Black_Friday_Notebook_3

September 16, 2019

0.1 Black Friday Notebook 3

[152]:
```
# IST 718 Big Data Analytics, Final Project: Black Friday
# Notebook Name: Black Friday Notebook 3
# Notebook Contents: Association Rules, Random Forest, Naive Bayes,
# Linear Regression, Decision Tree Regressor, K Neighbors Regressor
# Written for Watson IBM Cloud using Python 3.5 with Spark 2.1
# Group Members: Arielle Dortch, Lauren Foltz, Stephen Clark Washington III,␣
 ↪Inga Fleishhacker

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

[88]:
```
# Import Libraries
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 )

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

```
Head of Black Friday Data:
```

[88]:

	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	\
0	2	0	3	
1	2	0	1	
2	2	0	12	
3	2	0	12	
4	4+	0	8	

	Product_Category_2	Product_Category_3	Purchase
0	NaN	NaN	8370
1	6.0	14.0	15200
2	NaN	NaN	1422
3	14.0	NaN	1057
4	NaN	NaN	7969

[89]:
```
# Import more Libraries
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# For plot parameters
%matplotlib inline
from pylab import rcParams
```

0.2 Look at Black Friday Data

[90]:
```
# Look at shape of data:
print("Shape of Black Friday data is: ")
data.shape
```

```
Shape of Black Friday data is:
```

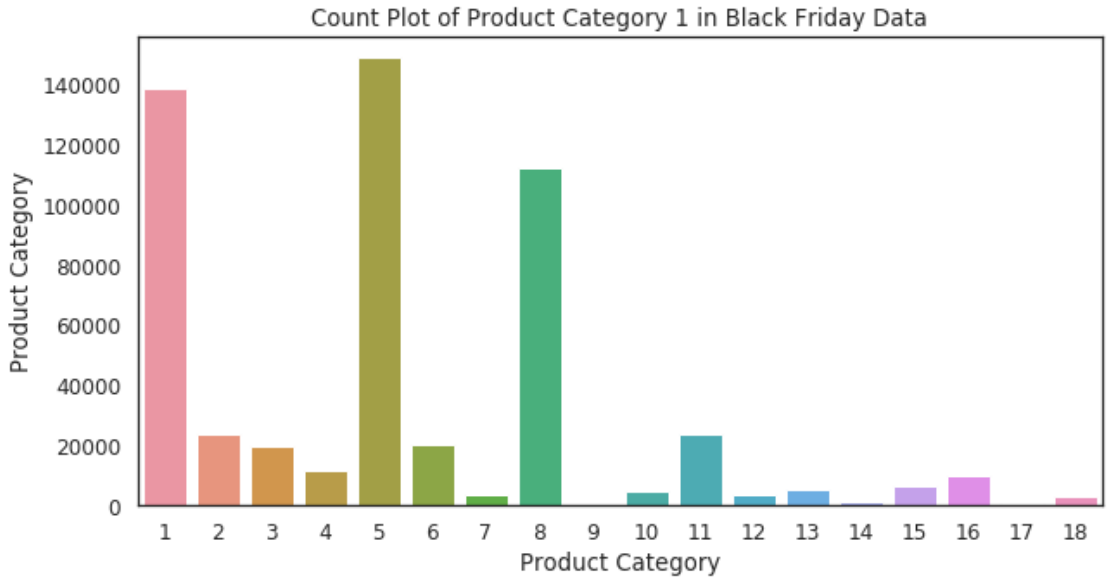
[90]:
```
(537577, 12)
```

[91]:
```
# Look at Data Types
print('Black Friday Data Types:')
print()
print(data.dtypes)
```

```
Black Friday Data Types:

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
Product_Category_2          float64
Product_Category_3          float64
Purchase                      int64
dtype: object
```

[92]:
```
# Look at count of product category 1
rcParams['figure.figsize'] = 10, 5 # set parameters
sns.set(font_scale=1.1) # set font size
sns.set_style("white") # set background to white
plt.title('Count Plot of Product Category 1 in Black Friday Data')
sns.countplot(x='Product_Category_1', data=data)
plt.xlabel('Product Category')
plt.ylabel('Product Category')
```

[92]:
```
Text(0,0.5,'Product Category')
```



[93]:
```
# Get counts
print("Counts for Product Category 1 in Black Friday Data:")
data['Product_Category_1'].value_counts()
```

```
Counts for Product Category 1 in Black Friday Data:
```

[93]:
```
5     148592
1     138353
8     112132
11     23960
2      23499
6      20164
3      19849
4      11567
16      9697
15      6203
13      5440
10      5032
12      3875
7       3668
18      3075
14      1500
17       567
9        404
Name: Product_Category_1, dtype: int64
```

0.3 Create Train and Test set (Label = Purchase)

[94]:
```
# Store data as "train"
train = data
```

[95]:
```
# Imported these earlier

#import pandas as pd
#import numpy as np
#import seaborn as sns
#import matplotlib.pyplot as plt
#import warnings
#warnings.filterwarnings("ignore")
```

[96]:
```
# Drop product categories 2 and 3 from "train"
train.drop(['Product_Category_2', 'Product_Category_3'], inplace=True, axis=1)
```

[97]:
```
ids_train = train['User_ID'].copy()
product_ids_train = train['Product_ID'].copy()
```

[98]:
```
cutoff_purchase = np.percentile(train['Purchase'], 99.9)  # 99.9 percentile
train.ix[train['Purchase'] > cutoff_purchase, 'Purchase'] = cutoff_purchase
```

[99]:
```
from sklearn.preprocessing import LabelEncoder

# Label Encoding User_IDs
le = LabelEncoder()
train['User_ID'] = le.fit_transform(train['User_ID'])

# Label Encoding Product_IDs
train['Product_ID'] = le.fit_transform(train['Product_ID'])
```

[100]:
```
train = pd.get_dummies(train, drop_first=True)

X = train.drop('Purchase', axis=1) # drop label from data
y = train['Purchase']
```

[101]:
```
from  sklearn.preprocessing  import StandardScaler

slc= StandardScaler()
X = slc.fit_transform(X)

# Spliting data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 0)
```

