Preprocessing the data:

Project by Anmol Sharma

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
from itertools import combinations
from collections import Counter
```

Merging different files:

In [2]:
```
df = pd.read_csv('c:/Users/Lenovo/PycharmProjects/Giraffe/Pandas-Data-Science-Tasks-master
                 '/Sales_Data/Sales_April_2019.csv')
files = [file for file in os.listdir('c:/Users/Lenovo/PycharmProjects/Giraffe/Pandas-Data-
```

Creating an empty dataframe to store all files:

In [3]:
```
merged = pd.DataFrame()

for file in files:
    #   print(file)
    df = pd.read_csv('c:/Users/Lenovo/PycharmProjects/Giraffe/Pandas-Data-Science-Tasks-ma
    merged = pd.concat([merged, df])

# Another method to print all component names within the list:
# print(*files, sep='\n')
```

In [4]:
```
# coverting to a CSV readable file
merged.to_csv('Merged.csv', index=False)

# Read updated dataframe:
all_data = pd.read_csv('Merged.csv')
```

Cleaning the data:

In [5]:
```
# Drop NaN rows: VERY IMP CODE:
nan_df = all_data[all_data.isna().any(axis=1)]
all_data = all_data.dropna(how='all')

# Finding 'Or' and deleting it, the negation will give us data which doesn't contain 'Or'
all_data = all_data.loc[~all_data['Order Date'].str.contains('Or')]
```

Modifying the data:

In [6]:
```
# Adding month column and then converting its components to integers; only int32 does the
all_data['Month'] = all_data['Order Date'].str[0: 2]
all_data['Month'] = all_data['Month'].astype('int32')
all_data['Quantity Ordered'] = all_data['Quantity Ordered'].astype('int32')
all_data['Price Each'] = pd.to_numeric(all_data['Price Each'])
```

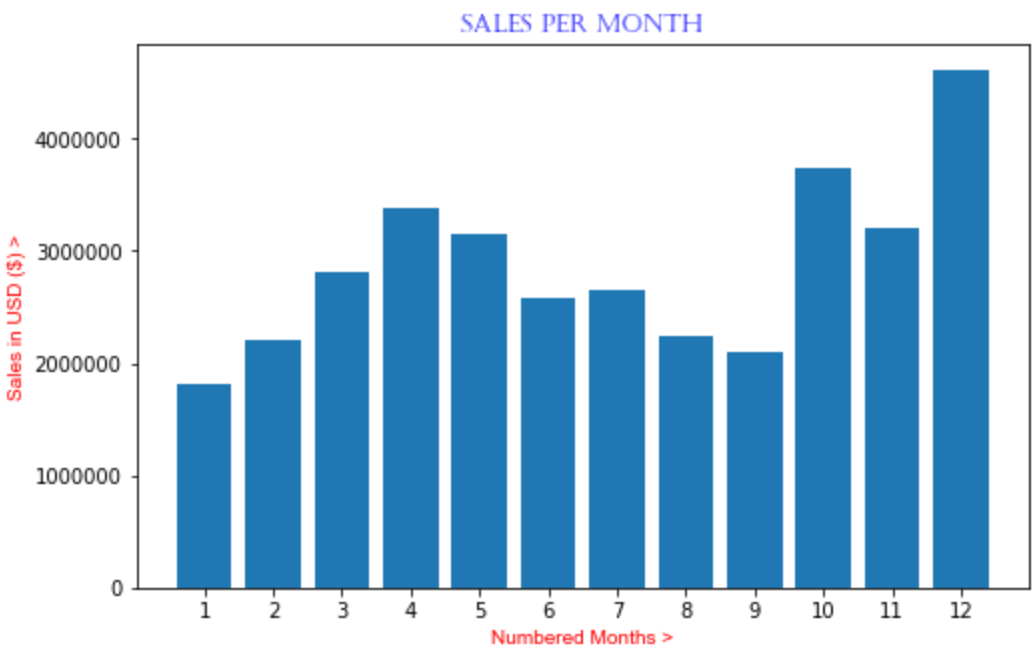
```
global orders

'''# For total no. of sales per month:
mon_count = 0

for Month in all_data['Month']:
    mon_count += 1
    if mon_count <= 12:
        mon = all_data.loc[all_data['Month'] == mon_count]
        orders = mon.sum()
        print(orders['Quantity Ordered'], mon_count,)'''

