

Product Sales Analysis Using Python

TEAM MEMBER

621521104098 : Praveenkumar.A

Phase_1: submission document

Project Title: **Product Sales Analysis Using Python**

Phase 1: Problem Definition and Design Thinking

In this part you will need to understand the problem statement and create a document on what have you understood and how will you proceed ahead with solving the problem.

Sales Analysis

Set up for success with sales analysis methods and techniques



Product Sales Analysis Using Python

OVERVIEW

In this post, I use Python Pandas & Python Matplotlib to analyze and answer business questions about 12 months worth of sales data. The data contains hundreds of thousands of electronics store purchases broken down by month, product type, cost, purchase address, etc. The dataset can be downloaded [here](#). In this analysis, I'm using Jupyter Notebook.

Source: <https://www.freepik.com/photos/business> Business photo created by cookie_studio —

PROBLEMS

1. What was the best month for sales? How much was earned that month?
2. What city sold the most products?
3. What times should we display advertisements to maximize likelihood of customer's buying products?
4. What products are most often sold together?
5. What products sold the most? Why do you think it did?

SOLUTION

1. What was the best month for sales? How much was earned that month?

First of all, we need to see what kind of data we are trying to analyze. We can simply do this

```
Import Necessary Libraries
In [3]: 1 import pandas as pd

Look at the first 5 datas
In [5]: 1 df = pd.read_csv("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data
2
3 df.head()
4

Out[5]:
   Order ID      Product  Quantity Ordered  Price Each  Order Date  Purchase Address
0  176558  USB-C Charging Cable           2       11.95 04/19/19 08:46  917 1st St, Dallas, TX 75001
1     NaN            NaN           NaN       NaN        NaN          NaN
2  176559  Bose SoundSport Headphones           1       99.99 04/07/19 22:30  682 Chestnut St, Boston, MA 02215
3  176560        Google Phone           1       600.0 04/12/19 14:38  669 Spruce St, Los Angeles, CA 90001
4  176560        Wired Headphones           1       11.99 04/12/19 14:38  669 Spruce St, Los Angeles, CA 90001
```

Figure 1. Code to show the top 5 data in Sales_April_2019.csv

We use pandas to read the csv file and create a data frame from it. The file's directory can be put anywhere. I personally put it in D. You can copy the directory and paste it in the syntax. At Figure 1, we can see that we have 6 columns. Now the first task is to merge all 12 months' worth of sales data (12 csv files) into a single csv file. To do that, we need to import new library called os.

```
In [3]: 1 import pandas as pd  
2 import os
```

Task 1: Merging 12 months of sales data into a single csv file.

```
In [7]: 1 df = pd.read_csv("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas\Data  
2  
3 files = [file for file in os.listdir("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science Task  
4 for file in files:  
5     print(file)  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2059  
2060  
2061  
2062  
2063  
2064  

```

Figure2.Importnew libraryos.

We need a library to read all csv files' title and call it using *for loop*. As you can see from Figure 2, we successfully read all the csv files' title and we're ready to merge it. To do that, we can simply do

Import Necessary Libraries

```
In [3]: 1 import pandas as pd  
2 import os
```

Task 1: Merging 12 months of sales data into a single csv file.

```
In [13]: 1 df = pd.read_csv("D:\\Self\\Online Course\\Solve real world Data science task\\Pandas-Data-Science-Tasks-master\\Pandas-Data
2
3 files = [file for file in os.listdir("D:\\Self\\Online Course\\Solve real world Data science task\\Pandas-Data-Science-Task
4
5 all_months_data = pd.DataFrame() #Creating empty dataframe called 'all_month_data'
6
7 for file in files:
8     df = pd.read_csv("D:\\Self\\Online Course\\Solve real world Data science task\\Pandas-Data-Science-Tasks-master\\Pandas-
9     all_months_data = pd.concat([all_months_data, df]) #Herging to the previous empty dataframe
10
11 #Checking the result
12 all_months_data.to_csv("all_data.csv", index=False) #since csv file contain 12 months data.
```

In [1]:

Figure3.Creatinganew file containsall12 monthsdata.

--DETAIL CODE-----

```
import pandas as pd
import os

df=pd.read_csv("D:\Self\OnlineCourse\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data-Science-Tasks-master\SalesAnalysis\Sales_Data\Sales_April_2019.csv")

files=[file for file in os.listdir("D:\Self\OnlineCourse\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data-Science-Tasks-master\SalesAnalysis\Sales_Data")]

all_months_data=pd.DataFrame()#Creating empty dataframe called 'all_month_data'

for file in files:
    df = pd.read_csv("D:\Self\Online Course\Solve real world Data science task\Pandas-Data-Science-Tasks-master\Pandas-Data-Science-Tasks-master\SalesAnalysis\Sales_Data/" + file)
    all_months_data=pd.concat([all_months_data,df])#Merging to the previous empty dataframe

#Checking the result
all_months_data.to_csv("all_data.csv",index=False)#single csv file contains 12 months data.
```

It will take a little longer time because of heavy computation. But once it's done, you can open the same directory folder and check the folder called "Output". You will see a new csv file contains all 12 months data.

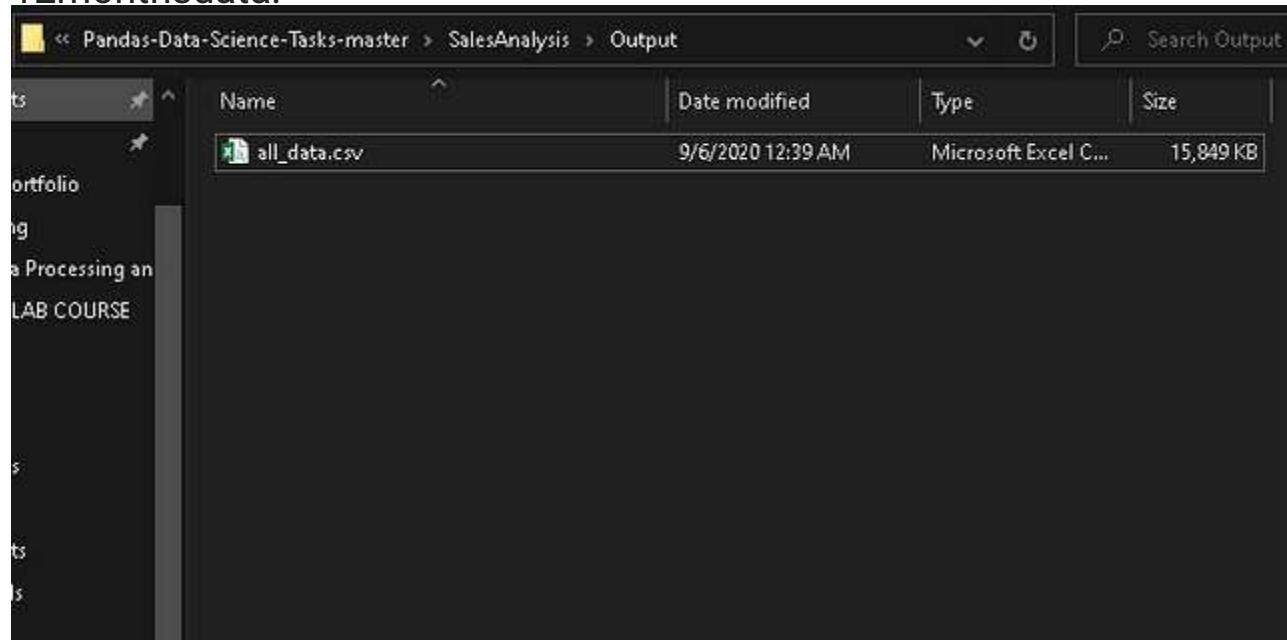


Figure 4. A new csv file contains all 12 months data.

After we create this new csv file, you can delete the previous code (if you want) and we will use this file to answer all of the problems.

Now we only use this code to read all of 12 months data.

```
Reading an updated dataframe
```

In [14]:

```
1 all_data=pd.read_csv("all_data.csv")
2 all_data.head()
```

Out [14]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
1	NaN	NaN	NaN	NaN	NaN	NaN
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001

Figure 5. Reading an updated dataframe

Now we're ready to answer the problem number 1. To remind you, the question is: **What was the best month for sales?**
How much was earned that month?

To answer this problem, obviously we need an additional column called "Month". If you look carefully at Figure 5, you will see the first 2 characters in "Order Date" values represent months. So the next task we will do is to add "Month" Column.

Task 2: Add "Month" Column

```
In [15]: 1 all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.  
2 all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer  
3 all_data.head()  
  
-----  
ValueError: Traceback (most recent call last)  
<ipython-input-15-19256a884797> in <module>  
      1 all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.  
----> 2 all_data['Month'] = all_data['Month'].astype('int32')  
      3 all_data.head()  
  
ValueError: cannot convert float NaN to integer
```

Figure 6. Adding "Month" Column

Now, we get an issue here. There are NaN values in our data. You could spot one of NaN value at Figure 1 or Figure 5 in index 1. Now we need to clean up the data by dropping rows of NaN. Let's spot more NaN value here. You don't have to do this, I am just curious.

Task 2: Add "Month" Column

```
In [16]: 1 nan_df = all_data[all_data.isna().any(axis=1)]  
2  
3 nan_df.head()  
4  
5 #all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.  
6 #all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer  
7 #all_data.head()
```

Out [16]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
356	NaN	NaN	NaN	NaN	NaN	NaN	NaN
735	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1433	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1553	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 7. Spotting the NaN Values in our data.

We use `.isna().any(axis=1)` to spot rows containing the NaN values (axis=0 to spot column containing NaN values). Now, we're going to remove it from our dataframe using `.dropna()` method.

```
Task 2: Add "Month" Column

In [17]: 1 all_data=all_data.dropna(how='all')
2
3 all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
4 all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer
5 all_data.head()

ValueError: Traceback (most recent call last)
<ipython-input-17-dcd59b77aa46> in <module>
      2
      3 all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
----> 4 all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer
      5 all_data.head()

ValueError: invalid literal for int() with base 10: 'Or'
```

Figure 8. Dropping the NaN values from our dataframe.

`.dropna()` method is successful, but we get a new issue here. There are values "Or" in our data. Let's find it first.

```
Task 2: Add "Month" Column

In [19]: 1 #Removing Nan Values in our data
2 all_data=all_data.dropna(how='all')
3
4 #Removing rows based on condition, finding 'Or' and delete it
5 df_dummy = all_data[all_data['Order Date'].str[0:2]=='Or']
6 df_dummy.head()
7
8 #all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
9 #all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer
10 #all_data.head()

Out[19]:
   Order ID Product Quantity Ordered Price Each Order Date Purchase Address Month
519  Order ID  Product  Quantity Ordered  Price Each  Order Date  Purchase Address  Or
1149  Order ID  Product  Quantity Ordered  Price Each  Order Date  Purchase Address  Or
1155  Order ID  Product  Quantity Ordered  Price Each  Order Date  Purchase Address  Or
2878  Order ID  Product  Quantity Ordered  Price Each  Order Date  Purchase Address  Or
2893  Order ID  Product  Quantity Ordered  Price Each  Order Date  Purchase Address  Or
```

Figure 9. Finding "Or" values in our dataframe.

We can see clearly from Figure 9 that the issue is the rows contain the same words as the title rows. So clearly 'Or' is coming from 'Order Date'. We need to drop this "Or" rows just simply change the equals sign ("==") to not equals sign ("!=").

Task 2: Add "Month" Column

```
In [20]: 1 #Removing Nan Values in our data
2 all_data=all_data.dropna(how='all')
3
4 #Removing rows based on condition, finding 'Or' and delete it
5 all_data = all_data[all_data['Order Date'].str[0:2]!='Or']
6
7 all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
8 all_data['Month'] = all_data['Month'].astype('int32') #Turning the data from string to integer
9 all_data.head()
```

Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month
0	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4
2	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4
3	Google Phone	1	600	04/12/19 14:39	669 Spruce St, Los Angeles, CA 90001	4
4	Wired Headphones	1	11.99	04/12/19 14:39	669 Spruce St, Los Angeles, CA 90001	4
5	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4

Figure10. Dropping all unnecessary values in our dataframe.

We can see clearly from Figure 10 that we successfully created "Month" column and make its datatype to integer.

Now, are we ready to answer the question? Not yet, we need obviously one more column called "Sales" Column. How can we get that? We get "Sales" by multiplying "Quantity Ordered" and "Price Each" values. Let's create it.

Task 2: Add "Month" Column

```
In [21]: 1 #Removing Nan Values in our data
2 all_data=all_data.dropna(how='all')
3
4 #Removing rows based on condition, finding 'Or' and delete it
5 all_data = all_data[all_data['Order Date'].str[0:2]!='Or']
6
7 #Add "Month" Column
8 all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
9 all_data['Month'] = all_data['Month'].astype('int32') #Turning the data from string to integer
10
11 #Add "Sales" Column
12 all_data['Sales'] = all_data['Quantity Ordered']*all_data['Price Each']
13
14 all_data.head()
```

```
-----  
TypeError: Traceback (most recent call last)  
~/anaconda3/lib/site-packages/pandas\core\ops\array_ops.py in na_arithmetic_op(left, right, op, str_rep)  
148     try:  
--> 149         result = expressions.evaluate(op, str_rep, left, right)  
150     except TypeError:  
  
TypeError: can't multiply sequence by non-int of type 'str'
```

Figure11. Adding "Sales" Column in our dataframe.

Now we encounter a new issue. The values of the column "QuantityOrdered" and "Price Each" are strings. So the next task is to

convert these columns to the correct type ("QuantityOrdered" is integer and "Price Each" is float). We're gonna use `pd.to_numeric()` method

to convert them to numeric.

```
Task 2: Add "Month" Column
In [22]: 1 #Removing Nan Values in our data
2 all_data=all_data.dropna(how='all')
3
4 #Removing rows based on condition, finding 'Or' and delete it
5 all_data = all_data[all_data['Order Date'].str[0:2]!='Or']
6
7 #Add "Month" Column
8 all_data['Month'] = all_data['Order Date'].str[0:2] #Get the first 2 characters.
9 all_data['Month'] = all_data['Month'].astype('int32') #turning the data from string to integer
10
11 #Convert 'Quantity Ordered' and 'Price Each' to numeric
12 all_data['Quantity Ordered'] = pd.to_numeric(all_data['Quantity Ordered']) #Becoming integer
13 all_data['Price Each'] = pd.to_numeric(all_data['Price Each']) #Becoming float
14
15 #Add "Sales" Column
16 all_data['Sales'] = all_data['Quantity Ordered']*all_data['Price Each']
17 all_data.head()
```

Out[22]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99

Figure12.Converting "QuantityOrdered" and "PriceEach" Columns to Numeric

Now the "Sales" Column is successfully created, we can answer the first question. What was the best month for sales? How much was earned that month? We can easily answer it by using `groupby('Month').sum()` method.

Question 1: What was the best month for sales? How much was earned that month?

```
In [23]: 1 all_data.groupby('Month').sum()  
Out[23]:
```

Month	Quantity Ordered	Price Each	Sales
1	10903	1.811768e+06	1.822257e+06
2	13449	2.188885e+06	2.202022e+06
3	17005	2.791208e+06	2.807100e+06
4	20558	3.367671e+06	3.390670e+06
5	18667	3.135125e+06	3.152607e+06
6	15253	2.562026e+06	2.577802e+06
7	16072	2.632540e+06	2.647776e+06
8	13448	2.230345e+06	2.244468e+06
9	13109	2.084992e+06	2.097560e+06
10	22703	3.715555e+06	3.736727e+06
11	19798	3.180601e+06	3.199603e+06
12	28114	4.598415e+06	4.613443e+06

Figure13. Grouping by month and summing the Sales.

Look carefully at Figure 13. We can clearly see that month 12 (December) is the highest sales in 2019 with approximately \$4,810,000. But we need to visualize it to make our business partners easier to understand. So we're going to import matplotlib and visualize our results with a bar chart.

Importing Matplotlib

```
In [27]: 1 import matplotlib.pyplot as plt
```

Visualizing our results

```
In [28]: 1 months = range(1,13) #For x axes  
2 results = all_data.groupby('Month').sum()  
3  
4 plt.bar(months, results['Sales'])  
5 plt.show()
```

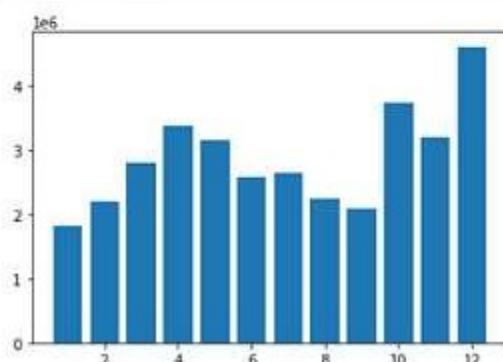


Figure14. Visualizing our results using matplotlib library.

It's good. But we need to make it look snazzy. So we're just gonna add a little code.

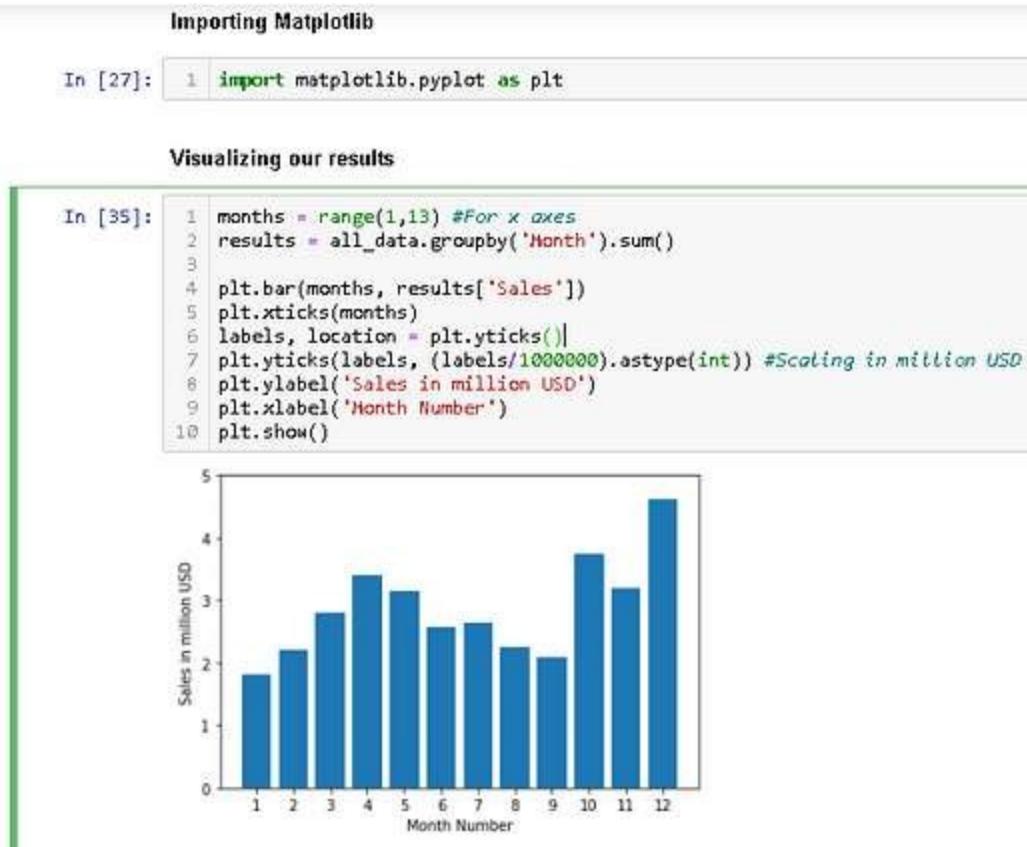


Figure 15. Improving the visualization.

Now, not only we can get the highest sales, but we can also get the lowest sales just by looking at it for a few seconds. As a data scientist, we have to figure out why a certain month is better than others.

Maybe the company spends more money on April so the product sales are increasing. Maybe the best product sales are on December because it's holiday and Christmas. Those are just my hypotheses, right? Now we don't have enough data to prove that hypothesis.

But we can take these as a consideration if you want to decide something that relates to product sales.

2. What city sold the most products?

To answer this question, obviously we need to create a new column called “City” column. How do we get that? As usual, we’re gonna check the top 5 data in our DataFrame to figure out where can we get our “City” column using `.head()` method.

Question 2: What city sold the most product?

Task 3: Add a “City” Column

In [11]:	1 all_data.head()																																																
Out[11]:	<table border="1"> <thead> <tr> <th>Order ID</th><th>Product</th><th>Quantity Ordered</th><th>Price Each</th><th>Order Date</th><th>Purchase Address</th><th>Month</th><th>Sales</th></tr> </thead> <tbody> <tr> <td>0 176558</td><td>USB-C Charging Cable</td><td>2</td><td>11.95</td><td>04/19/19 08:46</td><td>917 1st St, Dallas, TX 75001</td><td>4</td><td>23.90</td></tr> <tr> <td>2 176559</td><td>Bose SoundSport Headphones</td><td>1</td><td>99.99</td><td>04/07/19 22:30</td><td>682 Chestnut St, Boston, MA 02215</td><td>4</td><td>99.99</td></tr> <tr> <td>3 176560</td><td>Google Phone</td><td>1</td><td>600.00</td><td>04/12/19 14:38</td><td>669 Spruce St, Los Angeles, CA 90001</td><td>4</td><td>600.00</td></tr> <tr> <td>4 176560</td><td>Wired Headphones</td><td>1</td><td>11.99</td><td>04/12/19 14:38</td><td>669 Spruce St, Los Angeles, CA 90001</td><td>4</td><td>11.99</td></tr> <tr> <td>5 176561</td><td>Wired Headphones</td><td>1</td><td>11.99</td><td>04/30/19 09:27</td><td>333 8th St, Los Angeles, CA 90001</td><td>4</td><td>11.99</td></tr> </tbody> </table>	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	0 176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	2 176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	3 176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	4 176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	5 176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99
Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales																																										
0 176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90																																										
2 176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99																																										
3 176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00																																										
4 176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99																																										
5 176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99																																										

Figure 16. Showing Our Top 5 Updated DataFrame

As you can see at Figure 16, the “Purchase Address” Column contains the city. We can’t get it directly, we need to extract the data. We can use one of most useful function in pandas, `.apply()` method.

Question 2: What city sold the most product?