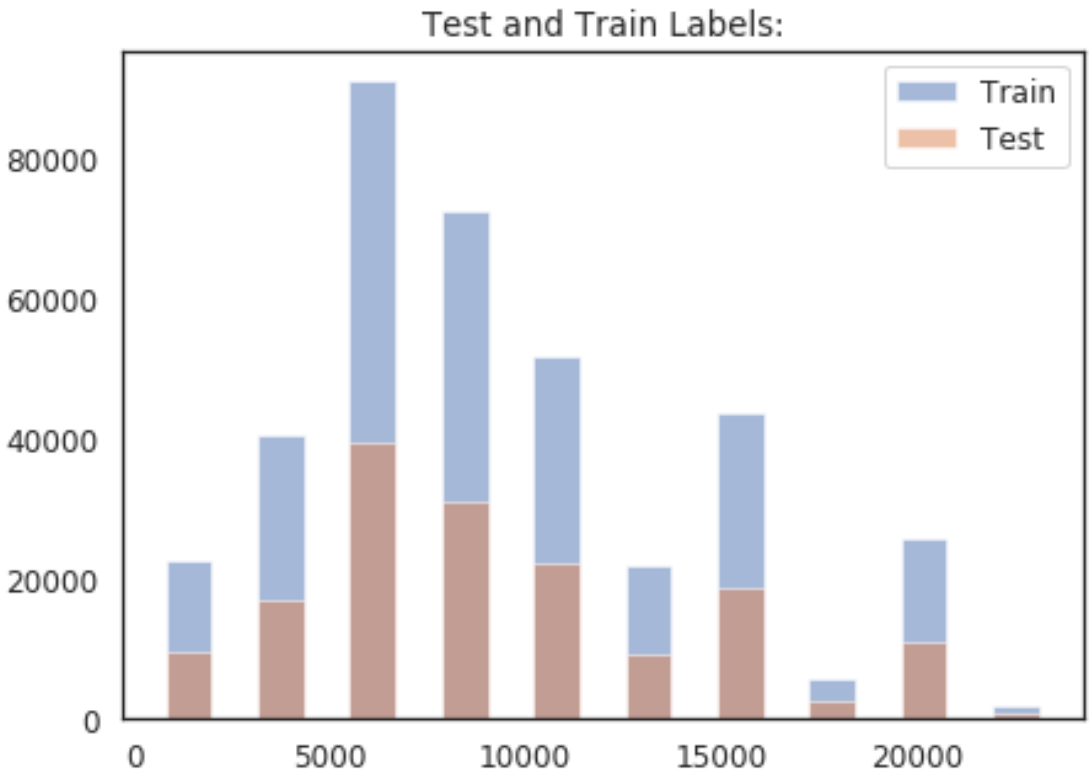
0.4 Look at at Train and Test Sets (Label = Purchase)

[102]:
```
# Print Shape of Train and Test Data
print("Shape of Train and Test Data:")
print("Shape of X_train is: ",X_train.shape)
print("Shape of y_train is: ",y_train.shape)
print("Shape of X_test is: ", X_test.shape)
print("Shape of y_test is: ", y_test.shape)
```

```
Shape of Train and Test Data:
Shape of X_train is:  (376303, 18)
Shape of y_train is:  (376303,)
Shape of X_test is:  (161274, 18)
Shape of y_test is:  (161274,)
```

[103]:
```
# Check to make sure Train and Test are balanced
rcParams['figure.figsize'] = 7, 5 # set parameters
plt.hist(y_train, rwidth=0.5, alpha=0.5, label = "Train")
plt.hist(y_test, rwidth=0.5, alpha=0.5, label = "Test")
plt.legend(loc="upper right")
plt.title("Test and Train Labels:")
```

[103]: Text(0.5,1,'Test and Train Labels:')



[104]:
```
# Look at head of Train Labels (Purchase)
print("Head of Train Labels:")
print()
y_train.head()
```

```
Head of Train Labels:
```

[104]:
```
371018    11872.0
117926    11639.0
517435     4273.0
361254    20232.0
17415      6858.0
Name: Purchase, dtype: float64
```

[105]:
```
# Look at head of Test Labels (Purchase)
print("Head of Test Labels:")
print()
y_test.head()
```

```
Head of Test Labels:
```

[105]:
```
225194    11394.0
297411    12781.0
394066    20068.0
504777    15359.0
347720    12665.0
Name: Purchase, dtype: float64
```

0.5 Association Rules

[108]:
```
lst=[]
for item in train['User_ID'].unique():
    lst2=list(set(train[train['User_ID']==item]['Product_ID']))
    if len(lst2)>0:
        lst.append(lst2)
```

[109]:
```
# Transform data for Association Rules
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

te=TransactionEncoder()
te_data=te.fit(lst).transform(lst)
df_x=pd.DataFrame(te_data,columns=te.columns_)
```

[110]:
```
# Get Frequent Items
# (this takes a while)
frequent_items=apriori(df_x,use_colnames=True,min_support=0.03)
```

[111]:
```
# Look at head of Frequent Items
print('Head of Frequent Items:')
frequent_items.head()
```

```
Head of Frequent Items:
```

[111]:

	support	itemsets
0	0.191818	(0)
1	0.062977	(1)
2	0.040401	(2)
3	0.086912	(5)
4	0.040401	(6)

[112]:
```
# Create rules
rules=association_rules(frequent_items,metric='lift',min_threshold=1)
rules.antecedents=rules.antecedents.apply(lambda x: next(iter(x)))
rules.consequents=rules.consequents.apply(lambda x: next(iter(x)))
rules=rules.sort_values('lift',ascending=False)
```

[113]:
```
# Look at Rules Head
print('Head of Association Rules:')
rules.head()
```

```
Head of Association Rules:
```

[113]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
24571	974	689	0.081141	0.062977	0.030555	0.376569	5.979429	0.025445	1.503009
24570	689	974	0.062977	0.081141	0.030555	0.485175	5.979429	0.025445	1.784800
24969	2091	314	0.072653	0.092344	0.033950	0.467290	5.060301	0.027241	1.703845
24968	314	2091	0.092344	0.072653	0.033950	0.367647	5.060301	0.027241	1.466502
20862	314	1180	0.092344	0.089798	0.041928	0.454044	5.056283	0.033636	1.667171

[114]:
```
# New Package: Plotly
# If needed, run below in anaconda prompt OR cmd under directory␣
 ↪Anaconda3\Scripts
# conda install -c https://conda.anaconda.org/plotly plotly
```

