Panda’s function

Pandas is a predominantly used python data analysis library. It provides many functions and methods to expedite the data analysis process. What makes pandas so common is its functionality, flexibility, and simple syntax.

Query

Example:

```
df.query('value_1 < value_2')
```

Insert

```
new_col = np.random.randn(10)
df.insert(2, 'new_col', new_col)
```

Cumsum

```
df['cumsum_2'] = df[['value_2','group']].groupby('group').cumsum()df
```

4. Sample

```
sample1 = df.sample(n=3)
sample1
```

```
sample2 = df.sample(frac=0.5)
sample2
```

Where

```
df['new_col'].where(df['new_col'] > 0 , 0)
```

```
df['new_col'].where(df['new_col'] > 0 , 0)np.where(df['new_col'] > 0, df['new_col'], 0)
```

6. Isin

```
years = ['2010','2014','2017']
df[df.year.isin(years)]
```

	group	year	new_col	value_1	value_2	cumsum_2
0	A	2010	0.332581	2	3	3
4	B	2014	0.078729	6	1	2
7	A	2017	-0.161005	8	5	15

7. Loc and iloc

```
df.iloc[:3, :2]
```

	group	year
0	A	2010
1	A	2011
2	B	2012

Selecting first 3 rows and first 2 columns with loc:

```
df.loc[:2, ['group','year']]
```

	group	year
0	A	2010
1	A	2011
2	B	2012

```
df.loc[[1,3,5], ['year','value_1']]
```

	year	value_1
1	2011	3
3	2013	9
5	2015	7

8. Pct_change

df.value_1.pct_change()

9. Rank

df['rank_1'] = df['value_1'].rank()
df

10. Melt

df_wide.melt(id_vars=['city'])

11. Explode

rows.

	ID	measurement	day
0	a	4	1
1	b	6	1
2	c	[2, 3, 8]	1

df1.explode('measurement').reset_index(drop=True)

12. Nunique

df.year.nunique()
10df.group.nunique()
3

```
df.nunique()
```

```
group        3
year        10
new_col     10
value_1      9
value_2      7
cumsum_2     9
rank_1       9
dtype: int64
```

13. Lookup

df['Person_point'] = df.lookup(df.index, df['Person'])
df

14. Infer_objects

	A	B	C	D
0	1	2.1	True	b
1	2	1.5	False	c
2	3	2	False	d
3	4	2.1	True	f

df2.dtypes
A object
B object
C object
D object
dtype: object

15. Memory_usage

df_large = pd.DataFrame({'A': np.random.randn(1000000),
'B': np.random.randint(100, size=1000000)})df_large.shape
(1000000, 2)

16. Describe

```
df.describe()
```

	year	new_col	value_1	value_2	cumsum_2	rank_1
count	10.00000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	2014.50000	-0.173626	4.800000	4.300000	7.500000	5.500000
std	3.02765	0.669093	2.780887	2.945807	5.681354	3.018462
min	2010.00000	-1.515015	0.000000	1.000000	1.000000	1.000000
25%	2012.25000	-0.322054	3.250000	2.250000	2.250000	3.375000
50%	2014.50000	0.096223	4.500000	3.500000	7.000000	5.250000
75%	2016.75000	0.237182	6.750000	5.750000	10.750000	7.750000
max	2019.00000	0.551692	9.000000	9.000000	17.000000	10.000000

17. Merge

		column_a	column_b	column_c
df1	0	1	a	True
	1	2	b	True
	2	3	c	False
	3	4	d	True
df2	0	1	a	False
	1	2	k	False
	2	9	l	False
	3	10	m	True

```
df_merge = pd.merge(df1, df2, on='column_a')
df_merge
```

	column_a	column_b_x	column_c_x	column_b_y	column_c_y
0	1	a	True	a	False
1	2	b	True	k	False

18. Select_dtypes

df.select_dtypes(include='int64')

	year	value_1	value_2	cumsum_2
0	2010	2	3	3
1	2011	3	3	6
2	2012	5	1	1
3	2013	9	4	10
4	2014	6	1	2
5	2015	7	9	11
6	2016	4	2	2
7	2017	8	5	15
8	2018	4	6	8
9	2019	0	9	17

df.select_dtypes(exclude='int64')

	group	new_col	rank_1
0	A	0.332581	2.0
1	A	0.122312	3.0
2	B	0.551692	6.0
3	A	-1.515015	10.0
4	B	0.078729	7.0
5	B	-0.375737	8.0
6	C	-1.159010	4.5
7	A	-0.161005	9.0
8	C	0.275472	4.5
9	C	0.113718	1.0

19. Replace

df.replace('A', 'A_1')

	group	year	new_col	value_1	value_2	cumsum_2	rank_1
0	A_1	2010	0.332581	2	3	3	2.0
1	A_1	2011	0.122312	3	3	6	3.0
2	B	2012	0.551692	5	1	1	6.0
3	A_1	2013	-1.515015	9	4	10	10.0
4	B	2014	0.078729	6	1	2	7.0
5	B	2015	-0.375737	7	9	11	8.0
6	C	2016	-1.159010	4	2	2	4.5
7	A_1	2017	-0.161005	8	5	15	9.0
8	C	2018	0.275472	4	6	8	4.5
9	C	2019	0.113718	0	9	17	1.0

df.replace({'A':'A_1', 'B':'B_1'})

	group	year	new_col	value_1	value_2	cumsum_2	rank_1
0	A_1	2010	0.332581	2	3	3	2.0
1	A_1	2011	0.122312	3	3	6	3.0
2	B_1	2012	0.551692	5	1	1	6.0
3	A_1	2013	-1.515015	9	4	10	10.0
4	B_1	2014	0.078729	6	1	2	7.0
5	B_1	2015	-0.375737	7	9	11	8.0
6	C	2016	-1.159010	4	2	2	4.5
7	A_1	2017	-0.161005	8	5	15	9.0
8	C	2018	0.275472	4	6	8	4.5
9	C	2019	0.113718	0	9	17	1.0

20. Applymap

```
def color_negative_values(val):
  color = 'red' if val < 0 else 'black'
  return 'color: %s' % color
```

Numpy Functions.

all ()

numpy.all(array,

axis = None,

out= None,

keepdims = class numpy._globals._NoValue at 0x40ba726c)

any()

numpy.any(a,

axis = None,

out = None,

keepdims = class numpy._globals._NoValue at 0x40ba726c)

take()

numpy.take(array, indices, axis = None, out = None, mode ='raise')

put()

numpy.put(array, indices, p_array, mode = 'raise')

apply_along_axis()

numpy.apply_along_axis(1d_func, axis, array, *args, **kwargs)

apply_over_axes()

numpy.apply_over_axes(func, array, axes)

argmin()

numpy.argmin(array, axis = None, out = None)

argmax()

numpy.argmax(array, axis = None, out = None)

nanargmin()

numpy.nanargmin(array, axis = None)

arange()

arange([start,] stop[, step,][, dtype])

place()

numpy.place(array, mask, vals)

extract()

numpy.extract(condition, array)

compress()

numpy.compress(condition, array, axis = None, out = None)

rot90()

rot90(array, k = 1, axes = (0, 1))

tile()

numpy.tile(arr, repetitions)

reshape()

numpy.tile(arr, repetitions)

ravel()

numpy.ravel(array, order = 'C')

isinf()

numpy.isinf(array [, out])

dot()

dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])

vdot()

vector_a = 2 + 3j

vector_b = 4 + 5j

```
product = geek.vdot(vector_a,
vector_b)

print("Dot Product  : ",
product)
```