

```
import pandas as pd
```

Problem 1: Take the data sets Append_1.csv, and Append_2.csv and append the two sets together. Name the new data set Append.

```
append_1 = pd.read_csv("Append_1.csv", index_col=0)
append_2 = pd.read_csv("Append_2.csv", index_col=0)
```

append_1

	Id	Score
1	78917851	13
2	34554367	77
3	22173883	10

append_2

	Id	Score
1	56993289	72
2	26856261	51
3	33921834	99
4	97613637	63
5	78816868	28
6	67731229	17

```
Append = pd.concat([append_1, append_2])
Append.to_csv("Append.csv")
Append
```

	Id	Score
1	78917851	13
2	34554367	77
3	22173883	10
1	56993289	72
2	26856261	51
3	33921834	99
4	97613637	63
5	78816868	28
6	67731229	17


Problem 2: Take the data sets Merge_1.csv and Merge_2.csv and perform an inner join, left join, right join, full join. Name the resulting data sets, Inner, Left, Right, and Full.

```
merge_1 = pd.read_csv("Merge_1.csv", index_col=0)
merge_2 = pd.read_csv("Merge_2.csv", index_col=0)
```

merge_1


	Id	Score
4	68134933	71
7	22113381	69
9	31937926	98
2	17245265	41
3	42428425	9
10	92922546	67
1	31674694	96

merge_2

	Id	Score	
8	23525437	54	
7	22113381	69	
9	31937926	98	
2	17245265	41	
10	92922546	67	
6	38672872	76	
1	31674694	96	


Inner

```
Inner = pd.merge(merge_1, merge_2, on='Id', how='inner')
Inner
```

	Id	Score_x	Score_y	
0	22113381	69	69	
1	31937926	98	98	
2	17245265	41	41	
3	92922546	67	67	
4	31674694	96	96	


Left

```
Left = pd.merge(merge_1, merge_2, on='Id', how='left')
Left
```

	Id	Score_x	Score_y	
0	68134933	71	NaN	
1	22113381	69	69.0	
2	31937926	98	98.0	
3	17245265	41	41.0	
4	42428425	9	NaN	
5	92922546	67	67.0	
6	31674694	96	96.0	

Right

```
Right = pd.merge(merge_1, merge_2, on='Id', how='right')
Right
```

	Id	Score_x	Score_y	
0	23525437	NaN	54	
1	22113381	69.0	69	
2	31937926	98.0	98	
3	17245265	41.0	41	
4	92922546	67.0	67	
5	38672872	NaN	76	
6	31674694	96.0	96	

Full

```
Full = pd.merge(merge_1, merge_2, on='Id', how='outer')
Full
```

	Id	Score_x	Score_y
0	68134933	71.0	NaN
1	22113381	69.0	69.0
2	31937926	98.0	98.0
3	17245265	41.0	41.0
4	42428425	9.0	NaN
5	92922546	67.0	67.0



Problem 3. Take the Filter.csv data set, and filter the data so that the new data set has

(i) only rows where Id is a vowel and (ii) only columns where the column means of the original data are positive. Name the new dataset Vowels.

```
Filter = pd.read_csv("Filter.csv", index_col=0)
Filter
```

	Id	V1	V2	V3	V4	V5	V6	V7	V8	V9	
1	a	-3.131767	-1.296154	-0.840714	-0.511305	-0.276927	-0.030081	0.228564	0.509134	0.816408	1.209
2	a	-2.753743	-1.293420	-0.830057	-0.510220	-0.271765	-0.029370	0.228574	0.510424	0.817482	1.215
3	a	-2.717051	-1.288645	-0.828611	-0.509153	-0.271128	-0.027879	0.230532	0.511642	0.818404	1.224
4	a	-2.475701	-1.279962	-0.824249	-0.499681	-0.266096	-0.027428	0.235311	0.515277	0.819011	1.225
5	b	-2.450370	-1.267123	-0.822122	-0.498173	-0.264964	-0.019237	0.237964	0.515872	0.821571	1.236
...
96	x	-1.325443	-0.861595	-0.522726	-0.285047	-0.043476	0.221000	0.493902	0.801205	1.195280	2.519
97	x	-1.312452	-0.853865	-0.518978	-0.283813	-0.043063	0.221481	0.498393	0.801779	1.195526	2.535
98	y	-1.300866	-0.850192	-0.517301	-0.283729	-0.038558	0.225658	0.498915	0.809598	1.199147	2.562
99	z	-1.300555	-0.844371	-0.517238	-0.280365	-0.033863	0.225745	0.498943	0.809836	1.204194	2.583
100	z	-1.299469	-0.840888	-0.513931	-0.280236	-0.033170	0.226953	0.508562	0.812575	1.208722	2.594

100 rows × 11 columns

```
Filter["Id"].unique()

array(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
       'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'],
      dtype=object)
```

only rows where Id is a vowel

```
Vowels = Filter[(Filter["Id"] == 'a') | (Filter["Id"] == 'e') | (Filter["Id"] == 'i') | (Filter["Id"] == 'o') | (Filter["Id"] == 'u')]
Vowels
```

	Id	V1	V2	V3	V4	V5	V6	V7	V8	V9	V
1	a	-3.131767	-1.296154	-0.840714	-0.511305	-0.276927	-0.030081	0.228564	0.509134	0.816408	1.2097
2	a	-2.753743	-1.293420	-0.830057	-0.510220	-0.271765	-0.029370	0.228574	0.510424	0.817482	1.2154
3	a	-2.717051	-1.288645	-0.828611	-0.509153	-0.271128	-0.027879	0.230532	0.511642	0.818404	1.2247
4	a	-2.475701	-1.279962	-0.824249	-0.499681	-0.266096	-0.027428	0.235311	0.515277	0.819011	1.2253
21	e	-2.071032	-1.167847	-0.760093	-0.461729	-0.227863	0.033444	0.284396	0.559895	0.871592	1.3543
22	e	-2.063066	-1.158616	-0.757211	-0.457241	-0.225684	0.035520	0.289390	0.560585	0.873613	1.3598
35	i	-1.846590	-1.104803	-0.727083	-0.410712	-0.201916	0.072672	0.326803	0.603491	0.936952	1.4611
36	i	-1.816983	-1.102528	-0.722621	-0.410241	-0.199493	0.073014	0.330990	0.613375	0.937777	1.4698
55	o	-1.625952	-1.020138	-0.655135	-0.371713	-0.143934	0.140601	0.395279	0.662332	1.025431	1.6343
56	o	-1.624472	-1.011244	-0.652875	-0.371476	-0.143122	0.146144	0.395987	0.664093	1.032519	1.6354
57	o	-1.623765	-1.007926	-0.650770	-0.371241	-0.142016	0.149244	0.400418	0.664547	1.037078	1.6370
58	o	-1.613101	-1.006028	-0.643169	-0.370309	-0.140304	0.151200	0.403982	0.665422	1.039321	1.6455
83	u	-1.402569	-0.908152	-0.561200	-0.313108	-0.068641	0.193865	0.465258	0.769225	1.135336	2.1365
84	u	-1.399668	-0.906932	-0.560007	-0.310695	-0.064179	0.203735	0.466399	0.771325	1.138102	2.2047
85	u	-1.398094	-0.904328	-0.553800	-0.307978	-0.061697	0.204181	0.467515	0.782652	1.139875	2.2058

```
Vowels.mean()
```

```
<ipython-input-115-cd15463a6dea>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated;
Vowels.mean()
V1    -1.970904
V2    -1.097115
V3    -0.704506
V4    -0.412453
V5    -0.180318
V6     0.085924
V7     0.343293
V8     0.624228
V9     0.962593
V10    1.574673
dtype: float64
```

only columns where the column means of the original data are positive

```
index = ["Id"] + list(Vowels.mean()[Vowels.mean() > 0].index)
Vowels = Vowels[index]
Vowels
```

```
<ipython-input-116-eab92df07757>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions
index = ["Id"] + list(Vowels.mean()[Vowels.mean() > 0].index)
```

		Id	V6	V7	V8	V9	V10	
1	a	-0.030081	0.228564	0.509134	0.816408	1.209791		
2	a	-0.029370	0.228574	0.510424	0.817482	1.215493		
3	a	-0.027879	0.230532	0.511642	0.818404	1.224739		
4	a	-0.027428	0.235311	0.515277	0.819011	1.225300		
21	e	0.033444	0.284396	0.559895	0.871592	1.354335		
22	e	0.035520	0.289390	0.560585	0.873613	1.359896		
35	i	0.072672	0.326803	0.603491	0.936952	1.461178		
36	i	0.073014	0.330990	0.613375	0.937777	1.469812		
55	o	0.140601	0.395279	0.662332	1.025431	1.634369		
56	o	0.146144	0.395987	0.664093	1.032519	1.635442		
57	o	0.149244	0.400418	0.664547	1.037078	1.637068		
58	o	0.151200	0.403982	0.665422	1.039321	1.645503		
83	u	0.193865	0.465258	0.769225	1.135336	2.136592		
84	u	0.203735	0.466399	0.771325	1.138102	2.204738		
85	u	0.204181	0.467515	0.782652	1.139875	2.205843		

Problem 4. Take the Filter.csv dataset, and take a simple random sample of the data with ten rows. Keep all columns. Name the new dataset SRS_Filter.

```
SRS_Filter = Filter.sample(n=10)
SRS_Filter
```

		Id	V1	V2	V3	V4	V5	V6	V7	V8	V9
70	r	-1.512410	-0.962193	-0.594802	-0.339151	-0.107537	0.169531	0.435238	0.730242	1.094578	
49	l	-1.656177	-1.046861	-0.674250	-0.380643	-0.163883	0.121138	0.371061	0.648888	0.994604	
14	d	-2.194228	-1.220047	-0.786685	-0.484320	-0.244800	0.012640	0.268868	0.535639	0.850074	
19	d	-2.093890	-1.176263	-0.763212	-0.469929	-0.231471	0.027452	0.273076	0.546232	0.862737	
29	g	-1.907103	-1.127302	-0.746737	-0.433587	-0.212005	0.051428	0.307884	0.585474	0.910425	
99	z	-1.300555	-0.844371	-0.517238	-0.280365	-0.033863	0.225745	0.498943	0.809836	1.204194	
11	c	-2.248656	-1.228823	-0.796115	-0.487631	-0.252037	0.004767	0.255484	0.529901	0.840513	
72	s	-1.507544	-0.958284	-0.586552	-0.338557	-0.106609	0.171311	0.439231	0.734304	1.101666	
47	l	-1.674005	-1.057892	-0.680548	-0.386315	-0.167965	0.113935	0.356816	0.645855	0.987294	
83	u	-1.402569	-0.908152	-0.561200	-0.313108	-0.068641	0.193865	0.465258	0.769225	1.135336	

Problem 5. Randomly partition the Filter.csv data set into three subsets:

Train (70% of your data), Validation (15% of your data), Test (15% of your data). For each of the three subsets, print the first three rows and the dimensions of the data set.

This can be easily done using scikit learn's train_test split.

```
from sklearn.model_selection import train_test_split

train, rest = train_test_split(Filter, train_size=0.7)
val, test = train_test_split(rest, test_size=0.5)
```

Train

```
train.head(3)
```

	Id	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
34	h	-1.848870	-1.108606	-0.728289	-0.4111311	-0.204454	0.072663	0.326538	0.597154	0.93585	0.93585
2	a	-2.753743	-1.293420	-0.830057	-0.510220	-0.271765	-0.029370	0.228574	0.510424	0.81748	0.81748
52	m	-1.641384	-1.032022	-0.670981	-0.375518	-0.162222	0.126508	0.384024	0.652923	1.02186	1.02186

```
train.shape
```

(70, 11)

Validation

```
val.head(3)
```

	Id	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
62	p	-1.583904	-0.989101	-0.629909	-0.365573	-0.123204	0.156192	0.414330	0.692836	1.051478	1.69469
49	l	-1.656177	-1.046861	-0.674250	-0.380643	-0.163883	0.121138	0.371061	0.648888	0.994604	1.59210
90	w	-1.362826	-0.887245	-0.542298	-0.293752	-0.056517	0.212445	0.482841	0.793072	1.170550	2.29813

```
val.shape
```

(15, 11)

Test

```
test.head(3)
```

	Id	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
24	f	-1.977538	-1.145779	-0.750031	-0.453491	-0.222002	0.036691	0.295670	0.568302	0.890023	1.37587
76	s	-1.460825	-0.941320	-0.577046	-0.324442	-0.101695	0.177586	0.456526	0.750048	1.123588	1.88879
20	d	-2.076374	-1.175226	-0.762930	-0.465575	-0.231032	0.027937	0.284120	0.556447	0.869824	1.34826

```
test.shape
```

(15, 11)