Assignment 05

December 2, 2019

[1]:
```
#01
import pandas as pd
```

[2]:
```
#01
myDF =pd.read_csv('test1.csv')
```

[3]:
```
#01
myDF.head()
```

[3]:

	ID	Range	Value	Type	Zip Code	Model	Cost	Continent
0	10	192	50	F	NXEYEM	EX85	[56$]	Asia
1	20	299	21	M	VNFKZD	CT41	[18$]	Asia
2	30	378	28	X	IMGMFY	VN81	[90$]	North America
3	40	156	44	Q	RHRHJY	IR14	[89$]	Asia
4	50	468	33	J	HPQCMC	QX90	[95$]	Africa

[4]:
```
#02
myDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 950 entries, 0 to 949
Data columns (total 8 columns):
ID           950 non-null int64
Range        950 non-null int64
Value        950 non-null int64
Type         948 non-null object
Zip Code     941 non-null object
Model        950 non-null object
Cost         950 non-null object
Continent    950 non-null object
dtypes: int64(3), object(5)
memory usage: 59.5+ KB
```

[5]:
```
#02
myDF.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 950 entries, 0 to 949
```

```
Data columns (total 8 columns):
ID           950 non-null int64
Range        950 non-null int64
Value        950 non-null int64
Type         948 non-null object
Zip Code     941 non-null object
Model        950 non-null object
Cost         950 non-null object
Continent    950 non-null object
dtypes: int64(3), object(5)
memory usage: 313.1 KB
```

[6]:
```
#03
myDF.memory_usage()
```

[6]:
```
Index         128
ID           7600
Range        7600
Value        7600
Type         7600
Zip Code     7600
Model        7600
Cost         7600
Continent    7600
dtype: int64
```

[7]:
```
#03
myDF.memory_usage(deep=True)
```

[7]:
```
Index         128
ID           7600
Range        7600
Value        7600
Type        58840
Zip Code    59571
Model       57950
Cost        58900
Continent   62431
dtype: int64
```

[8]:
```
#04
myDF['Continent'] = myDF.Continent.astype('category')
```

[9]:
```
#04
myDF.memory_usage(deep=True)
```

[9]:
```
Index         128
ID           7600
Range        7600
Value        7600
```

```
Type          58840
Zip Code      59571
Model         57950
Cost          58900
Continent      1730
dtype: int64
```

[10]:
```
#05
myDF.Continent.cat.codes.head()
```

[10]:
```
0    2
1    2
2    5
3    2
4    0
dtype: int8
```

[11]:
```
#07
myDF =pd.read_csv('test1.csv')
```

[12]:
```
#07
myDF.head()
```

[12]:

	ID	Range	Value	Type	Zip Code	Model	Cost	Continent
0	10	192	50	F	NXEYEM	EX85	[56$]	Asia
1	20	299	21	M	VNFKZD	CT41	[18$]	Asia
2	30	378	28	X	IMGMFY	VN81	[90$]	North America
3	40	156	44	Q	RHRHJY	IR14	[89$]	Asia
4	50	468	33	J	HPQCMC	QX90	[95$]	Africa

[13]:
```
#07
pd.get_dummies(myDF, columns=['Continent'], drop_first=True)
```

[13]:

	ID	Range	Value	Type	Zip Code	Model	Cost	Continent_Antarctica	\
0	10	192	50	F	NXEYEM	EX85	[56$]	0	
1	20	299	21	M	VNFKZD	CT41	[18$]	0	
2	30	378	28	X	IMGMFY	VN81	[90$]	0	
3	40	156	44	Q	RHRHJY	IR14	[89$]	0	
4	50	468	33	J	HPQCMC	QX90	[95$]	0	
..	...	...	...	...	...	...	...	...	
945	9460	136	12	C	HGVBPV	NG88	[51$]	0	
946	9470	316	46	L	SHBAUD	IE12	[33$]	0	
947	9480	461	46	K	PUZUVO	HB82	[12$]	0	
948	9490	185	13	N	UMESZO	WJ90	[39$]	0	
949	9500	137	10	W	JOHHVL	QE45	[10$]	0	

	Continent_Asia	Continent_Australia	Continent_Europe	\
0	1	0	0	
1	1	0	0	
2	0	0	0	

```
3                    1                      0                    0
4                    0                      0                    0
..                 ...                    ...                  ...
945                  0                      0                    0
946                  0                      0                    0
947                  0                      0                    1
948                  0                      0                    0
949                  0                      0                    0

     Continent_North America  Continent_South America
0                          0                        0
1                          0                        0
2                          1                        0
3                          0                        0
4                          0                        0
..                       ...                      ...
945                        0                        1
946                        1                        0
947                        0                        0
948                        1                        0
949                        0                        1

[950 rows x 13 columns]
```

[14]:
```
#08
mySchool = pd.read_csv('test2.csv')
```

[15]:
```
#08
mySchool.head()
```

[15]:
```
   ID  Math  Physics Result              Time
0  10    82       88   Pass   2010-12-14 9:01
1  20    63       62   Pass  2016-07-26 20:02
2  30    59       53   Pass  2010-04-15 18:25
3  40    26       79   Fail  2010-09-10 22:48
4  50    39       96   Fail  2017-03-10 22:02
```

[16]:
```
#09
train = mySchool.sample(frac=0.70)
```

[17]:
```
#09
train.to_csv('train.csv')
```

[18]:
```
#09
test = mySchool.loc[~mySchool.index.isin(train.index), :]
test.to_csv('test.csv')
```

[19]:
```
#10
feature_cols = ['Math', 'Physics']
```

[20]:
```
#10
X = train.loc[:, feature_cols]
```

[21]:
```
#11
Y= train.Result
```

[22]:
```
#12
from sklearn.linear_model import LogisticRegression
```

[23]:
```
#12
logreg = LogisticRegression(solver='lbfgs')
```

[24]:
```
#12
logreg.fit(X,Y)
```

[24]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[25]:
```
#13
X_test = test.loc[:, feature_cols]
```

[26]:
```
#13
Y_actual_test=test.loc[:,'Result']
```

[27]:
```
#14
Y_pred_test = logreg.predict(X_test)
```

[28]:
```
#15
Y_pred_test
```

[28]:
```
array(['Pass', 'Fail', 'Fail', 'Pass', 'Fail', 'Pass', 'Fail', 'Pass',
       'Fail', 'Fail', 'Fail', 'Fail', 'Fail', 'Fail', 'Fail', 'Fail',
       'Fail', 'Fail', 'Fail', 'Fail', 'Pass', 'Fail', 'Fail', 'Fail',
       'Pass', 'Pass', 'Fail', 'Fail', 'Fail', 'Pass'], dtype=object)
```

[29]:
```
#15
Y_actual_test
```

[29]:
```
6     Pass
9     Fail
11    Fail
12    Pass
13    Fail
14    Fail
15    Fail
16    Pass
21    Fail
22    Fail
28    Fail
29    Fail
```

```
31    Fail
35    Fail
37    Fail
39    Fail
52    Fail
55    Fail
58    Pass
62    Fail
65    Pass
66    Fail
70    Fail
74    Fail
77    Pass
81    Pass
84    Fail
92    Fail
93    Fail
94    Pass
Name: Result, dtype: object
```

[30]:
```
#df = pd.concat([Y_actual_test,pd.DataFrame(Y_pred_test)], axis=1)
```

[31]:
```
#df.shape
```

[32]:
```
#17
train.to_pickle('train.pkl')
```

[33]:
```
#17
test.to_pickle('test.pkl')
```

[34]:
```
#18
mySchool['Time'] = pd.to_datetime(mySchool.Time)
```

[35]:
```
#18
mySchool.dtypes
```

[35]:
```
ID                  int64
Math                int64
Physics             int64
Result             object
Time       datetime64[ns]
dtype: object
```

[36]:
```
#18
mySchool.head()
```

[36]:

	ID	Math	Physics	Result	Time
0	10	82	88	Pass	2010-12-14 09:01:00
1	20	63	62	Pass	2016-07-26 20:02:00
2	30	59	53	Pass	2010-04-15 18:25:00
3	40	26	79	Fail	2010-09-10 22:48:00

```
4  50    39       96   Fail 2017-03-10 22:02:00
```

[37]:
```
#19
mySchool['Hour'] = mySchool['Time'].dt.hour
```

[38]:
```
#19
mySchool['Year'] = mySchool['Time'].dt.year
```

[39]:
```
#19
mySchool.head()
```

[39]:
```
   ID  Math  Physics Result                Time  Hour  Year
0  10    82       88   Pass 2010-12-14 09:01:00     9  2010
1  20    63       62   Pass 2016-07-26 20:02:00    20  2016
2  30    59       53   Pass 2010-04-15 18:25:00    18  2010
3  40    26       79   Fail 2010-09-10 22:48:00    22  2010
4  50    39       96   Fail 2017-03-10 22:02:00    22  2017
```

[40]:
```
#20
ts = pd.to_datetime('1/1/2015')
```

[41]:
```
#20
mySchool.loc[mySchool.Time >= ts, :].head()
```

[41]:
```
     ID  Math  Physics Result                Time  Hour  Year
1    20    63       62   Pass 2016-07-26 20:02:00    20  2016
4    50    39       96   Fail 2017-03-10 22:02:00    22  2017
5    60    79       90   Pass 2017-01-11 19:37:00    19  2017
8    90    83       98   Pass 2017-06-08 02:58:00     2  2017
12  130    72       62   Pass 2017-06-30 08:45:00     8  2017
```

[42]:
```
#21
mySchool.Year.value_counts()
```

[42]:
```
2010    19
2017    16
2011    13
2018    10
2015    10
2014     9
2013     8
2012     7
2016     5
2019     2
2009     1
Name: Year, dtype: int64
```

[43]:
```
#21
mySchool.Year.value_counts().sort_index()
```

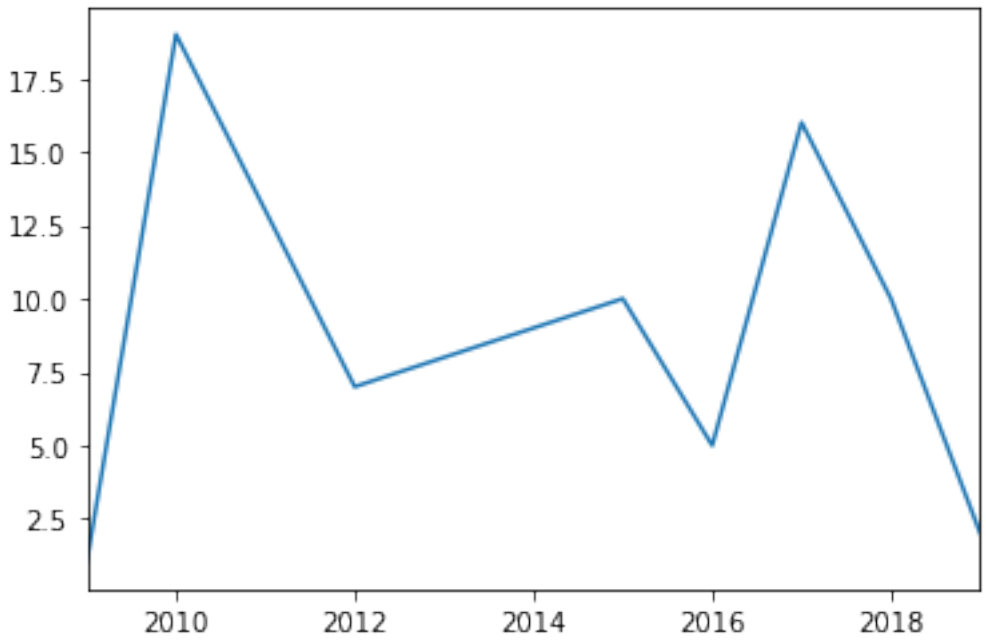
[43]:
```
2009     1
2010    19
```

```
2011    13
2012     7
2013     8
2014     9
2015    10
2016     5
2017    16
2018    10
2019     2
Name: Year, dtype: int64
```

