Weather Project Package Report

Michael Caldwell & Nan Bai

March 20, 2017

Introduction

The overall purposes of the weatherProject package is to empower users to be able to retrieve weather data utilizing Weather Underground (https://www.wunderground.com/) and subsequently visualize that data in order to observe differences between microclimates within the vacinity of a city.

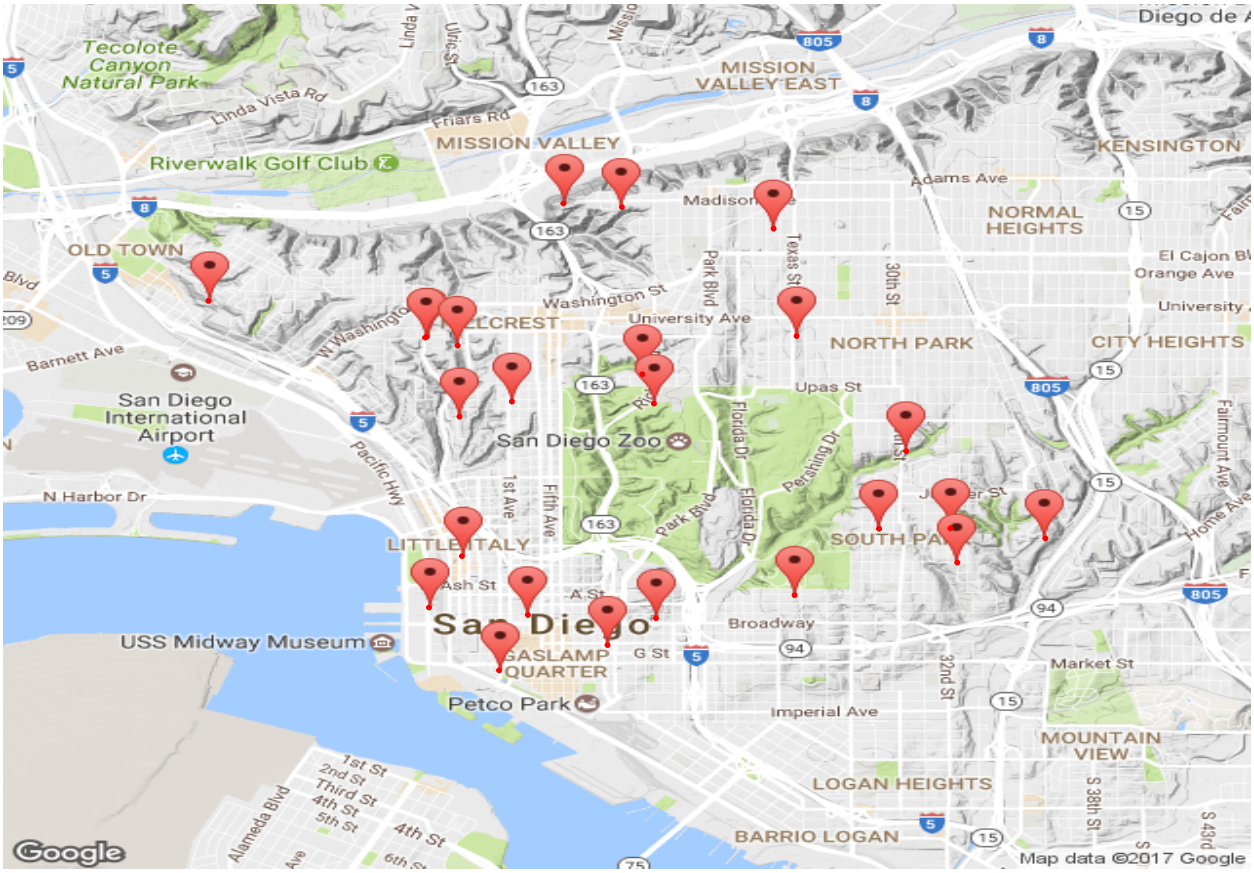
In order to achieve that end state, there are various functions included with the package, which will have examples demonstrated in the following sections. Those functions do things like retrieve latitude/longitude for a city, get other cities within the vacinity of a city, retrieve personal weather stations (PWSs) for those cities, get weather data related to PWSs, and create both map graphics and standard plots based on that data. The mapping data is based on the RgooleMaps package, which may be outside the normal usage of our audience. As part of the weatherProject package, a new R6 class (Pws) has been created and included to contain one or more PWSs and related attributes.

In order to get the full usage of this package, as user does need to sign up for 2 different types of online keys (both of which are free) and are required parameters for many of the functiosn. First, a key to access the wunderground API via https://www.wunderground.com/weather/api. Second, in order to overcome some of the current limitations of the wunderground API, a username to geonames via http://www.geonames.org/login.

Using the longlat and findNearbyCities functions to get Area of Interest

There are many constraints to the current, free version of the wunderground API such as only allowing 10 calls per minute or 500 per day. Within many of the functions we have wait times of 10 seconds entered between calls and ability to limit number of items being returned by one function before using those results in a call to another function.

However, one thing that was unanticipated is that due to the way wunderground returned that information within its limitations led to many overlapping PWSs for major metropolitan cities In other words, there are now many PWSs reporting within each city, so that unless you are dealing with a rural area, wunderground will actually cap the number of PWSs returned to you before you have any that are even a mile apart. As a result, the end state weather information and visualization of variation are uninteresing. Here is an example of a pin map using the wunderground standard returned PWSs for San Diego, CA and RgoogleMaps for visualization.