# Adding sales column:
all_data['Sales'] = all_data['Quantity Ordered'] * all_data['Price Each']
all_data['Sales'] = pd.to_numeric(all_data['Sales'])
```

Analysing the data:

Plotting Sales per month:

In [7]:

```
# Summing up data:

mon_count = 0
d = []
for Month in all_data['Month']:
    mon_count += 1
    if mon_count <= 12:
        mon = all_data.loc[all_data['Month'] == mon_count]
        orders = mon.sum()['Sales']
        d.append(orders)

# Using groupby func:
sum_up = all_data.groupby('Month').sum()
# print(sum_up)

x1 = range(1, 13)
plt.figure(figsize=(8, 5))
plt.bar(x1, sum_up['Sales'])
# In place of sum_up['Sales'], we can write d. Same results.
plt.xticks(x1)

# Disabling scientific notations like 1e6 on yaxis, for xaxis write useOFFset=False:
plt.ticklabel_format(style='plain')
plt.title('Sales Per Month', fontdict={'fontname': 'castellar', 'color': 'blue'})
plt.ylabel('Sales in USD ($) >', fontdict={'fontname': 'arial', 'color': 'red'})
plt.xlabel('Numbered Months >', fontdict={'fontname': 'arial', 'color': 'red'})
plt.show()
```

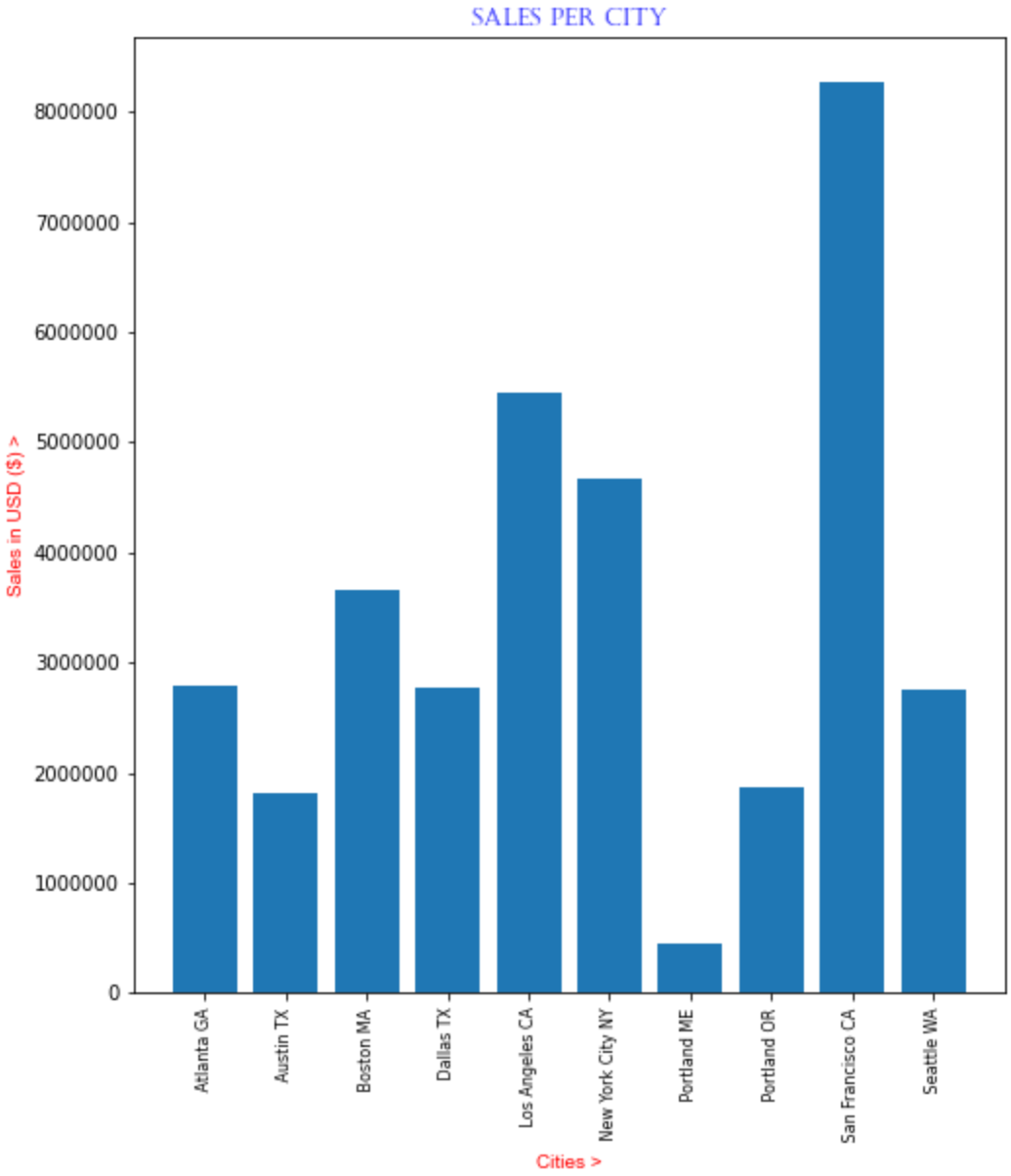


Plotting sales per city:

In [8]:

```
'''cities = ['New York City', 'Dallas', 'Portland', 'Austin', 'San Francisco', 'Los Angele
d1 = []
for city in cities:
    info = all_data.loc[all_data['Purchase Address'].str.contains(city)].sum()
    res = info['Quantity Ordered']
    d1.append(res)
print(d1)
print(cities)'''

# Pro way by creating city column; IMPORTANT- ".apply" method:
# After lambda x, we can create a func to do the task too, like we did in tkinter.

'''all_data['Cities'] = all_data['Purchase Address'].apply(lambda x: x.split(',')[1])
print(all_data['Cities'])'''


# Due to the problem of duplicate names, we gotta add state too, in Cities:

def get_city(address):
    return address.split(',')[1]


def get_state(address):
    return address.split(',')[2].split(' ')[1]


all_data['Cities'] = all_data['Purchase Address'].apply(lambda x: get_city(x) + " " + get_
# print(all_data['Cities'])
results = all_data.groupby('Cities').sum()

# Getting unique values from a column IMP - unique() function, not efficient it messes up
# city = all_data['Cities'].unique(), therefore:

city = [city for city, df in all_data.groupby('Cities')]
# print(city)


plt.figure(figsize=(8, 9))
plt.bar(city, results['Sales'])
plt.title('Sales Per City', fontdict={'fontname': 'castellar', 'color': 'blue'})
```

