Swift TabularData & DataFrame

ddas.tech/swift-tabulardata-dataframe/

August 7, 2023

In this post we will explore **Swift TabularData & DataFrame** using the data set https://www.kaggle.com/datasets/abhijitdahatonde/real-world-smartphones-dataset

One of the foundation block of TabularData is DataFrame which is a collection that arranges data in rows and columns.

Row from index 900 to the end

	brand_name <String>	model <String>	price <Int>	avg_rating <Double>	5G_or_not <Int>	processor_brand <String>	num_cores <Int>	processor_speed <Double>	battery_capaci… <Int>	13 more
900	xiaomi	Xiaomi Redmi N…	14,590	8.0	0	helio	8	2.05	5,000	
901	xiaomi	Xiaomi Redmi N…	15,499	8.1	0	helio	8	2.05	5,000	
902	xiaomi	Xiaomi Redmi N…	15,824	8.4	0	helio	8	2.05	5,000	
903	xiaomi	Xiaomi Redmi N…	11,999	8.0	0	helio	8	2.05	5,000	
904	xiaomi	Xiaomi Redmi N…	16,999	7.9	1	dimensity	8	2.4	5,000	
...										

34 rows, 22 columns

Table of Contents

For the sample given below we have used the following formatting option

```
let formattingOptions =  FormattingOptions(maximumLineWidth: 150,
                                           maximumCellWidth: 15,
                                           maximumRowCount: 5)
```

Creating a DataFrame from Dictionary Literal

```
var df: DataFrame = [
    "id":[1,2,3,4],
    "firstname":["John","Jack","Jill","Mary"],
    "lastname":["Doe","Doe","LNU","Jane"]
    //Every Column Must have the same number of elements.
]
print(df.description(options: formattingOptions))
```

	id <Int>	firstname <String>	lastname <String>
0	1	John	Doe
1	2	Jack	Doe
2	3	Jill	LNU
3	4	Mary	Jane

4 rows, 3 columns

Creating a DataFrame from Columns

```
var df = DataFrame()
let idColumn = Column(name: "id", contents: [1,2,3])
let nameColumn = Column(name: "name", contents: ["Jack","Jill","Mary"])
df.append(column: idColumn)
df.append(column: nameColumn)
print(df.description(options: formattingOptions))

//Every Column needs to have the same number of elements
//The column names must be unique
```

	id <Int>	name <String>
0	1	Jack
1	2	Jill
2	3	Mary

3 rows, 2 columns

Combining DataFrame Columns

```
var df: DataFrame = [
    "id":[1,2,3,4],
    "firstname":["John","Jack","Jill","Mary"],
    "lastname":["Doe","Doe","LNU","Jane"]
    //Every Column Must have the same number of elements.
]
print(df.description(options: formattingOptions))
df.combineColumns("firstname", "lastname", into: "fullname") { (fname:String?,
lname:String?) -> String? in
    guard let fn = fname, let ln = lname else {
        return ""
    }
    return fn + " " + ln
}
print(df.description(options: formattingOptions))
```

	id <Int>	firstname <String>	lastname <String>
0	1	John	Doe
1	2	Jack	Doe
2	3	Jill	LNU
3	4	Mary	Jane

4 rows, 3 columns

	id <Int>	fullname <String>
0	1	John Doe
1	2	Jack Doe
2	3	Jill LNU
3	4	Mary Jane

4 rows, 2 columns

First Name and Last Name Column has been combined into Full Name Column

Loading DataFrame from a csv

```
func loadDataFrameFromCSV() -> DataFrame{
    var df: DataFrame = DataFrame()
    if let filePath = Bundle.main.url(forResource: "smartphones", withExtension:
"csv"){
        let options = CSVReadingOptions(hasHeaderRow: true,
                                        delimiter: ",")

        guard let fileUrl = URL(string: filePath.absoluteString) else {
            fatalError("Error creating Url")
        }
        df = try! DataFrame(
            contentsOfCSVFile: fileUrl,
            options: options)

   }
    return df
}
```

Loading a DataFrame from CSV (limited number of rows)

```
func loadDataFrameFromCSV(numberOfRows:Int){
    if let filePath = Bundle.main.url(forResource: "smartphones", withExtension:
"csv"){
        let options = CSVReadingOptions(hasHeaderRow: true,
                                        delimiter: ",")

        guard let fileUrl = URL(string: filePath.absoluteString) else {
            fatalError("Error creating Url")
        }
        let df1 = try! DataFrame(
            contentsOfCSVFile: fileUrl,
            rows: 0 ..< numberOfRows,
            options: options )
        print(df1.description(options: formattingOptions))
   }
}
```

loadDataFrameFromCSV(numberOfRows: 10)

	brand_name <String>	model <String>	price <Int>	avg_rating <Double>	5G_or_not <Int>	processor_brand <String>	num_cores <Int>	processor_speed <Double>	battery_capaci… <Int>	13 more
0	asus	Asus ROG Phone…	39,999	8.7	1	snapdragon	8	2.9	6,000	
1	asus	Asus ROG Phone…	71,999	8.6	1	snapdragon	8	3.2	6,000	
2	asus	Asus ROG Phone…	72,999	8.8	1	dimensity	8	3.2	6,000	
3	asus	Asus ROG Phone…	89,999	8.8	1	snapdragon	8	3.2	6,000	
4	asus	Asus ROG Phone…	107,990	nil	1	dimensity	8	3.2	6,000	
...										

10 rows, 22 columns

Loading a DataFrame from CSV (stated row count & fixed columns)

```
func loadDataFrameFromCSV(numberOfRows:Int, columns:[String]){
    if let filePath = Bundle.main.url(forResource: "smartphones", withExtension:
"csv"){
        let options = CSVReadingOptions(hasHeaderRow: true,
                                        delimiter: ",")

        guard let fileUrl = URL(string: filePath.absoluteString) else {
            fatalError("Error creating Url")
        }
        let df1 = try! DataFrame(
            contentsOfCSVFile: fileUrl,
            columns:columns,
            rows: 0 ..< numberOfRows,
            types: ["brand_name" : .string,"model":.string], //This is optional and
can be used to speed up loading if we are aware of the data type of a column.
            options: options )
        print(df1.description(options: formattingOptions))
   }
}
```

CSV Reading Options

hasHeaderRow: Bool - Defaults to true

nilEncodings: Set<String>
Defaults to `["", "#N/A", "#N/A N/A", "#NA", "N/A", "NA", "NULL", "n/a", "null"]`.

trueEncodings: Set<String>
Defaults to `["1", "True", "TRUE", "true"]`.

falseEncodings: Set<String>
Defaults to `["0", "False", "FALSE", "false"]`.

floatingPointType: CSVType
Defaults to ``CSVType/double``.

dateParsers: [(String) -> Date?]
An array of closures that parse a date from a string.

ignoresEmptyLines: Bool
A Boolean value that indicates whether to ignore empty lines.
Defaults to `true`.

usesQuoting: Bool
A Boolean value that indicates whether to enable quoting.
When `true`, the contents of a quoted field can contain special characters, such as the field
delimiter and newlines. Defaults to `true`.

usesEscaping: Bool
A Boolean value that indicates whether to enable escaping.
When `true`, you can escape special characters, such as the field delimiter, by prefixing them with
the escape character, which is the backslash (`\`) by default. Defaults to `false`.

delimiter: Character { get }
Defaults to comma (`,`).
The character that separates data fields in a CSV file, typically a comma.

escapeCharacter: Character { get }
Defaults to backslash(`\`).

DataFrame summary()

```
print(df.summary().description(options: formattingOptions1))
```

	column <String>	mean <Double>	std <Double>	min <Double>	max <Double>	median <Double>	Q1 <Double>	Q3 <Double>	mode <Array<An…	uniqueCou… <Int>	noneCount <Int>	someCount <Int>
0	brand_name	nil	nil	nil	nil	nil	nil	nil	["xiaomi"]	45	0	934
1	model	nil	nil	nil	nil	nil	nil	nil	["Asus RO…	934	0	934
2	price	29,395.75…	36,616.70…	3,499.0	650,000.0	19,499.0	12,990.0	32,742.25	[14999]	355	0	934
3	avg_rating	7.833533	0.745394	6.0	8.9	8.0	7.4	8.4	[8.4]	31	99	835
4	5G_or_not	0.54818	0.49794	0.0	1.0	1.0	0.0	1.0	[1]	2	0	934
5	processor…	nil	nil	nil	nil	nil	nil	nil	["snapdra…	12	0	934
6	num_cores	7.849946	0.760472	4.0	8.0	8.0	8.0	8.0	[8]	3	1	933
7	processor…	2.40436	0.453533	1.2	3.2	2.3	2.05	2.84	[2.0]	33	28	906
8	battery_c…	4,866.108…	985.767579	1,900.0	22,000.0	5,000.0	4,500.0	5,000.0	[5000]	75	0	934
9	fast_char…	0.861884	0.345206	0.0	1.0	1.0	1.0	1.0	[1]	2	0	934
10	fast_char…	46.227154	34.306242	10.0	240.0	33.0	18.0	66.0	[33]	33	168	766
11	ram_capac…	6.623126	2.780143	1.0	18.0	6.0	4.0	8.0	[8]	9	0	934
12	internal_…	134.997859	87.74135	8.0	1,024.0	128.0	64.0	128.0	[128]	8	0	934
13	screen_si…	6.559283	0.31767	3.54	8.03	6.58	6.5	6.67	[6.5]	75	0	934
14	refresh_r…	92.752677	28.898534	60.0	240.0	90.0	60.0	120.0	[60]	6	0	934
15	num_rear_…	2.8394	0.773288	1.0	4.0	3.0	2.0	3.0	[3]	4	0	934
16	os	nil	nil	nil	nil	nil	nil	nil	["android…	3	0	934
17	primary_c…	51.775054	32.948263	2.0	200.0	50.0	48.0	64.0	[50.0]	18	0	934
18	primary_c…	16.843918	11.076068	0.0	60.0	16.0	8.0	20.0	[16]	19	5	929
19	extended_…	0.66167	0.473395	0.0	1.0	1.0	0.0	1.0	[1]	2	0	934
20	resolutio…	2,204.824…	518.190827	480.0	3,840.0	2,400.0	1,612.0	2,408.0	[2400]	57	0	934
21	resolutio…	1,073.319…	294.517723	480.0	2,460.0	1,080.0	1,080.0	1,080.0	[1080]	33	0	934

22 rows, 12 columns

DataFrame – List of Columns

```
let columns = df.columns
let columnNames = columns.map{$0.name}
print("Column Names")
print(columnNames)
```

```
["brand_name", "model", "price", "avg_rating", "5G_or_not", "processor_brand",
"num_cores", "processor_speed", "battery_capacity", "fast_charging_available",
"fast_charging", "ram_capacity", "internal_memory", "screen_size", "refresh_rate",
"num_rear_cameras", "os", "primary_camera_rear", "primary_camera_front",
"extended_memory_available", "resolution_height", "resolution_width"]
```

DataFrame – Accessing a row of data

```
//Accessing a row of data
let rowData = self.df[row:3]
print("Row at index 3")
print(rowData.description(options: formattingOptions))
```

DataFrame – Accessing Rows of Data from an index to the end of data frame

```
//Accessing rows of data from an index to the end
let rd1 = self.df[900...]
print("Row from index 900 to the end")
print(rd1.description(options: formattingOptions))
```

Row from index 900 to the end

	brand_name <String>	model <String>	price <Int>	avg_rating <Double>	5G_or_not <Int>	processor_brand <String>	num_cores <Int>	processor_speed <Double>	battery_capaci… <Int>	13 more
900	xiaomi	Xiaomi Redmi N…	14,590	8.0	0	helio	8	2.05	5,000	
901	xiaomi	Xiaomi Redmi N…	15,499	8.1	0	helio	8	2.05	5,000	
902	xiaomi	Xiaomi Redmi N…	15,824	8.4	0	helio	8	2.05	5,000	
903	xiaomi	Xiaomi Redmi N…	11,999	8.0	0	helio	8	2.05	5,000	
904	xiaomi	Xiaomi Redmi N…	16,999	7.9	1	dimensity	8	2.4	5,000	
...										