Task 3: Add a “City” Column

In [15]:	1 all_data['City'] = all_data['Purchase Address'].apply(lambda x: x.split(',')[-1]) 2 3 all_data.head()																																																						
Out[15]:	<table border="1"> <thead> <tr> <th>Order ID</th><th>Product</th><th>Quantity Ordered</th><th>Price Each</th><th>Order Date</th><th>Purchase Address</th><th>Month</th><th>Sales</th><th>City</th></tr> </thead> <tbody> <tr> <td>0 176558</td><td>USB-C Charging Cable</td><td>2</td><td>11.95</td><td>04/19/19 08:46</td><td>917 1st St, Dallas, TX 75001</td><td>4</td><td>23.90</td><td>Dallas</td></tr> <tr> <td>2 176559</td><td>Bose SoundSport Headphones</td><td>1</td><td>99.99</td><td>04/07/19 22:30</td><td>682 Chestnut St, Boston, MA 02215</td><td>4</td><td>99.99</td><td>Boston</td></tr> <tr> <td>3 176560</td><td>Google Phone</td><td>1</td><td>600.00</td><td>04/12/19 14:38</td><td>669 Spruce St, Los Angeles, CA 90001</td><td>4</td><td>600.00</td><td>Los Angeles</td></tr> <tr> <td>4 176560</td><td>Wired Headphones</td><td>1</td><td>11.99</td><td>04/12/19 14:38</td><td>669 Spruce St, Los Angeles, CA 90001</td><td>4</td><td>11.99</td><td>Los Angeles</td></tr> <tr> <td>5 176561</td><td>Wired Headphones</td><td>1</td><td>11.99</td><td>04/30/19 09:27</td><td>333 8th St, Los Angeles, CA 90001</td><td>4</td><td>11.99</td><td>Los Angeles</td></tr> </tbody> </table>	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	0 176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas	2 176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston	3 176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles	4 176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles	5 176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles
Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City																																															
0 176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas																																															
2 176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston																																															
3 176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles																																															
4 176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles																																															
5 176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles																																															

Figure 17. Using `.apply()` Method to Extract The Data

To make it easier, we can use this

Question 2: What city sold the most product?

Task 3: Add a "City" Column

In [17]:	Code																																																												
	<pre>1 #Function 2 def get_city(address): 3 return address.split(',')[1] 4 5 #Extract the city and the state 6 all_data['City'] = all_data['Purchase Address'].apply(lambda x: get_city(x)) 7 8 all_data.head()</pre>																																																												
Out[17]:	<table><thead><tr><th></th><th>Order ID</th><th>Product</th><th>Quantity Ordered</th><th>Price Each</th><th>Order Date</th><th>Purchase Address</th><th>Month</th><th>Sales</th><th>City</th></tr></thead><tbody><tr><td>0</td><td>176558</td><td>USB-C Charging Cable</td><td>2</td><td>11.95</td><td>04/19/19 08:45</td><td>917 1st St, Dallas, TX 75001</td><td>4</td><td>23.90</td><td>Dallas</td></tr><tr><td>1</td><td>176559</td><td>Bose SoundSport Headphones</td><td>1</td><td>99.99</td><td>04/07/19 22:30</td><td>682 Chestnut St, Boston, MA 02215</td><td>4</td><td>99.99</td><td>Boston</td></tr><tr><td>2</td><td>176560</td><td>Google Phone</td><td>1</td><td>600.00</td><td>04/12/19 14:38</td><td>669 Spruce St, Los Angeles, CA 90001</td><td>4</td><td>600.00</td><td>Los Angeles</td></tr><tr><td>3</td><td>176560</td><td>Wired Headphones</td><td>1</td><td>11.99</td><td>04/12/19 14:38</td><td>669 Spruce St, Los Angeles, CA 90001</td><td>4</td><td>11.99</td><td>Los Angeles</td></tr><tr><td>4</td><td>176561</td><td>Wired Headphones</td><td>1</td><td>11.99</td><td>04/30/19 09:27</td><td>333 8th St, Los Angeles, CA 90001</td><td>4</td><td>11.99</td><td>Los Angeles</td></tr></tbody></table>		Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:45	917 1st St, Dallas, TX 75001	4	23.90	Dallas	1	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston	2	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles	3	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles	4	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles
	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City																																																				
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:45	917 1st St, Dallas, TX 75001	4	23.90	Dallas																																																				
1	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston																																																				
2	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles																																																				
3	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles																																																				
4	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles																																																				

Figure18. Usingdef functionto MakeTheCodeNeater

applyandlambdausuallyusedtocreatenewcolumnbasedonothercolumn (example .apply(lambda x: x*2. It means every input x inother column will be changed to x*2 in a new column). In this casewe create “City” column based on “Purchase Address” column andwe split the data into 3 part. The first one is before the first comma(index = 0), the second one is between the commas (index = 1), andthe third on is after the last comma (index = 2). As we need toextractthecitydata,weuse[1]tostateittoindex1.

AsyoucanseeatFigure17,wesuccessfullycreateda“City”column.So are we ready to answer the second question? Not yet. We get anissue here. It's not error, it's the value of the “City” Column. This isjust a rare case when there are 2 cities are named exactly the same.Example someone in New England and someone in West Coastwould think Portland in different way. Someone in New Englandthinks Portland as Portland Maine and someone in West CoastthinksPortlandasPortlandOregon.Soinourdatasetweactually

had the overlapping cities between these two. So, we should also grab both states.

Question 2: What city sold the most product?

Task 3: Add a "City" Column

```
In [18]:  
1 #Function  
2 def get_city(address):  
3     return address.split(',')[1]  
4  
5 def get_state(address):  
6     return address.split(',')[2].split(' ')[1]  
7  
8 #Extract the city and the state  
9 all_data['City'] = all_data['Purchase Address'].apply(lambda x: get_city(x) + ' ' + get_state(x))  
10  
11 all_data.head()  
  
Out[18]:  


|   | Order ID | Product                    | Quantity Ordered | Price Each | Order Date     | Purchase Address                     | Month | Sales  | City           |
|---|----------|----------------------------|------------------|------------|----------------|--------------------------------------|-------|--------|----------------|
| 0 | 176558   | USB-C Charging Cable       | 2                | 11.95      | 04/19/19 08:46 | 917 1st St, Dallas, TX 75001         | 4     | 23.90  | Dallas TX      |
| 2 | 176559   | Bose SoundSport Headphones | 1                | 99.99      | 04/07/19 22:30 | 682 Chestnut St, Boston, MA 02215    | 4     | 99.99  | Boston MA      |
| 3 | 176560   | Google Phone               | 1                | 600.00     | 04/12/19 14:38 | 669 Spruce St, Los Angeles, CA 90001 | 4     | 600.00 | Los Angeles CA |
| 4 | 176560   | Wired Headphones           | 1                | 11.99      | 04/12/19 14:38 | 669 Spruce St, Los Angeles, CA 90001 | 4     | 11.99  | Los Angeles CA |
| 5 | 176561   | Wired Headphones           | 1                | 11.99      | 04/30/19 09:27 | 333 8th St, Los Angeles, CA 90001    | 4     | 11.99  | Los Angeles CA |


```

Figure 19. Extracting the state to "City" Column

The function `get_state()` basically works as explained before. But in this function, we separate the data again into three parts. The first one is before the whitespace (`index = 0`), the second one is between the whitespaces (`index = 1`), and the third one is after the last whitespace (`index = 2`). So that's why we use `.split(' ')[1]` in the second split.

Now we're ready to answer the second question, **what city sold the most product?** As we did before, we're going to group it by the city and summing all the values based on the group.

```
In [19]: 1 results2 = all_data.groupby('City').sum()
2 results2
```

City	Quantity Ordered	Price Each	Month	Sales
Atlanta GA	16602	2.779908e+06	104794	2.795499e+06
Austin TX	11153	1.809874e+06	69829	1.819582e+06
Boston MA	22528	3.637410e+06	141112	3.681642e+06
Dallas TX	16730	2.752628e+06	104620	2.767975e+06
Los Angeles CA	33209	5.421435e+06	208325	5.452571e+06
New York City NY	27932	4.635371e+06	175741	4.664317e+06
Portland ME	2750	4.471893e+05	17144	4.497583e+05
Portland OR	11303	1.860558e+06	70621	1.870732e+06
San Francisco CA	50239	8.211462e+06	315520	8.262204e+06
Seattle WA	16553	2.733296e+06	104941	2.747755e+06

Figure20. Grouping The DataframebyTheCity

It's too messy, but if you look carefully you can see that SanFranciscois the highest sold product of all cities with approximately \$8,200,000. We clearly need to visualize it because it's so hard to conclude anything just based on that numbers and also it will make our business partner easier to understand.

```
In [22]: 1 #We've already import the matplotlib
2
3 cities = all_data['City'].unique()
4
5 plt.bar(cities, results2['Sales'])
6 plt.xticks(cities, rotation='vertical', size = 8)
7 labels, location = plt.yticks()
8 plt.yticks(labels, (labels/1000000).astype(int)) #Scaling in million USD
9 plt.ylabel('Sales in million USD')
10 plt.xlabel('City Name')
11 plt.show()
```

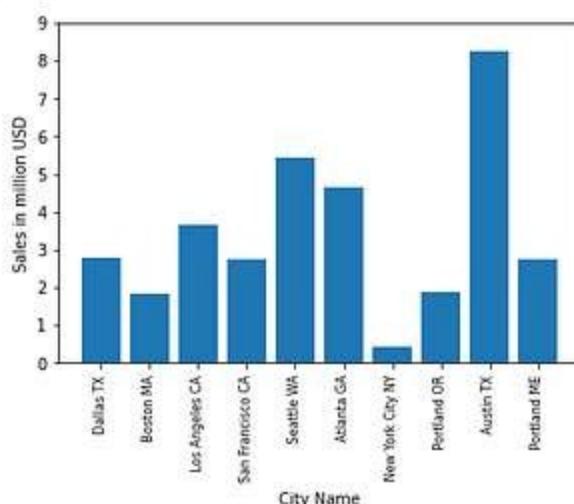


Figure21. Plottingthe Sales Grouped byCities.

Now we successfully plot it. But there is a big issue here. If you notice that the values (Figure 20) and the plot (Figure 21) are not synchronized. The highest sales should be San Francisco. What's wrong with our code?

There's an issue between `.unique()` method and `plt.bar()`. Their city order are different. We're going to synchronize the order by simply fixing the variable 'cities'.

```
In [23]: 1 #We've already import the matplotlib
2
3 #Fixing the cities order
4 cities = all_data['City'].unique()
5 cities = [city for city, df in all_data.groupby('City')]
6
7 plt.bar(cities, results2['Sales'])
8 plt.xticks(cities, rotation='vertical', size = 8)
9 labels, location = plt.yticks()
10 plt.yticks(labels, (labels/1000000).astype(int)) #Scaling in million USD
11 plt.ylabel('Sales in million USD')
12 plt.xlabel('City Name')
13 plt.show()
```

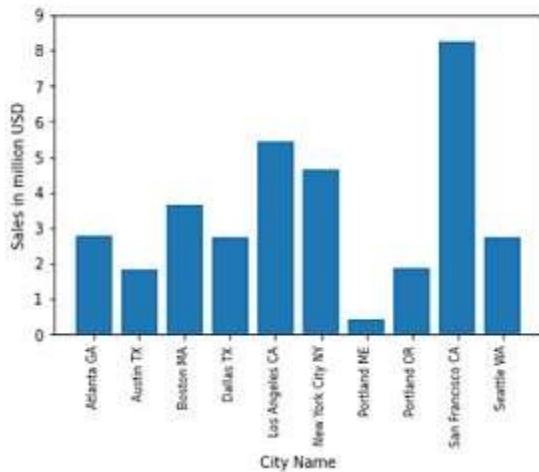


Figure 22. Fixing The Code

Now, we fix the issue and successfully plot it. As a data scientist, we need to figure out why San Francisco is the highest sale compared to other cities. Maybe Silicon Valley needs more electronic products.

Maybe the advertisement is better in San Francisco. We can use this data to improve the sales of business.

3. What Time Should We Display Advertisements to Maximize Likelihood of Customer's Buying Product?

As usual, to remind what our data look like, we use the `.head()` method to show the top 5 of our updated DataFrame.

Question 3: What time should we display advertisements to maximize likelihood of customer's buying product?

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas TX
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston MA
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles CA

Figure 23. Showing our Top 5 updated DataFrame

If we're gonna use our data to answer this question, we need to aggregate the period in 24 hours distribution. Look carefully at Figure 23. In "Order Date" column, there are times data. We could extract it like we did before. But to make it more consistent, we need to convert the "Order Date" Column into date time object.

We're gonna use `pd.to_datetime()` method.

Question 3: What time should we display advertisements to maximize likelihood of customer's buying product?

Task 4: Aggregate the period in 24-hours distribution

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order_Date_DTO
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas TX	2019-04-19 08:46:00
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 22:30:00
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:38:00
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-30 09:27:00

Figure 24. Converting the "Order Date" Column into Date-Time Object

It will take a little bit longer because of the heavy calculation. Now we can create a new column called "Hour" contain the extraction of "Order_Date.DTO" data. We only need the hours data, so we can extract them by doing this.

Task 4: Aggregate the period in 24-hours distribution

```
In [30]: 1 #Create new column in date-time Object (DTO)
2 all_data['Order_Date.DTO'] = pd.to_datetime(all_data['Order Date'])
3
4 #Extraction the hours data
5 all_data['Hour'] = all_data['Order_Date.DTO'].dt.hour
6
7 all_data.head()
```

Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order Date DTO	Hour
0 176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas TX	2019-04-19 08:46:00	8
2 176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 22:30:00	22
3 176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:38:00	14
4 176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00	14
5 176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-30 09:27:00	9

Figure25.ExtractingTheHours Data IntoTheNewColumn

Now we can answer the third question, what time should we display advertisements to maximize likelihood of customer's buying product? To answer this, we're going to group it by hours and count the number of orders.

```
In [48]: 1 results3 = all_data.groupby(['Hour']).count()
2 results3
```

13	12129	12129	12129	12129	12129	12129	12129	12129	12129	12129
14	10984	10984	10984	10984	10984	10984	10984	10984	10984	10984
15	10175	10175	10175	10175	10175	10175	10175	10175	10175	10175
16	10384	10384	10384	10384	10384	10384	10384	10384	10384	10384
17	10899	10899	10899	10899	10899	10899	10899	10899	10899	10899
18	12280	12280	12280	12280	12280	12280	12280	12280	12280	12280
19	12905	12905	12905	12905	12905	12905	12905	12905	12905	12905
20	12228	12228	12228	12228	12228	12228	12228	12228	12228	12228
21	10921	10921	10921	10921	10921	10921	10921	10921	10921	10921
22	8822	8822	8822	8822	8822	8822	8822	8822	8822	8822
23	6275	6275	6275	6275	6275	6275	6275	6275	6275	6275

Figure26. Grouping the data by the hours

If we want to answer the third question, we only need the “QuantityOrdered” column. Now let’s visualize it. We want it to be the linechart because this spesific data (hours) are more logical to show using linechart than barchart because the data has to be continue.

```
In [41]: 1 #Plotting
2 results3 = all_data.groupby(['Hour'])['Quantity Ordered'].count()
3 hours = [hour for hour, df in all_data.groupby('Hour')]
4
5 plt.plot(hours, results3)
6 plt.xticks(hours)
7 plt.xlabel('Hour')
8 plt.ylabel('Number of Orders')
9 plt.grid()
10 plt.show()
```

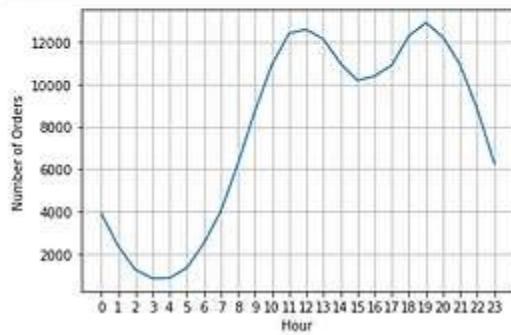


Figure27. Visualizing the Number of Orders in 24 hours format

As you can see from Figure27, there are approximately 2 peaks at the data. They are 12(12PM) and 19(7PM). It makes sense since most people shopping during the day. From this data, we can suggest to our business partner to advertise their product right before 12PM and/or 7PM. It could be 11.30AM and/or 6.30PM.