If you are familiar with San Diego, then it will be clear that the maximum PWSs that will be returned are all within one microclimate and will show very little variation.

To counter this a combination of google maps API via the included longlat function and the subsequent usage of those coordinates in the findNearbyCities function that utilizes the geonames function as in the following call. Here to shield the protected information, it uses the variable myusername to represent the geocode username. The distance is 100km. The total number of cities being returned is capped at 5, because from every step after this one, calls will be made to wunderground and the daily limits will come into play.

```
mycities <- findNearbyCities(
  username = mygeousername
  ,coord = longlat('San Diego, CA')
  ,distance = '100'
  ,maxRows = '5'
)
print(mycities)
```

```
## [1] "CA/San_Diego"     "CA/Coronado"      "CA/National_City"
## [4] "CA/Chula_Vista"   "CA/Lemon_Grove"
```

Now we have set up a vector of cities to subsequently go in and get a better distribution of PWSs from the wunderground data.

Using the findPWS function and R6 Pws Class to Retrieve Personal Weather Station (PWS) data for Cities

The next step to drive toward the end result of seeing weather differences between the microclimates is now to get related PWSs for the chosen cities. This can be done simply by calling findPWS as in the following example. In the example, again, we are going to limit the returned results to ensure that the calls to wunderground do not pile up. Due to the limit of 1 put in for each city chosen, a maximum of 5 PWSs can result in this example. Again, a key is being masked here by the use of mywukey. The returned structure is a tibble.

```
mypws <- findPWS(myKey=mywukey, nearbyCities=mycities, maxPerCity = 1)
head(mypws)
```

```
## Source: local data frame [5 x 11]
## Groups: citseq [5]
##
##           neighborhood          city state country          id      lat
##                  <chr>         <chr> <chr>   <chr>       <chr>    <dbl>
## 1        Core-Columbia     San Diego    CA      US KCASANDI299 32.71674
## 2 San Diego Yacht Club     San Diego    CA      US KCASANDI186 32.71794
## 3                      National City    CA      US      ME3219 32.67050
## 4           San Diego     San Diego    CA      US KCASANDI643 32.58209
## 5         Mount Vernon   Lemon Grove    CA      US   KCALEMON8 32.72964
## # ... with 5 more variables: lon <dbl>, distance_km <dbl>,
## #   distance_mi <dbl>, citseq <int>, pwsNo <int>
```

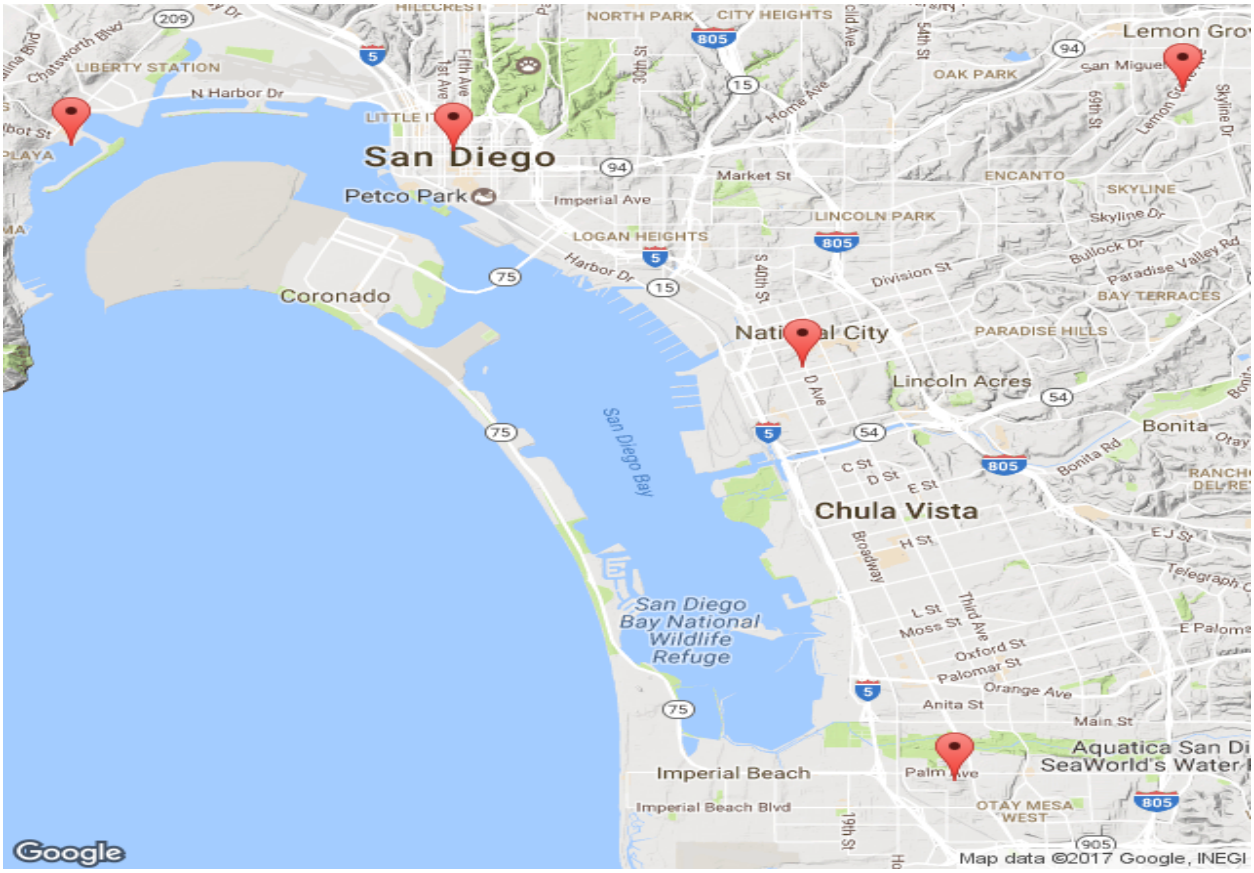
With this basic output, we can now actually proceed to utilize the R6 class, Pws, which is part of the package. Feeding it into the class has the benefit of some data validation, but moreover the ability to plot the data by using the function defined within the class. In the following examples, the data is pushed into the class,

```
pwsr6 <- Pws$new(
  id              = mypws$id
  ,lat            = mypws$lat
  ,lon            = mypws$lon
  ,neighborhood   = mypws$neighborhood
  ,city           = mypws$city
  ,state          = mypws$state
  ,country        = mypws$country
  ,distance_km    = mypws$distance_km
  ,distance_mi    = mypws$distance_mi
)
pwsr6$print()
```

```
## # A tibble: 5 × 9
##            id      lat       lon         neighborhood          city state
##         <chr>    <dbl>     <dbl>                <chr>         <chr> <chr>
## 1 KCASANDI299 32.71674 -117.1623        Core-Columbia     San Diego    CA
## 2 KCASANDI186 32.71794 -117.2295 San Diego Yacht Club     San Diego    CA
## 3      ME3219 32.67050 -117.1010                      National City    CA
## 4 KCASANDI643 32.58209 -117.0741           San Diego     San Diego    CA
## 5   KCALEMON8 32.72964 -117.0340         Mount Vernon   Lemon Grove    CA
## # ... with 3 more variables: country <chr>, distance_km <dbl>,
## #   distance_mi <dbl>
```