34 rows, 22 columns

DataFrame – Accessing a column by index

```
let columnData = df[column:1]
print("Column at Index 1")
print(columnData)
```

DataFrame – Accessing a Column by Name

```
//Acessing column by name
let modelCol = df["model"]
print("Columnn with name = model")
print(modelCol)
```

Accessing Multiple Column by Names

```
//Accessing couple of column names
let multiCols = df.selecting(columnNames: "brand_name","model")
print("Getting multiple columns from the dataframe")
print(multiCols.description(options: formattingOptions))
```

DataFrame Filter

```
func filterDataFrame(){
    let df1 = self.df.filter(on: "brand_name", String.self) {$0 == "google"}
     .filter(on: "processor_brand", String.self) {$0 == "snapdragon"}

    print(df1.description(options: formattingOptions))
}

func filterDataFrame1(){
    let df1 = self.df.filter(on: "brand_name", String.self) {$0 == "huawei"}
        .filter(on: "price", Int.self) {$0! > 100000}
        .filter(on: "processor_brand", String.self) {$0 == "snapdragon"}
    print(df1.description(options: formattingOptions))
}
```

DataFrame Filter & Sort

```
func filterDataFrame2(){
    let df1 = self.df.filter(on: "price", Int.self) {$0! > 100000 && $0! < 120000}
    .sorted(on: "price",order: .descending)
    print(df1.description(options: formattingOptions))
}
```

	brand_name <String>	model <String>	price <Int>	avg_rating <Double>	5G_or_not <Int>	processor_brand <String>	num_cores <Int>	processor_speed <Double>	battery_capaci… <Int>	13 more
0	samsung	Samsung Galaxy…	119,990	8.5	1	snapdragon	8	nil	5,100	
1	vivo	Vivo X Fold 2	119,990	nil	1	snapdragon	8	3.2	4,800	
2	samsung	Samsung Galaxy…	118,999	nil	1	snapdragon	8	3.0	5,000	
3	vivo	Vivo X Fold 5G…	118,990	nil	1	snapdragon	8	3.0	4,600	
4	samsung	Samsung Galaxy…	114,990	nil	1	snapdragon	8	3.2	5,000	
...										

15 rows, 22 columns

The price column is filtered and sorted in descending

DataFrame Grouped

```
    let gdf = self.df.grouped(by: "brand_name")
    let gdfCount = gdf.counts(order: .descending)
    print(gdfCount)
```

	brand_name <String>	count <Int>
0	xiaomi	134
1	samsung	132
2	vivo	111
3	realme	97
4	oppo	88
5	motorola	52
6	oneplus	42
7	poco	41
8	tecno	33
9	iqoo	32
10	infinix	29
11	huawei	16
12	google	14
13	honor	13
14	nokia	13
15	itel	10
16	sony	9
17	asus	7
18	nubia	6
19	nothing	5
...		

45 rows, 2 columns

```
let gdf1 = self.df.grouped(by: "brand_name").means("avg_rating", Double.self, order:
.descending)
    print(gdf1)
```

	brand_name <String>	mean(avg_rating) <Double>
0	leitz	8.9
1	lenovo	8.8
2	sharp	8.8
3	asus	8.7
4	lg	8.7
5	royole	8.7
6	doogee	8.6
7	zte	8.55
8	nothing	8.52
9	blu	8.5
10	nubia	8.433333
11	tesla	8.3
12	oneplus	8.223684
13	iqoo	8.221875
14	sony	8.142857
15	motorola	8.010417
16	huawei	8.0
17	oukitel	7.933333
18	xiaomi	7.8776
19	oppo	7.873171
...		

45 rows, 2 columns