[115]:
```
# Visualize Rules

import plotly.plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot

init_notebook_mode(connected=True)

import networkx as nx

nx_data=rules[rules.lift>=3]
GA=nx.
 ↪from_pandas_edgelist(nx_data,source='antecedents',target='consequents',edge_attr='lift')
pos=nx.kamada_kawai_layout(GA,weight='lift')
# pos = nx.nx_agraph.graphviz_layout(GA)
# pos = nx.nx_agraph.graphviz_layout(GA, prog='dot')
edge_trace = go.Scatter(
    x=[],
    y=[],
    line=dict(width=0.5,color='#888'),
    hoverinfo='none',
    mode='lines')

for edge in GA.edges():
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    edge_trace['x'] += tuple([x0, x1, None])
    edge_trace['y'] += tuple([y0, y1, None])

node_trace = go.Scatter(
    x=[],
    y=[],
    text=[],
    mode='markers',
    hoverinfo='text',
    marker=dict(
        showscale=True,
        colorscale='YlGnBu',
        reversescale=True,
 color=[],
        size=10,
        colorbar=dict(
            thickness=15,
            title='Node Connections',
            xanchor='left',
            titleside='right'
        ),
```

```
        line=dict(width=2)))

for node in GA.nodes():
    x, y = pos[node]
    node_trace['x'] += tuple([x])
    node_trace['y'] += tuple([y])

for node,adjacencies in enumerate(GA.adjacency()):
    node_trace['marker']['color']+=tuple([len(adjacencies[1])])
    node_info = str(adjacencies[0])+' - # of connections:␣
 ↪'+str(len(adjacencies[1]))
    node_trace['text']+=tuple([node_info])

fig = go.Figure(data=[edge_trace, node_trace],
             layout=go.Layout(
                title='<br>Network graph',
                titlefont=dict(size=16),
                showlegend=False,
                hovermode='closest',
                   margin=dict(b=20,l=5,r=5,t=40),
                annotations=[ dict(
                    text="Python code: <a href='https://plot.ly/
 ↪ipython-notebooks/network-graphs/'> https://plot.ly/ipython-notebooks/
 ↪network-graphs/</a>",
                    showarrow=False,
                    xref="paper", yref="paper",
                    x=0.005, y=-0.002 ) ],
                xaxis=dict(showgrid=False, zeroline=False,␣
 ↪showticklabels=False),
                yaxis=dict(showgrid=False, zeroline=False,␣
 ↪showticklabels=False)))

iplot(fig, filename='networkx')
```

0.6 Create New Train and Test set (Label = Product Category 1)

[116]:
```
# Start with fresh data set. Store as "BF"
BF = data
```

[117]:
```
# Drop Stay in Current City Years
BF.drop(['Stay_In_Current_City_Years'], axis=1, inplace=True)
```

[118]:
```
# Get dummies for Gender, Age, and City Category
BF = pd.get_dummies(BF, columns=['Gender', 'Age', 'City_Category'],␣
 ↪drop_first=True)
```

[119]:
```
# Prep data for splitting
X2 = BF.drop('Product_Category_1', axis=1)
y2 = BF['Product_Category_1']

# Spliting data
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size = 0.3,␣
↪random_state = 0)
```

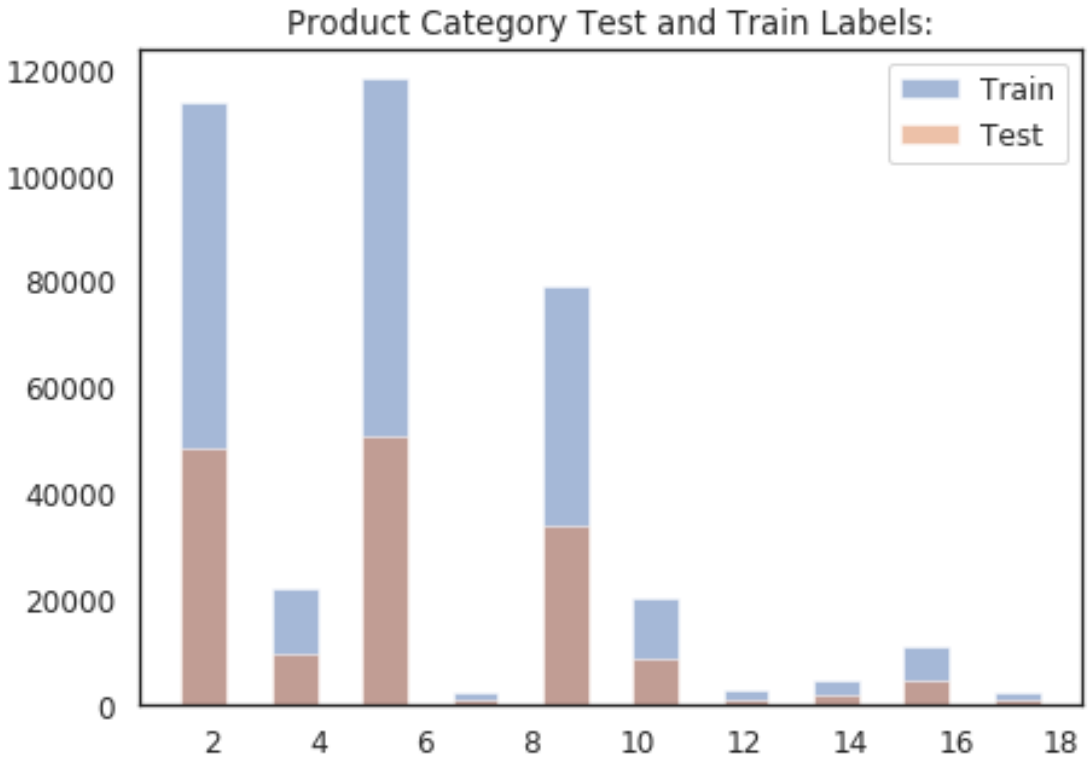
0.7 Look at Train and Test Sets (Product Category)

[122]:
```
# Print Shape of Prod Cat Train and Test Data
print("Shape of Prod Cat Train and Test Data:")
print("Shape of X_train is: ",X_train.shape)
print("Shape of y_train is: ",y_train.shape)
print("Shape of X_test is: ", X_test.shape)
print("Shape of y_test is: ", y_test.shape)
```

```
Shape of Prod Cat Train and Test Data:
Shape of X_train is:  (376303, 14)
Shape of y_train is:  (376303,)
Shape of X_test is:  (161274, 14)
Shape of y_test is:  (161274,)
```

