Arooba Jamil Khokhar
278077
Exercise # 5
Distributed Data Analytics

Exercise 1A

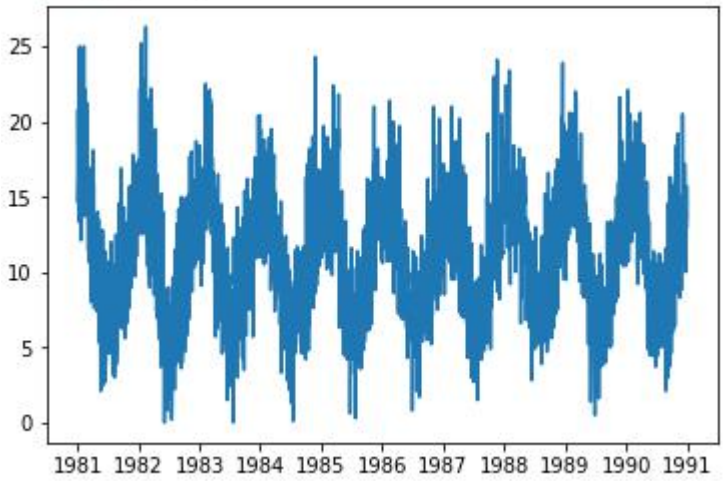
Check if the two datasets fulfill the stationary assumptions? Ideally you should write a function name stationary check and pass in a time series. This function should plot original series, rolling average and rolling std

DATASET(daily-minimum-temperatures-in-me)

It is a Stationary data set they do not have trend or Seasonal Effect

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from pandas import Series
from matplotlib import pyplot
from statsmodels.tsa.stattools import adfuller

temp = pd.read_csv('daily-minimum-temperatures-in-me.csv', parse_dates=True, header=0, index_
series = temp["#Melbourne"]
plt.plot(series)

```

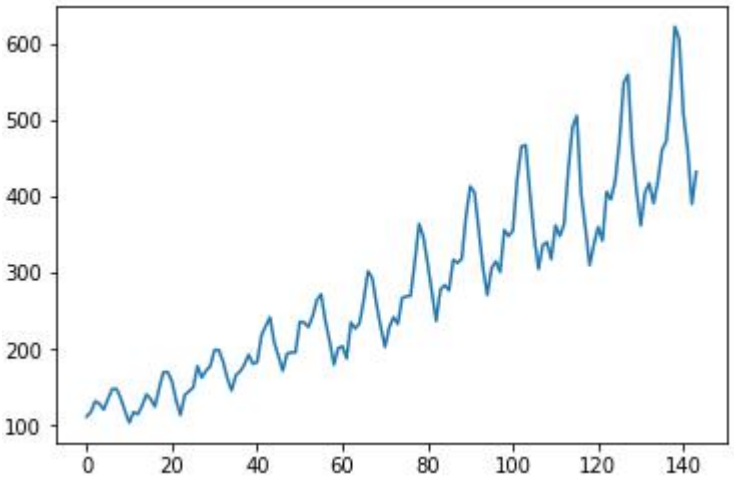
```
)ut[111]: [<matplotlib.lines.Line2D at 0x19b06b426d8>]
```



DATASET (AIRLINE PASSENGER)

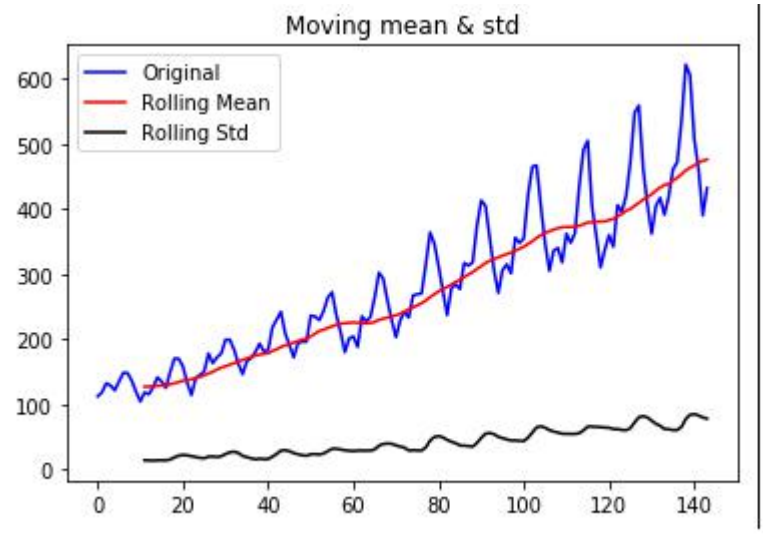
It is non stationary data set that show seasonal effects, trends, and other structures that depend on the time index

