Reading the data

In [0]:

```
#Reading the CSV format

df = spark.read.format('csv').option('inferSchema',True).option('header',True).load('/FileStore/tables/BigMart_Sales-1.csv')

#Reading the JSON format

df1 = spark.read.format('json').option('Inferschema',True).option('header',True)\
    .option('multiline',False).load('/FileStore/tables/drivers.json')

#To display the data which is loaded
df.display()
df.show()

#To display the schema
df.printSchema
```

StructType and DDL Schema

In [0]:

```
#Creating our own predefined schema

df_schema='''

Item_Identifier string,
Item_Weight STRING,
Item_Fat_Content string,
Item_Visibility double,
Item_Type string,
Item_MRP double,
Outlet_Identifier string,
Outlet_Establishment_Year integer,
Outlet_Size string,
Outlet_Location_Type string,
Outlet_Type string,
Item_Outlet_Sales double

'''

df = spark.read.format('csv').schema(df_schema).option('header',True).load('/FileStore/tables/BigMart_Sales-1.csv')

#Importing libaries to use StructType Schema
from pyspark.sql.types import *
from pyspark.sql.functions import *

df = StructType([
    StructField('Item_Identifier', StringType(), True),
    StructField('Item_Weight', StringType(), True),
    StructField('Item_Fat_Content', StringType(), True),
    StructField('Item_Visibility', DoubleType(), True),
    StructField('Item_Type', StringType(), True),
    StructField('Item_MRP', DoubleType(), True),
    StructField('Outlet_Identifier', StringType(), True),
    StructField('Outlet_Establishment_Year', IntegerType(), True),
    StructField('Outlet_Size', StringType(), True),
    StructField('Outlet_Location_Type', StringType(), True),
    StructField('Outlet_Type', StringType(), True),
    StructField('Item_Outlet_Sales', DoubleType(), True)
]
)
```

```
df_1 = spark.read.format('csv').schema(df).option('Header',True).load('/FileStore/tables
/BigMart_Sales-1.csv')
```

Transforming the Data

In [0]:

```
#Select

df.select('Item_identifier','Item_Weight','Item_Fat_content').display()
df.select(col('Item_Identifier'),col('Item_Weight')).display()

#Alias

df.select(col("Item_identifier").alias("Item_Number")).display()

#Filter

df.filter(col('Item_fat_content')=='Regular').display()
df.filter((col('Item_Weight')<10) & (col('Item_Type')=='Soft Drinks') ).display()
df.filter((col('outlet_size').isNull()) & (col('outlet_location_type').isin ('Tier 1','Ti
er 2'))).display()
df.filter((col('Item_Identifier')=='FDH17') & (col('item_weight').isNotNull()) & (col('O
utlet_Identifier').isin('OUT017'))).display()

#Renaming the column

df.withColumnRenamed('Item_Fat_content','Fat_content').display()

#Adding new Column

df.withColumn('Flag',lit('new')).display()
df.withColumn('Total Cost',col('Item_weight')*col('Item_MRP') ).display()
df.withColumn("Item_fat_content",regexp_replace(col('Item_fat_content'),'Regular','Reg'))
\
    .withColumn('Item_fat_content',regexp_replace(col('Item_fat_content'),"Low Fat",'Lf')
).display()

#TypeCasting

df=df.withColumn('Item_weight',col('Item_weight').cast(StringType()))

#Sorting

df.sort(col('Item_weight').desc()).display()
df.sort(['item_weight','item_visibility'],ascending=[0,0]).display()
df.sort(['item_weight','item_type'],ascending=[0,1]).display()
df.sort(['Item_visibility','Item_type'],asceding=[0,0]).display()

#Limit

df.limit(10).display()

#Drop

df.drop('Item_Weight','Item_Identifier').display()

#DropDuplicates

df.dropDuplicates().display()
df.drop_duplicates(subset=['Item_type']).display()

#Distinct

df.distinct().display()
```

Union and UnionByName

In [0]:

```
#Creating Data frames

data1=[('1','Shreyas'),('2','Ram')]
schema1='id String, name String'

df1=spark.createDataFrame(data1,schema1)


data2=[('3','Vivek'),('4','Kumar')]
schema2='id String,name String'

df2=spark.createDataFrame(data2,schema2)

#Using Union
df1.union(df2).display()

#Creating the data frame
data2=[('Vivek','3'),('Himani','4')]
schema2='name String,id String'

df2=spark.createDataFrame(data2,schema2)
df2.display()

#Using UnionByName
df1.unionByName(df2).display()
#Since There is a mismatch we are using the UnionByName
```

id	name
1	Shreyas
2	Ram
3	Vivek
4	Kumar

name	id
Vivek	3
Himani	4

id	name
1	Shreyas
2	Ram
3	Vivek
4	Himani

String Functions

In [0]:

```
#Using initcap

df.select(initcap('Item_type')).display()
df.select(lower('Item_type')).display()
df.select(upper('Item_type')).display()
df.select(initcap('Item_type').alias("Upper_name")).display()
```

Date Functions

- **Current_date()**
- **date_add() or date_sub()**
- **date_diff()**
- **date_format()**

- date_format()

In [0]:

```
df=df.withColumn('CurrentDate',current_date())
df=df.withColumn('added_date',date_add('CurrentDate',7))
df=df.withColumn('SubstractedDate',date_sub('currentDate',7))
df=df.withColumn('Date_difference',datediff('substractedDate','added_date'))
df=df.withColumn('formatted_date',date_format('SubstractedDate','MM-dd-yyyy'))
```

Handling null

- **fillna() function**
- **dropna() function**

In [0]:

```
df.dropna('all').display()
df.dropna('any').display()
df.dropna(subset=['Item_weight']).display()
df.fillna('Not Exist').display() #Only applicable for StringType
df.fillna(0).display() #Applicable for IntegerType
df.fillna('NA',subset=['outlet_size']).display()
```

Split and Indexing

In [0]:

```
df.withColumn('outlet_type',split('outlet_type',' ')).display() #Here Space ' ' is a del
imiter
df.withColumn('Outlet_type',split('outlet_type',' ')[1]).display()
```

Explode and Array_Contains

In [0]:

```
dataf.withColumn('outlet_type',explode('outlet_type')).display()
dataf.withColumn('ContainsinList',array_contains('outlet_type','Type1')).display() #Retu
rn boolean values
```

Data Aggregation

- **GroupBy**

In [0]:

```
df.groupBy('Item_type').agg(sum('iTEM_MRP').alias("Total_cost"),avg('iTEM_MRP')).display
()
df.groupBy('Item_type','Item_Fat_content').agg(round(sum('item_mrp'),2),avg('item_mrp'))
.show()
```

CollectList

In [0]:

```
data = [('user1','book1'),
        ('user1','book2'),
        ('user2','book2'),
        ('user2','book4'),
        ('user3','book1')]

schema = 'user string, book string'
```

```
df_book = spark.createDataFrame(data,schema)

df_book.display()


df_book.groupBy('user').agg(collect_list('book').alias("list")).display()
```

user	book
user1	book1
user1	book2
user2	book2
user2	book4
user3	book1

user	list
user1	List(book1, book2)
user2	List(book2, book4)
user3	List(book1)

Pivot

In [0]:

```
df.groupBy('Item_type').pivot('outlet_size').agg(sum('item_mrp')).display()
```

When_Otherwise

In [0]:

```
df5=df.withColumn('vegornonveg',when(col('item_type')=='Meat','NON-VEG').otherwise('VEG')
).display()
```

Joins

- **inner**
- **left and right**
- **anti**

In [0]:

```
dataj1 = [('1','gaur','d01'),
          ('2','kit','d02'),
          ('3','sam','d03'),
          ('4','tim','d03'),
          ('5','aman','d05'),
          ('6','nad','d06')]

schemaj1 = 'emp_id STRING, emp_name STRING, dept_id STRING'

df1 = spark.createDataFrame(dataj1,schemaj1)

dataj2 = [('d01','HR'),
          ('d02','Marketing'),
          ('d03','Accounts'),
          ('d04','IT'),
          ('d05','Finance')]

schemaj2 = 'dept_id STRING, department STRING'

df2 = spark.createDataFrame(dataj2,schemaj2)
```

```
df1.join(df2,df1['dept_id']==df2['dept_id'],'inner').display()
df1.join(df2,df1['dept_id']==df2['dept_id'],'left').display()
df1.join(df2,df1['dept_id']==df2['dept_id'],'anti').display()
```

emp_id	emp_name	dept_id	dept_id	department
1	gaur	d01	d01	HR
2	kit	d02	d02	Marketing
3	sam	d03	d03	Accounts
4	tim	d03	d03	Accounts
5	aman	d05	d05	Finance

emp_id	emp_name	dept_id	dept_id	department
1	gaur	d01	d01	HR
2	kit	d02	d02	Marketing
3	sam	d03	d03	Accounts
4	tim	d03	d03	Accounts
5	aman	d05	d05	Finance
6	nad	d06	null	null

emp_id	emp_name	dept_id
6	nad	d06

Row Functions

- **Rownum()**
- **Rank()**
- **DenseRank()**

In [0]:

```
#Importing the libraries

from pyspark.sql.window import Window

#row_number
df.withColumn('rownumber',row_number().over(Window.orderBy('Item_Identifier'))).display(
)

#Rank and Dense Rank

df.withColumn('rank',rank().over(Window.orderBy(col('Item_identifier').desc())))\
    .withColumn('dense_rank',dense_rank().over(Window.orderBy(col('item_identifier').des
c())))\
        .display()
```

Cummulative Sum

In [0]:

```
df.withColumn('Cummulative_Sum',round(sum('Item_MRP').over(Window.orderBy('Item_Identifi
er').rowsBetween(Window.unboundedPreceding,Window.currentRow)),2)).display()
```

User Defined Functions (UDF)

In [0]:

```
def my_function(x):
  return x*x

my_user = udf(my_function)

df.withColumn('User_def_fun',my_user('Item_outlet_sales')).display()
```

Data Writing

- **Append**
- **Overwrite**
- **Error**
- **Ignore**

In [0]:

```
df.write.format('csv').save('/FileStore/tables/CSV/data.csv')

#Append

df.write.format('csv')\
    .mode('append')\
    .option('path','/FileStore/tables/CSV/data.csv')\
    .save()


#overwrite
df.write.format('csv')\
    .mode('overwrite')\
        .save('/FileStore/Tables/CSV/data.csv')

#ignore

#overwrite
df.write.format('csv')\
    .mode('ignore')\
        .save('/FileStore/Tables/CSV/data.csv')

#Error
df.write.format('csv')\
    .mode('error')\
        .save('/FileStore/Tables/CSV/data.csv')
#We can see the error because the file is already exists.
```

Parquet File Format

In [0]:

```
df.write.format('parquet').mode('overwrite').save('\Filestore/tables/CSV/data.csv')
```

Table

In [0]:

```
df.write.format('parquet').mode('overwrite').saveAsTable('My_Table')
```

Spark SQL

Creating a Temporary view

In [0]:

```
df.createTempView('view')
```

In [0]:

```
%sql
select count(*) AS TotalNoOfRecords from view;
```

TotalNoOfRecords
8523

Storing the o/p of SparkSql into a dataframe

In [0]:

```
df_sql_op=spark.sql("select * from view where item_mrp<200")
```