purchase by Occupation by City

#change parameters before plotting
%matplotlib inline
from pylab import rcParams
rcParams['figure.figsize'] = 15,5 #a larger number on the left makes the image␣
 ↪wider

var = data.groupby(['Occupation','City_Category']).Purchase.mean()
var.unstack().plot(kind='bar',stacked=False,  color=['firebrick','steelblue',␣
 ↪'goldenrod'], grid=False)
plt.title("Average Purchase by Occupation and City Category")
plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")
```

[92]: <matplotlib.legend.Legend at 0x7fb10b13dcf8>



[93]:
```
# Show Summed Purchase by Occupation by City


rcParams['figure.figsize'] = 15,5 #set parameters
var = data.groupby(['Occupation','City_Category']).Purchase.sum()
var.unstack().plot(kind='bar',stacked=False,  color=['firebrick','steelblue',␣
↪'goldenrod'], grid=False)
plt.title("Summed Purchase by Occupation and City Category")
plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")
```

[93]:
```
<matplotlib.legend.Legend at 0x7fb10b0d5fd0>
```



0.5 K-Means Clustering Model: City Category

[94]:
```
#body = client_525f5d2245b6446693c23e44b7178e2a.
↪get_object(Bucket='finalproject-donotdelete-pr-pr6xudhaekii34',Key='ProductCombinedKmeans2.
↪csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
```

```
#if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__,␣
 ↪body )
body = client_525f5d2245b6446693c23e44b7178e2a.
 ↪get_object(Bucket='finalproject-donotdelete-pr-pr6xudhaekii34',Key='dataoutput4.
 ↪csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__,␣
 ↪body )


modelf = pd.read_csv(body)
modelf.head()

print("Length:")
print(len(modelf))
```

```
Length:
537577
```

[95]:
```
# Import more libraries
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler

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

```
/usr/local/src/conda3_runtime/home/envs/DSX-
Python35-Spark/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning:
numpy.dtype size changed, may indicate binary incompatibility. Expected 96, got
88
  return f(*args, **kwds)
```

[96]:
```
from scipy.stats import uniform
modelf = modelf.dropna()
np.random.seed(1234)
modelf['runiform'] = uniform.rvs(loc = 0, scale = 1, size = len(modelf))
#modelf['runiform'] = uniform.rvs(loc = 0, scale = 1)
modelf_train = modelf[modelf['runiform'] >= 0.33]
modelf_test = modelf[modelf['runiform'] < 0.33]

# check training data frame
print('*****train data frame*******')
#print('\nmodelf_train data frame (rows, columns): ',modelf_train.shape)
```

```
print(modelf_train.info())
print('\n')

# check test data frame
print('*****test data frame*******')
#print('\nmodelf_test data frame (rows, columns): ',modelf_test.shape)
print(modelf_test.info())
print('\n')

project.save_data(data=modelf_train.to_csv(index=False),file_name='train.
 ↪csv',overwrite=True)
project.save_data(data=modelf_test.to_csv(index=False),file_name='test.
 ↪csv',overwrite=True)

print("Train Shape:")
print(modelf_train.shape)
print()
print("Test Shape:")
print(modelf_test.shape)
```

```
*****train data frame*******
<class 'pandas.core.frame.DataFrame'>
Int64Index: 360078 entries, 1 to 537576
Data columns (total 8 columns):
Occupation            360078 non-null int64
Marital_Status        360078 non-null int64
Product_Category_1    360078 non-null int64
Purchase              360078 non-null int64
GenderRecode          360078 non-null int64
AgeRecode             360078 non-null int64
CityRecode            360078 non-null int64
runiform              360078 non-null float64
dtypes: float64(1), int64(7)
memory usage: 24.7 MB
None


*****test data frame*******
<class 'pandas.core.frame.DataFrame'>
Int64Index: 177499 entries, 0 to 537575
Data columns (total 8 columns):
Occupation            177499 non-null int64
Marital_Status        177499 non-null int64
Product_Category_1    177499 non-null int64
Purchase              177499 non-null int64
GenderRecode          177499 non-null int64
AgeRecode             177499 non-null int64
```

```
CityRecode            177499 non-null int64
runiform              177499 non-null float64
dtypes: float64(1), int64(7)
memory usage: 12.2 MB
None


Train Shape:
(360078, 8)

Test Shape:
(177499, 8)
```

[97]:
```
X = np.array(modelf_train.drop(['CityRecode'], 1).astype(float))
y = np.array(modelf_train['CityRecode'])
```

[98]:
```
kmeans = KMeans(n_clusters=3) # Trying 3, since there are 3 city categories
kmeans.fit(X)
```

[98]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

[99]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto', # Trying 3␣
 ↪clusters
    random_state=None, tol=0.0001, verbose=0)
```

[99]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

[100]:
```
# How many correct predictions?
correct = 0
for i in range(len(X)):
    predict_me = np.array(X[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == y[i]:
        correct += 1

print(correct/len(X))

# For 7 clusters, about 16%
# For 3 clusters, about 25%
```

```
0.2488127572359322
```

0.6 K-Means Clustering Model: Occupation

[101]:
```
#body = client_525f5d2245b6446693c23e44b7178e2a.
 ↪get_object(Bucket='finalproject-donotdelete-pr-pr6xudhaekii34',Key='ProductCombinedKmeans2.
 ↪csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
#if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__,␣
 ↪body )
body = client_525f5d2245b6446693c23e44b7178e2a.
 ↪get_object(Bucket='finalproject-donotdelete-pr-pr6xudhaekii34',Key='dataoutput4.
 ↪csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__,␣
 ↪body )


modelf = pd.read_csv(body)
modelf.head()

print("Length:")
print(len(modelf))
```

```
Length:
537577
```

[102]:
```
from scipy.stats import uniform
modelf = modelf.dropna()
np.random.seed(1234)
modelf['runiform'] = uniform.rvs(loc = 0, scale = 1, size = len(modelf))
#modelf['runiform'] = uniform.rvs(loc = 0, scale = 1)
modelf_train = modelf[modelf['runiform'] >= 0.33]
modelf_test = modelf[modelf['runiform'] < 0.33]
# check training data frame
print('*****train data frame*******')
#print('\nmodelf_train data frame (rows, columns): ',modelf_train.shape)
print(modelf_train.info())
print('\n')
# check test data frame
print('*****test data frame*******')
#print('\nmodelf_test data frame (rows, columns): ',modelf_test.shape)
print(modelf_test.info())
print('\n')

project.save_data(data=modelf_train.to_csv(index=False),file_name='train.
 ↪csv',overwrite=True)
project.save_data(data=modelf_test.to_csv(index=False),file_name='test.
 ↪csv',overwrite=True)
```

```
print("Train Shape:")
print(modelf_train.shape)
print()
print("Test Shape:")
print(modelf_test.shape)
```

```
*****train data frame*******
<class 'pandas.core.frame.DataFrame'>
Int64Index: 360078 entries, 1 to 537576
Data columns (total 8 columns):
Occupation            360078 non-null int64
Marital_Status        360078 non-null int64
Product_Category_1    360078 non-null int64
Purchase              360078 non-null int64
GenderRecode          360078 non-null int64
AgeRecode             360078 non-null int64
CityRecode            360078 non-null int64
runiform              360078 non-null float64
dtypes: float64(1), int64(7)
memory usage: 24.7 MB
None


*****test data frame*******
<class 'pandas.core.frame.DataFrame'>
Int64Index: 177499 entries, 0 to 537575
Data columns (total 8 columns):
Occupation            177499 non-null int64
Marital_Status        177499 non-null int64
Product_Category_1    177499 non-null int64
Purchase              177499 non-null int64
GenderRecode          177499 non-null int64
AgeRecode             177499 non-null int64
CityRecode            177499 non-null int64
runiform              177499 non-null float64
dtypes: float64(1), int64(7)
memory usage: 12.2 MB
None


Train Shape:
(360078, 8)

Test Shape:
(177499, 8)
```

[103]:
```
X = np.array(modelf_train.drop(['Occupation'], 1).astype(float))
y = np.array(modelf_train['Occupation'])
```

[104]:
```
# Select Number of Clusters
kmeans = KMeans(n_clusters=21) # Trying 21, since there are 21 occupations
kmeans.fit(X)
```

[104]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=21, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

[105]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=21, n_init=10, n_jobs=1, precompute_distances='auto', # Trying␣
 ↪21 clusters
    random_state=None, tol=0.0001, verbose=0)
```

[105]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=21, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

[106]:
```
# How many correct predictions?
correct = 0
for i in range(len(X)):
    predict_me = np.array(X[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == y[i]:
        correct += 1

print(correct/len(X))

# For 7 clusters, about 8%
# For 21 clusters, about 6%
```

```
0.05956487205549909
```

[ ]:
```
# End of Black Friday Notebook 1
```