[123]:
```
#Check to make sure Prod Cat Train and Test are balanced
rcParams['figure.figsize'] = 7, 5 # set parameters
plt.hist(y_train, rwidth=0.5, alpha=0.5, label = "Train")
plt.hist(y_test, rwidth=0.5, alpha=0.5, label = "Test")
plt.legend(loc="upper right")
plt.title("Product Category Test and Train Labels:")
```

[123]:
```
Text(0.5,1,'Product Category Test and Train Labels:')
```



[124]:
```
# Look at head of Product Category Train Labels (Product Category 1)
print("Head of Product Category Train Labels:")
print()
y_train.head()
```

```
Head of Product Category Train Labels:
```

[124]:
```
371018    1
117926    1
517435    1
361254    6
17415     5
Name: Product_Category_1, dtype: int64
```

[125]:
```
# Look at head of Product Category Test Labels (Product Category 1)
print("Head of Product Category Test Labels:")
print()
y_test.head()
```

```
Head of Product Category Test Labels:
```

[125]:
```
225194     1
297411     7
394066     6
504777     1
347720    15
Name: Product_Category_1, dtype: int64
```

[126]:
```
# Get counts
print("Counts for Product Category 1 in Train Set: ")
y_train.value_counts()
```

```
Counts for Product Category 1 in Train Set:
```

[126]:
```
5     103782
1      97194
8      78514
11     16766
2      16452
6      14152
3      13835
4       8080
16      6735
15      4370
13      3809
10      3439
12      2726
7       2535
18      2153
14      1081
17       391
9        289
Name: Product_Category_1, dtype: int64
```

[127]:
```
# Get counts
print("Counts for Product Category 1 in Test Set: ")
y_test.value_counts()
```

```
Counts for Product Category 1 in Test Set:
```

[127]:
```
5     44810
1     41159
8     33618
11     7194
2      7047
3      6014
6      6012
4      3487
```

```
16     2962
15     1833
13     1631
10     1593
12     1149
7      1133
18      922
14      419
17      176
9       115
Name: Product_Category_1, dtype: int64
```

0.8 Random Forest

[128]:
```
# Import Libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

[129]:
```
# Create Model
rf_model = RandomForestClassifier(n_estimators=64, n_jobs=-1)
```

[130]:
```
# Train Model
rf_model.fit(X_train, y_train.values.ravel())
```

[130]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=64, n_jobs=-1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

[131]:
```
# Make Predictions
y_pred = rf_model.predict(X_test)
```

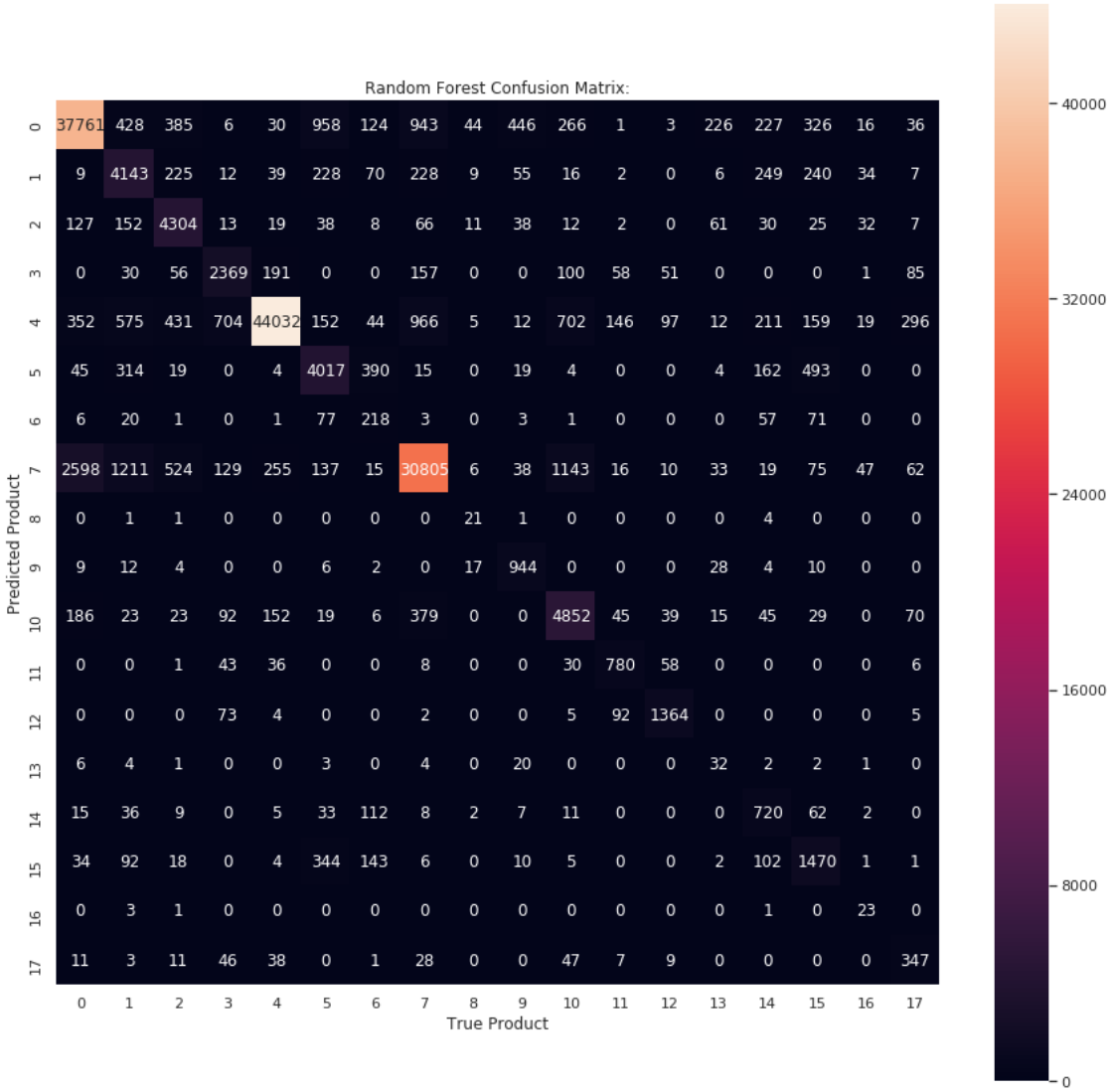
[132]:
```
# Print Accuracy Score
print('Random Forest Accuracy Score:')
print(accuracy_score(y_test, y_pred))
```