Remember, this chart is the total orders of all cities. Maybe you could make a specific chart for a specific city and planning the advertisement better for that city.

4. What Products Are Most Often Sold Together?

We're gonna take a look to our top 5 data as usual.

Question 4: What products are most often sold together?														
In [42]:	1	all_data.head()												
Out[42]:	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order_Date.DTO	Hour			
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas TX	2019-04-19 08:46:00	8			
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 22:30:00	22			
3	176560	Google Phone	1	600.00	04/12/19 14:30	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:30:00	14			
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00	14			
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-30 09:27:00	9			

Figure 28. Showing Our Top 5 Updated Dataframe

Look carefully at Figure 28. We can see that “Order ID” indicate the transaction. So by grouping the product by the Order ID, we are able to know what products are often sold together. We’re gonna use .duplicated() method to find duplicate values of “Order ID”.

Task 5: Make a new column called "Product Bundle"														
In [43]:	1	#Make a new dataframe to separate the duplicated values of Order ID												
Out[43]:	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order_Date.DTO	Hour			
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:38:00	14			
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00	14			
18	176574	Google Phone	1	600.00	04/03/19 19:42	20 Hill St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-03 19:42:00	19			
19	176574	USB-C Charging Cable	1	11.95	04/03/19 19:42	20 Hill St, Los Angeles, CA 90001	4	11.95	Los Angeles CA	2019-04-03 19:42:00	19			
30	176585	Bose SoundSport Headphones	1	99.99	04/07/19 11:31	823 Highland St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 11:31:00	11			
31	176585	Bose SoundSport Headphones	1	99.99	04/07/19 11:31	823 Highland St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 11:31:00	11			
32	176586	AAA Batteries (4-pack)	2	2.99	04/10/19 17:00	365 Center St, San Francisco, CA 94101	4	5.98	San Francisco CA	2019-04-10 17:00:00	17			

33	176586	Google Phone	1	600.00	04/10/19 17:00	365 Center St, San Francisco, CA 94016	4	600.00	San Francisco CA	2019-04-10 17:00:00	17
119	176672	Lightning Charging Cable	1	14.95	04/12/19 11:07	778 Maple St, New York City, NY 10001	4	14.95	New York City NY	2019-04-12 11:07:00	11
120	176672	USB-C Charging Cable	1	11.95	04/12/19 11:07	778 Maple St, New York City, NY 10001	4	11.95	New York City NY	2019-04-12 11:07:00	11
129	176681	Apple Airpods Headphones	1	150.00	04/20/19 10:39	331 Cherry St, Seattle, WA 98101	4	150.00	Seattle WA	2019-04-20 10:39:00	10
130	176681	ThinkPad Laptop	1	999.99	04/20/19 10:39	331 Cherry St, Seattle, WA 98101	4	999.99	Seattle WA	2019-04-20 10:39:00	10
138	176689	Bose SoundSport Headphones	1	99.99	04/24/19 17:15	659 Lincoln St, New York City, NY 10001	4	99.99	New York City NY	2019-04-24 17:15:00	17
139	176689	AAABatteries (4-pack)	2	2.99	04/24/19 17:15	659 Lincoln St, New York City, NY 10001	4	5.98	New York City NY	2019-04-24 17:15:00	17
189	176739	34in Ultrawide Monitor	1	379.99	04/05/19 17:38	730 6th St, Austin, TX 73301	4	379.99	Austin TX	2019-04-05 17:38:00	17
190	176739	Google Phone	1	600.00	04/05/19 17:38	730 6th St, Austin, TX 73301	4	600.00	Austin TX	2019-04-05 17:38:00	17
225	176774	Lightning Charging Cable	1	14.95	04/25/19 15:06	372 Church St, Los Angeles, CA 90001	4	14.95	Los Angeles CA	2019-04-25 15:06:00	15
226	176774	USB-C Charging Cable	1	11.95	04/25/19 15:06	372 Church St, Los Angeles, CA 90001	4	11.95	Los Angeles CA	2019-04-25 15:06:00	15

Figure29.ShowingOurTop20 DuplicatedDataframe

Now we want to create a column called “Product Bundle” that contain example *Google Phone* and *Wired Headphone* (transaction17650) at the same line. We’re gonna use the *.transform()* method to join values from two rows into a single row.

Question 4: What products are most often sold together?

Task 5: Make a new column called “Product Bundle”

```
In [44]: 1 #Make a new dataframe to separate the duplicated values of Order ID
2 new_all = all_data[all_data['Order ID'].duplicated(keep=False)]
3
4 #Joining few products with the same Order ID into the same line.
5 new_all['Product_Bundle'] = new_all.groupby('Order ID')['Product'].transform(lambda x: ','.join(x))
6
7 new_all.head()
```

Out[44]:

Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order Date DTO	Hour	Product_Bundle
3 176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:38:00	14	Google Phone,Wired Headphones
4 176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00	14	Google Phone,Wired Headphones
18 176574	Google Phone	1	600.00	04/03/19 19:42	20 Hill St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-03 19:42:00	19	Google Phone,USB-C Charging Cable
19 176574	USB-C Charging Cable	1	11.95	04/03/19 19:42	20 Hill St, Los Angeles, CA 90001	4	11.95	Los Angeles CA	2019-04-03 19:42:00	19	Google Phone,USB-C Charging Cable
30 176585	Bose SoundSport Headphones	1	99.99	04/07/19 11:31	823 Highland St, Boston, MA 02215	4	99.99	Boston MA	2019-04-07 11:31:00	11	Bose SoundSport Headphones,Bose SoundSport Hea...

Figure30.JoiningFewProductsWithTheSameOrderID IntoTheSame Line

It's good, but we have an issue here. We have the same order at least twice because we did merge them in every situation in groupby without dropping the duplicate values. Now let's drop the rows with duplicate values.

Question 4: What products are most often sold together?

Task 5: Make a new column called "Product Bundle"

```
In [18]: 1 #Make a new dataframe to separate the duplicated values of Order ID
2 new_all = all_data[all_data['Order ID'].duplicated(keep=False)]
3
4 #Joining few products with the same Order ID into the same line.
5 new_all['Product_Bundle'] = new_all.groupby('Order ID')['Product'].transform(lambda x: ','.join(x))
6
7 #Dropping the duplicate values
8 new_all = new_all[['Order ID','Product_Bundle']].drop_duplicates()
9
10 new_all.head()
```

Out[18]:	Order ID	Product_Bundle
3	176560	Google Phone,Wired Headphones
10	176574	Google Phone,USB-C Charging Cable
30	176585	Bose SoundSport Headphones,Bose SoundSport He...
32	176586	AAA Batteries (4-pack),Google Phone
119	176672	Lightning Charging Cable,USB-C Charging Cable

Figure31.Droppingrows withduplicatevalues

Now, we need to count the pair of products. We need new libraries because they have all we need to count all the combination of products bundle. We're gonna use *itertools* and *collections* libraries.

Task 6: Counting the Product bundles

```
In [21]: 1 #Importing libraries
2 from itertools import combinations
3 from collections import Counter
4
5 count = Counter()
6
7 for row in new_all['Product_Bundle']:
8     row_list = row.split(',')
9     count.update(Counter(combinations(row_list,2))) #Counting all the 2 products bundle
10 print(count)

Counter({('iPhone', 'Lightning Charging Cable'): 1005, ('Google Phone', 'USB-C Charging Cable'): 987, ('iPhone', 'Wired Headphones'): 447, ('Google Phone', 'Wired Headphones'): 414, ('Vareebadd Phone', 'USB-C Charging Cable'): 361, ('iPhone', 'Apple Airpods Headphones'): 360, ('Google Phone', 'Bose SoundSport Headphones'): 220, ('USB-C Charging Cable', 'Wired Headphones'): 160, ('Vareebadd Phone', 'Wired Headphones'): 143, ('Lightning Charging Cable', 'Wired Headphones'): 92, ('Lightning Charging Cable', 'Apple Airpods Headphones'): 81, ('Vareebadd Phone', 'Bose SoundSport Headphones'): 80, ('USB-C Charging Cable', 'Bose SoundSport Headphones'): 77, ('Apple Airpods Headphones', 'Wired Headphones'): 69, ('Lightning Charging Cable', 'USB-C Charging Cable'): 58, ('Lightning Charging Cable', 'AA Batteries (4-pack)'): 55, ('Lightning Charging Cable', 'Lightning Charging Cable'): 54, ('Bose SoundSport Headphones', 'Wired Headphones'): 53, ('AA Batteries (4-pack)', 'Lightning Charging Cable'): 51, ('AAA Batteries (4-pack)', 'USB-C Charging Cable'): 50, ('Apple Airpods Headphones', 'AAA Batteries (4-pack)'): 48, ('AA Batteries (4-pack)', 'AAA Batteries (4-pack)'): 48, ('USB-C Charging Cable', 'USB-C Charging Cable'): 48, ('AAA Batteries (4-pack)', 'AAA Batteries (4-pack)'): 48, ('USB-C Charging Cable', 'AAA Batteries (4-pack)'): 45, ('Wired Headphones', 'USB-C Charging Cable'): 45, ('AA Batteries (4-pack)', 'Hired Headphones'): 44, ('AAA Batteries (4-pack)', 'Lightning Charging Cable'): 44, ('AAA Batteries (4-pack)', 'Wired Headphones'): 43, ('Wired Headphones', 'AAA Batteries (4-pack)'): 43, ('USB-C Charging Cable', 'Lightning Charging Cable'): 42, ('AA Batteries (4-pack)', 'Apple Airpods Headphones'): 41, ('AAA Batteries (4-pack)', 'AA Batteries (4-pack)'): 39, ('Wired Headphones', 'AA Batteries (4-pack)'): 39, ('Lightning Charging Cable', 'Bose SoundSport Headphones'): 39, ('USB-C Charging Cable', 'AA Batteries (4-pack)'): 38, ('Bose SoundSport Headphones', 'AAA Batteries (4-pack)'): 37, ('AA Batteries (4-pack)', 'USB-C Charging Cable'): 37, ('Wired Headphones', 'Lightning Charging Cable'): 37, ('Lightning Charging Cable', 'AAA Batteries (4-pack)'): 36, ('Apple Airpods Headphones', 'Lightning Charging Cable'): 35, ('Wired Headphones', 'Hired Headphones'): 35, ('AA Batteries (4-pack)', 'AA Batteries (4-pack)'): 35})
```

Figure 32. Counting the Product Bundle

It's too messy, let's just showing the top 10 data

Task 6: Counting the Product bundles

```
In [23]: 1 #Importing Libraries
2 from itertools import combinations
3 from collections import Counter
4
5 count = Counter()
6
7 for row in new_all['Product_Bundle']:
8     row_list = row.split(',')
9     count.update(Counter(combinations(row_list,2))) #Counting all the 2 products bundle
10
11 count.most_common(10)
12 
```

```
Out[23]: [(['iPhone', 'Lightning Charging Cable'), 1005),
          ('Google Phone', 'USB-C Charging Cable'), 987),
          ('iPhone', 'Wired Headphones'), 447),
          ('Google Phone', 'Wired Headphones'), 414),
          ('Vareebadd Phone', 'USB-C Charging Cable'), 361),
          ('iPhone', 'Apple Airpods Headphones'), 368),
          ('Google Phone', 'Bose SoundSport Headphones'), 220),
          ('USB-C Charging Cable', 'Wired Headphones'), 168),
          ('Vareebadd Phone', 'Wired Headphones'), 143),
          ('Lightning Charging Cable', 'Wired Headphones'), 92)]
```

Figure33. ShowingThe Top 10-Product Bundles

Now we can clearly see that the most often products sold together are iPhone and Lightning Charging Cable with 1005 transactions. We could count the 3 product bundles by just changing the *count.update* index into 3.

Task 6: Counting the Product bundles

```
In [24]: 1 #Importing Libraries
2 from itertools import combinations
3 from collections import Counter
4
5 count = Counter()
6
7 for row in new_all['Product_Bundle']:
8     row_list = row.split(',')
9     #count.update(Counter(combinations(row_list,2))) #Counting all the 2 products bundle
10    count.update(Counter(combinations(row_list,3))) #Counting all the 3 products bundle
11
12 count.most_common(10)
13 
```

```
Out[24]: [(['Google Phone', 'USB-C Charging Cable', 'Wired Headphones'), 87),
           ('iPhone', 'Lightning Charging Cable', 'Wired Headphones'), 62),
           ('iPhone', 'Lightning Charging Cable', 'Apple Airpods Headphones'), 47),
           ('Google Phone', 'USB-C Charging Cable', 'Bose SoundSport Headphones'), 35),
           ('Vareebadd Phone', 'USB-C Charging Cable', 'Wired Headphones'), 33),
           ('iPhone', 'Apple Airpods Headphones', 'Wired Headphones'), 27),
           ('Google Phone', 'Bose SoundSport Headphones', 'Wired Headphones'), 24),
           ('Vareebadd Phone', 'USB-C Charging Cable', 'Bose SoundSport Headphones'),
           16),
           ('USB-C Charging Cable', 'Bose SoundSport Headphones', 'Wired Headphones'),
           5),
           ('Vareebadd Phone', 'Bose SoundSport Headphones', 'Wired Headphones'), 5)]
```

Figure34. ShowingThe Top 103-Product Bundles

We can see the most often sold products (3 products) together are Google Phone, USB-C Charging Cable, and Wired Headphones with 87 transactions. It's not really significant compare to the 2-Product Bundle. So we're gonna ignore the 3-Product bundle.

What would we do with this data? Well, we could offer a smart deal to the customer that buy iPhone, you could recommend the charging cable with discount. That's one of the possibility and you can bundle there maining products if you need to.

5. What products sold the most? Why do you think it did?

As usual, let's see our data looks like again.

Question 5: What Product sold the most? Why do you think it did?

In [25]:	1 all_data.head()										
Out [25]:											
	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	City	Order Date DTO	Hour
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	4	23.90	Dallas TX	2019-04-19 08:46:00	8
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02115	4	99.99	Boston MA	2019-04-07 22:30:00	22
3	176560	Google Phone	1	600.00	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	600.00	Los Angeles CA	2019-04-12 14:38:00	14
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-12 14:38:00	14
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001	4	11.99	Los Angeles CA	2019-04-30 09:27:00	9

Figure 35. Showing Out Top 5 Updated Dataframe

We need to sum up the "QuantityOrdered" based on grouping the Product. So let's do it.

Question 5: What Product sold the most? Why do you think it did?

Task 7: Grouping by the product

In [27]:

```
1 product_group = all_data.groupby('Product')
2 product_group.sum()
```

Out[27]:

Product	Quantity Ordered	Price Each	Month	Sales	Hour
20in Monitor	4129	451068.99	20336	454148.71	59764
27in 4K Gaming Monitor	6244	2429037.70	44440	2435097.56	90916
27in FHD Monitor	7550	1125974.93	52558	1132424.50	107540
34in Ultrawide Monitor	6199	2348718.19	43304	2355558.01	89076
AA Batteries (4-pack)	27635	79015.68	145558	106118.40	298342
AAA Batteries (4-pack)	31017	61716.59	140370	92740.83	297332
Apple Airpods Headphones	15661	2332350.00	108477	2349150.00	223304
Bose SoundSport Headphones	13457	1332366.75	94113	1345565.43	192445
Flatscreen TV	4819	1440000.00	34224	1445700.00	68815
Google Phone	5532	3315000.00	39305	3319200.00	79479
LG Dryer	646	387600.00	4383	387600.00	9326
LG Washing Machine	666	399600.00	4523	399600.00	9785
Lightning Charging Cable	23217	323787.10	153092	347094.15	312529
Machbook Pro Laptop	4728	8030800.00	33548	8037600.00	68261
ThinkPad Laptop	4130	4127958.72	28950	4129950.70	59746
USB-C Charging Cable	23975	261740.85	154819	286501.25	314645
Vareebaddi Phone	2068	826000.00	14309	827200.00	29472
Wired Headphones	20557	226395.18	133397	246478.43	271720
iPhone	6849	4789400.00	47941	4794300.00	98657

Figure36. Groupingbythe Product

To make it easier to understand, let's visualize it.

Question 5: What Product sold the most? Why do you think it did?

Task 7: Grouping by the product

In [28]:

```
1 product_group = all_data.groupby('Product')
2
3 #Visualizing
4 quantity_ordered = product_group.sum()['Quantity Ordered']
5
6 products = [product for product, df in product_group]
7
8 plt.bar(products, quantity_ordered)
9 plt.ylabel('Quantity Ordered')
10 plt.xlabel('Product')
11 plt.xticks(products, rotation='vertical', size=8)
12 plt.show()
```

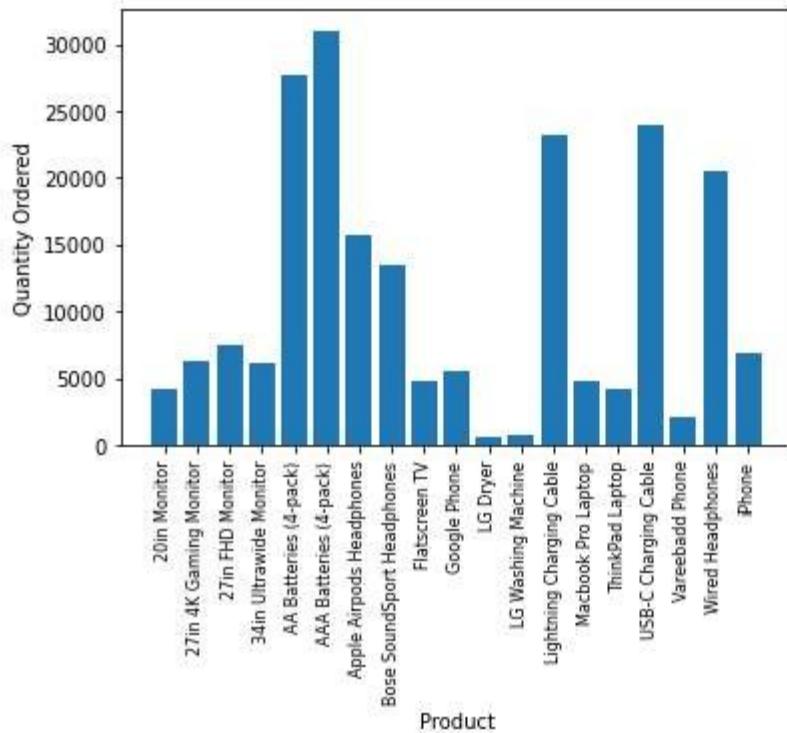


Figure37.VisualizingTheGroupedProduct.

Now we can see what product sold the most, it's AAA Batteries(4pack). We can also see that AA Batteries (4 pack), LightningChargingCable,USB-CChargingCable, and WiredHeadphones are sold more than other products. Why are they sold the most? The first impression is that they are cheaper than other products. As a data scientist, let's prove this hypothesis. We could do it by overlaying the graph by their actual price and see if they have direct correlation.

Task 8: Overlaying a second y-axis on existing chart

```
In [33]: 1 prices = all_data.groupby('Product').mean()['Price Each']
2
3 fig, ax1 = plt.subplots()
4
5 ax2 = ax1.twinx()
6 ax1.bar(products, quantity_ordered, color='g')
7 ax2.plot(products, prices, 'b-')
8
9 ax1.set_xlabel('Product Name')
10 ax1.set_ylabel('Quantity Ordered', color='g')
11 ax2.set_ylabel('Price ($)', color = 'b')
12 ax1.set_xticklabels(products, rotation='vertical', size=8)
13
14 plt.show()
```

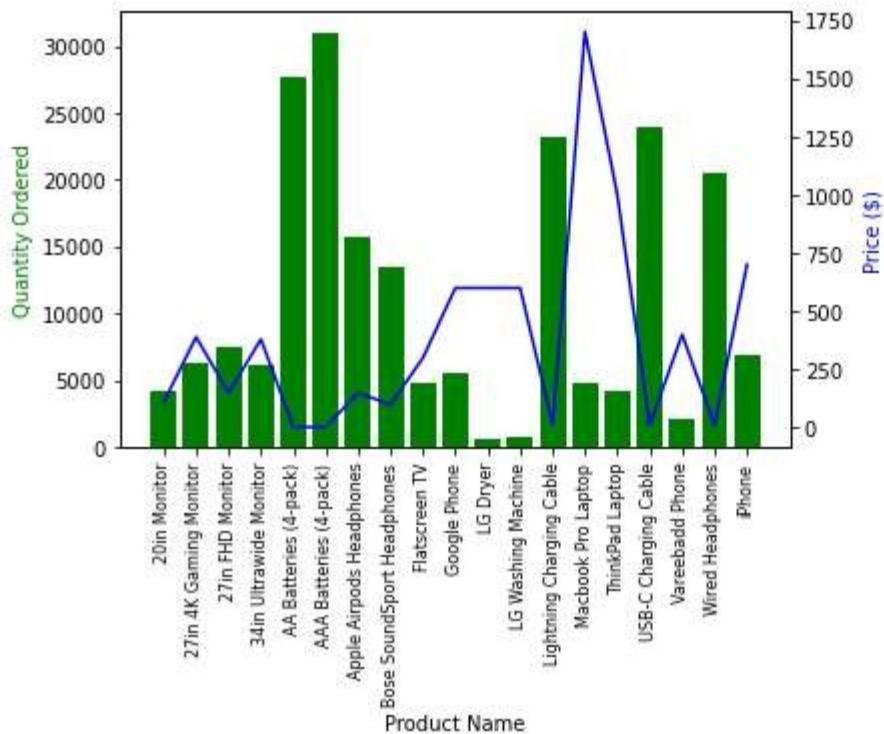


Figure38.OverlayingTheSecondy-axis

Now we will interpret our results. Our hypothesis is true if the high sold products have low price. From the graph we can see it is the case for AAA Batteries and all products except the Macbook Pro Laptop and ThinkPad Laptop. They have decent orders even though they are expensive. We can say that there are many people in the world need laptops. So the laptops are the exception because the laptops have high demand.

CONCLUSION

1. What was the best month for sales? How much was earned that month?

The best month for sales is December. The company earned approximately \$4,810,000.

2. What city sold the most product?

San Francisco is the city with the highest sales.

3. What times should we display advertisements to maximize likelihood of customer's buying products?

We can suggest to advertise the products right before 12PM and/or 7PM. It could be 11.30AM and/or 6.30PM.

4. What Products are most often sold together?

The most often products sold together are iPhone and Lightning Charging Cable with 1005 transactions.

5. What product sold the most? Why do you think it did?

AAA Batteries (4 pack) is the most sold product. Because it's cheap and another products and have high demand.