Pandas DataFrame Cheat Sheet

In [1]:
```
import pandas as pd
pd.set_option("display.precision", 4)
pd.set_option("display.max_rows", 15)
```

Create

In [2]:
```
pd.DataFrame({'ColA': [1, 2, 3], 'ColB': [4, 5, 6], 'ColC': [7, 8, 9]})
```

Out[2]:

	ColA	ColB	ColC
0	1	4	7
1	2	5	8
2	3	6	9

In [3]:
```
pd.DataFrame({'ColA': [1, 2, 3], 'ColB': [4, 5, 6], 'ColC': [7, 8, 9]},
             ['a', 'b',' c'])
```

Out[3]:

	ColA	ColB	ColC
a	1	4	7
b	2	5	8
c	3	6	9

In [4]:
```
pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

Out[4]:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

In [5]:
```
ps1 = pd.Series([1, 2, 3])
ps2 = pd.Series([4, 5, 6])
pd.DataFrame({'ColA': ps1, 'ColB': ps2})
```

Out[5]:

	ColA	ColB
0	1	4
1	2	5
2	3	6

In [6]:
```
ps1 = pd.Series([1, 2, 3], [1, 2, 3])
ps2 = pd.Series([1, 2, 3], [2, 1, 3])
```

```
pd.DataFrame({'ColA': ps1, 'ColB': ps2})
# NOTE: Series are re-ordered and matched by Indices!
```

Out[6]:

	ColA	ColB
1	1	2
2	2	1
3	3	3

In [7]:
```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                  index=['a', 'b',' c'],
                  columns=['ColA', 'ColB', 'ColC'])
df
```

Out[7]:

	ColA	ColB	ColC
a	1	2	3
b	4	5	6
c	7	8	9

Rename Column/Index

In [8]:
```
df.rename({'ColC': 'CCC'}, axis='columns')
```

Out[8]:

	ColA	ColB	CCC
a	1	2	3
b	4	5	6
c	7	8	9

In [9]:
```
df.rename({'b': 'B'}, axis='index')
```

Out[9]:

	ColA	ColB	ColC
a	1	2	3
B	4	5	6
c	7	8	9

Assign/Reset Index

In [10]:
```
df.index = ['A', 'B', 'C']
df
```

Out[10]:

	ColA	ColB	ColC
A	1	2	3
B	4	5	6
C	7	8	9

In [11]:
```
df.reset_index()
# Resets index and create new 'index' column to store old indexes
```

Out[11]:

	index	ColA	ColB	ColC
0	A	1	2	3
1	B	4	5	6
2	C	7	8	9

In [12]:
```
df.reset_index(drop=True)
# Dispense with creating 'index' column
```

Out[12]:

	ColA	ColB	ColC
0	1	2	3
1	4	5	6
2	7	8	9

Access Whole Column(s)

In [13]:
```
df
```

Out[13]:

	ColA	ColB	ColC
A	1	2	3
B	4	5	6
C	7	8	9

In [14]:
```
df['ColA']
```

Out[14]:
```
A    1
B    4
C    7
Name: ColA, dtype: int64
```

In [15]:
```
df.ColA
```

Out[15]:
```
A    1
B    4
C    7
Name: ColA, dtype: int64
```

In [16]:
```
df.iloc[:, 0]
```

Out[16]:
```
A    1
B    4
C    7
Name: ColA, dtype: int64
```

In [17]:
```
df.loc[:, 'ColA':'ColB']
```

Out[17]:

	ColA	ColB
A	1	2
B	4	5
C	7	8

In [18]:
```
df[['ColA', 'ColC']]
```

Out[18]:

	ColA	ColC
A	1	3
B	4	6
C	7	9

Access Whole Row(s)

In [19]:
```
df.iloc[0]
```

Out[19]:
```
ColA    1
ColB    2
ColC    3
Name: A, dtype: int64
```

In [20]:
```
df.loc['A']
```

Out[20]:
```
ColA    1
ColB    2
ColC    3
Name: A, dtype: int64
```

In [21]:
```
df.loc['A':'B']
```

Out[21]:

	ColA	ColB	ColC
A	1	2	3
B	4	5	6

In [22]:
```
df[0:2]
# NOTE: Returns first two *ROWS* rather than *COLS*!
```

Out[22]:

	ColA	ColB	ColC
A	1	2	3
B	4	5	6

In [23]:
```
df.loc[df['ColB'] > 4]
```

Out[23]:

	ColA	ColB	ColC
B	4	5	6
C	7	8	9

Access Single Cell

In [24]:
```
df.at['C', 'ColA']  # FASTEST
# df.loc['C', 'ColA']
# df.loc['C']['ColA']
# df['ColA']['C']
```

Out[24]:
```
7
```

In [25]:
```
df.iat[2, 0]  # FASTEST
# df.iloc[2, 0]
# df.iloc[2][0]
```

Out[25]:
```
7
```

In [26]:
```
df.at['C', 'ColA'] = 77
df
```

Out[26]:

	ColA	ColB	ColC
A	1	2	3
B	4	5	6
C	77	8	9

In [27]:
```
df.iat[2, 0] = 7
df
```

Out[27]:

	ColA	ColB	ColC
A	1	2	3
B	4	5	6
C	7	8	9

Access Partial Column

In [28]:
```
df.loc['A':'B', 'ColA']  # FASTEST
# df['ColA']['A':'B']
```

Out[28]:
```
A    1
B    4
Name: ColA, dtype: int64
```

In [29]:
```
df.iloc[0:2]['ColA']  # FASTEST
```