```
Random Forest Accuracy Score:
0.8569391222391706
```

[133]:
```
# Visualize Confusion Matrix

sns.set(font_scale=1) # set font size
sns.set_style("white") # set background to white
```

```
rcParams['figure.figsize'] = 15, 15 # set parameters

from sklearn.metrics import confusion_matrix
mat = confusion_matrix(y_test, y_pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=True) # alternate␣
 ↪color scheme
#sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,␣
 ↪cmap="Purples") # alternate color scheme
plt.title("Random Forest Confusion Matrix:")
plt.xlabel('True Product')
plt.ylabel('Predicted Product')
```

[133]: Text(111.5,0.5,'Predicted Product')



[134]:
```
# Find out which features are most important

#https://towardsdatascience.com/
 ↪running-random-forests-inspect-the-feature-importances-with-this-code-2b00dd72b92e
import pandas as pd
rf_feature_importances = pd.DataFrame(rf_model.feature_importances_,
                                      index = X_train.columns,
                                       columns=['importance']).
 ↪sort_values('importance',ascending=False)
```

[135]:
```
# Look at feature importance list
print('Random Forest Feature Importance:')
rf_feature_importances
```

```
Random Forest Feature Importance:
```

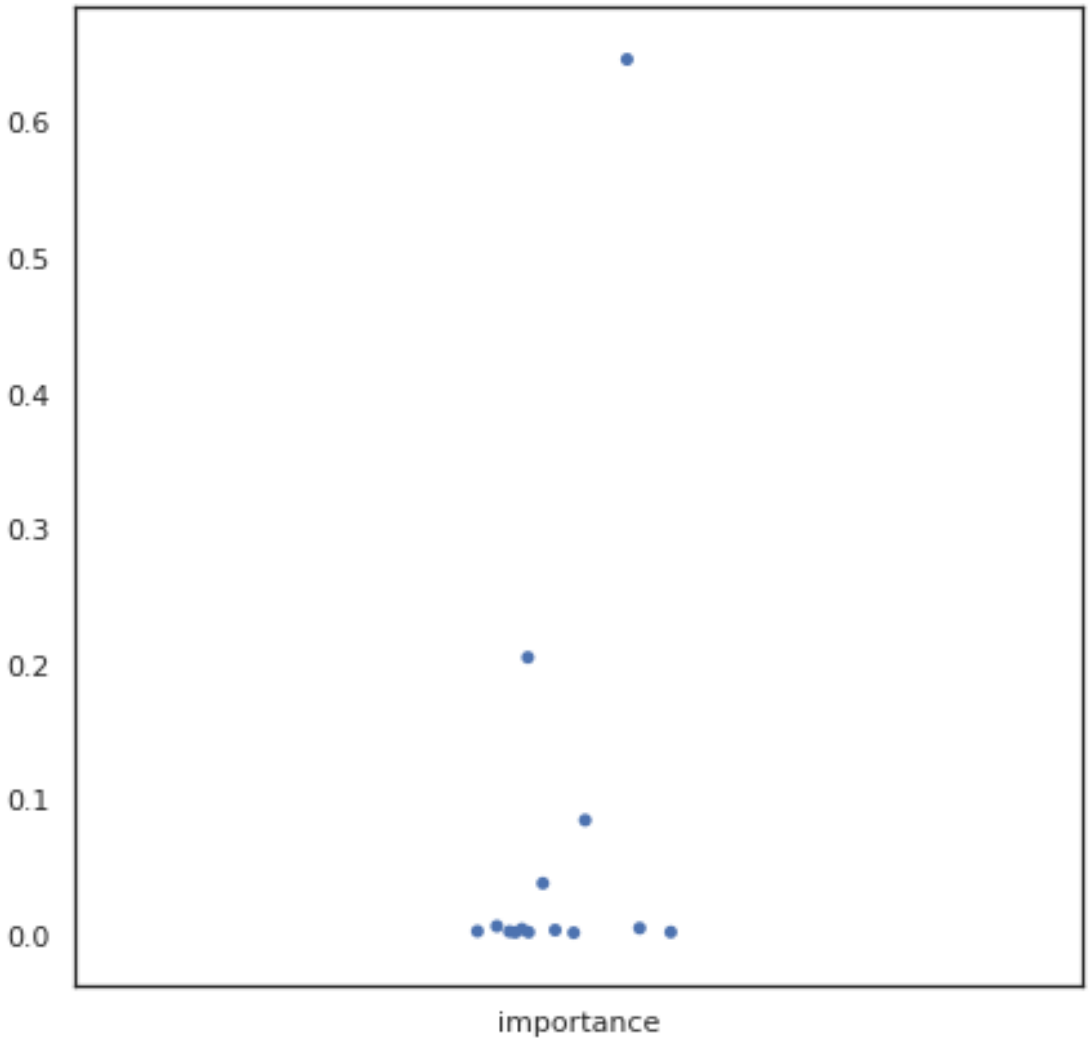
[135]:

	importance
Purchase	0.646761
Product_ID	0.204630
User_ID	0.084306
Occupation	0.037618
Marital_Status	0.005907
City_Category_B	0.004476
Gender_M	0.003754
City_Category_C	0.003013
Age_36-45	0.002194
Age_26-35	0.001961
Age_18-25	0.001544
Age_46-50	0.001457
Age_51-55	0.001329
Age_55+	0.001050

[136]:
```
# Plot Feature Importances
rcParams['figure.figsize'] = 7,7 # set parameters
print("Random Forest Feature Importance:")
sns.stripplot(data=rf_feature_importances)
```

```
Random Forest Feature Importance:
```

[136]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2ca311c470>
```