[44]:
```
#21
%matplotlib inline
```

[45]:
```
#21
mySchool.Year.value_counts().sort_index().plot()
```

[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x148669173c8>
```



[46]:
```
#22
mySchool.loc[mySchool.duplicated(subset=['Math','Physics'],keep='first'),:]
```

[46]:

	ID	Math	Physics	Result	Time	Hour	Year
16	170	85	72	Pass	2011-06-20 05:28:00	5	2011
33	340	60	10	Fail	2012-10-27 23:50:00	23	2012
47	480	7	52	Fail	2015-06-07 08:59:00	8	2015
64	650	76	7	Fail	2011-06-03 15:54:00	15	2011

[47]:
```
#23
mySchool.loc[mySchool.duplicated(subset=['Math','Physics'],keep=False),:]
```

[47]:

	ID	Math	Physics	Result	Time	Hour	Year
6	70	85	72	Pass	2011-06-20 05:28:00	5	2011
16	170	85	72	Pass	2011-06-20 05:28:00	5	2011
23	240	60	10	Fail	2012-10-27 23:50:00	23	2012
33	340	60	10	Fail	2012-10-27 23:50:00	23	2012
37	380	7	52	Fail	2015-06-07 08:59:00	8	2015
47	480	7	52	Fail	2015-06-07 08:59:00	8	2015
54	550	76	7	Fail	2011-06-03 15:54:00	15	2011
64	650	76	7	Fail	2011-06-03 15:54:00	15	2011

[48]:
```
#24
pd.get_option('display.max_rows')
```

[48]:
```
60
```

[49]:
```
#24
pd.set_option('display.max_rows', None)
```

[50]:
```
#24
pd.get_option('display.max_columns')
```

[50]:
```
20
```

[51]:
```
#25
pd.set_option('display.max_columns', 500)
```

[52]:
```
#25
pd.get_option('display.max_columns')
```

[52]:
```
500
```

[53]:
```
#25
pd.reset_option('display.max_columns')
```

[54]:
```
#26
pd.set_option('display.float_format', '{:,}'.format)
```

[55]:
```
#27
import numpy as np
```

[56]:
```
#27
arr=np.random.randint(0, 101, [3,4])
```

[57]:
```
#27
newDF=pd.DataFrame(arr, columns = ['Red', 'Green', 'Blue', 'Yellow'],␣
 ↪index=['10', '20', '30'])
```

[58]:
```
#27
newDF
```

[58]:

	Red	Green	Blue	Yellow
10	49	19	20	27

```
20    1     60    43      72
30   65     75    82       3
```

[59]:
```
#28
newDF.applymap(float)
```

[59]:
```
    Red  Green  Blue  Yellow
10 49.0   19.0  20.0    27.0
20  1.0   60.0  43.0    72.0
30 65.0   75.0  82.0     3.0
```

[60]:
```
#29
newDF.apply(np.max, axis=1)
```

[60]:
```
10    49
20    72
30    82
dtype: int64
```

[61]:
```
#30
newDF.apply(np.argmax, axis=1)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:56:
FutureWarning:
The current behaviour of 'Series.argmax' is deprecated, use 'idxmax'
instead.
The behavior of 'argmax' will be corrected to return the positional
maximum in the future. For now, use 'series.values.argmax' or
'np.argmax(np.array(values))' to get the position of the maximum
row.
  return getattr(obj, method)(*args, **kwds)
```

[61]:
```
10       Red
20    Yellow
30      Blue
dtype: object
```

[ ]:
```

```