```
df = pd.read_csv("AirPassengers.csv", header=0)
timeseries = df["#Passengers"]
plt.plot(timeseries)
```



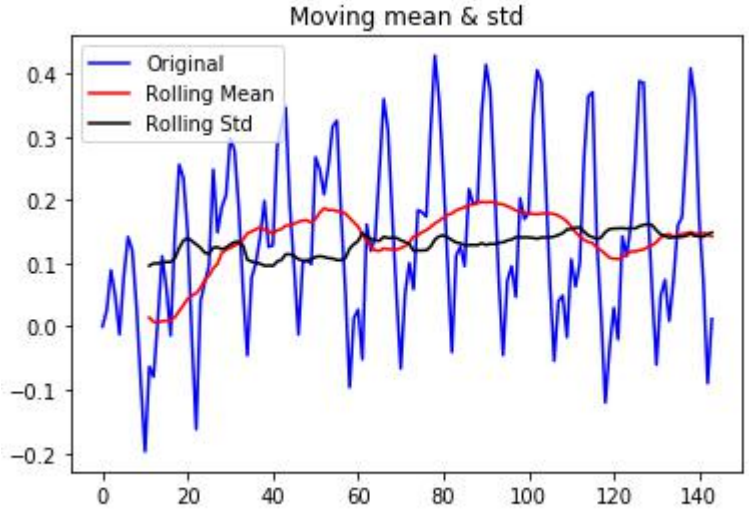
d write a function name stationary check and pass in a time series

```
def stationary_check(timeseries):
    rol_mean = pd.rolling_mean(timeseries, window=12)
    rol_std = pd.rolling_std(timeseries, window=12)

