Pandas Documentation

Installation

On Ananconda Distributions, Pandas can be installed using the following command:

```
conda install pandas
```

Pandas can also be installed from the command line using pip using

```
pip install pandas
```

Series and DataFrames

Pandas Series is a labelled 1d array capable of holding data of any type (int,str,float,objects,etc.) The axis labels are collectively called as index

Pandas DataFrames is a 2d array that is able to store heterogeneous data items which can input in the form of a dictionary

In [2]:
```
data = {
    'apple' : [3,1,4,5],
    'orange' : [1,5,6,8]
}
print(data)
print(type(data))
```

```
{'apple': [3, 1, 4, 5], 'orange': [1, 5, 6, 8]}
<class 'dict'>
```

In [3]:
```
df = pd.DataFrame(data)
```

In [4]:
```
df
```

Out[4]:

	apple	orange
0	3	1
1	1	5
2	4	6
3	5	8

Accessing Elements from a Pandas Series/DataFrame

Elements can from a pandas dataframe just like a dictionary

./media/image2.png

Reading CSV Files and inserting them, constructing Pandas Data Frames and viewing the data

Path can be a chosen as a raw string and feed into pandas via the read_csv(), read_tsv() or read_table() command as illustrated below:

The path variable must a reference to a tabular datasheet in the user's local machine

Once the DataFrame has been created, the top 5 rows can be viewed using the head() command

```
In [8]: df = pd.read_csv(path)
```

```
In [9]: df.head()
```

Out[9]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0

Moreover, the bottom rows can also be viewed using the tail command (). Also, a desirable number of rows can be passed into the bracket.

```
In [10]: df.tail(10)
```

Out[10]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
448	Gordon Hayward	Utah Jazz	20.0	SF	26.0	6-8	226.0	Butler	15409570.0
449	Rodney Hood	Utah Jazz	5.0	SG	23.0	6-8	206.0	Duke	1348440.0
450	Joe Ingles	Utah Jazz	2.0	SF	28.0	6-8	226.0	NaN	2050000.0
451	Chris Johnson	Utah Jazz	23.0	SF	26.0	6-6	206.0	Dayton	981348.0
452	Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Setting a Different Index

Pandas gives the user an option of setting indices to other data columns using the index_col parameter in the read_csv command

```
In [11]: df = pd.read_csv(path,index_col='Name')
         df.head()
```

Out[11]:

	Team	Number	Position	Age	Height	Weight	College	Salary
Name								
Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0

Getting Useful Insights into the data

The user can get description of Column names, nullability and data types of the columns using the info() command

```
In [15]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 1 to 1000
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Title               1000 non-null   object 
 1   Genre               1000 non-null   object 
 2   Description         1000 non-null   object 
 3   Director            1000 non-null   object 
 4   Actors              1000 non-null   object 
 5   Year                1000 non-null   int64  
 6   Runtime (Minutes)   1000 non-null   int64  
 7   Rating              1000 non-null   float64
 8   Votes               1000 non-null   int64  
 9   Revenue (Millions)  872 non-null    float64
 10  Metascore           936 non-null    float64
dtypes: float64(3), int64(3), object(5)
memory usage: 74.2+ KB
```

Further, detailed statistics can constructed from the data using the describe() command.

```
In [16]: df.describe()
```

Out[16]:

	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore
count	1000.000000	1000.000000	1000.000000	1.000000e+03	872.000000	936.000000
mean	2012.783000	113.172000	6.723200	1.698083e+05	82.956376	58.985043
std	3.205962	18.810908	0.945429	1.887626e+05	103.253540	17.194757
min	2006.000000	66.000000	1.900000	6.100000e+01	0.000000	11.000000
25%	2010.000000	100.000000	6.200000	3.630900e+04	13.270000	47.000000
50%	2014.000000	111.000000	6.800000	1.107990e+05	47.985000	59.500000
75%	2016.000000	123.000000	7.400000	2.399098e+05	113.715000	72.000000
max	2016.000000	191.000000	9.000000	1.791916e+06	936.630000	100.000000

The shape of the Data-Frame can be checked using shape command

```
In [17]: df.shape
Out[17]: (1000, 11)
```

Checking for Duplicates

The Data-Frames might contain duplicates and this can be checked using duplicated () cmd.

```
In [18]: df.duplicated()
Out[18]: Rank
         1       False
         2       False
         3       False
         4       False
         5       False
                 ...
         996     False
         997     False
         998     False
         999     False
         1000    False
         Length: 1000, dtype: bool

In [19]: sum(df.duplicated())
Out[19]: 0

In [20]: df1 = df.append(df)
         df1.shape
Out[20]: (2000, 11)

In [21]: df1.duplicated().sum()
Out[21]: 1000

In [22]: df2 = df1.drop_duplicates()

In [23]: print(df.shape)
         print(df1.shape)
         print(df2.shape)
         (1000, 11)
         (2000, 11)
         (1000, 11)
```

The columns can be viewed using df.columns

```
In [25]: # Columns

In [26]: df.columns

Out[26]: Index(['Title', 'Genre', 'Description', 'Director', 'Actors', 'Year',
               'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
               'Metascore'],
              dtype='object')

In [27]: len(df.columns)

Out[27]: 11

In [28]: df.describe()
```

Out[28]:

	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore
count	1000.000000	1000.000000	1000.000000	1.000000e+03	872.000000	936.000000
mean	2012.783000	113.172000	6.723200	1.698083e+05	82.956376	58.985043
std	3.205962	18.810908	0.945429	1.887626e+05	103.253540	17.194757
min	2006.000000	66.000000	1.900000	6.100000e+01	0.000000	11.000000
25%	2010.000000	100.000000	6.200000	3.630900e+04	13.270000	47.000000
50%	2014.000000	111.000000	6.800000	1.107990e+05	47.985000	59.500000
75%	2016.000000	123.000000	7.400000	2.399098e+05	113.715000	72.000000
max	2016.000000	191.000000	9.000000	1.791916e+06	936.630000	100.000000

```
In [29]: col = df.columns
         type(col)

Out[29]: pandas.core.indexes.base.Index

In [30]: type(list(col))

Out[30]: list

In [31]: # Renaming the Columns
         col

Out[31]: Index(['Title', 'Genre', 'Description', 'Director', 'Actors', 'Year',
               'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
               'Metascore'],
              dtype='object')
```

The columns can also be manipulated using df.columns

./media/image11.png

The columns can also be manipulated using df.columns

```
In [36]: df.rename(columns={
             'Runtime (Minutes)' : 'Runtime',
             'Revenue (Millions)' : 'Revenue'
         },inplace=True)

In [37]: df.columns

Out[37]: Index(['Title', 'Genre', 'Description', 'Director', 'Actors', 'Year',
               'Runtime', 'Rating', 'Votes', 'Revenue', 'Metascore'],
              dtype='object')

In [38]: col

Out[38]: Index(['Title', 'Genre', 'Description', 'Director', 'Actors', 'Year',
               'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
               'Metascore'],
              dtype='object')
```

Checking for NULL values in the dataset

1. To check for NULL values in the dataset use the isnull() or isna() cmd

2. To perform sum use the isnull().sum() cmd.

```
In [41]: df.isnull()
```

Out[41]:

	Title	Genre	Description	Director	Actors	Year	Runtime	Rating	Votes	Revenue	Metascore
Rank											
1	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...
996	False	False	False	False	False	False	False	False	False	True	False
997	False	False	False	False	False	False	False	False	False	False	False
998	False	False	False	False	False	False	False	False	False	False	False
999	False	False	False	False	False	False	False	False	False	True	False
1000	False	False	False	False	False	False	False	False	False	False	False

1000 rows × 11 columns

```
In [42]: df.isnull().sum()
```

```
Out[42]: Title            0
         Genre            0
         Description      0
         Director         0
         Actors           0
         Year             0
         Runtime          0
         Rating           0
         Votes            0
         Revenue        128
         Metascore       64
         dtype: int64
```

```
In [43]: df.isna().sum()
```

```
Out[43]: Title            0
         Genre            0
         Description      0
         Director         0
         Actors           0
         Year             0
         Runtime          0
         Rating           0
         Votes            0
         Revenue        128
         Metascore       64
         dtype: int64
```

Gathering Valuable insights into the dataset

1. To get the total counts of each category use the value_counts () cmd
2. To check check for uniqueness in the data use the unique (). This would return the number of unique values in a particular Pandas Series.

```
In [65]: df['Genre'].describe()

Out[65]: count                        1000
         unique                        207
         top       Action,Adventure,Sci-Fi
         freq                           50
         Name: Genre, dtype: object

In [66]: df['Genre'].value_counts().head(10)

Out[66]: Action,Adventure,Sci-Fi       50
         Drama                         48
         Comedy,Drama,Romance          35
         Comedy                        32
         Drama,Romance                 31
         Animation,Adventure,Comedy    27
         Action,Adventure,Fantasy      27
         Comedy,Drama                  27
         Comedy,Romance                26
         Crime,Drama,Thriller          24
         Name: Genre, dtype: int64

In [69]: len(df['Genre'].unique())

Out[69]: 207
```

The Correlation Matrix

The correlation matrix shows the relationship between various features and can be implemented using the corr () method.

```
In [71]: corrmat = df.corr()

In [72]: corrmat
```

Out[72]:

	Year	Runtime	Rating	Votes	Revenue	Metascore
Year	1.000000	-0.164900	-0.211219	-0.411904	-0.117562	-0.076077
Runtime	-0.164900	1.000000	0.392214	0.407062	0.247834	0.202239
Rating	-0.211219	0.392214	1.000000	0.511537	0.189527	0.604723
Votes	-0.411904	0.407062	0.511537	1.000000	0.607941	0.318116
Revenue	-0.117562	0.247834	0.189527	0.607941	1.000000	0.132304
Metascore	-0.076077	0.202239	0.604723	0.318116	0.132304	1.000000

Bibliography

1. TakenMind Course [Udemy]
2. Google
3. Stack Overflow
4. Wikipedia