0.9 Naive Bayes

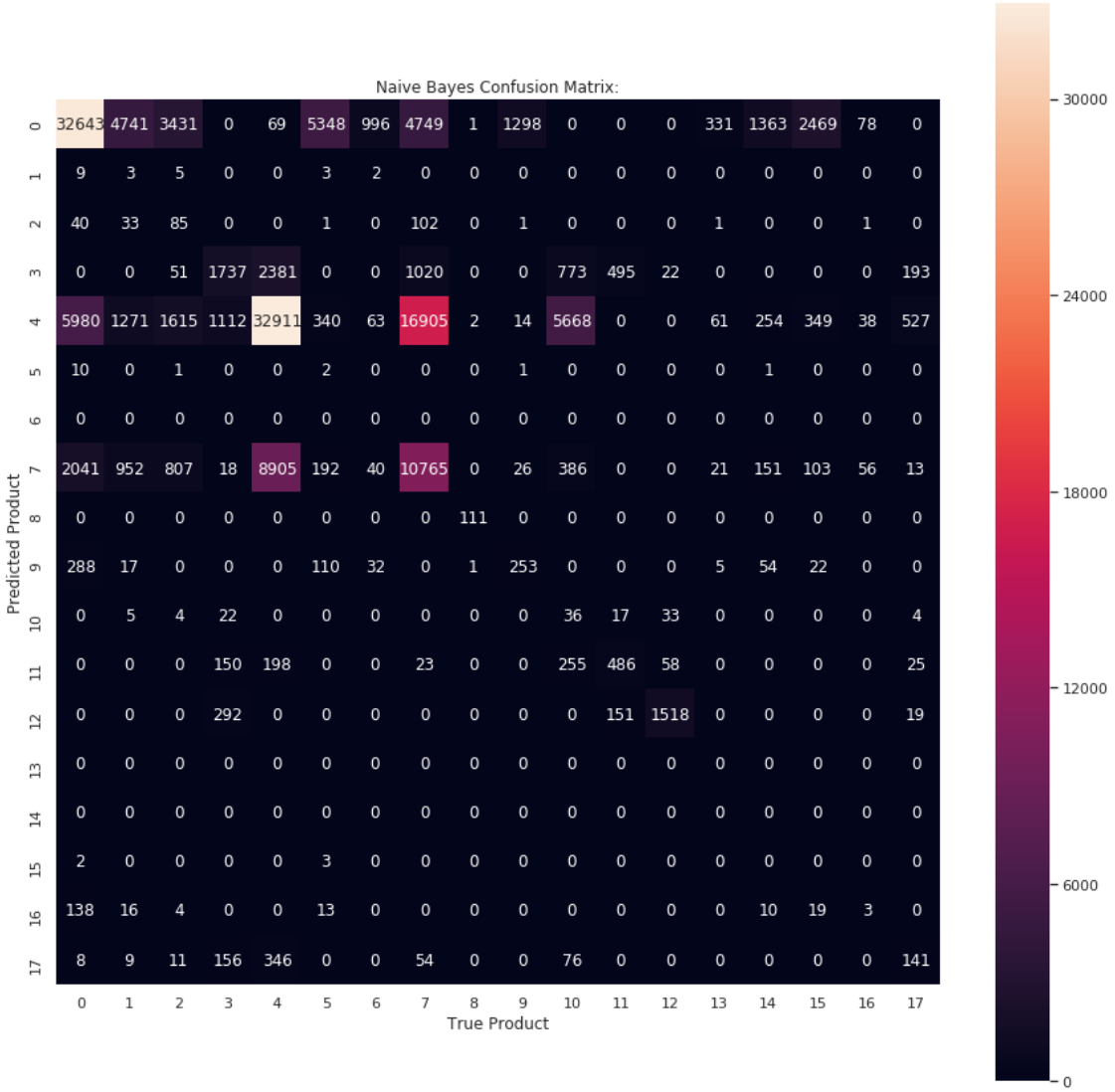
[137]:
```
# Naive Bayes using Product Category test/train set

from sklearn.naive_bayes import GaussianNB
#Create a Gaussian Classifier
nb_model = GaussianNB()

# Train Model
nb_model.fit(X_train, y_train)

# Make Predictions
y_pred2 = nb_model.predict(X_test)
```

[138]:
```
# Print Accuracy Score
print("Accuracy Score for Naive Bayes:")
print(accuracy_score(y_test, y_pred2))
```

```
Accuracy Score for Naive Bayes:
0.5003534357676996
```

[139]:
```
# Confusion Matrix

sns.set(font_scale=1) # set font size
sns.set_style("white") # set background to white
rcParams['figure.figsize'] = 15, 15 # set parameters

mat2 = confusion_matrix(y_test, y_pred2)
sns.heatmap(mat2.T, square=True, annot=True, fmt='d', cbar=True)
#sns.heatmap(mat2.T, square=True, annot=True, fmt='d', cbar=False,␣
 ↪cmap="Purples") #an alternate color scheme
plt.title("Naive Bayes Confusion Matrix:")
plt.xlabel('True Product')
plt.ylabel('Predicted Product')
```

[139]:
```
Text(111.5,0.5,'Predicted Product')
```



0.10 Linear Regression

[140]:
```
# Linear Regression using Product Category test/train set
from sklearn import linear_model
lm = linear_model.LinearRegression() # Create the model
lr_model = lm.fit(X_train, y_train) # Train the model
predictions = lm.predict(X_test) # Make predictions
```

[141]:
```
# Print Accuracy Score
print('Linear Regression Accuracy:')
lr_model.score(X_test,y_test)
```

```
Linear Regression Accuracy:
```

[141]: 0.10072609370736196

0.11 Decision Tree Regressor

[142]:
```
# Decision Tree Regressor using Product Category test/train set
from sklearn.tree import DecisionTreeRegressor
reg = DecisionTreeRegressor()
reg.fit(X_train,y_train)
print('Decision Tree Regressor Accuracy:')
r2_score = reg.score(X_test,y_test)
r2_score
```

Decision Tree Regressor Accuracy:

[142]: 0.9058314377590717

[143]:
```
# Find out which features are most important
reg_feature_importances = pd.DataFrame(reg.feature_importances_,
                                       index = X_train.columns,
                                        columns=['importance']).
 ↪sort_values('importance',ascending=False)
```

[144]:
```
# Look at feature importance list
print('Decision Tree Regressor Feature Importance:')
reg_feature_importances
```

Decision Tree Regressor Feature Importance:

[144]:

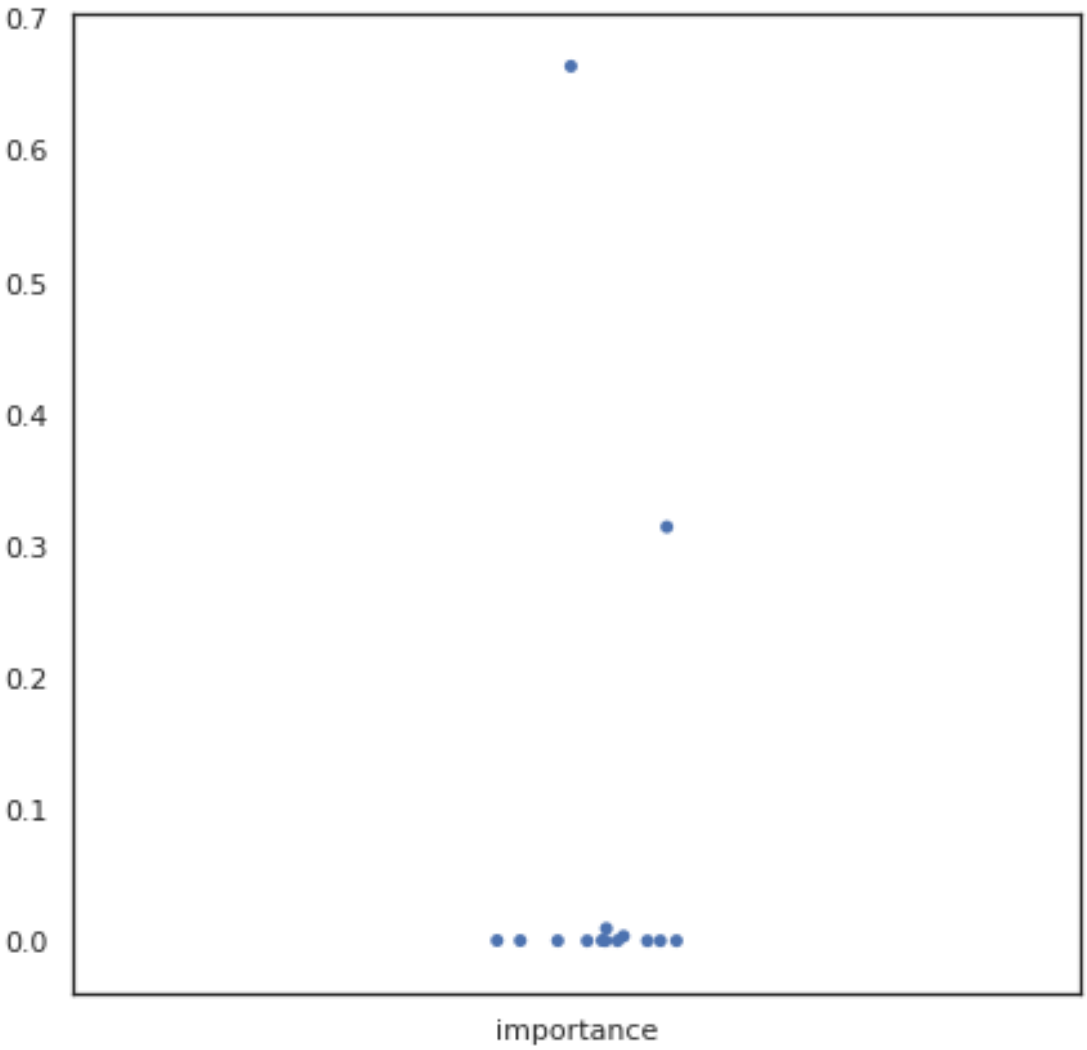
	importance
Purchase	0.663475
Product_ID	0.314375
User_ID	0.010020
Occupation	0.004207
Gender_M	0.000944
Marital_Status	0.000938
City_Category_C	0.000887
Age_26-35	0.000780
City_Category_B	0.000771
Age_18-25	0.000754
Age_51-55	0.000737
Age_36-45	0.000725
Age_55+	0.000715
Age_46-50	0.000674

[145]:
```
# Plot Feature Importance
rcParams['figure.figsize'] = 7,7 # set parameters
print("Decision Tree Regressor Feature Importance:")
```

```
sns.stripplot(data=reg_feature_importances)
```

```
Decision Tree Regressor Feature Importance:
```

[145]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2cb1f6d6d8>



0.12 K Neighbors Regressor

[146]:
```
# K Neighbors Regressor using Product Category test/train set

#from sklearn.model_selection import train_test_split
#train,test = train_test_split(data)
#X_train , y_train = train.iloc[:,train.columns!='Product_Category_1'],␣
 ↪train[['Product_Category_1']]
```

```
#X_test , y_test = test.iloc[:,test.columns!='Product_Category_1'],␣
 ↪test[['Product_Category_1']]

from sklearn.neighbors import KNeighborsRegressor
clf = KNeighborsRegressor(n_neighbors=18) # select n
clf.fit(X_train,y_train)
print('K Neighbors Regressor Score:')
r2_score = clf.score(X_test,y_test)
r2_score

# Accuracy 62.76% with n=12
# Accuracy 62.06% with n=18
```

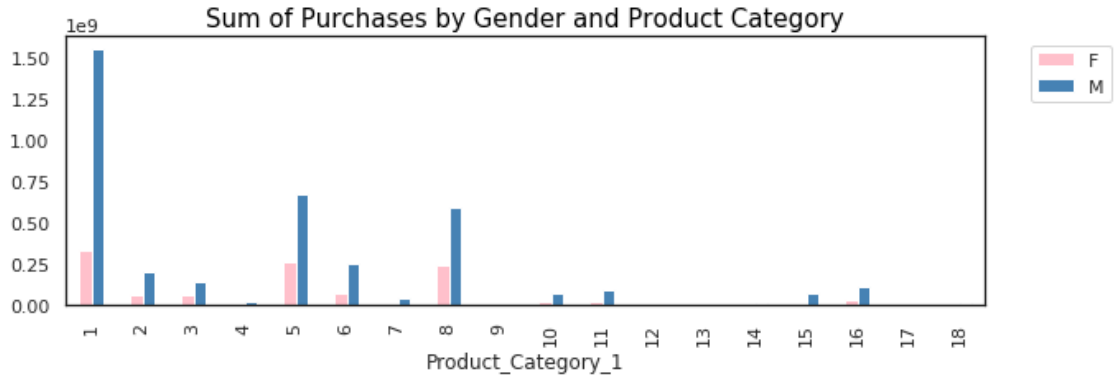
```
K Neighbors Regressor Score:
```

[146]:
```
0.6206212507979256
```

0.13 More Visuals

[147]:
```
# Sum of Purchases by Gender and Product Category
rcParams['figure.figsize'] = 10,3 # set parameters

var1 = data.groupby(['Product_Category_1', 'Gender']).Purchase.sum()
var1.unstack().plot(kind='bar',stacked=False, color=['pink','steelblue'],␣
 ↪grid=False)
plt.title('Sum of Purchases by Gender and Product Category', fontdict={'size' :␣
 ↪15, 'color':'black'})
plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")
```