Now it is plotted, first with a pin drops from before, the with the ids as labels (which is the default).

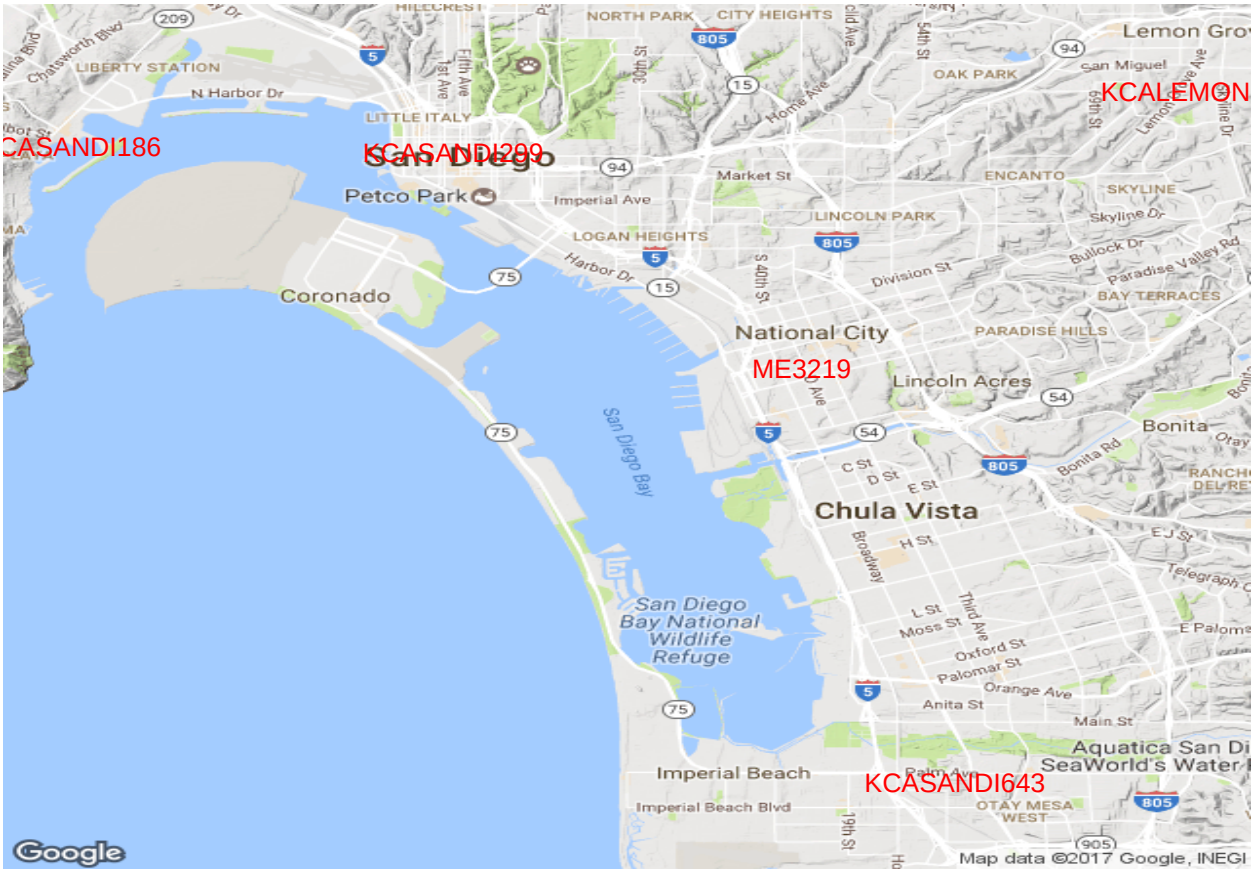
```
pwsr6$plot(plottype = 'p')
```



When compared to the plot at the beginning of this vignette, one can see that, even though we have limited the number of PWSs returned significantly, we have greater dispersion throughou the San Diego area.

Now we will tag the ids, by using the default plot options for the class.

```
pwsr6$plot()
```



This plot can then be used as a geographical reference in cases where we may later plot weather data without a map.

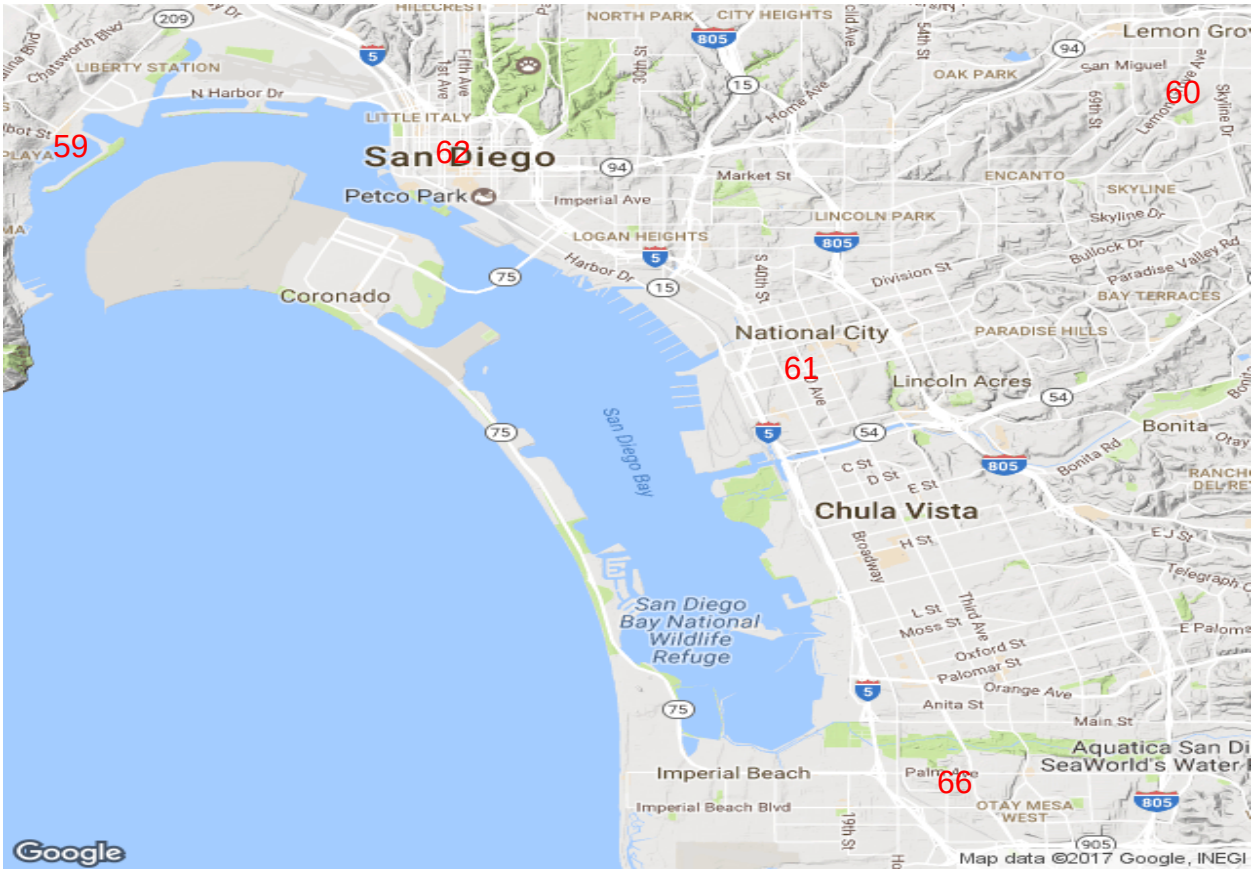
Using the queryHistory and plotWeather functions to retrieve

First we will query the history available by using a start and end date applied to a tibble extracted from our class using one of the functions. The nearbyStations parameter could just as easily have been provided by the original data set generated prior to moving into our class.

```
myweather <- queryHistory(
  myKey = mywukey
  ,nearbyStations = pwsr6$getTibble()
  ,startDate = '2017-03-17'
  ,endDate = '2017-03-20'
)
```