```
# df['ColA'][0:2]
```

Out[29]:
```
A    1
B    4
Name: ColA, dtype: int64
```

In [30]:
```
df.iloc[0:2, 0]
```

Out[30]:
```
A    1
B    4
Name: ColA, dtype: int64
```

In [31]:
```
df['ColA'][[0,2]]
```

Out[31]:
```
A    1
C    7
Name: ColA, dtype: int64
```

Access Partial Row

In [32]:
```
df.loc['A', 'ColA':'ColB']
```

Out[32]:
```
ColA    1
ColB    2
Name: A, dtype: int64
```

In [33]:
```
df.iloc[0]['ColA':'ColB']
```

Out[33]:
```
ColA    1
ColB    2
Name: A, dtype: int64
```

In [34]:
```
df.iloc[0, 0:2]
# df.iloc[0][0:2]
```

Out[34]:
```
ColA    1
ColB    2
Name: A, dtype: int64
```

In [35]:
```
df.iloc[0][[0,2]]
```

Out[35]:
```
ColA    1
ColC    3
Name: A, dtype: int64
```

Access Partial Frame

In [36]:
```
df.iloc[1:, 1:]
```

Out[36]:

	ColB	ColC
B	5	6
C	8	9

In [37]:
```
df.loc['B':'C', 'ColB':'ColC']
```

Out[37]:

	ColB	ColC
B	5	6
C	8	9

Dropping Rows / Columns

In [38]:
```
df
```

Out[38]:

	ColA	ColB	ColC
A	1	2	3
B	4	5	6
C	7	8	9

In [39]:
```
df.drop('A')
```

Out[39]:

	ColA	ColB	ColC
B	4	5	6
C	7	8	9

In [40]:
```
df.drop('ColA', axis=1)
```

Out[40]:

	ColB	ColC
A	2	3
B	5	6
C	8	9

In [41]:
```
df.drop(columns=['ColA', 'ColC'])
# df.drop(['ColA', 'ColC'], axis='columns')
```

Out[41]:

	ColB
A	2
B	5
C	8

Data Exploration

In [42]:
```
df.shape
```

Out[42]:
```
(3, 3)
```

In [43]:
```
df.values
```

Out[43]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [44]:
```
df.columns
```

Out[44]:
```
Index(['ColA', 'ColB', 'ColC'], dtype='object')
```

In [45]:
```
df.columns.values.tolist()
```

Out[45]:
```
['ColA', 'ColB', 'ColC']
```

In [46]:
```
df1 = df.copy()
df1.iat[0, 0] = np.NaN
df1.count()
# Count non-NA cells for each column or row
```

Out[46]:
```
ColA    2
ColB    3
ColC    3
dtype: int64
```

In [47]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3 entries, A to C
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   ColA    3 non-null      int64
 1   ColB    3 non-null      int64
 2   ColC    3 non-null      int64
dtypes: int64(3)
memory usage: 204.0+ bytes
```

In [48]:
```
df.describe()
```

Out[48]:

	ColA	ColB	ColC
count	3.0	3.0	3.0
mean	4.0	5.0	6.0
std	3.0	3.0	3.0
min	1.0	2.0	3.0
25%	2.5	3.5	4.5
50%	4.0	5.0	6.0
75%	5.5	6.5	7.5
max	7.0	8.0	9.0

In [49]:
```
df.agg(('min', 'max'))
```

Out[49]:

	ColA	ColB	ColC
min	1	2	3
max	7	8	9

In [50]:
```
df1 = pd.DataFrame({'ColA': ['one', 'two', 'one']})
df1.describe()
```

Out[50]:

	ColA
count	3
unique	2
top	one
freq	2

In [51]:
```
df['ColA'].value_counts()
# Returns Series containing Count of unique values with index set to values
```

Out[51]:
```
1    1
4    1
7    1
Name: ColA, dtype: int64
```

In [52]:
```
vc = df[['ColA', 'ColC']].value_counts()
vc
# Returns Series containing Count of unique multi-col value combos with index
# set to the combos
```

Out[52]:
```
ColA  ColC
1     3       1
4     6       1
7     9       1
dtype: int64
```

In [53]:
```
df.head(3)
```

Out[53]:

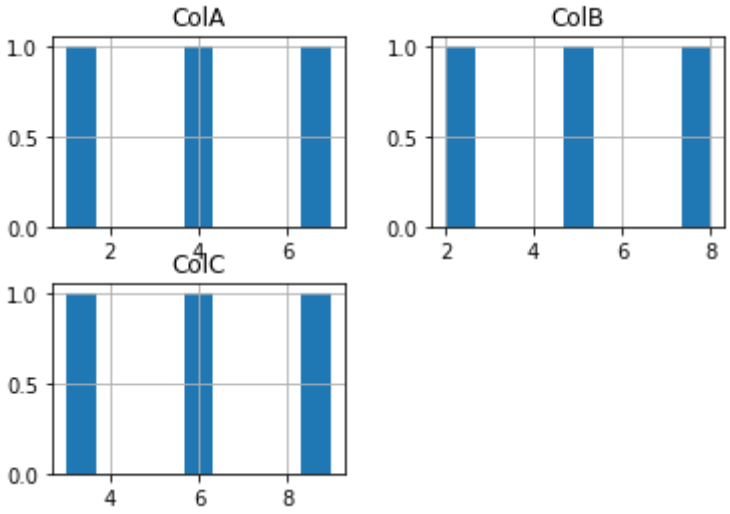
	ColA	ColB	ColC
A	1	2	3
B	4	5	6
C	7	8	9

In [54]:
```
df.tail(2)
```

Out[54]:

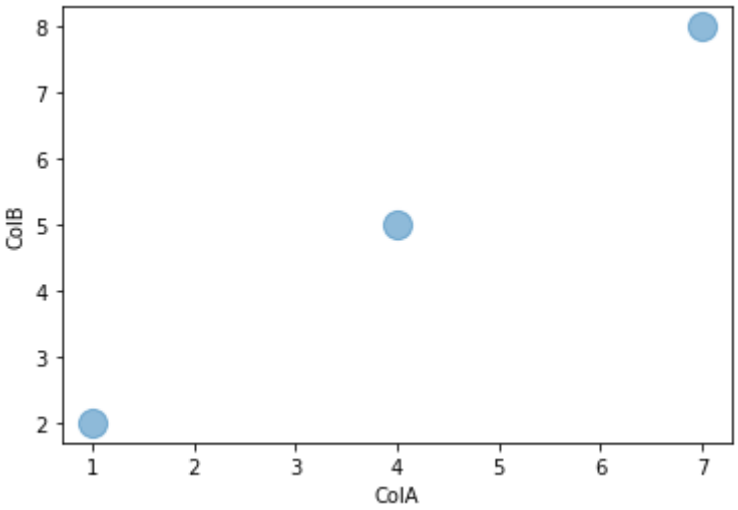
	ColA	ColB	ColC
B	4	5	6
C	7	8	9

In [55]:
```
df.hist(bins=9);
# Histograms of all columns
```



Data Visualization

In [56]:
```
df.plot(kind='scatter', x='ColA', y='ColB', s=200, alpha=0.5);
```

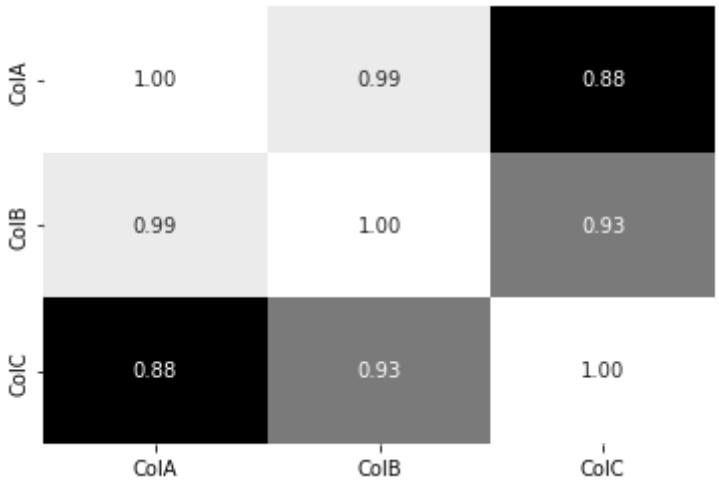


In [57]:
```
df1 = df.applymap(lambda x: np.sin(x))
df1.corr()
```

Out[57]:

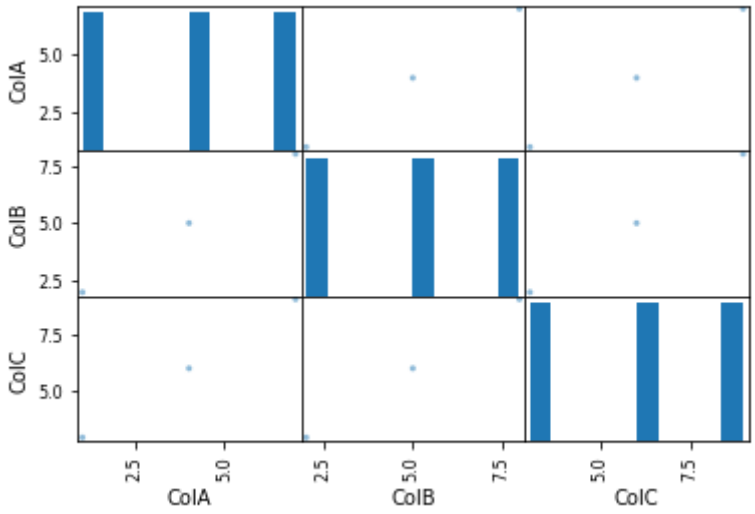
	ColA	ColB	ColC
ColA	1.0000	0.9899	0.8751
ColB	0.9899	1.0000	0.9348
ColC	0.8751	0.9348	1.0000

In [58]:
```
import seaborn as sns
sns.heatmap(abs(df1.corr()), annot=True, fmt=".2f", cmap='gray', cbar=False);
```

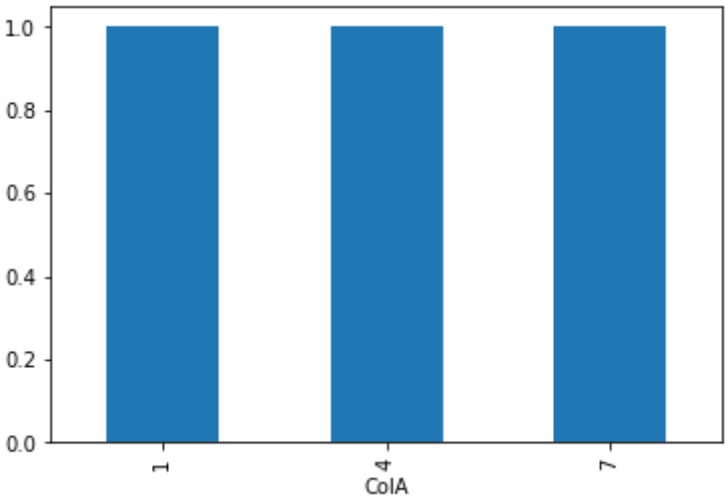


In [59]:
```
from pandas.plotting import scatter_matrix
print(df)
scatter_matrix(df);
# Matrix of scatter plots of all column-pairs
# Histograms plotted on diagonal
```

```
   ColA  ColB  ColC
A     1     2     3
B     4     5     6
C     7     8     9
```



In [60]:
```
df.groupby('ColA').count().iloc[:, 0].plot(kind='bar');
# Chart of value counts of particular column
```



Boolean Indexing

In [61]:
```
df
```

Out[61]:

	ColA	ColB	ColC
A	1	2	3
B	4	5	6
C	7	8	9

In [62]:
```
df['ColA'] > 2
```

Out[62]:
```
A    False
B     True
C     True
Name: ColA, dtype: bool
```

In [63]:
```
df.loc[df['ColA'] > 2, 'ColA']
# df['ColA'][df['ColA'] > 2]
```

Out[63]:
```
B    4
C    7
Name: ColA, dtype: int64
```

In [64]:
```
df > 4
```

Out[64]:

	ColA	ColB	ColC
A	False	False	False
B	False	True	True
C	True	True	True

In [65]:
```
gt4 = df > 4
df[~gt4]
```

Out[65]:

	ColA	ColB	ColC
A	1.0	2.0	3.0
B	4.0	NaN	NaN
C	NaN	NaN	NaN

In [66]:
```
df1 = df.copy()
df1[gt4] = -1
df1
```

Out[66]:

	ColA	ColB	ColC
A	1	2	3
B	4	-1	-1
C	-1	-1	-1

In [67]:
```
df[(df['ColA'] > 1) & (df['ColA'] < 10)]
```

Out[67]:

	ColA	ColB	ColC
B	4	5	6
C	7	8	9

In [68]:
```
df.loc[df['ColA'] > 1]
```

Out[68]:

	ColA	ColB	ColC
B	4	5	6
C	7	8	9

Sorting

In [69]:
```
df
```

Out[69]:

	ColA	ColB	ColC
A	1	2	3
B	4	5	6
C	7	8	9

In [70]:
```
df.sort_values('ColA', ascending=False, inplace=True)
df
```

Out[70]:

	ColA	ColB	ColC
C	7	8	9
B	4	5	6
A	1	2	3

In [71]:
```
df.sort_index(inplace=True)
df
```

Out[71]:

	ColA	ColB	ColC
A	1	2	3
B	4	5	6
C	7	8	9

In [72]:
```
df.rank()
# Compute numerical data ranks (1 - n) along axis, where equal values by
# default are assigned a rank that is the average of ranks of those values
```

Out[72]:

	ColA	ColB	ColC
A	1.0	1.0	1.0
B	2.0	2.0	2.0
C	3.0	3.0	3.0

Computation

In [73]:
```
df
```

Out[73]:

	ColA	ColB	ColC
A	1	2	3
B	4	5	6
C	7	8	9

In [74]:
```
df['ColA'].sum()
```

Out[74]: 12

In [75]:
```
df['ColA'].min()
```

Out[75]: 1

In [76]:
```
df['ColA'].max()
```

Out[76]: 7

In [77]:
```
df['ColA'].mean()
```

Out[77]:
```
4.000
```

In [78]:
```
df['ColA'].median()
```

Out[78]:
```
4.000
```

In [79]:
```
df['ColA'].mode()
# Returns a Series as there may be multiple modes
```

Out[79]:
```
0    1
1    4
2    7
Name: ColA, dtype: int64
```

In [80]:
```
df.cumsum(axis=1)
# Returns a same size DataFrame/Series containing the cumulative sum
```

Out[80]:

	ColA	ColB	ColC
A	1	3	6
B	4	9	15
C	7	15	24

Substitution

In [81]:
```
df['ColA'].map({1:'one', 4:'four'})
# Substituting each value in a Series with another value
```

Out[81]:
```
A     one
B    four
C     NaN
Name: ColA, dtype: object
```

In [82]:
```
df['ColA'].apply(lambda x: x**2)
# Invoke function on values of Series
```

Out[82]:
```
A     1
B    16
C    49
Name: ColA, dtype: int64
```

In [83]:
```
df.applymap(lambda x: x**2)
# Apply a function to a Dataframe elementwise
```

Out[83]:

	ColA	ColB	ColC
A	1	4	9
B	16	25	36
C	49	64	81

In [84]:
```
df.apply(sum)
# Apply a function to a Col / Row(axis=1) Series
```

Out[84]:
```
ColA    12
ColB    15
ColC    18
dtype: int64
```

Grouping

In [85]:
```
df = pd.DataFrame({'ColA': ['X', 'Y', 'X'], 'ColB': [1, 2, 3], 'ColC': [4, 5, 6
df
```

Out[85]:

	ColA	ColB	ColC
0	X	1	4
1	Y	2	5
2	X	3	6

In [86]:
```
df.groupby('ColA').sum()
# Group by unique values of given column(s), then apply an aggregation functior
# to each group, column-by-column
```

Out[86]:

	ColB	ColC
ColA		
X	4	10
Y	2	5

In [87]:
```
df.groupby('ColA')['ColB'].sum()
# df.groupby('ColA').sum()['ColB']
```

Out[87]:
```
ColA
X    4
Y    2
Name: ColB, dtype: int64
```

In [88]:
```
df.groupby('ColA').max()
```

Out[88]:

	ColB	ColC
ColA		
X	3	6
Y	2	5

In [89]:
```
df = pd.DataFrame({'ColA': ['X','X','Y','X'],
                   'ColB': ['Y','Y','Y','Z'],
                   'ColC': [1,2,3,4],
                   'ColD': [5,6,7,8]})
df
```

Out[89]:

	ColA	ColB	ColC	ColD
0	X	Y	1	5
1	X	Y	2	6
2	Y	Y	3	7
3	X	Z	4	8

In [90]:
```
df.groupby(['ColA', 'ColB'])['ColC'].count()
```

Out[90]:
```
ColA  ColB
X     Y       2
      Z       1
Y     Y       1
Name: ColC, dtype: int64
```