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rol_mean, color='red', label='Rolling Mean')
    std = plt.plot(rol_std, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Moving mean & std')
    plt.show(block=False)
stationary_check(timeseries)
```



If one of the datasets do not fulfill the stationary assumption can you use smoothing technique i.e. taking rolling average to make it stationary?

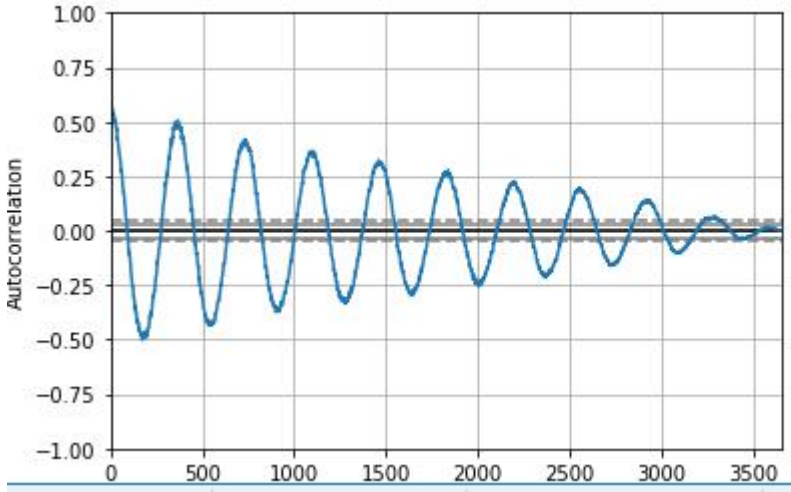
```
log_timeseries = np.log(timeseries)#take the log to reduce the values
expwighted_avg = pd.ewma(log_timeseries, halflife=12)
difference = log_timeseries - expwighted_avg
stationary_check (difference)
```



Exercise 1B:

• Write a python code that builds an ARIMA model

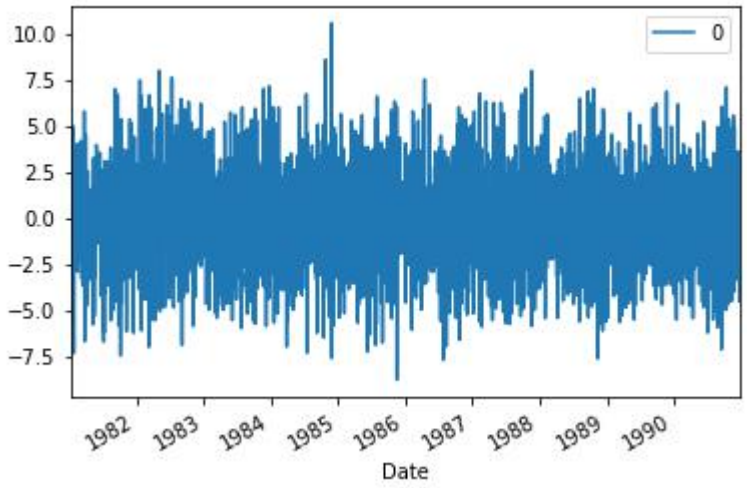
```
#series = read_csv('daily-minimum-temperatures-in-me.csv', header=0, parse_dates=[0], i
# fit model
model = ARIMA(series, order=(5,1,0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
# plot residual errors
residuals = DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
residuals.plot(kind='kde')
pyplot.show()
print(residuals.describe())
```




```
                              Lag
                         ARIMA Model Results
==============================================================================
Dep. Variable:          D.#Melbourne   No. Observations:                 3649
Model:                ARIMA(5, 1, 0)   Log Likelihood               -8495.810
Method:                       css-mle   S.D. of innovations              2.482
Date:                Sat, 02 Jun 2018   AIC                          17005.620
Time:                        01:36:00   BIC                          17049.036
Sample:                    02-01-1981   HQIC                         17021.082
                         - 12-31-1990
==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const              -0.0013      0.017     -0.076      0.940      -0.035       0.033
ar.L1.D.#Melbourne -0.3358      0.016    -20.469      0.000      -0.368      -0.304
ar.L2.D.#Melbourne -0.3911      0.017    -23.037      0.000      -0.424      -0.358
ar.L3.D.#Melbourne -0.2942      0.018    -16.804      0.000      -0.329      -0.260
ar.L4.D.#Melbourne -0.2077      0.017    -12.235      0.000      -0.241      -0.174
ar.L5.D.#Melbourne -0.1361      0.016     -8.298      0.000      -0.168      -0.104
                              Roots
```

```
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            0.7142           -1.1395j            1.3448           -0.1609
AR.2            0.7142           +1.1395j            1.3448            0.1609
AR.3           -1.6724           -0.0000j            1.6724           -0.5000
AR.4           -0.6410           -1.4207j            1.5586           -0.3175
AR.5           -0.6410           +1.4207j            1.5586            0.3175
-----------------------------------------------------------------------------
```

Split the data into train and validation. The time series data is usually split on the time not randomly. • Train the ARIMA model




```
# fit model
model = ARIMA(series, order=(5,1,0))

#3650

train_data = series[0:int(3650 * 0.8)]
validation_data = series[int(3650 * 0.8):]

print(len(train_data),len(validation_data))
```

use validation dataset to check the validation accuracy

```
forecast_array = model.forecast(steps=730)
output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
```

Exercise 2: Logistic Regression on the Olivetti faces dataset (5 points)

When program first starts, the loss is big. As more epochs are run, the accuracy continues to increase.

```
Epoch: 0001 cost= 6.033115935
Epoch: 0002 cost= 5.774913931
Epoch: 0003 cost= 4.939247543
Epoch: 0004 cost= 4.149745035
Epoch: 0005 cost= 3.630291149
Epoch: 0006 cost= 3.018514380
Epoch: 0007 cost= 2.564376166
Epoch: 0008 cost= 2.158875135
Epoch: 0009 cost= 1.831405316
Epoch: 0010 cost= 1.567071119
Epoch: 0011 cost= 1.353274047
Epoch: 0012 cost= 1.181682791
Epoch: 0013 cost= 1.042426887
Epoch: 0014 cost= 0.927122563
Epoch: 0015 cost= 0.830883344
Epoch: 0016 cost= 0.750456806
Epoch: 0017 cost= 0.683003995
Epoch: 0018 cost= 0.625729771
Epoch: 0019 cost= 0.576284829
Epoch: 0020 cost= 0.533035878
Epoch: 0021 cost= 0.494845811
Epoch: 0022 cost= 0.460887884
Epoch: 0023 cost= 0.430532374
Epoch: 0024 cost= 0.403282978
Epoch: 0025 cost= 0.378736738
Epoch: 0026 cost= 0.356559435
```

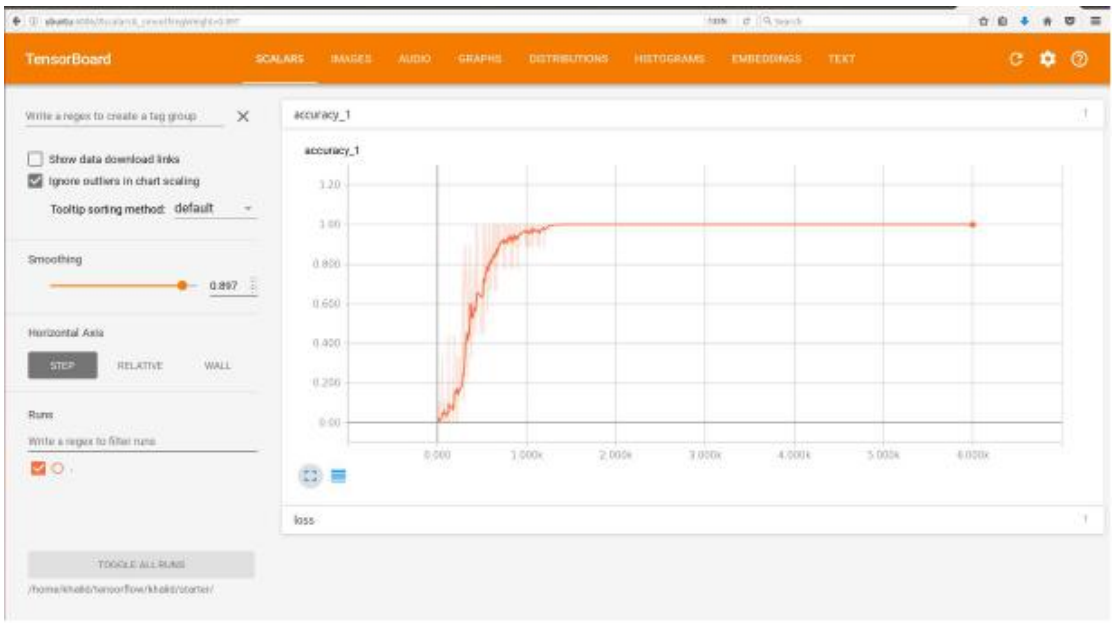
```
Epoch: 0083 cost= 0.070576954
Epoch: 0084 cost= 0.069539082
Epoch: 0085 cost= 0.068530526
Epoch: 0086 cost= 0.067550381
Epoch: 0087 cost= 0.066597353
Epoch: 0088 cost= 0.065670340
Epoch: 0089 cost= 0.064768341
Epoch: 0090 cost= 0.063890441
Epoch: 0091 cost= 0.063035597
Epoch: 0092 cost= 0.062203003
Epoch: 0093 cost= 0.061391753
Epoch: 0094 cost= 0.060601105
Epoch: 0095 cost= 0.059830243
Epoch: 0096 cost= 0.059078541
Epoch: 0097 cost= 0.058345291
Epoch: 0098 cost= 0.057629722
Epoch: 0099 cost= 0.056931305
Epoch: 0100 cost= 0.056249437
Epoch: 0101 cost= 0.055583531
Epoch: 0102 cost= 0.054933035
Epoch: 0103 cost= 0.054297426
Epoch: 0104 cost= 0.053676238
Epoch: 0105 cost= 0.053068976
Epoch: 0106 cost= 0.052475179
Epoch: 0107 cost= 0.051894396
Epoch: 0108 cost= 0.051326266
Epoch: 0109 cost= 0.050770289
```

```
Epoch: 0137 cost= 0.038946033
Epoch: 0138 cost= 0.038624752
Epoch: 0139 cost= 0.038308766
Epoch: 0140 cost= 0.037997918
Epoch: 0141 cost= 0.037692074
Epoch: 0142 cost= 0.037391157
Epoch: 0143 cost= 0.037095004
Epoch: 0144 cost= 0.036803558
Epoch: 0145 cost= 0.036516647
Epoch: 0146 cost= 0.036234235
Epoch: 0147 cost= 0.035956165
Epoch: 0148 cost= 0.035682361
Epoch: 0149 cost= 0.035412695
Epoch: 0150 cost= 0.035147145

Optimization Finished!

Now Testing on Test Set

Accuracy on Test Set is :  1.0
```

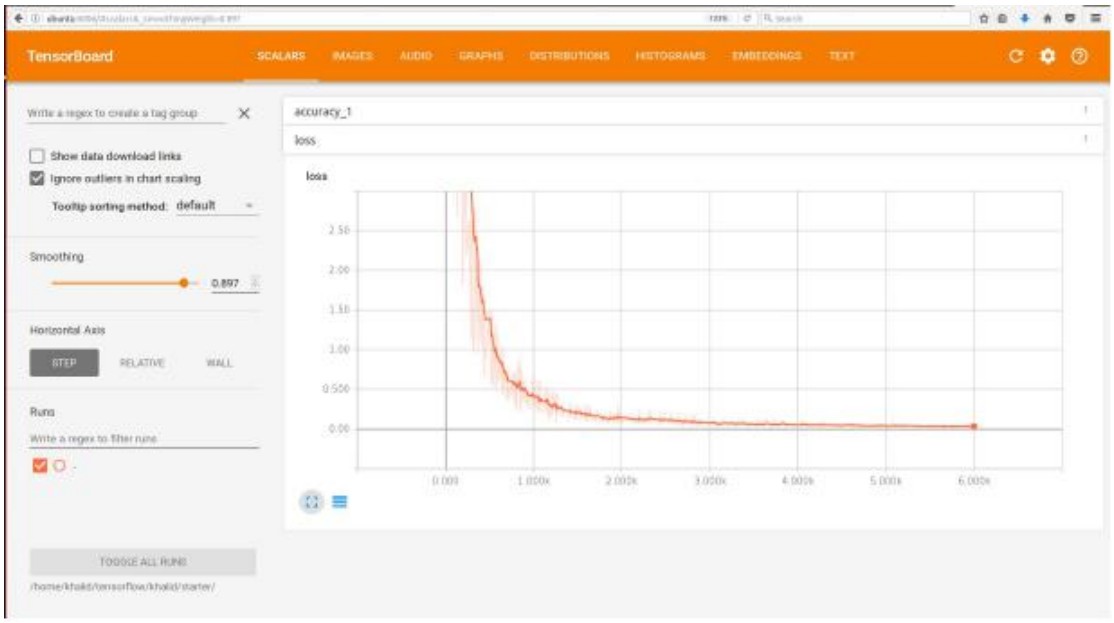
Scalar Values:

Below are scalar values for Accuracy and Loss in TensorBoard.

Accuracy:

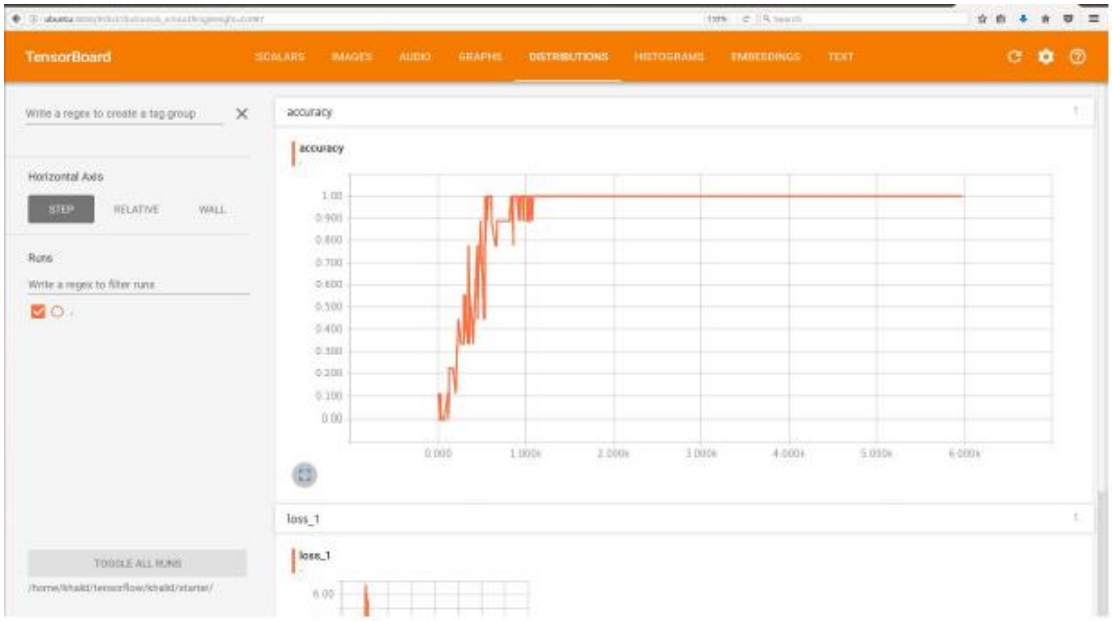


Loss:



Distributions:

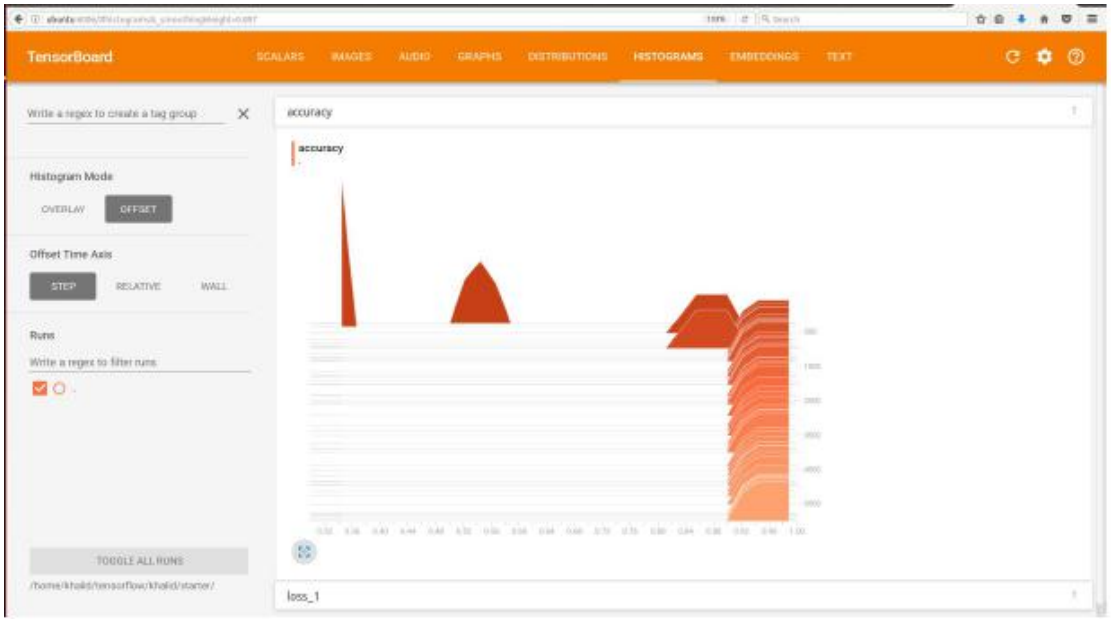
Accuracy:



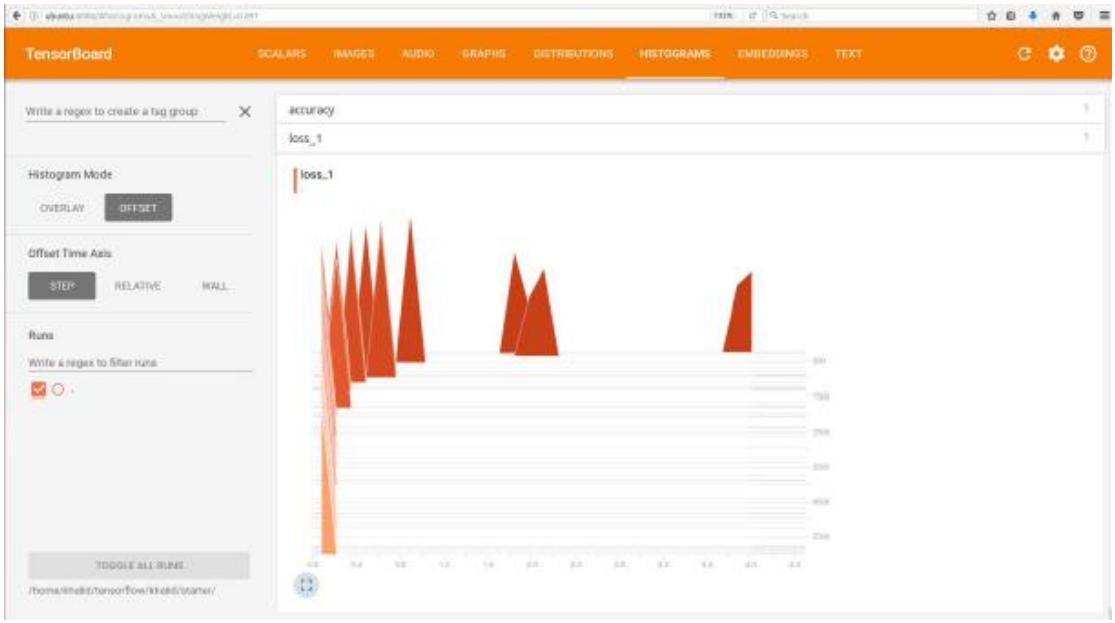
Histograms:

Below are histograms for Accuracy and Loss

Accuracy:



Loss:



Graph:

Below is the Tensor graph of the TensorBoard.