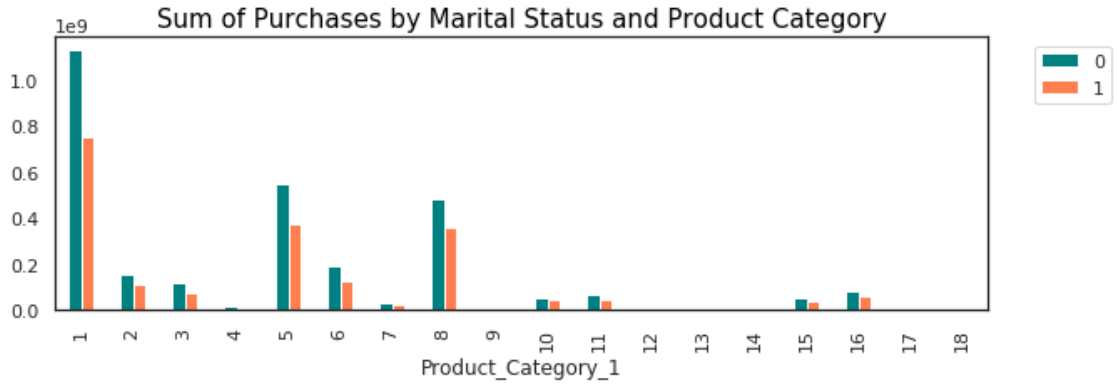
[147]:
```
<matplotlib.legend.Legend at 0x7f2cb1ec5ef0>
```



[148]:
```
# Sum of Purchases by Marital Status and Product Category
rcParams['figure.figsize'] = 10,3 # set parameters
```

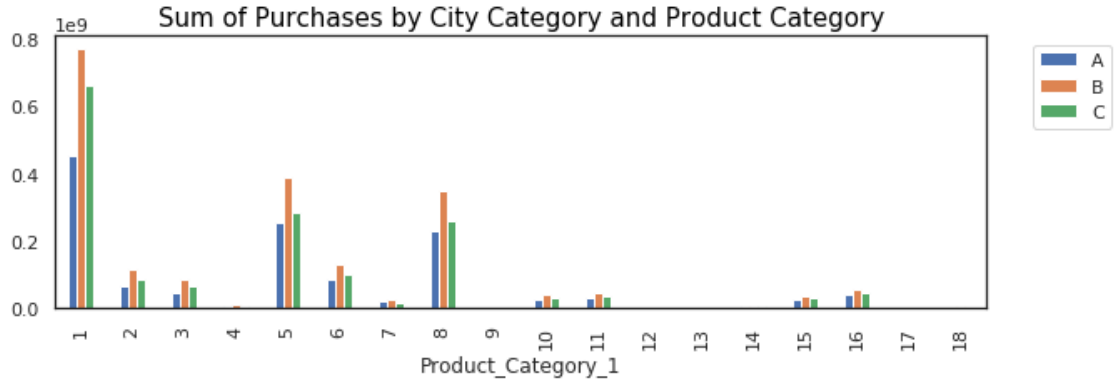
```
var2 = data.groupby(['Product_Category_1', 'Marital_Status']).Purchase.sum()
var2.unstack().plot(kind='bar',stacked=False, color=['teal','coral'],␣
 ↪grid=False)
plt.title('Sum of Purchases by Marital Status and Product Category',␣
 ↪fontdict={'size' : 15, 'color':'black'})
plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")
```

[148]: <matplotlib.legend.Legend at 0x7f2ca34aa588>



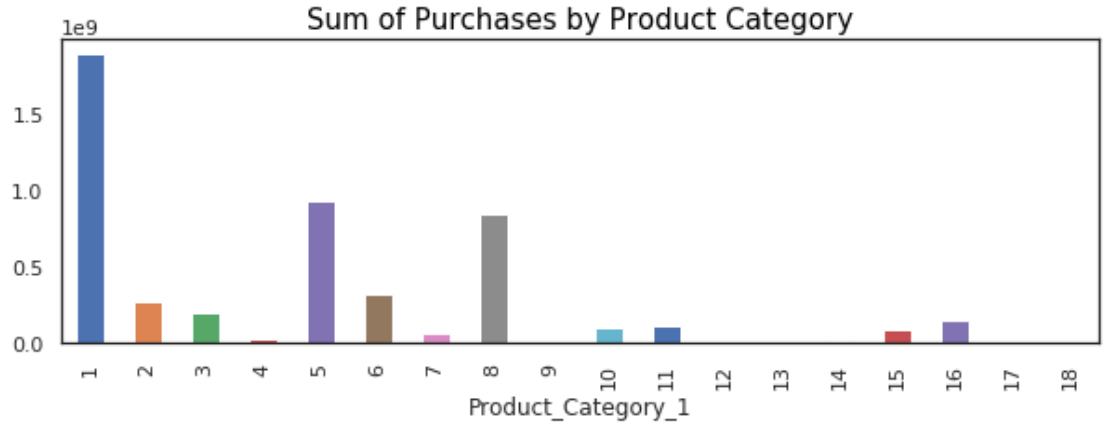
[149]:
```
#Sum of Purchases by City Category and Product Category
rcParams['figure.figsize'] = 10,3 # set parameters

var3 = data.groupby(['Product_Category_1', 'City_Category']).Purchase.sum()
var3.unstack().plot(kind='bar',stacked=False, grid=False)
plt.title('Sum of Purchases by City Category and Product Category',␣
 ↪fontdict={'size' : 15, 'color':'black'})
plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")
```

[149]: <matplotlib.legend.Legend at 0x7f2ca3460c18>



[150]:
```
#Sum of Purchases by Product Category
var4 = data.groupby('Product_Category_1').Purchase.sum()
plt.title('Sum of Purchases by Product Category', fontdict={'size' : 15,␣
 ↪'color':'black'})
var4.plot(kind='bar')
```

[150]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2ca343be80>



[ ]:
```
# End of Black Friday Notebook 3
```