```
# Rotation function:
plt.xticks(city, rotation='vertical', size=8)
plt.ylabel('Sales in USD ($) >', fontdict={'fontname': 'arial', 'color': 'red'})
# axis function:
plt.ticklabel_format(style='plain', axis='y')
plt.xlabel('Cities >', fontdict={'fontname': 'arial', 'color': 'red'})
plt.show()
```

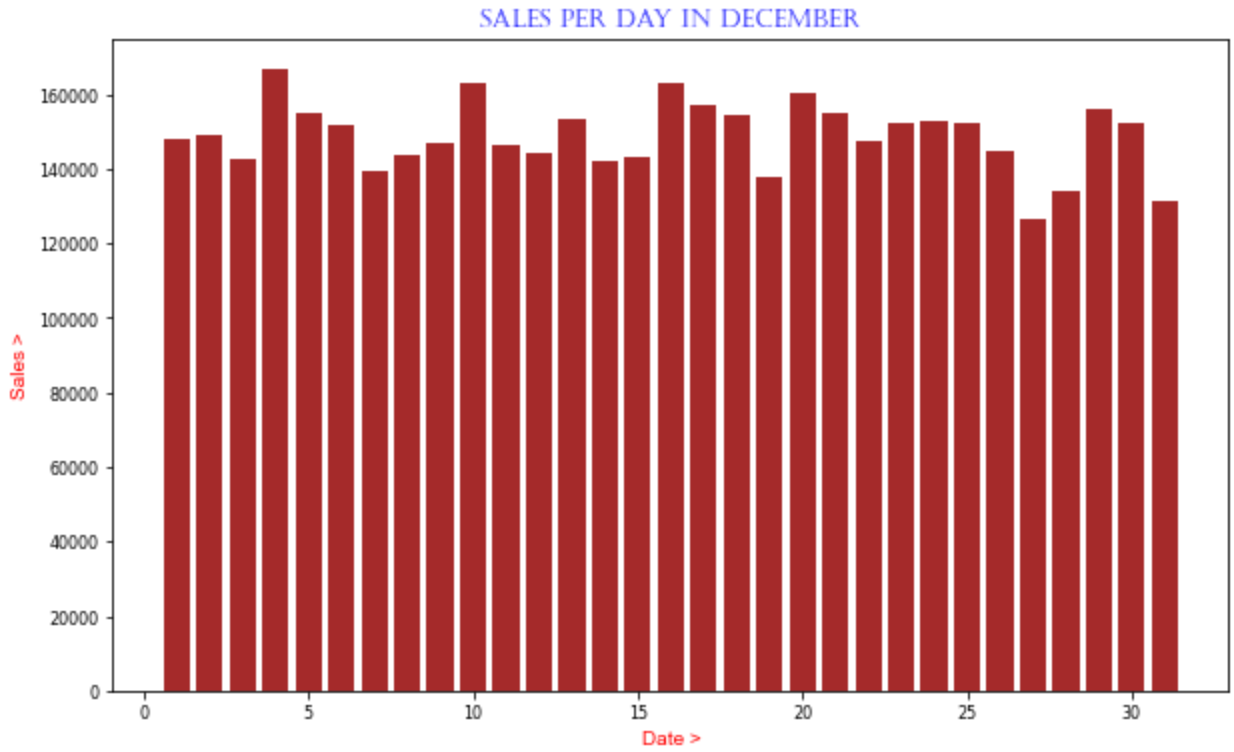


Sales in a specific month:

In [9]:

```
def get_dates(date):
    return date.split('/')[1]


all_data['Date'] = all_data['Order Date'].apply(lambda y: get_dates(y))
all_data['Date'] = pd.to_numeric(all_data['Date'])
# print(all_data['Date'])

dec = all_data.loc[all_data['Month'] == 12]
#print(dec.mean())

grp1 = dec.groupby('Date').sum()
grp2 = [x for x, df in dec.groupby('Date')]
```

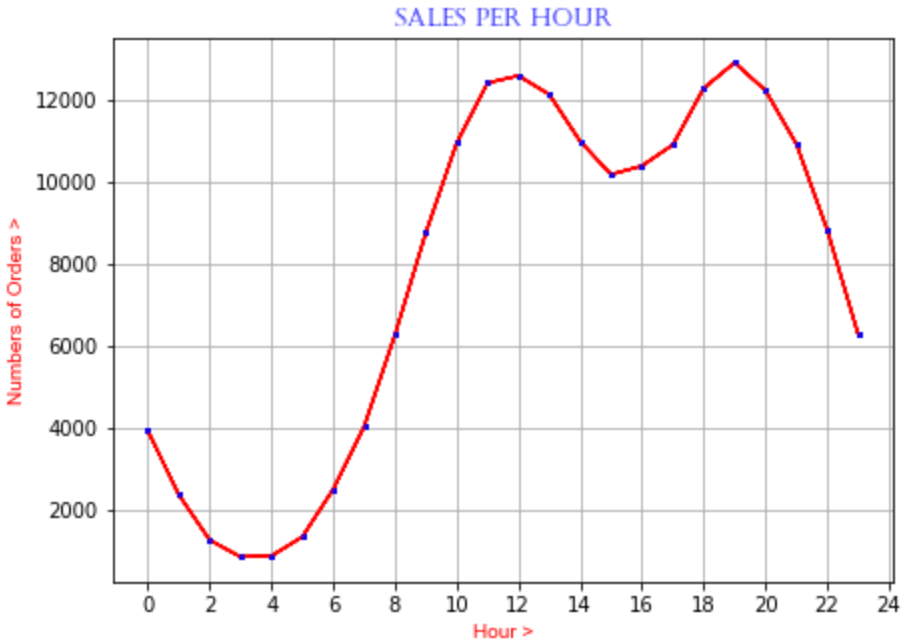
```
plt.figure(figsize=(10, 6))
plt.bar(grp2, grp1['Sales'], color='brown')
plt.title('Sales Per Day in December', fontdict={'fontname': 'castellar', 'color': 'blue'}
plt.xlabel('Date >', fontdict={'fontname': 'arial', 'color': 'red'})
plt.xticks(size=8)
plt.ylabel('Sales >', fontdict={'fontname': 'arial', 'color': 'red'})
plt.yticks(size=8)

plt.show()
```



Sales per hour:

In [10]:

```
# Pro Way for Time, to_daytime function:

all_data['Order Date'] = pd.to_datetime(all_data['Order Date'])
all_data['Hour'] = all_data['Order Date'].dt.hour

hours = [hour for hour, df in all_data.groupby('Hour')]
# print(all_data.groupby(['Hour']).count())
# By using .count we are just counting the rows:
plt.figure(figsize=(7, 5))
plt.plot(hours, all_data.groupby(['Hour']).count(), color="red", linewidth=1.1, marker="."
         markeredgecolor="blue")
plt.xticks(np.arange(0, 25, 2))
plt.title('Sales Per Hour', fontdict={'fontname': 'castellar', 'color': 'blue'})
plt.xlabel('Hour >', fontdict={'fontname': 'arial', 'color': 'red'})
plt.ylabel('Numbers of Orders >', fontdict={'fontname': 'arial', 'color': 'red'})
plt.grid()
plt.show()
```



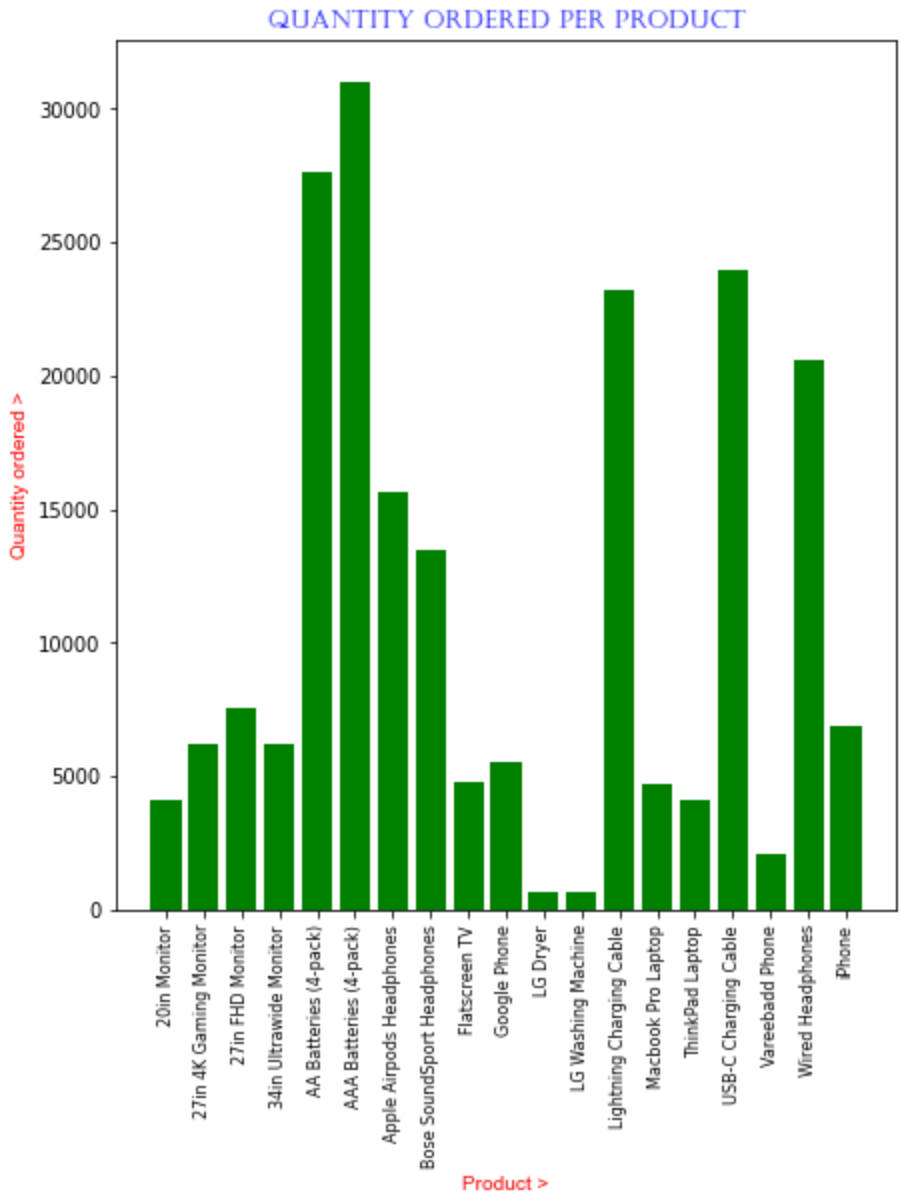
Products sold together, in one hour:

In [11]:

```
# Products sold together, in one hour:

'''d3 = []
hour_count = 0
for x in all_data['Hour']:
    hour_count += 1
    if hour_count <= 24:
        same = all_data.loc[all_data['Hour'] == hour_count]
        prod = same.sort_values(by='Hour')
        #print(same)
        print(prod)'''

# Pro way products sold together, through IDs, take duplicates and keep = False ensures no
same1 = all_data[all_data['Order ID'].duplicated(keep=False)]
same1['Grouped'] = same1.groupby('Order ID')['Product'].transform(lambda x: ','.join(x))

# Now dropping duplicate rows:
same1 = same1[['Order ID', 'Grouped']].drop_duplicates()
# print(same1)

# Counting most repeated pairs, import itertools -  Combinations, collections - counter:
# For explanations see the video at 1:00:00:
count1 = Counter()
for row in same1['Grouped']:
    row_list = row.split(',')
    count1.update(Counter(combinations(row_list, 2)))

# print(count1.most_common(10))

# The most sold product:
most = all_data.groupby(['Product']).sum()
#print(most['Quantity Ordered'])

prods = [prod for prod, df in all_data.groupby(['Product'])]

plt.figure(figsize=(7, 8))
plt.bar(prods, most['Quantity Ordered'], color='green')
plt.title('Quantity Ordered Per Product', fontdict={'fontname': 'castellar', 'color': 'blu
# Rotation function:
plt.xticks(prods, rotation='vertical', size=8)
```

```
plt.ylabel('Quantity ordered >', fontdict={'fontname': 'arial', 'color': 'red'})
# axis function:
plt.ticklabel_format(style='plain', axis='y')
plt.xlabel('Product >', fontdict={'fontname': 'arial', 'color': 'red'})
plt.show()
```

```
C:\Users\Lenovo\AppData\Local\Temp/ipykernel_12396/334426647.py:15: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  same1['Grouped'] = same1.groupby('Order ID')['Product'].transform(lambda x: ','.join(x))
```



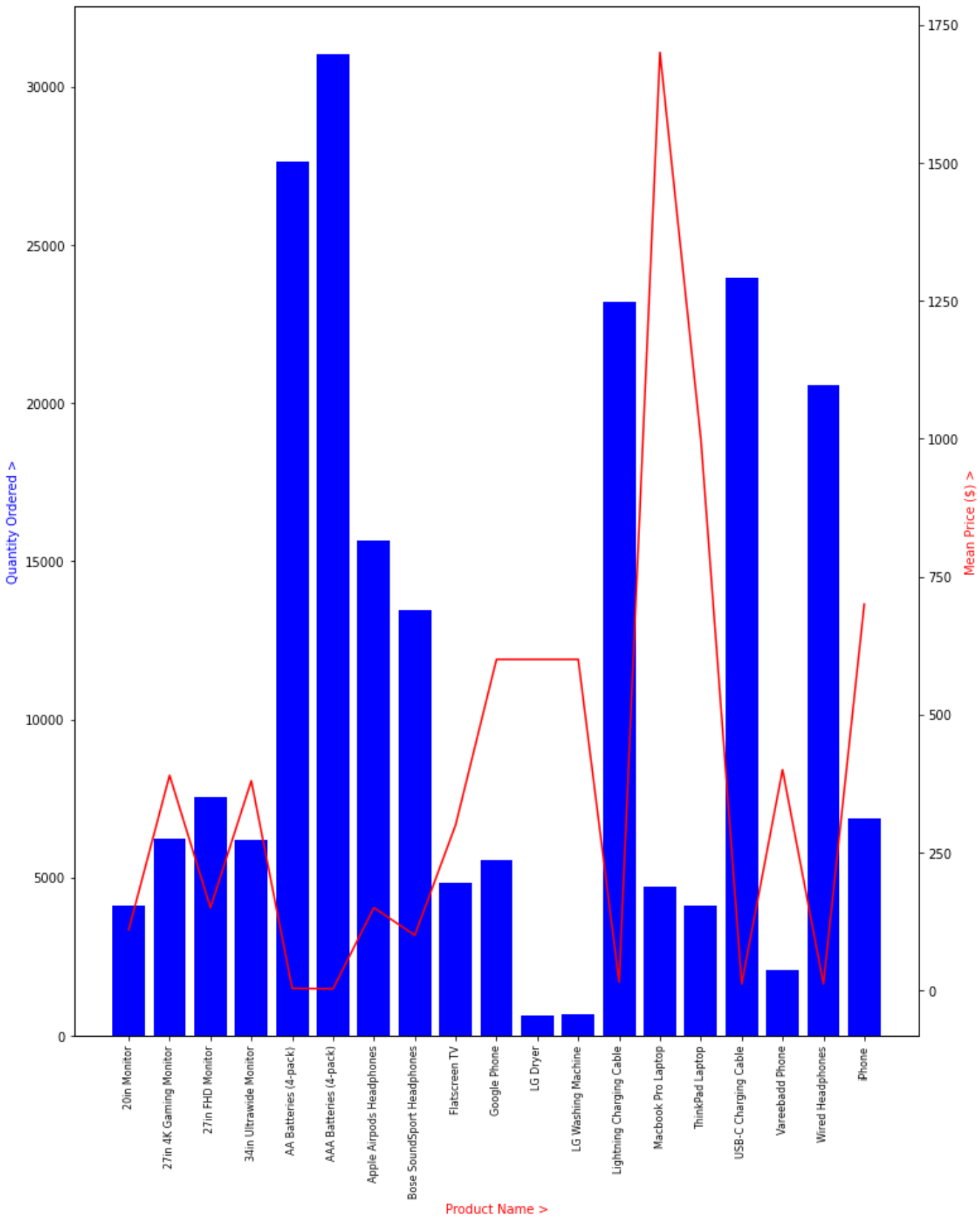
Subplot between price and quantity ordered of various products:

In [12]:

```
prices = all_data.groupby(['Product']).mean()['Price Each']
# print(prices)
fig, ax1 = plt.subplots()
fig.set_figheight(15)
fig.set_figwidth(12)
fig.suptitle('Quantity and Prices', fontsize=14, fontname='gill sans mt', color='purple')
ax2 = ax1.twinx()
```

```
ax1.bar(prods, most['Quantity Ordered'], color='blue')
ax2.plot(prods, prices, 'r-')

ax1.set_xlabel('Product Name >', color='r')
ax1.set_xticklabels(prods, rotation='vertical', size=8)
ax1.set_ylabel('Quantity Ordered >', color='blue')
ax2.set_ylabel('Mean Price ($) >', color='r')
fig.savefig('Quantity and Prices.png', dpi=1000)
plt.show()
```

```
C:\Users\Lenovo\AppData\Local\Temp/ipykernel_12396/311880613.py:12: UserWarning: FixedForm
atter should only be used together with FixedLocator
  ax1.set_xticklabels(prods, rotation='vertical', size=8)
```


Quantity and Prices
Quantity Ordered >
Mean Price ($) >
Product Name >
0
5000
10000
15000
20000
25000
30000
0
250
500
750
1000
1250
1500
1750
20in Monitor
27in 4K Gaming Monitor
27in FHD Monitor
34in Ultrawide Monitor
AA Batteries (4-pack)
AAA Batteries (4-pack)
Apple Airpods Headphones
Bose SoundSport Headphones
Flatscreen TV
Google Phone
LG Dryer
LG Washing Machine
Lightning Charging Cable
Macbook Pro Laptop
ThinkPad Laptop
USB-C Charging Cable
Vareebadd Phone
Wired Headphones
iPhone