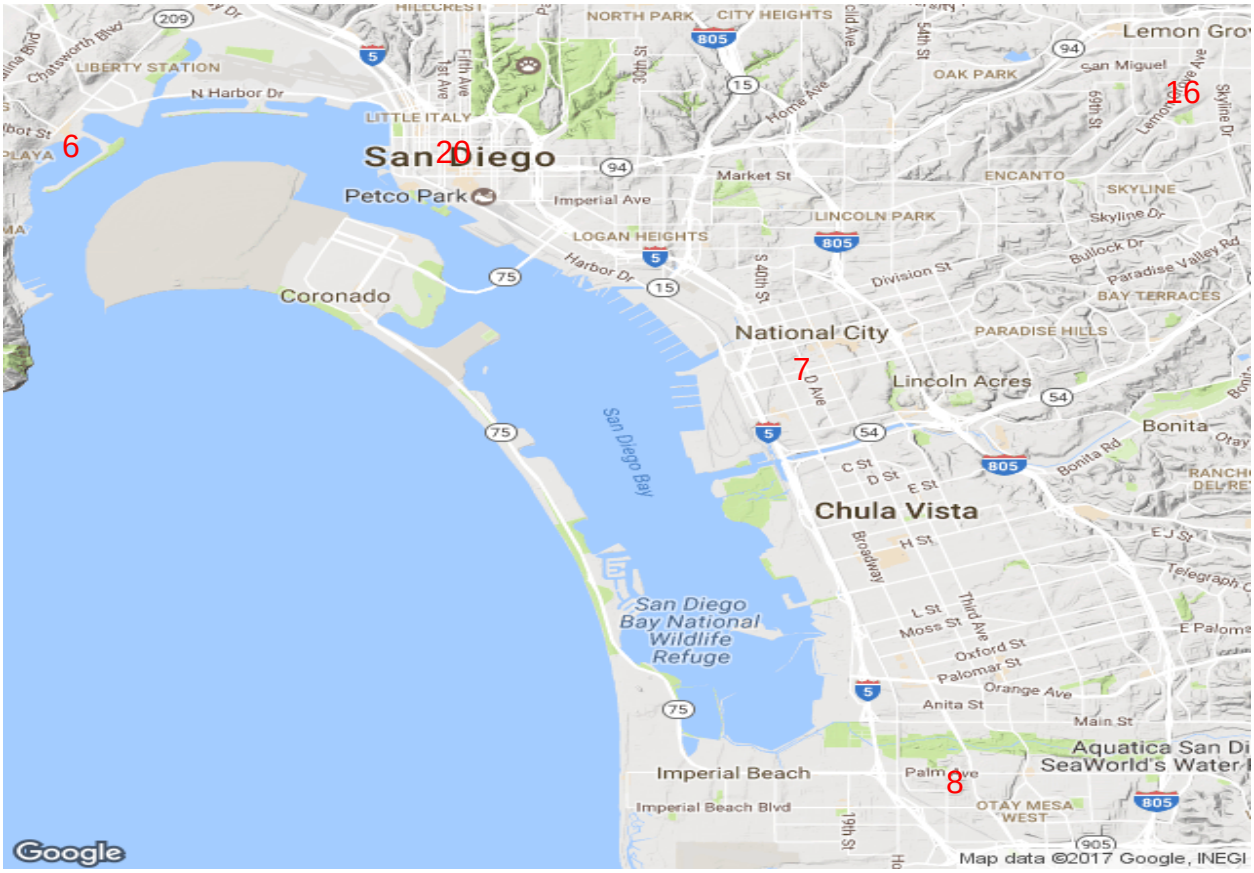
Now we will actually use the standard options for plotWeather. This will result in a map showing the average temperature (tempi) in English units, i.e. Faherenheit across the entire time period.

```
plotWeather(
  wtbl = myweather
)
```



We only see a few degress of separation in the average. However, given the differences in the terrain of the San Diego area, let's instead look at the range across the period, being the max - min.

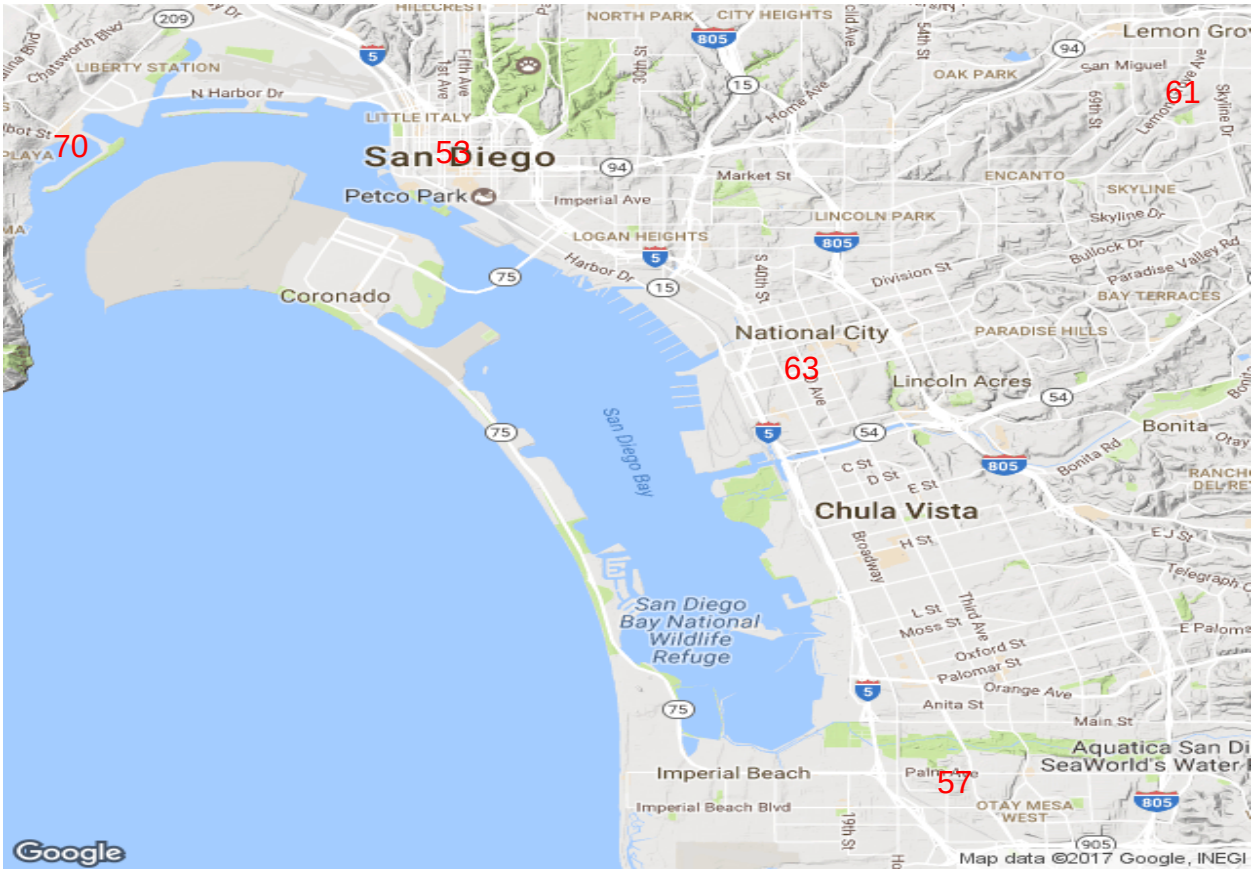
```
plotWeather(
  wtbl = myweather
  ,aggtype = "range"
)
```



Here we see much more differentiation, as the area most near the Pacific Ocean shows the least amount of variability, given the regulation of temperature that the massive body of water provides.

Now let us see if we can see any differentiation by looking at the minimum humidity across the time period.

```
plotWeather(
  wtbl = myweather
  ,wvar = "hum"
  ,aggtype = "min"
)
```



What is most telling here is the very low mimimum % humidity of 39 in the inland area, which is neither bordering the ocean, nor in the mountains, and is known to be desert-like.

The plotWeather function also provides the ability to see a traditional line graph of conditions across time, rather than just a singular aggregate value. The following funcion call will look.

```
plotWeather(
  wtbl = myweather
  ,plottype = "g"
)
```

Using the summarizeData function to subset the weatherData

Sometimes you might want to subset the queried weather data within a certain distance and within a time frame and calculate a certain weather parameter statistics, this is exactly the purpose of this summarizeData function. For example, for the above queried weather data, we want to limit the pws within 10km distance and we only want to see the mean temperature for the first two days for these weather stations. The following function is used to fulfil this purpose.

```
subsetData <- summarizeData(myweather, distance = 10, startDate = '2017-03-20', endDate = '2017-03-20',
subsetData
```

```
## # A tibble: 5 × 4
##            id      lat       lon     stat
##         <chr>    <dbl>     <dbl>    <dbl>
## 1   KCALEMON8 32.72964 -117.0340 15.56904
## 2 KCASANDI186 32.71794 -117.2295 15.24982
```

```
## 3 KCASANDI299 32.71674 -117.1623 16.88000
## 4 KCASANDI643 32.58209 -117.0741 18.76000
## 5      ME3219 32.67050 -117.1010 16.27917
```

A shiny app on top of these functions for easy adoption of this package

To facilitate the usage of this package, we built a shiny app on top of these functions. Users can launch the shiny app by calling the following function.

```
runShinyExample()
```

For the first tab Nearby PWS, user can input the city name, country name, distance from that location, maximum number of nearby cities, and the date range with the key to access wunderground API and username to access geoname webservice, and click query button. The following three actions will be performed and the results will be displayed in the main panel:

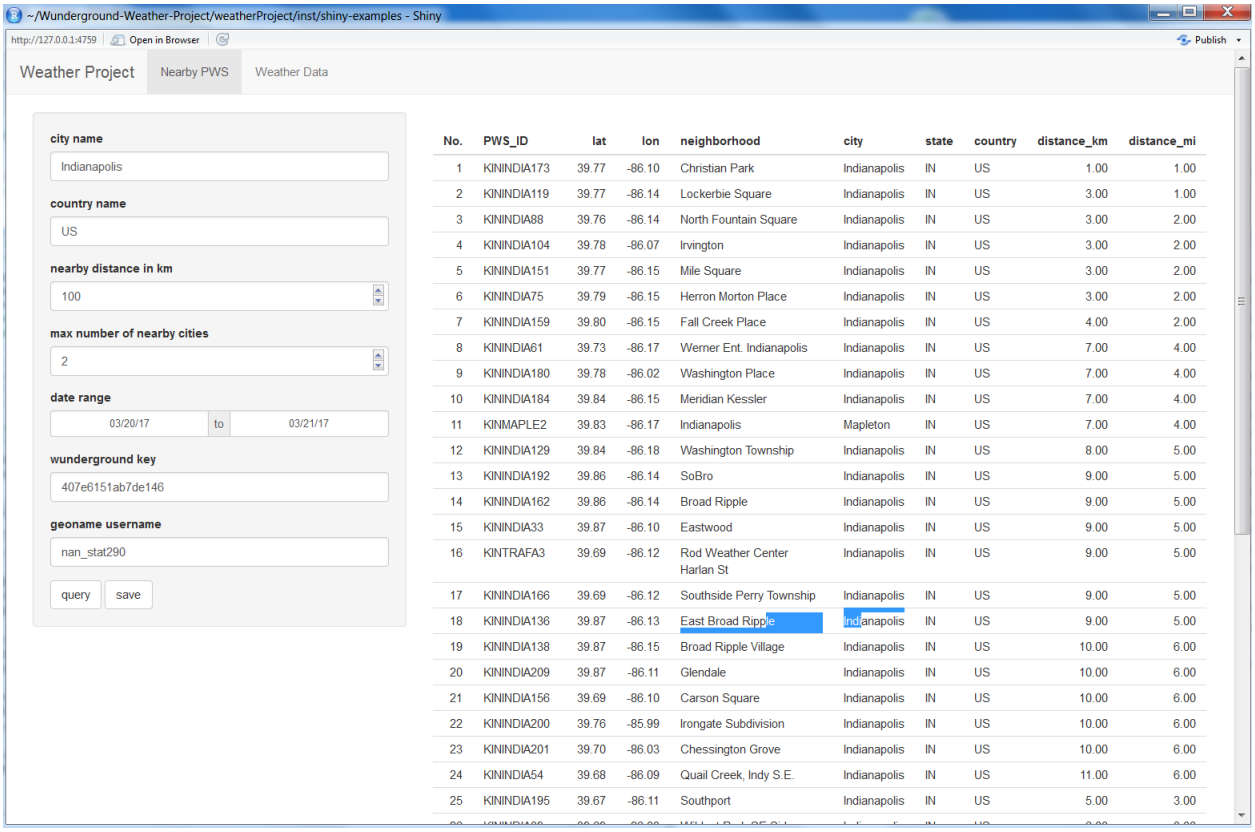
- find the longtitude and latitude of the location
- find nearby cities within the distance
- find all pws within these cities
- query weather data for all pws within the time frame

Some limitation were enforced due to the query limit from Wunderground website:

- The maximum date range is 5 days
- The maximum number of nearby cities is 10
- The maximum number of pws per city is 5

User can click save button to save the data. Two date files pwsData_yyyymmddhhmmss.rds and weatherData_yyyymmddhhmmss.rds will be saved with timestamp for future usage. pwsData.rds contains the data frame describing the pws, while weatherData.rds contains the weather data for all these pws.

For the second tab Weather Data, user can load the saved weatherData.rds file, specifiy the distance and date range to subset the weather data table. User can choose the weather parameter of interest, the unit (metric vs. english) and the statistics (mean/max/min/range) for visualizing on the map. The whole data table will be displayed below the graph as shown below.



No.	PWS_ID	lat	lon	neighborhood	city	state	country	distance_km	distance_mi
1	KINDIA173	39.77	-86.10	Christian Park	Indianapolis	IN	US	1.00	1.00
2	KINDIA119	39.77	-86.14	Lockerbie Square	Indianapolis	IN	US	3.00	1.00
3	KINDIA88	39.76	-86.14	North Fountain Square	Indianapolis	IN	US	3.00	2.00
4	KINDIA104	39.78	-86.07	Irvington	Indianapolis	IN	US	3.00	2.00
5	KINDIA151	39.77	-86.15	Mile Square	Indianapolis	IN	US	3.00	2.00
6	KINDIA75	39.79	-86.15	Herron Morton Place	Indianapolis	IN	US	3.00	2.00
7	KINDIA159	39.80	-86.15	Fall Creek Place	Indianapolis	IN	US	4.00	2.00
8	KINDIA61	39.73	-86.17	Werner Ent. Indianapolis	Indianapolis	IN	US	7.00	4.00
9	KINDIA180	39.78	-86.02	Washington Place	Indianapolis	IN	US	7.00	4.00
10	KINDIA184	39.84	-86.15	Meridian Kessler	Indianapolis	IN	US	7.00	4.00
11	KINMAPLE2	39.83	-86.17	Indianapolis	Mapleton	IN	US	7.00	4.00
12	KINDIA129	39.84	-86.18	Washington Township	Indianapolis	IN	US	8.00	5.00
13	KINDIA192	39.86	-86.14	SoBro	Indianapolis	IN	US	9.00	5.00
14	KINDIA162	39.86	-86.14	Broad Ripple	Indianapolis	IN	US	9.00	5.00
15	KINDIA33	39.87	-86.10	Eastwood	Indianapolis	IN	US	9.00	5.00
16	KINTRAFA3	39.69	-86.12	Rod Weather Center Harlan St	Indianapolis	IN	US	9.00	5.00
17	KINDIA166	39.69	-86.12	Southside Perry Township	Indianapolis	IN	US	9.00	5.00
18	KINDIA136	39.87	-86.13	East Broad Ripple	Indianapolis	IN	US	9.00	5.00
19	KINDIA138	39.87	-86.15	Broad Ripple Village	Indianapolis	IN	US	10.00	6.00
20	KINDIA209	39.87	-86.11	Glendale	Indianapolis	IN	US	10.00	6.00
21	KINDIA156	39.69	-86.10	Carson Square	Indianapolis	IN	US	10.00	6.00
22	KINDIA200	39.76	-85.99	Irongate Subdivision	Indianapolis	IN	US	10.00	6.00
23	KINDIA201	39.70	-86.03	Chessington Grove	Indianapolis	IN	US	10.00	6.00
24	KINDIA54	39.68	-86.09	Quail Creek, Indy S.E.	Indianapolis	IN	US	11.00	6.00
25	KINDIA195	39.67	-86.11	Southport	Indianapolis	IN	US	5.00	3.00

