Hands-on Lab: Historical Weather Forecast Comparison to Actuals

Scenario

You've been tasked by your team to create an automated Extract, Transform, Load (ETL) process to extract daily weather forecast and observed weather data and load it into a live report to be used for further analysis by the analytics team. As part of a larger prediction modelling project, the team wants to use the report to monitor and measure the historical accuracy of temperature forecasts by source and station.

As a proof-of-concept (POC), you are only required to do this for a single station and one source to begin with. For each day at noon (local time), you will gather both the actual temperature and the temperature forecasted for noon on the following day for Casablanca, Morocco.

At a later stage, the team anticipates extending the report to include lists of locations, different forecasting sources, different update frequencies, and other weather metrics such as wind speed and direction, precipitation, and visibility.

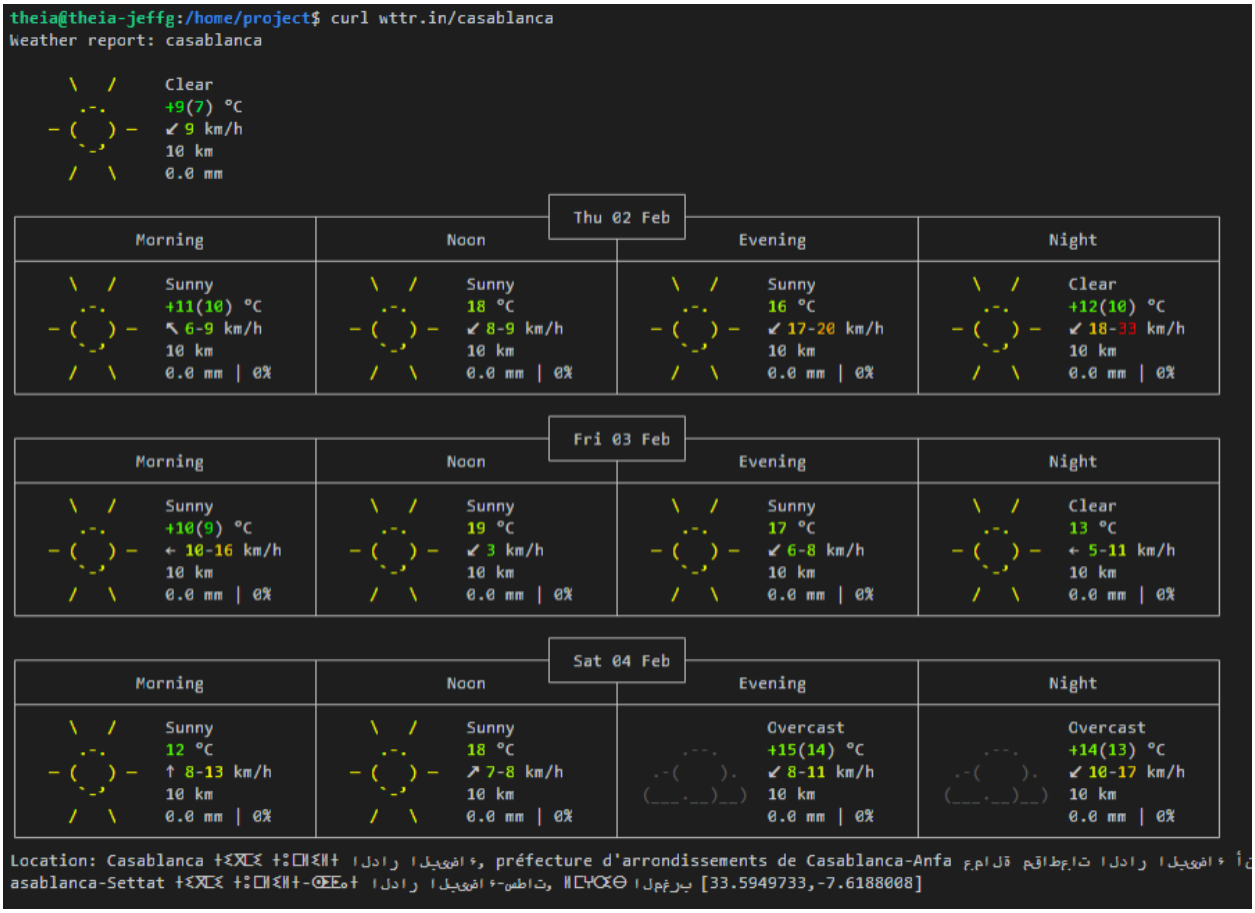
Data source

For this practice project, you'll use the weather data package provided by the open source project wttr.in, a web service that provides weather forecast information in a simple and text-based format. For further information, you can read more about the service on its GitHub Repo.

First, you'll use the `curl` command to scrape weather data via the *wttr.in* website. For example, to get data for Casablanca, enter:

1. `curl wttr.in/casablanca`

which prints the following to `stdout`:



Learning Objectives

After completing this practice project, you will be able to apply your new shell scripting skills in a real-world scenario to:

- Download raw weather data
- Extract data of interest from the raw data
- Transform the data as required
- Load the data into a log file using a tabular format
- Schedule the entire process to run automatically at a set time daily

Overview

Weather reporting tasks

You must extract and store the following data every day at noon, local time, for Casablanca, Morocco:

- The actual temperature (in degrees Celsius)
- The forecasted temperature (in degrees Celsius) for the following day at noon

Here is an example of what the the resulting weather report should look like:

year	month	day	obs_tmp	fc_temp
2023	1	1	10	11
2023	1	2	11	12
2023	1	3	12	10
2023	1	4	13	13
2023	1	5	10	9
2023	1	6	11	10
...	...	...	...	...

> Table 1. Example of weather report

Feel free to try completing the entire project on your own, or follow the exercises below to guide you through the process. Either way, check out the exercises, hints, and solutions.

