Google Drive <-> Google Colaboratory

In [1]:

```
# Install the PyDrive wrapper & import libraries.
# This only needs to be done once in a notebook.
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once in a notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
    100% |████████████████████████████████| 993kB 6.2MB/s
  Building wheel for PyDrive (setup.py) ... done
Uploaded file with ID 1xSpkqokICQVUd1bhPnGq_zVlZYJDgmQb
```

In [2]:

```
# Create & upload a text file.
uploaded = drive.CreateFile({'title': 'File2.txt'})
uploaded.SetContentString('Hello World')
uploaded.Upload()
print('Uploaded file with ID {}'.format(uploaded.get('id')))
```

```
Uploaded file with ID 1U0LJtwyr29QpJ6lVeDdwm0JGn1ASRPCJ
```

In [12]:

```
# List .txt files in the root.
#
# Search query reference:
# https://developers.google.com/drive/v2/web/search-parameters
listed = drive.ListFile({'q': "title contains '.csv' and 'root' in parents"}).GetList()
for file in listed:
  print('title {}, id {}'.format(file['title'], file['id']))
```

```
title mobile_cleaned.csv, id 1kE2SbrXuVLZcE0wnTCANZPsuMvej3YGz
```

In [0]:

```
# Download a file based on its file ID.
#
# A file ID looks like: laggVyWshwcyP6kEI-y_W3P8D26sz
file_id = '1kE2SbrXuVLZcE0wnTCANZPsuMvej3YGz'  # https://drive.google.com/open?id=1kE2SbrXuVLZcE0wnTCANZPsuMvej3YGz
downloaded = drive.CreateFile({'id': file_id})
print('Downloaded content "{}"'.format(downloaded.GetContentString()))
```

In [0]:

```
downloaded.GetContentFile('mobile_cleaned_local.csv')
```

In [7]:

```
!ls
```

```
adc.json  mobile_cleaned_local.csv  sample_data
```

In [0]:

```
import pandas as pd
```

Pandas

In [0]:

```
df = pd.read_csv('mobile_cleaned_local.csv')
```

In [11]:

```
df.head()
```

Out[11]:

	sim_type	aperture	gpu_rank	weight	stand_by_time	processor_frequency	thickness	flash_type	front_camera_resolution
0	0	12	55	155.0	250	1.3	10.5	5	2.00
1	0	1	55	132.0	300	1.3	10.6	5	0.30
2	0	9	55	142.0	329	1.5	8.5	5	2.00
3	0	8	55	152.0	385	1.3	8.0	5	2.00
4	1	1	55	234.0	385	1.3	7.9	5	1.92

5 rows × 40 columns

In [13]:

```
df.tail()
```

Out[13]:

	sim_type	aperture	gpu_rank	weight	stand_by_time	processor_frequency	thickness	flash_type	front_camera_resolution
104	3	10	14	192.0	540	1.8	9.4	2	2.0
105	0	5	3	157.0	400	2.3	7.7	5	5.0
106	3	10	6	192.0	384	1.8	7.3	2	5.0
107	3	10	12	129.0	250	1.4	6.9	2	1.2
108	2	8	3	158.0	400	2.2	7.4	6	8.0

5 rows × 40 columns

In [14]:

```
type(df)
```

Out[14]:

```
pandas.core.frame.DataFrame
```

In [0]:

```
dir(df)
```

In [16]:

```
len(df)
```

Out[16]:

```
109
```

In [17]:

```
df.shape
```

```
df.shape
```

Out[17]:

```
(109, 40)
```

In [18]:

```
df.loc[5]
```

Out[18]:

```
sim_type                              0.0
aperture                             14.0
gpu_rank                             55.0
weight                              179.0
stand_by_time                       280.0
processor_frequency                   1.3
thickness                             7.9
flash_type                            5.0
front_camera_resolution               5.0
auto_focus                            3.0
screen_size                           5.5
frames_per_second                    30.0
FM                                    3.0
no_of_reviews_in_gsmarena_in_week     6.0
os                                    0.0
phone_height                        150.0
screen_protection                     5.0
sim_size                              3.0
price                              5999.0
talk_time                            22.0
video_resolution                    720.0
display_resolution                    0.0
removable_battery                     0.0
display_type                          2.0
primary_camera_resolution             8.0
battery_type                          1.0
ram_memory                            1.0
internal_memory                       7.0
brand_rank                            4.0
no_of_cores                           6.0
micro_sd_slot                         4.0
screen_pixel_density                  7.0
water_proof_rate                      3.0
phone_width                          71.0
expandable_memory                    32.0
version                               6.0
usb_type                              3.0
battery_capacity                   2900.0
processor_rank                      165.0
is_liked                              0.0
Name: 5, dtype: float64
```

In [0]:

```
df_short = df[23:29]
```

In [23]:

```
df_short.shape
```

Out[23]:

```
(6, 40)
```

In [24]:

```
df_short.head()
```

Out[24]:

	sim_type	aperture	gpu_rank	weight	stand_by_time	processor_frequency	thickness	flash_type	front_camera_resolution

	sim_type	aperture	gpu_rank	weight	stand_by_time	processor_frequency	thickness	flash_type	front_camera_resolution
23	3	1	43	97.0	345	1.2	5.1	5	5.0
24	0	10	29	150.0	322	1.5	8.2	5	5.0
25	0	8	43	202.0	914	1.2	10.6	2	5.0
26	0	8	43	170.0	456	1.2	10.8	2	5.0
27	0	8	38	155.0	350	1.3	9.3	5	5.0

5 rows × 40 columns

In [0]:

```
df_thin = df[['stand_by_time', 'expandable_memory', 'price', 'battery_capacity', 'is_liked']]
```

In [26]:

```
df_thin.shape
```

Out[26]:

```
(109, 5)
```

In [27]:

```
df_thin.head()
```

Out[27]:

	stand_by_time	expandable_memory	price	battery_capacity	is_liked
0	250	64.0	3870	2000	1
1	300	32.0	4059	2000	1
2	329	32.0	4777	2500	0
3	385	32.0	5799	3000	1
4	385	32.0	5990	3000	0

In [0]:

```
df_liked = df_thin[df_thin['is_liked'] == 1]
```

In [29]:

```
df_liked.shape
```

Out[29]:

```
(92, 5)
```

In [34]:

```
df_thin['price'].describe()
```

Out[34]:

```
count      109.000000
mean     19373.211009
std      14039.197220
min       3870.000000
25%       8999.000000
50%      14614.000000
75%      24999.000000
max      64500.000000
Name: price, dtype: float64
```

In [35]:

```
df_thin.describe()
```

Out[35]:

	stand_by_time	expandable_memory	price	battery_capacity	is_liked
count	109.00000	109.000000	109.000000	109.000000	109.000000
mean	404.66055	104.513761	19373.211009	2841.779817	0.844037
std	176.44206	275.799767	14039.197220	655.003963	0.364496
min	160.00000	0.000000	3870.000000	1560.000000	0.000000
25%	264.00000	0.000000	8999.000000	2470.000000	1.000000
50%	360.00000	32.000000	14614.000000	2900.000000	1.000000
75%	500.00000	128.000000	24999.000000	3100.000000	1.000000
max	1093.00000	2048.000000	64500.000000	5000.000000	1.000000

In [36]:

```
df_thin[df_thin['is_liked'] == 1]['price'].mean()
```

Out[36]:

```
19393.239130434784
```

In [37]:

```
df_thin[df_thin['is_liked'] == 0]['price'].mean()
```

Out[37]:

```
19264.823529411766
```

In [0]:

```
g = df_thin.groupby(['is_liked'])
```

In [39]:

```
for key, df_key in g:
  print(key)
  print(df_key)
```

```
0
     stand_by_time  expandable_memory  price  battery_capacity  is_liked
2              329               32.0   4777              2500         0
4              385               32.0   5990              3000         0
5              280               32.0   5999              2900         0
11             300              128.0   6990              2600         0
22             354              128.0   7999              2400         0
38             490               32.0   9999              2100         0
53             345               64.0  14300              2950         0
60             840                0.0  15689              4100         0
74             390              128.0  21999              2800         0
77             620              128.0  22999              3100         0
78             618                0.0  24499              3600         0
80             598                0.0  24999              3000         0
83             504              200.0  25500              2600         0
90             500                0.0  34999              3760         0
91             240                0.0  34999              1624         0
95             580              256.0  37766              2840         0
108            400                0.0  27999              3000         0
1
     stand_by_time  expandable_memory  price  battery_capacity  is_liked
0              250               64.0   3870              2000         1
1              300               32.0   4059              2000         1
3              385               32.0   5799              3000         1
6              230              128.0   5999              1700         1
7              182               32.0   6599              2000         1
8              182               32.0   6599              2000         1
```

9	435	32.0	6649	3000	1
10	514	32.0	6749	4000	1
12	280	32.0	6999	2500	1
13	198	32.0	6999	2200	1
14	200	32.0	6999	2500	1
15	680	256.0	6999	2500	1
16	576	128.0	7340	2200	1
17	264	32.0	7499	2300	1
18	180	128.0	7590	2000	1
19	160	32.0	7790	2230	1
20	450	32.0	7899	4000	1
21	264	32.0	7914	2900	1
23	345	0.0	8490	2000	1
24	322	32.0	8499	2750	1
25	914	64.0	8999	5000	1
26	456	128.0	8999	3000	1
27	350	64.0	8999	3000	1
28	617	32.0	9399	2100	1
29	775	32.0	9499	3100	1
30	218	128.0	9700	2420	1
31	270	32.0	9715	2900	1
32	250	32.0	9999	2470	1
33	264	0.0	9999	4050	1
34	265	0.0	9999	4050	1
..	...	...	...	...	...
70	170	128.0	19890	2600	1
71	360	0.0	19999	3100	1
72	250	0.0	20397	1560	1
73	687	128.0	21300	2600	1
75	635	200.0	21999	2930	1
76	360	0.0	22999	3300	1
79	180	128.0	24900	2900	1
81	410	128.0	24999	3000	1
82	590	128.0	25500	2930	1
84	250	128.0	27580	2850	1
85	200	128.0	29900	3220	1
86	420	64.0	29990	3200	1
87	410	2048.0	30947	3000	1
88	440	0.0	31999	3450	1
89	354	0.0	33900	2550	1
92	360	0.0	35900	2700	1
93	250	0.0	36499	1810	1
94	250	0.0	36999	1810	1
96	410	32.0	38000	3000	1
97	340	256.0	39890	2900	1
98	362	64.0	40900	2600	1
99	250	0.0	48329	1810	1
100	242	200.0	48900	3000	1
101	240	0.0	49499	1715	1
102	330	0.0	50895	3450	1
103	600	256.0	52699	3430	1
104	540	256.0	54900	3410	1
105	400	200.0	56900	3600	1
106	384	0.0	59000	2750	1
107	250	0.0	64500	1810	1

[92 rows x 5 columns]

In [43]:

```
df_thin.groupby(['is_liked']).describe()
```

Out[43]:

	battery_capacity								expandable_memory		...	price		s
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	c
is_liked														
0	17.0	2874.941176	591.777035	1624.0	2600.0	2900.0	3000.0	4100.0	17.0	68.235294	...	25500.00	37766.0	
1	92.0	2835.652174	668.850998	1560.0	2457.5	2900.0	3100.0	5000.0	92.0	111.217391	...	24924.75	64500.0	

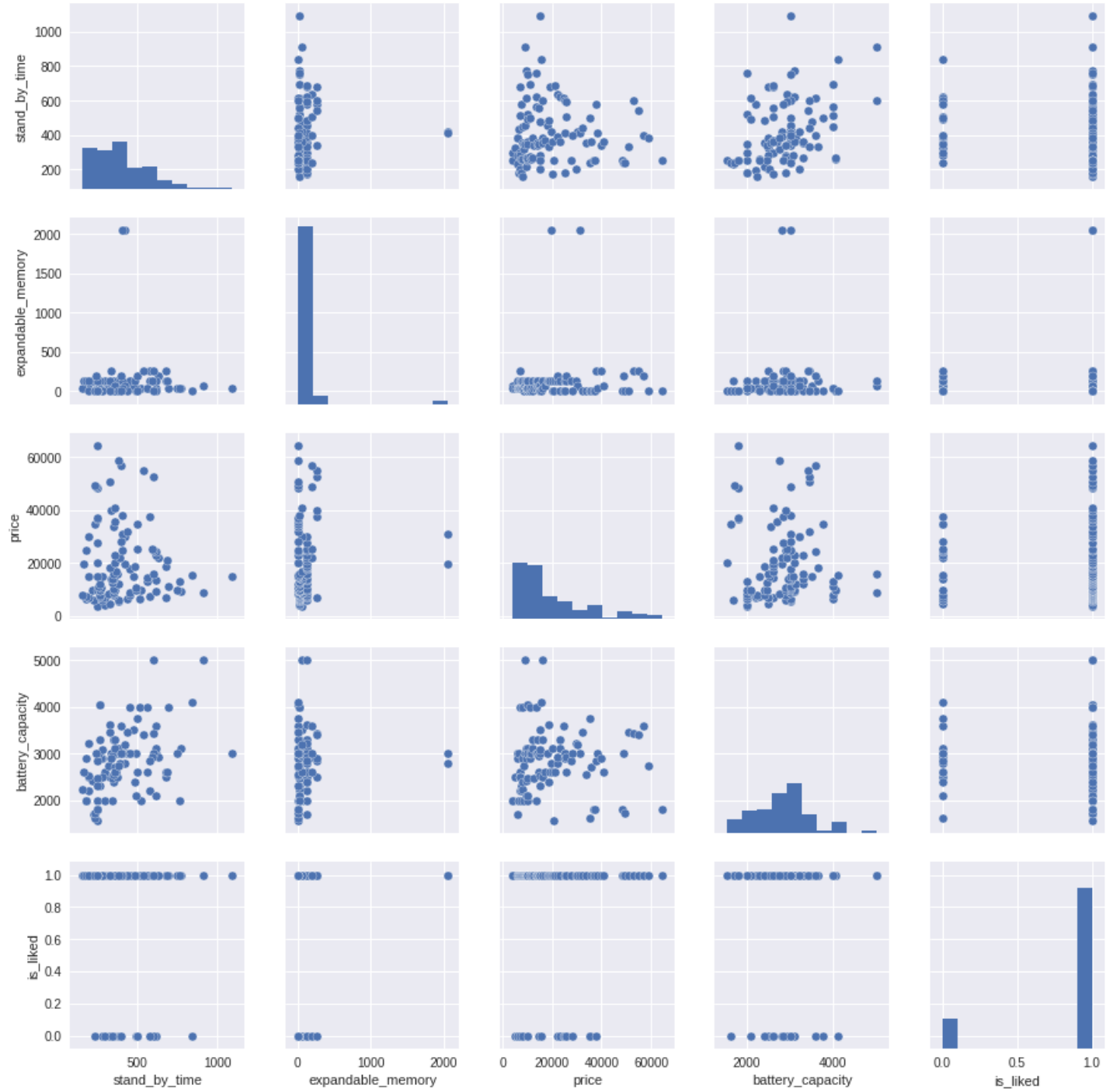
	battery_capacity								expandable_memory		...	price		s
2 rows × 32 columns	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	c

In [0]:

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

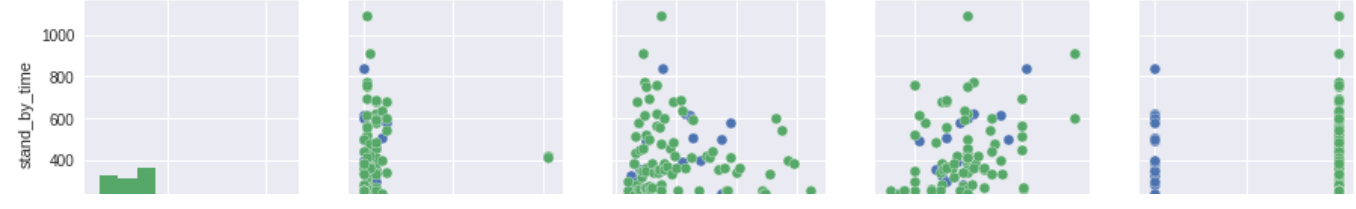
In [45]:

```
ax = sns.pairplot(df_thin, diag_kind='hist')
```



In [46]:

```
ax = sns.pairplot(df_thin, diag_kind='hist', hue='is_liked')
```



Debugging

In [0]:

```
import random
```

In [0]:

```
def factorial(x):
  if (x == 0):
    return 1
  return x * factorial(x - 1)
```

In [49]:

```
factorial(5)
```

Out[49]:

```
120
```

In [0]:

```
def code_to_debug():
  # import pdb; pdb.set_trace()

  for i in range(10):
    x = random.random()
    factorial(x)
```

In [61]:

```
%xmode Verbose
```

```
Exception reporting mode: Verbose
```

In [62]:

```
code_to_debug()
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-62-35361d661c6e> in <module>()
----> 1 code_to_debug()
        global code_to_debug = <function code_to_debug at 0x7fdc1ead88c8>

<ipython-input-59-84611e850098> in code_to_debug()
      4   for i in range(10):
      5     x = random.random()
----> 6     factorial(x)
        global factorial = <function factorial at 0x7fdc1ead89d8>
        x = 0.9542626647624946

<ipython-input-47-6a1d5582b0f3> in factorial(x=0.9542626647624946)
      2   if (x == 0):
      3     return 1
----> 4   return x * factorial(x - 1)
        x = 0.9542626647624946
        global factorial = <function factorial at 0x7fdc1ead89d8>

... last 1 frames repeated, from the frame below ...

<ipython-input-47-6a1d5582b0f3> in factorial(x=-0.04573733523750545)
      2   if (x == 0):
      3     return 1
----> 4   return x * factorial(x - 1)
        x = -0.04573733523750545
        global factorial = <function factorial at 0x7fdc1ead89d8>

RecursionError: maximum recursion depth exceeded in comparison
```

In [0]:

```
def factorial_debugged(x):
  if (not isinstance(x, int)):
    print('This method only supports integers')
    return -1
  if (x == 0):
    return 1
  return x * factorial(x - 1)
```

In [0]:

```
def code_to_debug():
  import pdb; pdb.set_trace()

  for i in range(10):
    x = random.random()
    factorial_debugged(x)
```

In [58]:

```
code_to_debug()
```

```
> <ipython-input-57-3364bd0836cb>(4)code_to_debug()
-> for i in range(10):
(Pdb) ?

Documented commands (type help <topic>):
========================================
EOF    c          d        h         list      q        rv       undisplay
a      cl         debug    help      ll        quit     s        unt
alias  clear      disable  ignore    longlist  r        source   until
args   commands   display  interact  n         restart  step     up
```

```
b      condition  down     j        next      return   tbreak   w
break  cont       enable   jump     p         retval   u        whatis
bt     continue   exit     l        pp        run      unalias  where

Miscellaneous help topics:
==========================
exec  pdb

(Pdb) help c
c(ont(inue))
        Continue execution, only stop when a breakpoint is encountered.
(Pdb) c
This method only supports integers
This method only supports integers
This method only supports integers
This method only supports integers
This method only supports integers
This method only supports integers
This method only supports integers
This method only supports integers
This method only supports integers
This method only supports integers
```

In [0]: