```
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
6	31674694	96.0	96.0
7	23525437	NaN	54.0
8	38672872	NaN	76.0

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	
1	a	-3.131767	-1.296154	-0.840714	-0.511305	-0.276927	-0.030081	0.228564	0.5091
2	a	-2.753743	-1.293420	-0.830057	-0.510220	-0.271765	-0.029370	0.228574	0.5104
3	a	-2.717051	-1.288645	-0.828611	-0.509153	-0.271128	-0.027879	0.230532	0.5116
4	a	-2.475701	-1.279962	-0.824249	-0.499681	-0.266096	-0.027428	0.235311	0.5152
5	b	-2.450370	-1.267123	-0.822122	-0.498173	-0.264964	-0.019237	0.237964	0.5158
...	...	...	...	...	...	...	...	...	
96	x	-1.325443	-0.861595	-0.522726	-0.285047	-0.043476	0.221000	0.493902	0.8012
97	x	-1.312452	-0.853865	-0.518978	-0.283813	-0.043063	0.221481	0.498393	0.8017
98	y	-1.300866	-0.850192	-0.517301	-0.283729	-0.038558	0.225658	0.498915	0.8095
99	z	-1.300555	-0.844371	-0.517238	-0.280365	-0.033863	0.225745	0.498943	0.8098
100	z	-1.299469	-0.840888	-0.513931	-0.280236	-0.033170	0.226953	0.508562	0.8125

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	V
1	a	-3.131767	-1.296154	-0.840714	-0.511305	-0.276927	-0.030081	0.228564	0.50913
2	a	-2.753743	-1.293420	-0.830057	-0.510220	-0.271765	-0.029370	0.228574	0.51042
3	a	-2.717051	-1.288645	-0.828611	-0.509153	-0.271128	-0.027879	0.230532	0.51164
4	a	-2.475701	-1.279962	-0.824249	-0.499681	-0.266096	-0.027428	0.235311	0.51527

```
Vowels.mean()
```

```
<ipython-input-18-cd15463a6dea>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') i
  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-19-eab92df07757>:1: FutureWarning: Dropping of nuisance columns in Dat
  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	V
41	j	-1.768754	-1.084088	-0.709889	-0.403803	-0.181488	0.091924	0.341611	0.61902
93	x	-1.337392	-0.871296	-0.529506	-0.293274	-0.050818	0.214479	0.488918	0.79720
94	x	-1.334215	-0.868354	-0.527451	-0.288823	-0.050732	0.217681	0.490113	0.79983
97	x	-1.312452	-0.853865	-0.518978	-0.283813	-0.043063	0.221481	0.498393	0.80177

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
18	d	-2.102028	-1.178832	-0.765685	-0.476587	-0.234067	0.020574	0.272390	0.54453
74	s	-1.463690	-0.948343	-0.585414	-0.334283	-0.104285	0.173921	0.442712	0.74083
95	x	-1.330359	-0.868267	-0.523089	-0.287184	-0.045273	0.218168	0.491043	0.80056

```
train.shape
```

```
(70, 11)
```

Validation

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

	Id	V1	V2	V3	V4	V5	V6	V7	V
7	b	-2.347337	-1.260646	-0.814551	-0.494618	-0.257148	-0.002708	0.243666	0.52176
64	q	-1.543357	-0.983950	-0.627542	-0.360100	-0.118509	0.158597	0.424618	0.70891
44	k	-1.718103	-1.077033	-0.705715	-0.396122	-0.172683	0.105170	0.347219	0.63401

```
val.shape
```

```
(15, 11)
```

Test

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

	Id	V1	V2	V3	V4	V5	V6	V7	V8
51	m	-1.643792	-1.039570	-0.672395	-0.377073	-0.163041	0.124118	0.375872	0.65225
86	v	-1.379663	-0.899883	-0.553627	-0.304022	-0.060937	0.204793	0.473697	0.78497
21	e	-2.071032	-1.167847	-0.760093	-0.461729	-0.227863	0.033444	0.284396	0.55989

```
test.shape
```

```
(15, 11)
```

✓ 0s completed at 10:45 ● ✕