> Tip:

- At each step of the process, test your code to ensure it does what you intended. For more complicated steps, break the task down into smaller, more managable steps that you can test individually. You can test code on the command line or by running your script as you develop it. The context will indicate the best approach.

Estimated time needed: **30** minutes

Learning objectives

In this practice project, you will:

- Initialize your log file
- Write a Bash script to download, extract, and load raw data into a report
- Add some basic analytics to your report
- Schedule your report to update daily
- Measure and report on historical forecasting accuracy

We've broken this project down into manageable steps. Feel free to try any or all of these on your own; however, we recommend checking your work with the details provided.

Exercise 1 - Initialize your weather report log file

1.1 Create a text file called `rx_poc.log`

`rx_poc.log` will be your POC weather report log file, or a text file which contains a growing history of the daily weather data you will scrape. Each entry in the log file corresponds to a row as in **Table 1**.

▼ Click here for Hint

> *Use the `touch` command or open a new text file from the GUI.*

▼ Click here for Solution

> *Create `rx_poc.log` by entering the following in your terminal:*

```
1  touch rx_poc.log
```

1.2 Add a header to your weather report

Your header should consist of the column names from **Table 1**, delimited by tabs.
Write the header to your weather report.

▼ Click here for Hint

> *Use the `echo` command with the `-e` option, and include tab separators `\t` in a string of names.*
> *Consider why the `-e` option is needed.*

▼ Click here for Solution

> *Use a shell variable and command substitution:*

```
1  header=$(echo -e "year\tmonth\tday\tobs_temp\tfc_temp")
2  echo $header>rx_poc.log
```

> ***OR** more directly, use `echo` and redirection:*

```
1  echo -e "year\tmonth\tday\tobs_temp\tfc_temp">rx_poc.log
```

> ***Tip**: Although it might seem redundant, it is better practice to use variables in these cases. Variables make for much cleaner code, which is easier to understand and safer to modify by others or even yourself at a later date. Using meaningful names for your variables also makes the code more "self-documenting."*

Exercise 2 - Download the raw weather data

2.1. Create a text file called `rx_poc.sh` and make it an executable Bash script

▼ Click here for Hint 1

Include a shebang.

▼ Click here for Hint 2

Use the `chmod` *command.*

▼ Click here for Solution 1

Create the file `rx_poc.sh`

```
1  touch rx_poc.sh
```

Include the Bash shebang on the first line of `rx_poc.sh` *:*

```
1  #! /bin/bash
```

▼ Click here for Solution 2

Make your script executable by running the following in the terminal:

```
1  chmod u+x rx_poc.sh
```

Verify your changes using the `ls` *command with the* `-l` *option.*

2.2. Assign the city name to `Casablanca` for accessing the weather report

▼ Click here for Hint

Use the assignment operator

▼ Click here for Solution

```
1  city=Casablanca
```

2.3 Obtain the weather information for Casablanca

▼ Click here for Hint

Use the `curl` *command with the* `--output` *option. Saved the output to a file named* `weather_report` *.*

▼ Click here for Solution

Edit `rx_poc.sh` *to include:*

```
1  curl -s wttr.in/$city?T --output weather_report
```

Exercise 3 - Extract and load the required data

3.1. Edit `rx_poc.sh` to extract the required data from the raw data file and assign them to variables `obs_temp` and `fc_temp`

Extracting the required data is a process that will take some trial and error until you get it right. Study the weather report you obtained in Step 2.3, determine what you need to extract, and look for patterns.

You are looking for ways to 'chip away' at the weather report by:

- Using shell commands to extract only the data you need (the **signal**)
- Filtering everything else out (the **noise**)
- Combining your filters into a pipeline (recall the use of **pipes** to chain **filters** together)

▼ Click here for a Hint to get started

> *Extract only those lines that contain temperatures from the weather report, and save the result to variables representing the temparature output.*

3.1.1. Extract the current temperature, and store it in a shell variable called `obs_temp`

Remember to validate your results.

You may have noticed by now that the temperature values extracted from *wttr.in* are surrounded by special formatting characters. These "hidden" characters cause the numbers to display in specific color - for example, when you use the `cat` command to display your log file.

Unfortunately you cannot perform arithmetic calculations on such formatted text, so you will need to extract the values from the surrounding formatting so you can make use of them later in this lab.

▼ Click here for Hint 1

> *Which line is the current temperature on?*

▼ Click here for Hint 2

> *Are there any characters you can use as a delimiter to appropriately parse the line into fields?*

▼ Click here for Solution

> *While adding the following lines to `rx_poc.sh`, ensure you understand what each filter accomplishes by using the command line. Try adding one filter at a time to see what the outcome is as you develop the pipeline.*

```
#To extract Current Temperature
obs_temp=$(curl -s wttr.in/$city?T | grep -m 1 '°.' | grep -Eo -e '-?[[:digit:]].*')
echo "The current Temperature of $city: $obs_temp"
```

- The first line uses the curl command to fetch weather information from wttr.in for the specified city ($city). It then uses a combination of grep and grep -Eo to extract the current temperature in degrees Celsius and assigns it to the variable obs_temp.
- The second line (`echo $obs_temp`) prints the current temperature to the console.

3.1.2. Extract tomorrow's temperature forecast for noon, and store it in a shell variable called `fc_temp`

▼ Click here for Hint

> *Provided you understand the previous pipeline, you will be able to solve this problem through experimentation.*

▼ Click here for Solution

Add to `rx_poc.sh` :

```
# To extract the forecast tempearature for noon tomorrow
fc_temp=$(curl -s wttr.in/$city?T | head -23 | tail -1 | grep '°.' | cut -d 'C' -f2 | grep -Eo -e '-?[[:digit:]].*')
echo "The forecasted temperature for noon tomorrow for $city : $fc_temp C"
```

- The first line uses a combination of head and tail to narrow down to the line containing tomorrow's noon forecast temperature. grep and cut are employed to isolate and format the temperature information, and, the numeric part of the temperature is captured using grep -Eo.
- The second line (echo $fc_temp) prints the forecast temperature for noon tomorrow to the console.

3.2. Store the current day, month, and year in corresponding shell variables

▼ Click here for Hint

> *Use command substitution and the* `date` *command with the correct formatting options.*
> *The time zone for Casablanca is UTC+1. To get the local time for Casablanca, you can set the time-zone environment variable,* `TZ` *, as follows:*

```
TZ='Morocco/Casablanca'
```

▼ Click here for Solution

Add to `rx_poc.sh` :

```
#Assign Country and City to variable TZ
TZ='Morocco/Casablanca'

# Use command substitution to store the current day, month, and year in corresponding shell variables:
day=$(TZ='Morocco/Casablanca' date -u +%d)
month=$(TZ='Morocco/Casablanca' date +%m)
year=$(TZ='Morocco/Casablanca' date +%Y)
```

> ***Note***: *You might be wondering why we didn't just set* `hour` *to a value of 12, since we want to get the time at noon.*
> *However, if we did, we would lose the ability to verify that we are actually running the code at the correct local time. You should think of the local time as a measurement rather than a set number.*

3.3. Merge the fields into a tab-delimited record, corresponding to a single row in Table 1

Append the resulting record as a row of data to your weather log file.

▼ Click here for Hint

> *How did you create the header to initialize your log file?*

▼ Click here for Solution

Add to `rx_poc.sh` :

```
record=$(echo -e "$year\t$month\t$day\t$obs_temp\t$fc_temp C")
echo $record>>rx_poc.log
```

Exercise 4 - Schedule your Bash script `rx_poc.sh` to run every day at noon local time

4.1. Determine what time of day to run your script

Recall that you want to load the weather data coresponding to noon, local time, in Casablanca every day.

First, check the time difference between your system's default time zone and UTC.

▼ Click here for Hint 1

> *Use the `date` command twice with appropriate options; once to get your system time, and once to get UTC.*

▼ Click here for Solution

> *Running the following commands gives you the info you need to get the time difference between your system and UTC.*

```
$ date
Mon Feb 13 11:28:12 EST 2023
$ date -u
Mon Feb 13 16:28:16 UTC 2023
```

> *Now you can determine how many hours difference there are between your system's local time and that of Casablanca. In the example above, we see that the system time relative to UTC is UTC+5 (i.e. 16 - 11 = 5).*
>
> *We know Casablanca is UTC+1, so the system time relative to Casablanca is 4 hours earlier. Thus to run your script at noon Casablanca time, you need to run it at 8 am.*

4.2 Create a cron job that runs your script

▼ Click here for Hint

> *Edit the crontab file and review the crontab syntax description included in the file.*

▼ Click here for Solution

> *Edit the crontab file:*

```
crontab -e
```

> *Add the following line at the end of the file:*

```
0 8 * * * /home/project/rx_poc.sh
```

> *Save the file and quit the editor.*

4.3 Full solution

For reference, here is a Bash script that creates the raw weather report. Try to follow all the steps on your own before looking!

▼ Click here for Full Solution

```
#! /bin/bash

#Assign city name as Casablanca
city=Casablanca

#Obtain the weather report for Casablanca
curl -s wttr.in/$city?T --output weather_report

#To extract Current Temperature
obs_temp=$(curl -s wttr.in/$city?T | grep -m 1 '°.' | grep -Eo -e '-?[[:digit:]].*')
echo "The current Temperature of $city: $obs_temp"

# To extract the forecast tempearature for noon tomorrow
fc_temp=$(curl -s wttr.in/$city?T | head -23 | tail -1 | grep '°.' | cut -d 'C' -f2 | grep -Eo -e '-?[[:digit:]].*')
echo "The forecasted temperature for noon tomorrow for $city : $fc_temp C"

#Assign Country and City to variable TZ
TZ='Morocco/Casablanca'


# Use command substitution to store the current day, month, and year in corresponding shell variables:
day=$(TZ='Morocco/Casablanca' date -u +%d)
month=$(TZ='Morocco/Casablanca' date +%m)
year=$(TZ='Morocco/Casablanca' date +%Y)


# Log the weather
record=$(echo -e "$year\t$month\t$day\t$obs_temp\t$fc_temp C")
echo $record>>rx_poc.log
```

Exercise 5 - Create a script to report historical forecasting accuracy

Now that you've created an ETL shell script to gather weather data into a report, let's create another script to measure and report the accuracy of the forecasted temperatures against the actuals.

- To start, create a tab-delimited file named `historical_fc_accuracy.tsv`.

Insert the following code into the file to include a header with column names:

```
1  echo -e "year\tmonth\tday\tobs_temp\tfc_temp\taccuracy\taccuracy_range" > historical_fc_accuracy.tsv
```

One key difference between this report and the previous report you generated is that the forecast temperature will now be aligned with the date the forecast is for. As a result, the date will be in the same row as the observed temperature for that date, rather than the previous row on the day that the forecast was made.

- Also create an executable Bash script called `fc_accuracy.sh`.

Rather than scheduling your new script to run periodically, think of it as a tool you can use to generate the historical forecast accuracy on demand.

5.1. Determine the difference between today's forecasted and actual temperatures

Rather than writing the script to process all of the data at once, let's simplify by solving the problem for just one instance. Later you can modify the script to handle the general case of mupltiple days.

5.1.1. Extract the forecasted and observed temperatures for today and store them in variables

▼ Click here for Hint 1

> *Look up the record from yesterday.*

▼ Click here for Hint 2

> *Extract the forecast from the appropriate field.*

▼ Click here for Solution

```
1  yesterday_fc=$(tail -2 rx_poc.log | head -1 | cut -d " " -f5)
```

5.1.2. Calculate the forecast accuracy

▼ Click here for Hint

> *First extract today's observed temperature. Then calculate the difference between the forecasted and observed temperatures.*

▼ Click here for Solution

```
today_temp=$(tail -1 rx_poc.log | cut -d " " -f4)
accuracy=$(($yesterday_fc-$today_temp))
echo "accuracy is $accuracy"
```

> *Tip: Your weather report must have at least two days of data for this calulation to make sense.*
> *To test your code you can simply append some artificial data to your weather report,* `rx_poc.log` .

5.2. Assign a label to each forecast based on its accuracy

Let's set the accuracy labels according to the range that the accuracy fits most tightly within, according to the following table. Validate your result.

accuracy range	accuracy label
+/- 1 deg	excellent
+/- 2 deg	good
+/- 3 deg	fair
+/- 4 deg	poor

▼ Click here for Hint 1

> *Use two conditions to compare the accuracy sizes to each positive and negative integer range, accordingly.*

▼ Click here for Solution

```
if [ -1 -le $accuracy ] && [ $accuracy -le 1 ]
then
   accuracy_range=excellent
elif [ -2 -le $accuracy ] && [ $accuracy -le 2 ]
then
    accuracy_range=good
elif [ -3 -le $accuracy ] && [ $accuracy -le 3 ]
then
    accuracy_range=fair
else
    accuracy_range=poor
fi

echo "Forecast accuracy is $accuracy"
```

5.3. Append a record to your historical forecast accuracy file.

▼ Click here for Hint

> *Extract the right row and remaining data you need to populate all fields.*

▼ Click here for Solution

```
row=$(tail -1 rx_poc.log)
year=$( echo $row | cut -d " " -f1)
month=$( echo $row | cut -d " " -f2)
day=$( echo $row | cut -d " " -f3)
echo -e "$year\t$month\t$day\t$today_temp\t$yesterday_fc\t$accuracy\t$accuracy_range" >> historical_fc_accuracy.tsv
```

5.4. Full solution for handling a single day

Below is the final script of `fc_accuracy.sh` for handling the accuracy calculations based on just one instance, or day.

▼ Click here for Solution

```
#! /bin/bash

yesterday_fc=$(tail -2 rx_poc.log | head -1 | cut -d " " -f5)
today_temp=$(tail -1 rx_poc.log | cut -d " " -f4)
accuracy=$(($yesterday_fc-$today_temp))

echo "accuracy is $accuracy"

if [ -1 -le $accuracy ] && [ $accuracy -le 1 ]
then
          accuracy_range=excellent
elif [ -2 -le $accuracy ] && [ $accuracy -le 2 ]
   then
             accuracy_range=good
      elif [ -3 -le $accuracy ] && [ $accuracy -le 3 ]
      then
                accuracy_range=fair
          else
                    accuracy_range=poor
fi

echo "Forecast accuracy is $accuracy_range"

row=$(tail -1 rx_poc.log)
year=$( echo $row | cut -d " " -f1)
month=$( echo $row | cut -d " " -f2)
day=$( echo $row | cut -d " " -f3)
echo -e "$year\t$month\t$day\t$today_temp\t$yesterday_fc\t$accuracy\t$accuracy_range" >> historical_fc_accuracy.tsv
```

Exercise 6 - Create a script to report weekly statistics of historical forecasting accuracy

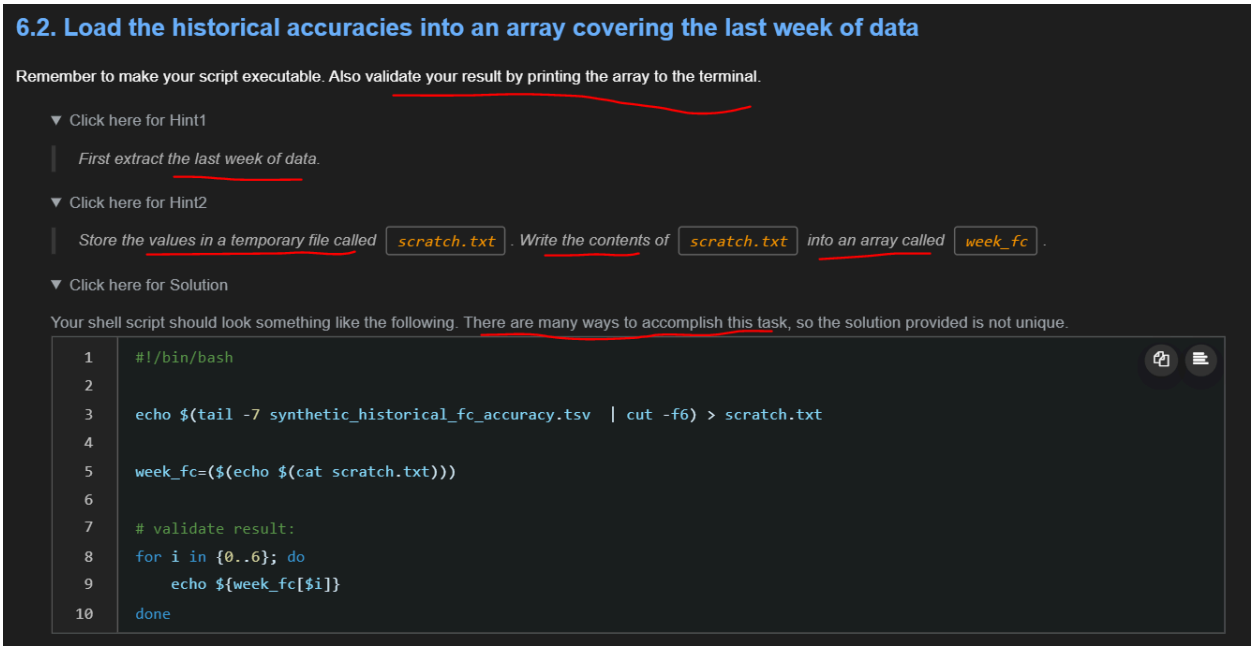
In this exercise, you will download a synthetic historical forecasting accuracy report and calculate some basic statistics based on the latest week of data.

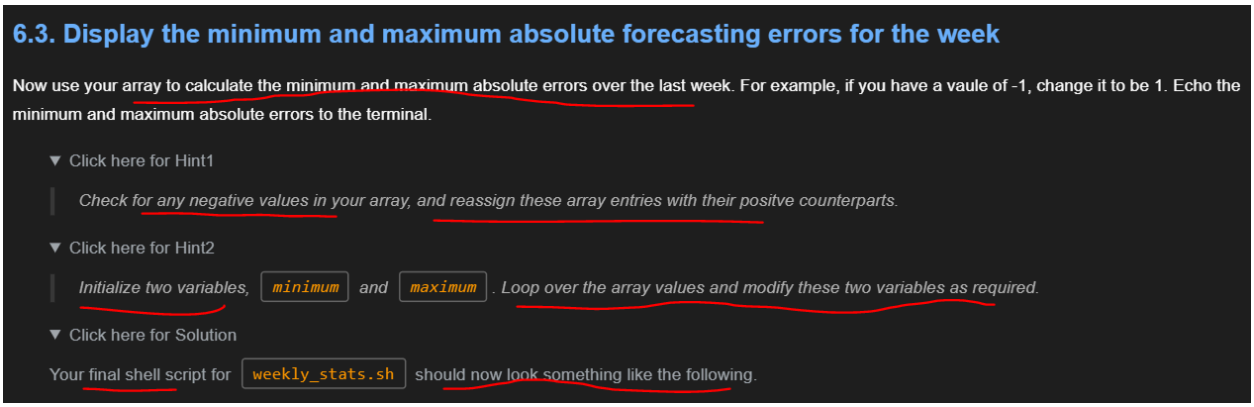
Begin by creating an executable bash script called `weekly_stats.sh`.

6.1. Download the synthetic historical forecasting accuracy dataset

Run the following command in the terminal to download the dataset to your current working directory.

1. `wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMSkillsNetwork-LX0117EN-Coursera/labs/synthetic_historical_fc_accuracy.tsv`





```bash
#!/bin/bash

echo $(tail -7 synthetic_historical_fc_accuracy.tsv  | cut -f6) > scratch.txt

week_fc=($(echo $(cat scratch.txt)))

# validate result:
for i in {0..6}; do
    echo ${week_fc[$i]}
done

for i in {0..6}; do
  if [[ ${week_fc[$i]} < 0 ]]
  then
    week_fc[$i]=$(((-1)*week_fc[$i]))
  fi
  # validate result:
  echo ${week_fc[$i]}
done

minimum=${week_fc[1]}
maximum=${week_fc[1]}
for item in ${week_fc[@]}; do
   if [[ $minimum > $item ]]
   then
     minimum=$item
   fi
   if [[ $maximum < $item ]]
   then
     maximum=$item
   fi
done

echo "minimum ebsolute error = $minimum"
echo "maximum absolute error = $maximum"
```

Summary

Congratulations! You've just completed a challenging, real-world practice project using many of the concepts you've learned from this course. The knowledge you've gained has prepared you to solve many practical real world problems. You're almost finished with this course now, and the final step in your journey is to complete the peer-reviewed Final Project.

In this lab, you learned how to:

- Initialize your weather report log file
- Write a Bash script that downloads the raw weather data, and extracts and loads the required data

- Schedule your Bash script `rx_poc.sh` to run every day at noon local time
- Apply advanced Bash scripting to produce reporting metrics
- Create a script to report historical forecasting accuracy
- Create a script to report the minimum and maximum absolute errors for the week

sample of the output and the code: