

# Web Apps using Package 'shiny'

# Contents

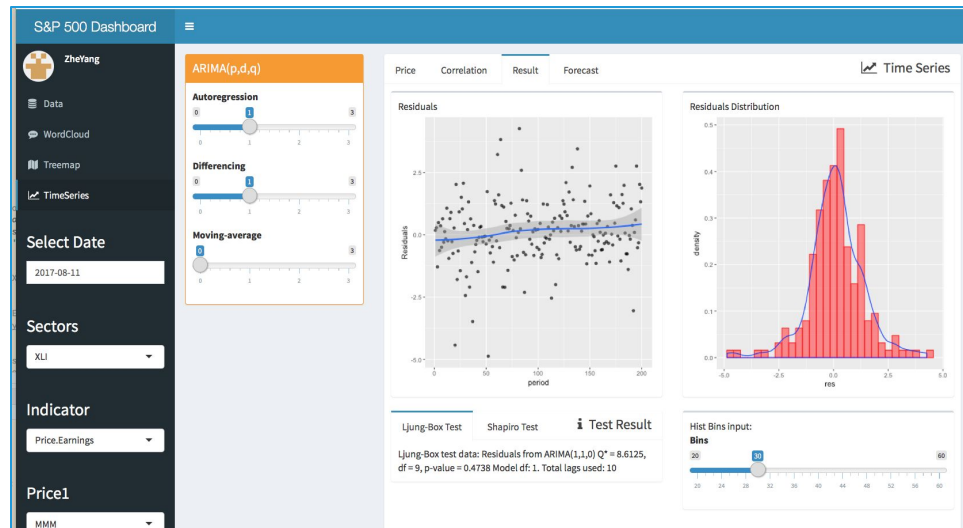
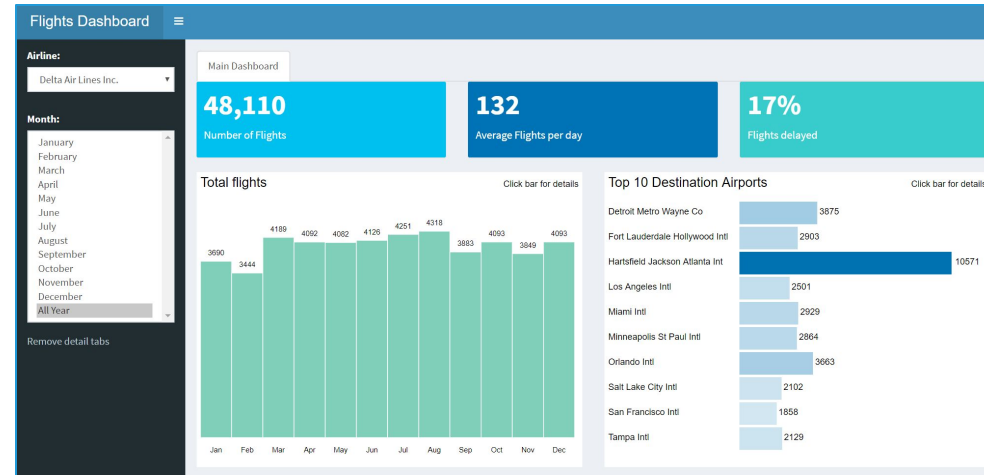
1. Introduction to Shiny
2. Understanding the Structure of Shiny App!
3. Layout of the App
4. How UI and SERVER Interact?

# Introduction to Shiny

- Shiny is an R application from RStudio that makes it incredibly easy to build interactive web applications (apps) in R.
- It provides elegant and powerful web framework for building interactive web applications using R.
- Shiny apps are easy to write. No web development skills are required.
- It helps you turn your analyses into interactive web applications without requiring HTML, CSS, or JavaScript knowledge, but if you know them you can also extend your Shiny apps with CSS themes, htmlwidgets, and JavaScript actions.
- **Shiny is an extremely powerful tool.** Unlike the more traditional workflow of creating presentations, you can now create an app that allows your readers to change the assumptions underlying your analysis and see the results immediately. It **reduces the effective time by more than 80%** and makes documentation fun and interactive!

# Introduction to Shiny

Here's how a Shiny dashboard looks.



Let us now see the basic steps for making your own app

# New to Shiny!

First you'll have to install the shiny package from CRAN on RStudio.

# Install and load the package “shiny”

```
install.packages("shiny")  
library(shiny)
```

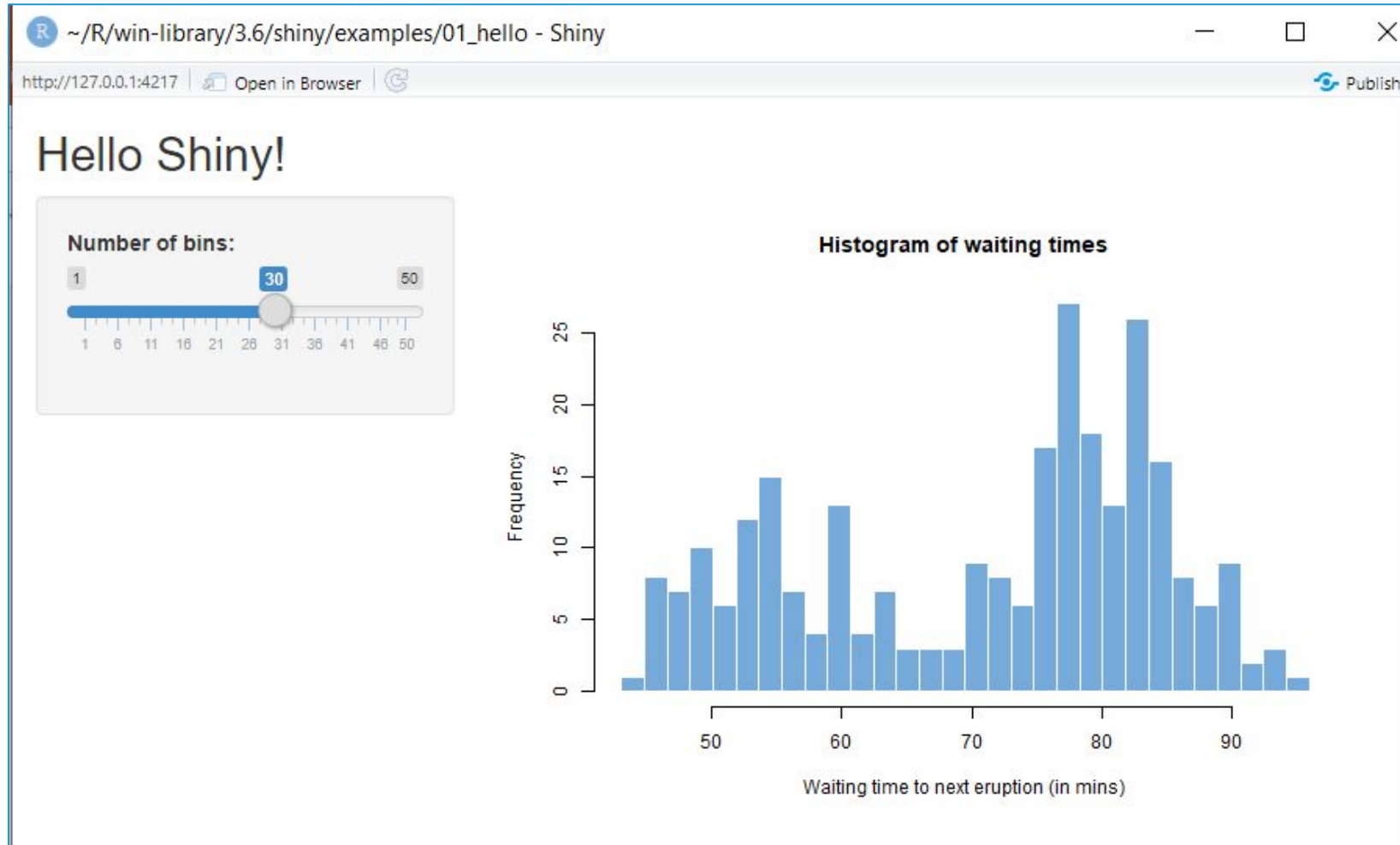
The **shiny** package has some inbuilt examples that each demonstrates how shiny works. Run the **Hello Shiny** example with the code given below :

```
runExample("01_hello")
```

This example displays a histogram of R's faithful dataset with a configurable number of bins. Number of bins can be changed with the slider bar, and the app will immediately respond to the input given by user.

# New to Shiny!

Hello Shiny Example:



# Understand the Structure of Shiny App!

A Shiny app consists of 2 main components :

- User Interface script (ui.R)
- Server script (server.R)


- The **user - interface script** controls the layout and the appearance of your app. It is defined and stored with the name **ui.R**.
- The **server script** contains the instructions that your computer needs, to build the app. It is defined and stored with the name **server.R**
- The user interface and the server interact with each other through input and output objects.
- The instructions for creating input objects are in the UI.

# Understand the Structure of Shiny App!

- Create a new directory named **Myshinyapp** in your working directory. Then copy - paste your **ui.R** and **server.R** scripts within **Myshinyapp**.
- You can check your working directory by **getwd()** function and set it where you'd like by **setwd()** or manually on R studio by clicking on Session.

Alternate way to create a Shiny app is by defining UI and SERVER in a single file named app.R. This file must return an object created by the shinyApp() function like this :

```
ui<-fluidPage()  
server<-function(input,output){ }  
  
shinyApp(ui,server)
```



This method is more appropriate for smaller applications.



Note: Both the ui and server files work in conjunction and should be in the same working directory along with any input datasets (if any) used in your app.



# Layout of the app

The user interface consists of 2 basic Panels:

- Title Panel
- Sidebar Layout
  - Sidebar panel
  - Main panel

The basic structure of the user-interface and the server script is:

# ui.R

```
library(shiny)
fluidPage(
  titlePanel("This is the title panel"),
  sidebarLayout(
    sidebarPanel("This is the sidebar panel"),
    mainPanel("This is the main panel")
  )
)
```

# server.R

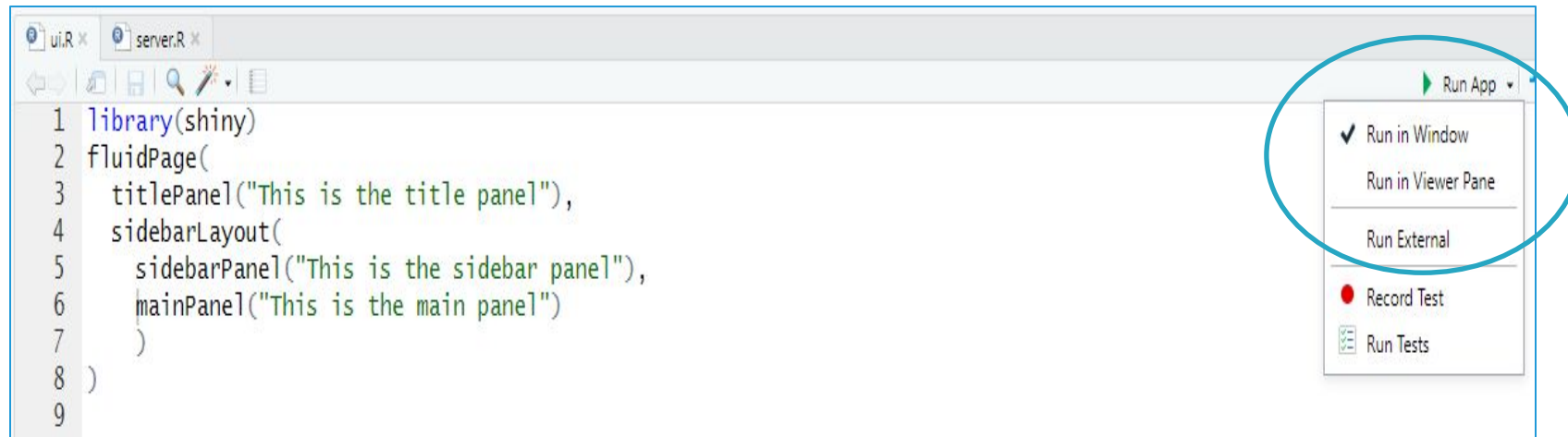
```
function(input,output){
}
```

- ❑ The **ui.R** script uses the **fluidPage()** function to make the app page responsive.
- ❑ **fluidPage()** can enable the app to work on any platforms without affecting the layout of the app. It automatically adjusts to the dimensions of the browser window.

We define server by creating a function and arguments input and output are supplied to refer to inputs and outputs defined in **ui.R**

# Layout of the app

Notice the green **Run App** button appear when the file is saved. This button also allows you to control whether the app runs in a browser window, in the RStudio Viewer pane, or in an external RStudio window.



Click on 'Run App' to run the app.

# Layout of the app



Most of the inputs of the shiny app are presented in the sidebar panel with the output in the main panel. However this is subject to change according to the requirement of the app.

# More Features that can be added to Layout

## – `fluidRow()` & `column()`

- **Sidebar Layout:**

Our **Myshinyapp** app has a layout which provides a sidebar for inputs and a large main area for output.

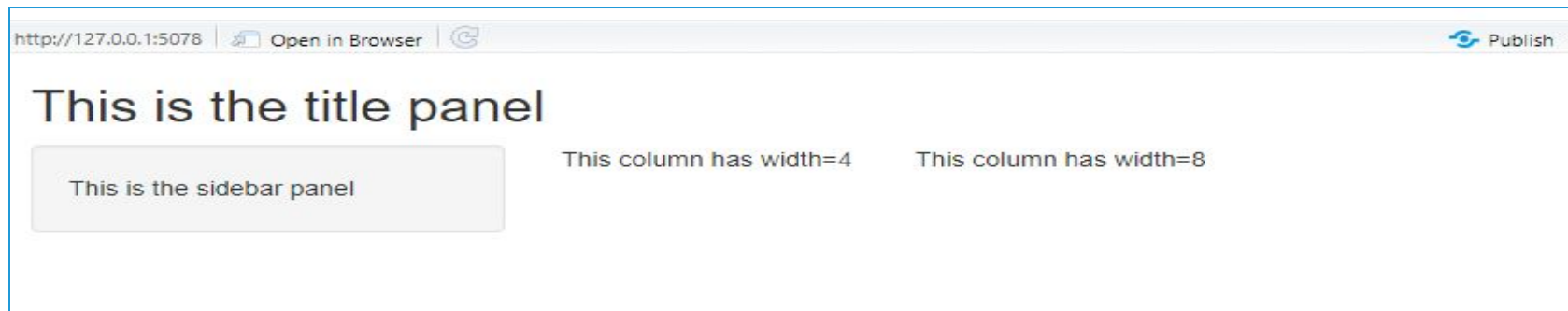
- **Fluid Grid System:**

To create a layout based on the fluid system you use the `fluidPage()` function which we have already seen in our **Myshinyapp** example. This grid system uses 12 column grid layout which flexibly be subdivided into rows and columns. To create rows use `fluidRow()` function and include columns defined by the `column()` function. Grid layouts can be used anywhere within a `fluidPage()` and can even be nested within each other.

# More Features that can be added to Layout

## – fluidRow() & column()

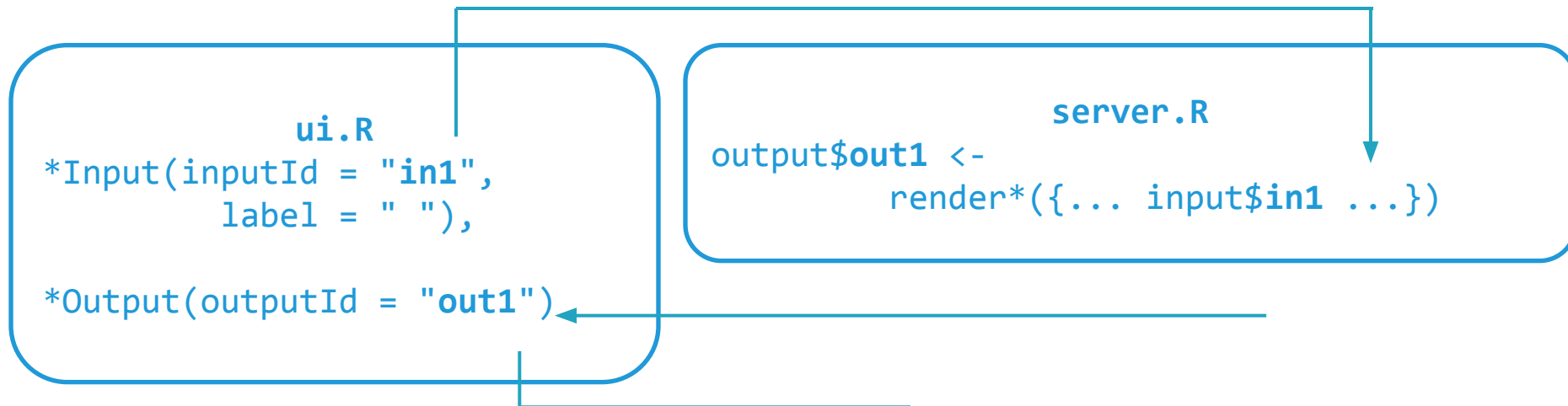
```
fluidPage(  
  titlePanel("This is the title panel"),  
  sidebarLayout(  
    sidebarPanel("This is the sidebar panel"),  
    mainPanel(fluidRow(  
      column(4,"This column has width=4"),  
      column(8,"This column has width=8")  
    )  
  ))  
)
```



You can move the columns to the right by specifying the number of columns to offset to **offset=** argument. Each unit of offset increases the left-margin of a column by a whole column.

# How UI and SERVER Interact?

The user interface and the server interact with each other through input and output objects.



- **Input objects** are a way for users to interact with Shiny App. Inputs in UI are given by adding widgets with functions like **`selectInput()`** or **`radioButtons()`** and are used within **`render*()`** functions in the server object to create output objects.
- **Output objects** are placed in the UI using output functions like **`plotOutput()`** or **`textOutput()`**.

# Web Apps using Package 'shiny' - II

# Contents

1. Case Study
2. App Design
3. Create an Empty Shiny App
4. Build the Basic UI
5. Add Widgets to Your App
6. Add Placeholders for Output in UI
7. Build the Object for Output in SERVER
8. Launch Your App!
9. Share Your App!
10. More Shiny App Features: `tabsetPanel()`, `Package DT`, `global.R`, `helper.R`, `reactive()`



# Case Study

## Background

- A Telecom company wants to analyse the data to provide solution to managerial problems.

## Objective

To create a shiny application which will

- Identify usage pattern of the customers across various demographics
- Identify volume of business generated across various demographics
- Develop a dashboard for the in-house analytics team
- Design efficient ways to communicate analysis insights with the marketing team

# Case Study

## Data Description

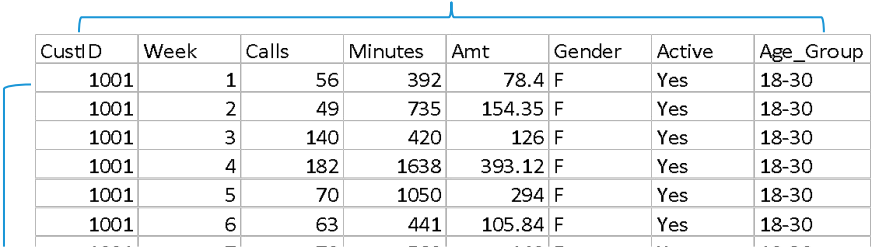
Telecom data for 24 weeks was provided by the client at customer level.

Telecom data:

- 21902 rows & 5 columns
- Contains customer level information like Age\_Group, Gender, and whether the customer has been active in the past six months.
- Contains Number of Calls, Time Spent on Calls (in Minutes) and the Amount Spent (in Rs.) which are recorded for the past 6 months (24 weeks) for each customer.

# Data Snapshot

## Variables

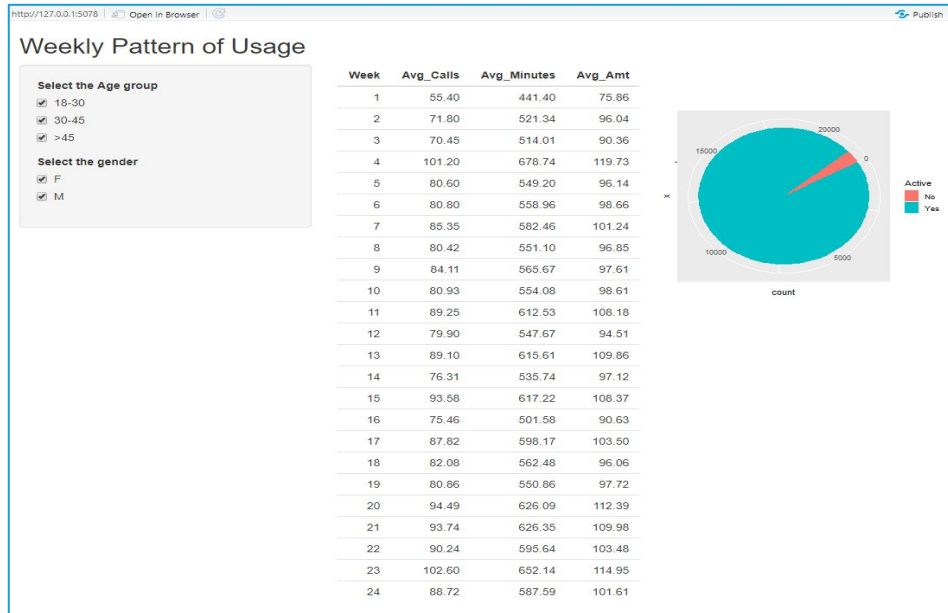


CustID	Week	Calls	Minutes	Amt	Gender	Active	Age_Group
1001	1	56	392	78.4	F	Yes	18-30
1001	2	49	735	154.35	F	Yes	18-30
1001	3	140	420	126	F	Yes	18-30
1001	4	182	1638	393.12	F	Yes	18-30
1001	5	70	1050	294	F	Yes	18-30
1001	6	63	441	105.84	F	Yes	18-30

Columns	Description	Type	Measurement	Possible values
CustID	Customer ID	Integer	-	-
Week	Week Number	Integer	1 – 24	24
Calls	Number of Calls	Integer	-	Positive values
Minutes	Time Spent on Calls in Minutes	Integer	minutes	Positive values
Amt	Amount Spent (in Rs.)	Numeric	Rs.	Positive values
Gender	Gender of customer	Character	M, F	2
Active	Active status	Character	Yes, No	2
Age_Group	Age Group	Character	18-30, 30-45, >45	3

# App Design

We'll now create the app which will look like this. It will allow you to see a table and pie chart. Table will show week wise average calls, minutes & amount for the chosen **Age Group** and **Gender** and a pie chart will represent active customers information graphically.



Working of the app:

Whenever the user changes the choice for Age Group and Gender, the output will be rebuilt using the new value.

# Create an Empty Shiny App

All Shiny apps follow the same template:

# ui.R

```
library(shiny)
fluidPage()
```

# server.R

```
function(input,output){
}
```

- Save the **ui.R** and **server.R** in a folder which is saved in your working directory. Save the folder with the name “**Myshinyapp**”.
- After saving the file, RStudio should recognize that this is a Shiny app and you should see the usual ‘**Run App**’ button’.

# Build the Basic UI

```
# ui.R
```

```
library(shiny)
fluidPage(
  titlePanel("Weekly Pattern of Usage")
)
```

**titlePanel()** adds a title on the top left corner of the app and sets it as “official” title of the page.

- Add a layout:

```
# Add the following code after titlePanel()
```

```
sidebarLayout(
  sidebarPanel(),
  mainPanel()
)
```

- ❑ **sidebarLayout()** provides a simple two-column layout with a smaller sidebar and a larger main panel.
- ❑ All the arguments inside **fluidPage()** need to be separated by commas.

The layout of our app will be such that all the inputs (using widgets) that the user can manipulate will be in the sidebar, and the results will be shown in the main panel on the right.

Now we'll define some widgets in our sidebar panel.



Note: when adding **sidebarLayout()** put ‘,’ after **titlePanel()**,

# Add Widgets to Your App

Widget:

- Control widgets are web elements that users can interact with. Widgets provide input to your Shiny app. As the input provided to the widget changes, the value also changes.
- Eg: In our ui.R script, the buttons where you select the variables are the widgets. They let the user select the inputs for which the table and pie chart is provided in the main panel.
- The widget we have used here is `checkboxGroupInput()` which gives user a list of choices to select from, the code for which is :

```
checkboxGroupInput(inputId="age",  
                label="Select the Age group",  
                choices=c("18-30", "30-45", "45-60", "60-70"),  
                selected=c("18-30", "30-45"),  
checkboxGroupInput(inputId="gender",  
                label="Select the gender",  
                choices=c("F", "M"),  
                selected=c("F", "M"))
```

Add the above code in `sidebarPanel()`

- ❑ **inputId=** is the name with which the widget is stored. This is for internal computational purposes.
- ❑ **label=** is used just for user-interface, it can also be blank.
- ❑ **choices=** is the list of values to select from.
- ❑ **selected=** is the initially selected value.

# Standard Shiny Widgets

Function	Widget
<code>actionButton</code>	Action Button
<code>checkboxGroupInput</code>	A group of check boxes
<code>checkboxInput</code>	A single check box
<code>dateInput</code>	A calendar to aid date selection
<code>dateRangeInput</code>	A pair of calendars for selecting a date range
<code>fileInput</code>	A file upload control wizard
<code>helpText</code>	Help text that can be added to an input form
<code>numericInput</code>	A field to enter numbers
<code>radioButtons</code>	A set of radio buttons
<code>selectInput</code>	A box with choices to select from
<code>sliderInput</code>	A slider bar
<code>submitButton</code>	A submit button
<code>textInput</code>	A field to enter text



# Standard Shiny Widgets

## Basic widgets

### Buttons

Action

Submit

### Single checkbox

☒ Choice A

### Checkbox group

☒ Choice 1

☐ Choice 2

☐ Choice 3

### Date input

2014-01-01

### Date range

2014-01-24 to 2014-01-24

### File input

Choose File No file chosen

### Help text

Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

### Numeric input

1

### Radio buttons

☒ Choice 1

☐ Choice 2

☐ Choice 3

### Select box

Choice 1

### Sliders

0 50 100

0 25 75 100

### Text input

Enter text...

# Add Placeholders for Output in UI

- After creating all the inputs, we should add elements to the UI to display the outputs. Outputs can be any objects that R creates such as a plot, a table, or text.
- We will add a placeholder for the outputs in the UI that will determine where the output will be displayed and what its ID is.
- Each output needs to be built in the server code to be able to respond to the users input.

# Add Placeholders for Output in UI

# Add placeholder for table and piechart in the mainPanel()

```
mainPanel(tableOutput("weeks"),plotOutput("pie"))
```



This will add two placeholders in the UI for a table and a pie named **weeks** and **pie** respectively.

- Notice that `tableOutput()` and `plotOutput()` takes an argument, the character string **"weeks"** and **"pie"** respectively.
- Each of the `*Output` functions require a single argument : a character string that Shiny will use as the name to connect to server.

# Output Functions for UI Script

Each function creates a specific type of output	
Output Function	creates
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
uiOutput	raw HTML
verbatimTextOutput	text

# Check Point: What Our App Looks Like After Implementing UI

So far our complete app looks like this

```
# ui.R
```

```
library(shiny)
fluidPage(
  titlePanel("Weekly Pattern of Usage"),
  sidebarLayout(
    sidebarPanel(
      checkboxGroupInput(inputId="age",
        label="Select the Age group",
        choices=c("18-30", "30-45", ">45"),
        selected=c("18-30", "30-45", ">45")),
      checkboxGroupInput(inputId="gender",
        label="Select the gender",
        choices=c("F", "M"),
        selected=c("F", "M"))
    ),
    mainPanel(
      fluidRow(
        column(6, tableOutput("weeks")),
        column(6, plotOutput("pie"))
      )
    )
  )
#) server.R
function(input,output){}
```

# Build the Objects for Output in SERVER

- Recall that we created two output placeholders: weeks(a table) and pie(a plot).
- **server.R** will include the code which will tell Shiny what kind of table or plot to display.
- There are three rules to build an output in Shiny :
  - Save the output object into the **output** list (remember that every server function has an **output** argument) which should match the ID defined for your object in your UI script.
  - Build the object with a **render\*** function, where **\*** is the type of output
  - Access input values using the **input** list

# Build the Objects for Output in SERVER

```
# server.R
```

```
# Import Telecom data.
```

```
teledata <- read.csv("Telecom data.csv",header = TRUE)
```

```
# Load required libraries
```

```
library(ggplot2)
```

```
library(dplyr)
```

```
function(input,output){
```

```
  output$weeks <- renderTable({
```

```
    xy <- subset(teledata, (Age_Group %in% input$age) & (Gender %in%  
input$gender))
```

```
    summarise(group_by(xy,Week),Avg_Calls=mean(Calls),Avg_Minutes=mean(Min  
utes),Avg_Amt=mean(Amt))
```

```
  })
```

Here, we are saving the output list (**output\$weeks** & **output\$pie**), using a render\* function to build the output(**renderTable({})** & **renderPlot({})**) and accessing an input value (**input\$age** & **input\$gender**)

**input\$age** and **input\$gender** stores the value provided to the widget by the user, which is supplied to **Age\_Group** and **Gender** variables to create subset **xy**.

# Build the Objects for Output in SERVER

# server.R continued

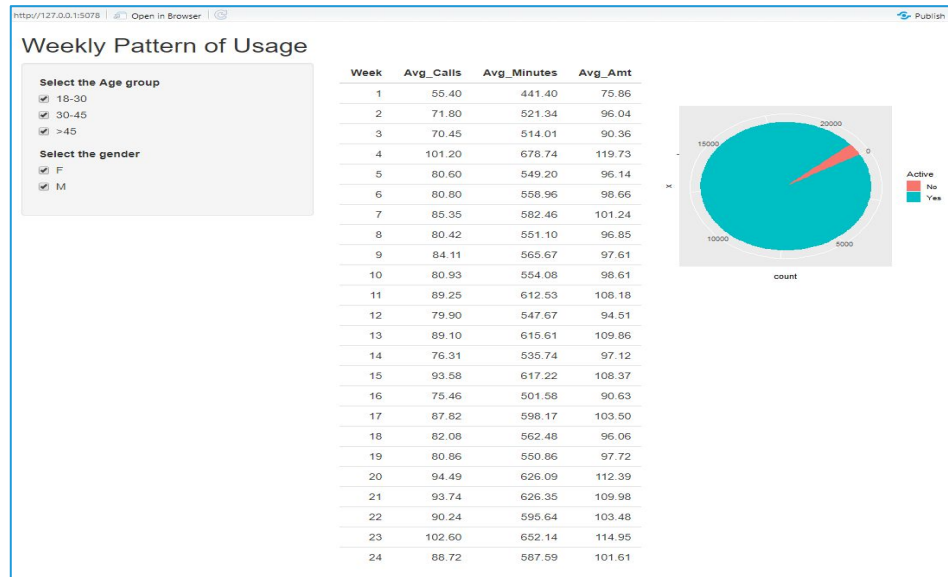
```
output$pie <- renderPlot({  
  xy <- subset(teledata, (Age_Group %in% input$age) & (Gender %in%  
input$gender))  
  pie <- ggplot(xy, aes(x="",fill=Active))+  
    geom_bar(width = 1)+  
    coord_polar(theta = "y", start = pi/3)  
  pie  
})  
}
```



# Launch Your App!

You can launch your app by clicking on 'Run App' button or by running the below command in your R console

```
runApp("Myshinyapp")
```



Note : If the directory is not set, then give the complete path in runApp() function while executing it in R console.

# Launch Your App!

You can also choose to launch your app, highlighting the code each time each time it is running by adding the argument **display="showcase"** in the **runApp()** command:

```
runApp("Myshinyapp",display="showcase")
```

Weekly Pattern of Usage

Select the Age group

- ☒ 18-30
- ☒ 30-45
- ☒ >45

Select the gender

- ☒ F
- ☒ M

Week	Avg_Calls	Avg_Minutes	Avg_Amt
1	55.40	441.40	75.86
2	71.80	521.34	96.04
3	70.45	514.01	90.36
4	101.20	678.74	119.73
5	80.60	549.20	96.14
6	80.80	558.96	98.66
7	85.35	582.46	101.24
8	80.42	551.10	96.85
9	84.11	565.67	97.61
10	80.93	554.08	98.61
11	89.25	612.53	108.18
12	79.90	547.67	94.51
13	89.10	615.61	109.86
14	76.31	535.74	97.12
15	93.58	617.22	108.37
16	75.46	501.58	90.63
17	87.82	598.17	103.50
18	82.08	562.48	96.06
19	80.86	550.86	97.72
20	94.49	626.09	112.39
21	93.74	626.35	109.98
22	90.24	595.64	103.48
23	102.60	652.14	114.95
24	88.72	587.59	101.61

Active

- ☒ No
- ☒ Yes

```
serverR    ui.R

# Import Telecom data.
teledata <- read.csv("Telecom data.csv",header = TRUE)

# Load required libraries
library(ggplot2)
library(dplyr)

function(input,output){

  output$weeks <- renderTable({

    xy <- subset(teledata, (Age_Group %in% input$age) & (Gender %in% input$gender))
    summarise(group_by(xy,Week),Avg_Calls=mean(Calls),Avg_Minutes=mean(Minutes),Avg_Amt=mean(Amt))

  })

  output$pie <- renderPlot({

    xy <- subset(teledata, (Age_Group %in% input$age) & (Gender %in% input$gender))
    pie <- ggplot(xy, aes(x="",fill=Active)) +
      geom_bar(width = 1) +
      coord_polar(theta = "y", start = pi/3)
    pie
  })

}
```

# Share Your App!

Once you have made the app, you can share it with others easily. **There are two options:**

Share your Shiny app as files:

1. **Share a copy of server.R and ui.R files**, as well as any supplementary materials used in your app (E.g. data files, images, etc.) with anyone, but it will work only if your users have R on their computer. The user can place these files in their working directory and launch the app in R with the same commands you used on your computer.

```
runApp("Myshinyapp")
```

# Share Your App!

## 2. Share as a Web Page:

If you want to share your app with a number of people then the most easy and user friendly way is to share as a web page.

RStudio provides a platform called **shinyapps.io** which lets you upload your app.

### To upload the app:

- Go to <http://www.shinyapps.io/> and create a free or professional account.
- Make sure all your app files are in an isolated folder saved in your working directory.
- In order to upload the app on shinyapp.io, you need to install package devtools and rsconnect.

# Share Your App!

Run following command to upload the app.

```
install.packages("devtools")
install.packages("rsconnect")
library(rsconnect)

rsconnect::setAccountInfo(name=<name>, token=<token>, secret=<token>)

rsconnect::deployApp("Myshinyapp", account=<account name>)
```

- ❑ **rsconnect::setAccountInfo()** configures a ShinyApps account for publishing from this system.
- ❑ **name=** Name of account to save
- ❑ **token=** User token for the account
- ❑ **secret=** User secret for the account
- ❑ The **deployApp()** function automatically syncs up to the Shinyapps.io server, and opens the shinyapps.io website on your browser. Once your app is uploaded, it is on the internet and can be viewed by anybody with access to it.

# More Shiny App Features – tabsetPanel()

## 1. Multiple tabs in UI:

Using `tabsetPanel()` in `ui.R` script you can add multiple tabs and navigate between them.

```
shinyUI(fluidPage(  
  tabsetPanel(  
    tabPanel("Tab 1", "Hello there!"),  
    tabPanel("Tab 2", " "),  
    tabPanel("Tab 3", " ")  
  )  
))
```



# More Shiny App Features – Package DT

## 2. Beautiful and Interactive Tables:

- With `tableOutput()` + `renderTable()`, shiny creates static tables.
- Using package **DT**, you can replace the default table with a much sleeker and interactive table.
- To use package **DT**, first install and load the package and then replace `tableOutput()` with `dataTableOutput()` & replace `renderTable()` with `renderDataTable()`.

# More Shiny App Features – global.R

## 3. Global Objects:

- If there are objects that you want to have available to both `ui.R` and `server.R`, you can place them in `global.R` and call it with `'source("global.R")'` command at the beginning of `ui.R` and `server.R`.
- Defining the objects in `global.R` loads them into global environment of R session.
- `global.R` can also be used to add codes which are repeating in `server.R`. This is for computational ease and also provides a less messy looking `server.R` file.



# More Shiny App Features – helper.R

## 3. **Helper Objects:**

- The helpers script file supports the SERVER and UI files to continue working smoothly. They can help you to install a package, calculate intermittent values for analysis and plots.
- Coding that needs to be repeated or have to be taken from a common source is put in helpers.R file. This is for computational ease. It also results in a less messy server.R file.
- The `helpers.R` file is accessed through the use of `source("helpers.R")` command in the server.R file.

# More Shiny App Features – reactive()

## 3. **Reactive function :**

- **reactive()** function in shiny is used when one wants the outputs to be changed as the input selection are changed at user-interface level.
- When a user changes something in the app, the app sends the new input value to the server, which triggers a flush event and invalidates all the input's dependents. Then all the reactive endpoints (like observers and outputs) are refreshed, which may also trigger a refresh of the reactive conductors (or reactive expressions defined with the reactive function).

# Shiny – Case Study (Telecom App)

# Contents

1. Making of a Shiny App – Recap
2. The Telecom App – Structure
3. User Interface – ui.R
4. Server – server.R
5. Deploy the App

# Making of a Shiny App - Recap

- Structure of the app consists of a user interface and server script. The user interface script is used for layout and appearance purposes.
- It is generally saved as **ui.R** file on the working directory. While the server script contains the instructions that the computer needs to build an app. The server script is usually named as **server.R** and saved in the working directory for the app being developed.

# Data Snapshot

Telecom Data

Variables

CustID	Week	Calls	Minutes	Amt	Gender	Active	Age_Group
1001	1	56	392	78.4	F	Yes	18-30
1001	2	49	735	154.35	F	Yes	18-30
1001	3	140	420	126	F	Yes	18-30
1001	4	182	1638	393.12	F	Yes	18-30

Columns	Description	Type	Measurement	Possible values
CustID	Customer ID	Integer	-	-
Week	Week Number	Integer	1 – 24	24
Calls	Number of Calls	Integer	-	Positive values
Minutes	Time Spent on Calls in Minutes	Integer	minutes	Positive values
Amt	Amount Spent (in Rs.)	Numeric	Rs.	Positive values
Gender	Gender of customer	Charater	M, F	2
Active	Active status	Charater	Yes, No	2
Age_Group	Age Group	Charater	18-30, 30-45, >45	3

1001	24	70	980	264.6	F	Yes	18-30
------	----	----	-----	-------	---	-----	-------



Here we continue to use previous data Telecom Data.

# The Telecom App Structure

- Title panel has name of the application.
- Sidebar layout has sidebar panel with input widgets and main panel with outputs.
- Main panel of the app has four tabs:
  1. **Data:**
    - Table for weekly usage pattern of the customers across various demographics where one imports data & can also download the table.
    - Pie chart of Active customers
  2. **Weekwise Summary:** Summary of data.
  3. **Weekly Usage:** Line chart showing weekly usage pattern of the customers across various demographics.
  4. **Volume of Business.**

Let's have a look at the app design first and then begin to create our app.

# App Design – Tab 1

On opening the App & before importing data :

http://127.0.0.1:3290 | Open in Browser | Publish

## Weekly Pattern of Usage

Select the Age group

☒ 18-30

☒ 30-45

☒ >45

Select the gender

☒ F

☒ M

## Customer pattern

Data

Weekwise Summary

Weekly usage

Volume of Business

Browse...

No file set

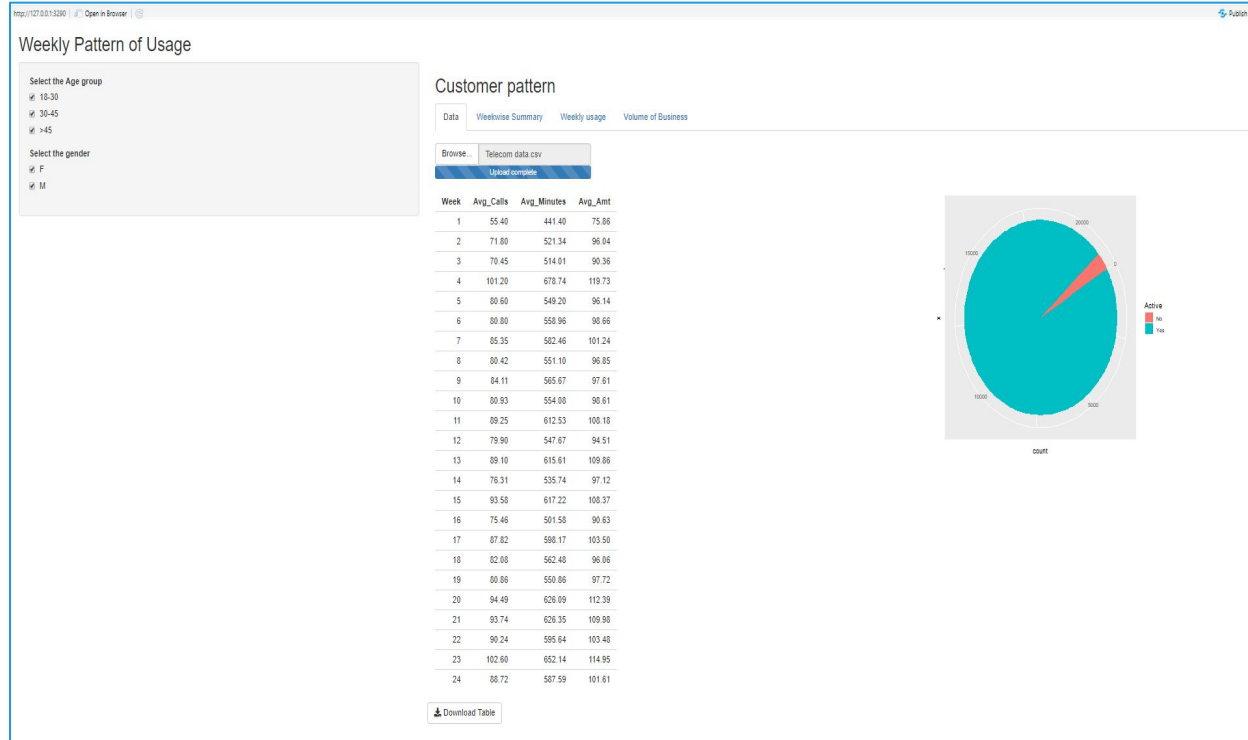
Error: 'file' must be a character string or connection

Error: 'file' must be a character string or connection



Download Table



# App Design – Tab 1



# App Design – Tab 2

<http://127.0.0.1:3290> | [Open in Browser](#) |   Publish ▾

## Weekly Pattern of Usage

**Select the Age group**

☒ 18-30

☒ 30-45

☒ >45

**Select the gender**

☒ F

☒ M

**Select your week number**

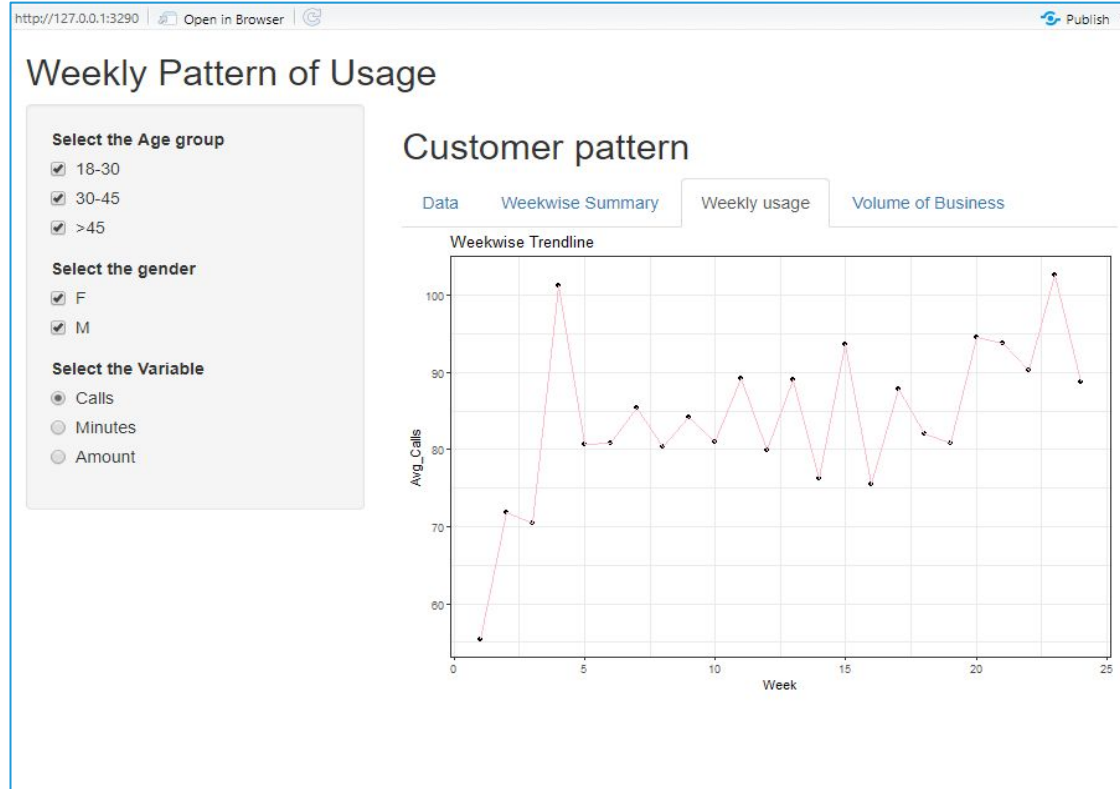
1 ▾

## Customer pattern

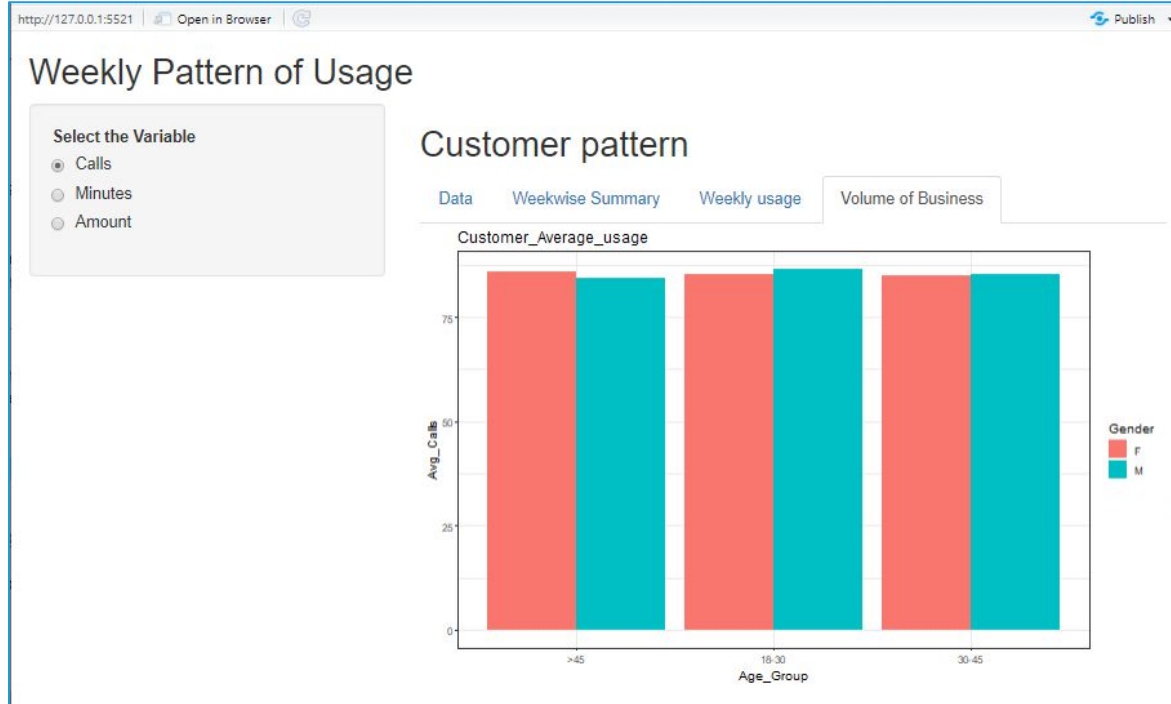
[Data](#) | [Weekwise Summary](#) | [Weekly usage](#) | [Volume of Business](#)

Calls	Minutes	Amt	Gender	Active	Age_Group
Min. : 7.00	Min. : 28.0	Min. : 2.52	F:72	No : 8	>45 : 0
1st Qu.:56.00	1st Qu.: 178.5	1st Qu.: 26.60	M:68	Yes:132	18-30:68
Median :63.00	Median : 392.0	Median : 49.84			30-45:72
Mean :55.40	Mean : 441.4	Mean : 75.86			
3rd Qu.:64.75	3rd Qu.: 630.0	3rd Qu.:117.60			
Max. :70.00	Max. :1050.0	Max. :274.05			

# App Design – Tab 3



# App Design – Tab 4



Note : Always have the R codes ready before starting shiny app coding.

# ui.R

```
# ui.R
```

```
#Load the required packages.
```

```
library(shiny)
```

```
library(shinyjs)
```

- ❑ Package **shiny** is used for creating Shiny App.
- ❑ Package **shinyjs** allows us to perform JavaScript operations in Shiny apps that greatly improves our apps without having to know any JavaScript.

```
fluidPage(
```

← **fluidPage()** makes the app page responsive.

```
useShinyjs(),
```

← Every time any function from the **shinyjs()** is used, **useShinyjs()** has to be mentioned.

# ui.R

# ui.R continued

```
titlePanel("Weekly Pattern of Usage"),
  sidebarLayout(
    sidebarPanel(
      hidden(checkboxGroupInput("age", "Select the Age
group", choices=c("18-30", "30-45", ">45"), selected=c("18-30", "30-45", ">4
5"))),
      hidden(checkboxGroupInput("gender", "Select the
gender", choices=c("F", "M"), selected=c("F", "M"))),
```

The **titlePanel** and the **sidebarPanel** consists of widgets for age, gender, week number and variable.

- ☐ **hidden()** function from **shinyjs** package makes the Shiny tag invisible when Shiny app starts.
- ☐ **checkboxGroupInput()** creates a group of checkboxes.
- ☐ **First argument** is the ID with which **server.R** will identify the input
- ☐ **Second argument** is label for the widget
- ☐ **choices=** is for the list of values to show check boxes for
- ☐ **selected=** is for the value that should be initially selected.

# ui.R

```
hidden(  
  selectInput("ID","Select your week number",choices=c(1:24))),  
hidden(  
  radioButtons("var",label="Select the Variable",  
    choices=c("Calls"=2,"Minutes"=3,"Amount"=4)))  
)
```

- ❑ **selectInput()** creates a select list control that can be used to choose a single or multiple items from a list of values.
- ❑ **radioButtons()** creates a set of radio buttons used to select an item from a list.

- With this **sidebarPanel()** body ends and now we will define **mainPanel()**.

# ui.R

# Define the main panel with various tabs

```
mainPanel(  
  h2("Customer pattern"),  
  tabsetPanel(id="tabs",type="tabs",  
    tabPanel("Data",value="data",  
      fluidRow(column(width=4,  
        fileInput("file1","", multiple = TRUE,  
          accept = c("text/csv",  
"text/comma-separated-values,text/plain", ".csv")  
        )  
      )),  
      fluidRow(column(6,tableOutput("weeks")),  
        column(6,plotOutput("pie"))),  
      fluidRow(downloadButton('download',"Download Table",  
class = "butt"))  
    ),  
  )  
)
```



# ui.R

# Define the main panel with various tabs

```
tabPanel("Weekwise Summary",value="summa", verbatimTextOutput("sum")),  
tabPanel("Weekly usage",value="Usag",plotOutput("usage")),  
tabPanel("Volume of Business",value="vol",plotOutput("bar"))  
  )  
)  
))
```

- ❑ **mainPanel()** creates a main panel containing output elements that can in turn be passed to **sidebarLayout()**.
- ❑ HTML content can be added to Shiny app by placing it inside a **\*Panel()** function
- ❑ **h2()** is a Shiny's HTML tag function which creates a second level header.
- ❑ **tabsetPanel()** is to create multiple tabs with **tabPanel()**. It takes the arguments as a set of **tabPanels**, **id=** for the server logic to determine which of the current tabs is active and **type=** for the type of tabs.
- ❑ In **tabPanel()** first argument is the display title for the tab,
- ❑ **value=** value that should be sent when **tabsetPanel()** reports that this tab is selected,
- ❑ **last argument** is the placeholder for the output.

# server.R

# Load the required packages

```
library(ggplot2)
library(dplyr)
library(reshape2)
```

```
function(input,output,session){ ←
```

Defining the **function** and **session** argument for shiny server. This is for the use of **ObserveEvent()** command & **shinyjs()** function.

# Coding for tab1 i.e. Data

```
# import data
```

```
imported_data <- reactive({ ←
```

**reactive()** function helps executing codes based on the data imported.

```
df <- read.csv(input$file1$datapath,header = TRUE)
df$Gender <- as.factor(df$Gender)
df$Active <- as.factor(df$Active)
df$Age_Group <- as.factor(df$Age_Group)
return(df)
})
```

# server.R

# Coding for tab1 i.e. Data

```
week_Avg <- reactive({
  teledata <- imported_data()

  xy <- subset(teledata, (Age_Group %in% input$age) & (Gender %in%
input$gender))

  summarise(group_by(xy, Week), Avg_Calls=mean(Calls), Avg_Minutes=mean(Min
utes), Avg_Amt=mean(Amt))

})

output$weeks<-renderTable({
  week_Avg()
})
```

- ❑ **renderTable()** displays table as an output.
- ❑ **xy** is the subset of **imported\_data()** where the age and gender inputs are taken from the user through the widgets.
- ❑ **summarise(group\_by())** is displaying the weekly Average no. calls, Average time spent on calls (in Minutes) and Average Amount spent by the customers.

# server.R

# Coding for tab1 i.e. Data

```
output$download <- downloadHandler(  
  filename = function(){  
    paste("Weekly Summary.csv")  
  },  
  content = function(file){  
    write.csv(week_Avg(), file, row.names = FALSE)  
  },  
  contentType = "text/csv"  
)
```



- **downloadHandler()** allows content from the Shiny application to be made available to the user as file downloads.
- **filename=** A string of the filename, including extension, that the user's web browser should default to when downloading the file; or a function that returns such a string.
- **content=** A function that takes a single argument file that is a file path (string) of a nonexistent temp file, and writes the content to that file path.
- **contentType=** A string of the download's content type, for example "text/csv" or "image/png"

# server.R

```
# Coding for tab1 i.e. Data
```

```
# Pie Chart representing Active Status of Customers :
```

```
output$pie <- renderPlot({  
  teledata <- imported_data()  
  
  xy <- subset(teledata, (Age_Group %in% input$age) & (Gender %in%  
input$gender))  
  
  pie <- ggplot(xy, aes(x="", fill=Active))+  
    geom_bar(width = 1)+  
    coord_polar(theta = "y", start = pi/3)  
  pie  
})
```

# server.R

# Coding for tab2 i.e Weekwise Summary

```
output$sum<-renderPrint({  
  teledata <- imported_data()  
  
  summary(subset(teledata,(Week %in% input$ID) & (Age_Group %in%  
input$age) & (Gender %in% input$gender), select=c(-CustID,-Week)))  
  
  })
```

- ❑ Here we have defined the function for giving summary as the output **"as-it-is "**, hence used **renderPrint()** and not **renderText()**.
- ❑ It will print the **summary()** of the subset of **teledata** where the age, gender and week inputs are taken from the user through input widgets.

# server.R

# Coding for tab3 i.e. Weekly Usage Summary

```
output$usage<-renderPlot({
  teledata <- imported_data()
  xy<-subset(teledata,(Age_Group %in% input$age) & (Gender %in%
input$gender))

  xyz <- as.data.frame(summarise(group_by(xy,Week),
                                Avg_Calls=mean(Calls), Avg_Minutes=mean(Minutes),
                                Avg_Amt=mean(Amt)))

  qplot(x=Week, y=xyz[,as.numeric(input$var)],data=xyz, xlab="Week",
        ylab=paste(names(xyz[as.numeric(input$var)])),
        main="Weekwise Trendline")+
    geom_line(size=0.7,colour="pink") + theme_bw()
})
```

- ❑ **renderPlot()** displays a plot as an output.
- ❑ **xy** is the subset of **teledata** which uses the age, gender taken from the user through the widgets.
- ❑ **xyz** has the weekly Average no. calls, Average time spent on called (in Minutes) and Average Amount spent by the customers.
- ❑ **qplot()** is displaying line chart representing **xyz** for the selected variable.

# server.R

# Coding for tab4 i.e. Volume of Business

```
output$bar <- renderPlot({  
  teledata <- imported_data()←  
  
  xy<-subset(teledata,(Age_Group %in% input$age) & (Gender %in%  
  input$gender))  
  
  xyz <- as.data.frame(summarise(group_by(xy,Week),  
                                Avg_Calls=mean(Calls), Avg_Minutes=mean(Minutes),  
                                Avg_Amt=mean(Amt)))  
  
  teledata_new <- subset(teledata, select=c(-Week))  
  
  data <-with(teledata_new, tapply(teledata_new[,as.numeric(input$var)],  
                                list(Gender,Age_Group),mean))  
  
  data.m <- melt(data,id.vars=Gender)  
  colnames(data.m)[1] <- "Gender"
```

**renderPlot()** displays plot.

Object **data** uses the third input – variable.



Note: All inputs are taken from the user at one time but are used at different stages in the backend.



# server.R

# Coding for tab4 i.e. Volume of Business

```
ggplot(data.m, aes(Var2,value)) +  
geom_bar(aes(fill=Gender), position="dodge", stat="identity")+  
labs(title="Customer_Average_usage")+  
labs(x="Age_Group",y=names(xyz[as.numeric(input$var)]))+  
theme_bw()
```

```
})
```

# server.R

```
observeEvent(input$tabs, {  
  if(input$tabs=="data"){  
    show("age")  
    show("gender")  
    hide("ID")  
    hide("var")  
  })  
observeEvent(input$tabs, {  
  if(input$tabs=="summa"){  
    show("age")  
    show("gender")  
    show("ID")  
    hide("var")  
  })  
observeEvent(input$tabs, {  
  if(input$tabs=="Usag"){  
    show("age")  
    show("gender")  
    show("var")  
    hide("ID")  
  })  
})
```

- ❑ **observeEvent()** is an event handler used for giving condition/s for a particular event to happen.
- ❑ Here we have given conditions for all the tabs

- ❑ **show()** makes the widgets for **age** and **gender** visible
- ❑ **hide()** makes **var** and **ID** invisible when **data** tab is selected.
- ❑ Likewise, we have given the conditions for other tabs

# server.R

```
observeEvent(input$tabs, {  
  if(input$tabs=="vol"){  
    show("var")  
    hide("age")  
    hide("gender")  
    hide("ID")  
  })  
}
```

# Deploy the App

To upload the app:

- Go to <http://www.shinyapps.io/> and create a free or professional account.
- Make sure all your app files are in an isolated folder saved in your working directory.
- In order to upload the app on shinyapp.io, you need to install package **devtools** and **rsconnect**.

# This R file is created for making the app live

```
library(rsconnect)
rsconnect::setAccountInfo(name=<name>, token=<token> secret=<token>)

rsconnect::deployApp("TelecomApplication")
```

- ❑ **deployApp()** function automatically syncs up to the shinyapps.io server, and opens the shinyapps.io website on your browser.
- ❑ Once your app is uploaded, it is on the internet and can be viewed by anybody with access to it.