Figure 1:

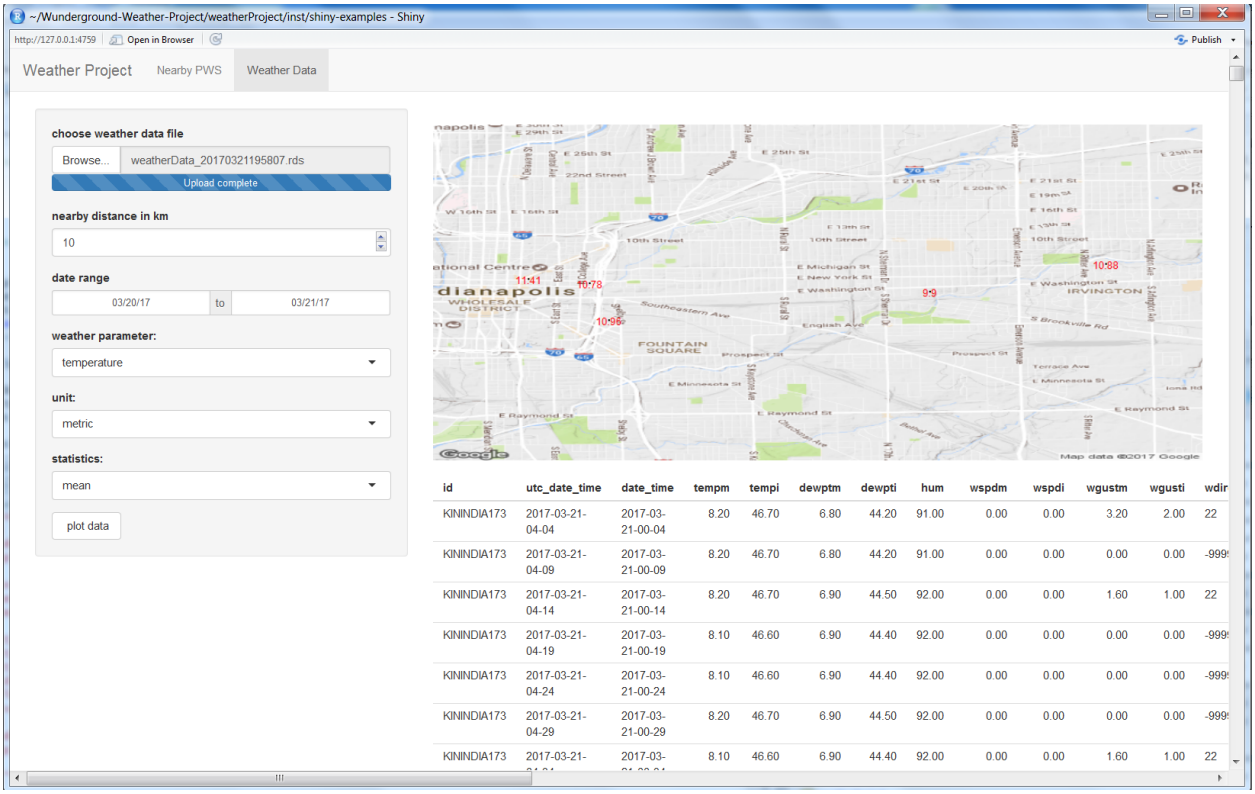

~/Wunderground-Weather-Project/weatherProject/inst/shiny-examples - Shiny
http://127.0.0.1:4759
Open in Browser
Publish
Weather Project
Nearby PWS
Weather Data
choose weather data file
Browse...
weatherData_20170321195807.rds
Upload complete
nearby distance in km
10
date range
03/20/17
to
03/21/17
weather parameter:
temperature
unit:
metric
statistics:
mean
plot data
id
utc_date_time
date_time
tempm
tempi
dewptm
dewpti
hum
wspdm
wspdi
wgustm
wgusti
wdir
KINDIA173
2017-03-21-04-04
2017-03-21-00-04
8.20
46.70
6.80
44.20
91.00
0.00
0.00
3.20
2.00
22
KINDIA173
2017-03-21-04-09
2017-03-21-00-09
8.20
46.70
6.80
44.20
91.00
0.00
0.00
0.00
0.00
-999
KINDIA173
2017-03-21-04-14
2017-03-21-00-14
8.20
46.70
6.90
44.50
92.00
0.00
0.00
1.60
1.00
22
KINDIA173
2017-03-21-04-19
2017-03-21-00-19
8.10
46.60
6.90
44.40
92.00
0.00
0.00
0.00
0.00
-999
KINDIA173
2017-03-21-04-24
2017-03-21-00-24
8.10
46.60
6.90
44.40
92.00
0.00
0.00
0.00
0.00
-999
KINDIA173
2017-03-21-04-29
2017-03-21-00-29
8.20
46.70
6.90
44.50
92.00
0.00
0.00
0.00
0.00
-999
KINDIA173
2017-03-21-
2017-03-
8.10
46.60
6.90
44.40
92.00
0.00
0.00
1.60
1.00
22

Figure 2: