Data Grid View Summary

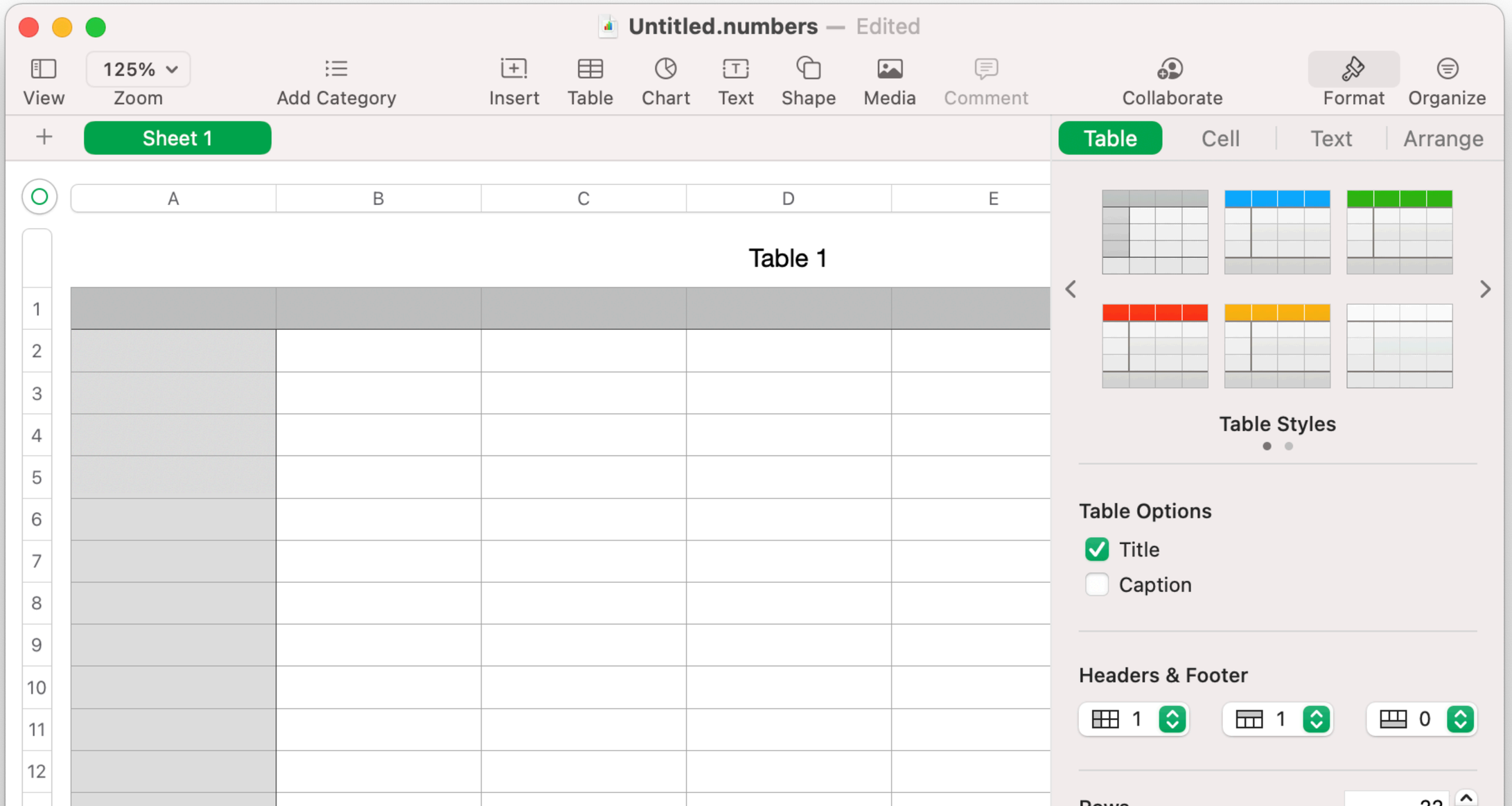
Creating Spreadsheet-like view for SwiftUI

Ray Fix, A Flock of Swifts, 2020.11.28

Agenda

- Problem statement
- Data model
- Solution using pure SwiftUI
- Solution using UIViewRepresentable
- Future improvements

Spreadsheet



Problem Statement

- Display a two dimensional grid of data
- Column and row headers independent of data
- Scrolling in both vertical and horizontal directions
- Automatically size the column width so that the text doesn’t truncate
- [Column and row headers are sticky]

Simplifying Assumptions

- Make the dimensions immutable
- Data grid only displays the data, doesn’t allow for edits
- The data is just strings (no special formatting)
- Each cell is a single line (not multiple lines)
- The data is loaded in all at once

Data Model

```
struct DataGrid {
  private(set) var rowNames: [String]
  private(set) var columnNames: [String]
  private var data: [[String?]]

  /// …

 }
```

Data Model

```
var columns: Range<Int> {
  0..<columnNames.count
}

var rows: Range<Int> {
  0..<rowNames.count
}
```

Data Model

```
func isContained(row: Int, column: Int) -> Bool {
  rows.contains(row) && columns.contains(column)
}
```

```
 subscript(row row: Int, column column: Int) -> String? {
   get {
     guard isContained(row: row, column: column) else {
       return nil
     }
     return data[row][column]
   }
 }
```

Data Model

```
func column(at columnIndex: Int) -> AnySequence<String?> {

    AnySequence(
       sequence(state: 0) { rowIndex in

           guard rows.contains(rowIndex) else {
             return nil
           }
           defer {
             rowIndex += 1
           }
           return self[row: rowIndex, column: columnIndex]

       }
    )

}
```

Pure SwiftUI

```
struct DataGridView: View {
  let dataGrid: DataGrid

  private func dataCell(_ string: String, column: Int) -> some View {
    Text(string)
      .padding()
      .frame(width: 150, height: 30, alignment: .leading)
      .border(Color.black, width: 0.5)
  }
  private func headerCell(_ string: String, column: Int) -> some View {
    dataCell(string, column: column)
      .foregroundColor(.white)
      .background(Color.gray)
  }

  var body: some View {
    :
  }
}
```

Pure SwiftUI

```
var body: some View {
  ScrollView([.vertical, .horizontal], showsIndicators: true) {
    HStack(alignment: .center, spacing: 0) {
      VStack(spacing: 0) {
        headerCell(“”, column: 0)
        ForEach(Array(dataGrid.rowNames.enumerated()), id: \.offset) { offset, name in
          headerCell(name, column: 0)
        }
      }
      ForEach(dataGrid.columnNames.indices, id: \.self) { columnIndex in
        VStack(spacing: 0) {
          headerCell(dataGrid.columnNames[columnIndex], column: columnIndex+1)
          ForEach(dataGrid.rowNames.indices, id: \.self) { rowIndex in
            dataCell(dataGrid[row: rowIndex, column: columnIndex] ?? "NA",
                     column: columnIndex+1)
          }
        }
      }
    }.border(Color.black, width: 1)
  }
}
```

Pure SwiftUI

```
struct MaxColumnWidthKey: PreferenceKey {


  static var defaultValue: [Int: CGFloat] = [:]


  static func reduce(value: inout [Int : CGFloat], nextValue: () -> [Int : CGFloat]) {

    for (column, width) in nextValue() {
      if (value[column] ?? 0) < width {
        value[column] = width
      }
    }
  }

}
```

Pure SwiftUI

```swift
@State private var columnWidths: [Int: CGFloat] = [:]

private func dataCell(_ string: String, column: Int) -> some View {
  Text(string)
    .padding()
    .background(GeometryReader { proxy in
      Color.clear.preference(key: MaxColumnWidthKey.self, value: [column: proxy.size.width])
    })
    .frame(width: columnWidths[column], height: 30, alignment: .leading)
    .border(Color.black, width: 0.5)
}
```

Pure SwiftUI

```
.onPreferenceChange(MaxColumnWidthKey.self) {
    columnWidths = $0
}
```

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
Row 1	0.17060477412883335	0.6454650064554366	0.6367239028633905	0.3678640233216768	0.584025781242763	0.3148717
Row 2	0.9313284321744758	0.571237995863783	0.04731936571250128	0.26687538998633387	0.11145230251134475	0.5483185
Row 3	0.0942413539464001	0.658893527988487	0.4805422100563065	0.021727919993320133	0.1613290025750742	0.5512443
Row 4	0.12431904802867033	0.7171102444414845	0.5868203131101164	0.5211410614876242	0.8620663370732363	0.6233198
Row 5	0.8024244393326566	0.34320561339840105	0.41884807019745607	0.29980012703540715	0.04994633238464652	0.4574035
Row 6	0.6368130627057995	0.3430884873054306	0.46575780156585966	0.3296840518951838	0.15342830791984208	0.2386134
Row 7	0.05990629705563133	0.07832278248126312	0.9876820823521045	0.6608797596349678	0.1955959764051084	0.6164319
Row 8	0.08982858402986371	0.3220456635476555	0.5418002497001858	0.14029990997045338	0.06455832129110906	0.9951815
Row 9	0.6539632043985092	0.8445242715381357	0.3736862309072151	0.3077208568211717	0.8445179735980373	0.7086959
Row 10	0.06530157823507632	0.7947958908298112	0.07440900739811296	0.14053274196533594	0.7135076887846628	0.1020499
Row 11	0.07875222146891903	0.08011820796354863	0.8380802414395621	0.9934512004914563	NA	0.4914236
Row 12	0.23692759777382555	0.9441331923569253	0.6467504769024409	0.6337880784491011	0.45163717877820875	0.7381302
Row 13	0.768097572430043	0.9312829409781275	0.3666635929393942	0.7286097328449123	0.3329875347689478	0.7125032
Row 14	0.7236454838322331	0.9251324194636346	0.25437584451254314	0.4633102371320209	0.7775035264901818	0.4587849
Row 15	0.8118742973660499	0.18868849536187549	0.8156727832877061	0.432587522888112	0.6136716647115174	0.1355733
Row 16	0.7822358394603237	0.37047107218469777	0.2760080918918829	0.25495100484714184	0.7376451194346748	0.9310862
Row 17	0.8383881312674157	0.6426631650068432	0.4342957632381329	0.2530093206064373	0.2923690103327553	0.7883162
Row 18	0.2643420014409259	0.6339097637208602	0.9833116357753912	0.9310719401781754	0.9611032113172168	0.5300367
Row 19	0.11514279173320696	0.6432157356844594	0.37912101676322263	0.35760119786183	0.5294649968820436	0.8968327
Row 20	0.4583615907377059	0.24126476487569948	0.004792832208174169	0.40673341317179856	0.784782954287938	0.1849775
Row 21	0.8533972078602207	0.64604479724165	0.5986144609155948	0.18367563281765786	0.9875215018577921	0.4600221
Row 22	0.5011127089691159	0.7087212127634375	0.2251251150940924	0.9179584119143127	0.9548852191016904	0.4821280
Row 23	0.22221360236059362	0.9434215196531801	0.34683633224625865	0.8395716900732381	0.5889296843936057	0.1415128
Row 24	0.1017873686627947	0.9688922453505217	0.9183625118071214	0.7613444697953197	0.41139317219161187	0.2184035
Row 25	0.015890158239007413	0.8994450408226992	0.4248587839571847	0.42166672681561923	0.6535161802424424	0.9897864
Row 26	0.07201290614107747	0.12173530866291882	0.010492991612612634	0.32038507465951493	0.5046112143662665	0.8782239
Row 27	0.6961198669050478	0.4081443624328073	0.8648963655929526	0.07510877282085704	0.6558917275691352	0.6588543
Row 28	0.43392389657080976	0.3686920160587245	0.12409687244605139	0.4199888465197924	0.7352504562170697	0.4929690
Row 29	0.4882723585707991	0.11705805215800957	0.23358205363640228	0.5827651260215182	0.8431207395758262	0.6996888
Row 30	0.548261022546377	0.5211062464100419	0.7057062643874602	0.9252621217384517	0.4596054073089937	0.1043480
Row 31	0.9432677248469495	0.34324004416743303	0.8406105304888968	0.4775230333024213	0.35734445532634496	0.6946672

Using UIViewRepresentable

```
struct DataGridView: UIViewRepresentable {

  let dataGrid: DataGrid

  func makeUIView(context: Context) -> some UIView {
    RFGridView(dataGrid: dataGrid, missing: "NA")
  }

  func updateUIView(_ uiView: UIViewType, context: Context) {
  }
}
```

RFGridView

```
private final class RFGridView: UIView {

  private var dataGrid: DataGrid
  private var columnHeaderHeight: CGFloat
  private var rowHeaderWidth: CGFloat

  private var rowsView: UICollectionView
  private var columnsView: UICollectionView
  private var contentView: UICollectionView

  private var font: UIFont
  private var missing: String

}
```

Delegate

Links the scrolling together

```
extension RFGridView: UICollectionViewDelegate {
  func scrollViewDidScroll(_ scrollView: UIScrollView) {

    let contentOffset = scrollView.contentOffset

    switch scrollView {
    case contentView:
      rowsView.contentOffset.y = contentOffset.y
      columnsView.contentOffset.x = contentOffset.x
    case columnsView:
      contentView.contentOffset.x = contentOffset.x
    case rowsView:
      contentView.contentOffset.y = contentOffset.y
    default:
      break
    }
  }
}
```

Sections and Items

```
private extension IndexPath {
  init(gridRow: Int, gridColumn: Int) {
    self.init(item: gridColumn, section: gridRow)
  }
  var gridRow: Int { section }
  var gridColumn: Int { item }
}
```

Custom Cell

```
private final class RFGridCell: UICollectionViewCell {
  var label: UILabel!

  override init(frame: CGRect) {
    label = UILabel(frame: frame)
    label.textAlignment = .center
    label.font = UIFont.boldSystemFont(ofSize: 17)
    super.init(frame: frame)
    contentView.addSubview(label)
    contentView.layer.borderWidth = 0.5
    contentView.layer.borderColor = UIColor.black.cgColor
  }

  override func layoutSubviews() {
    super.layoutSubviews()
    label.frame = contentView.bounds
  }

  required init?(coder: NSCoder) {
    fatalError("init(coder:) has not been implemented")
  }
}
```

Data Grid

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
Row 1	0.8875145979230562	0.9351372190158398	0.607382522246032	0.2711921623552447	0.8680654242966968	0.9133490396853611
Row 2	0.16540312055912088	0.7762651818779113	0.5232815669261168	0.3301244727227761	0.9763396062775184	0.6460328861802294
Row 3	0.8863412607957853	0.15440308599008523	0.6345936120492842	0.3077987342703976	0.42632389691042283	0.3736199023633069
Row 4	0.3852115656170759	0.32074692875921795	0.8062370076566603	0.14347815500798133	0.22339120494843945	0.5113009044957652
Row 5	0.8873391914579881	0.6496401887356347	0.45789292587465846	0.21651143576003595	0.8232370452805927	0.0634423908839868
Row 6	0.05743853466146176	0.42125769412897274	0.37575018124872495	0.39485011670793646	0.3188164363191959	0.6643972707142795
Row 7	0.33283813040993615	0.9552314768988057	0.7382969685378761	0.4299682110757317	0.6527922405460066	0.15664578696519216
Row 8	0.04208274456113281	0.9061418261593177	0.13370885558503542	0.12997220342141946	0.9805903959447244	0.12749177251680022
Row 9	0.36966691382287187	0.5797222852733763	0.7657511958995379	0.1286239321996534	0.09151967255115656	0.37429883396657715
Row 10	0.02301468156765285	0.4242589741180278	0.20380200403740278	0.07602488897064263	0.5676909618222302	0.4569268580320698
Row 11	0.48693813624263316	0.8027057913708642	0.2084711651248935	0.4188178361960315	0.4354245789423531	0.2273347381905877
Row 12	0.6297513168496895	0.8294712803049029	0.8871957193489951	0.35904121580082193	0.7302318096196214	0.9665518192633372
Row 13	0.9242246842518836	0.964685148409884	0.24112212052248483	0.6650898119505217	0.6489567218120282	0.7567879396501068
Row 14	0.6524775631827732	0.9996699209125567	0.384165593361673	0.5445649602468654	0.6384929164177298	0.5953229639832008
Row 15	0.497820385709533	0.4823929974515502	0.41581960191691447	0.04697245174885123	0.6003435934026351	0.06852592988747186
Row 16	0.21299216875662608	0.5064521137829239	0.9772071939141598	0.0362737447403807	0.17894525332396016	0.3505338135062511
Row 17	0.9704174726173969	0.7852814824423104	0.8903117944915059	0.3848312593971307	0.8700224218326705	0.5977557843221164
Row 18	0.3066331185005292	0.6722384942150698	0.7357514139539127	0.6934444793822934	0.8559301856302589	0.4629837066167112

Future Improvements

PRs welcome…

- Make accessible from Swift Package Manager
- More full featured data model
- User adjustable column sizes
- Editable cells
- Editable dimensions
- Big data support