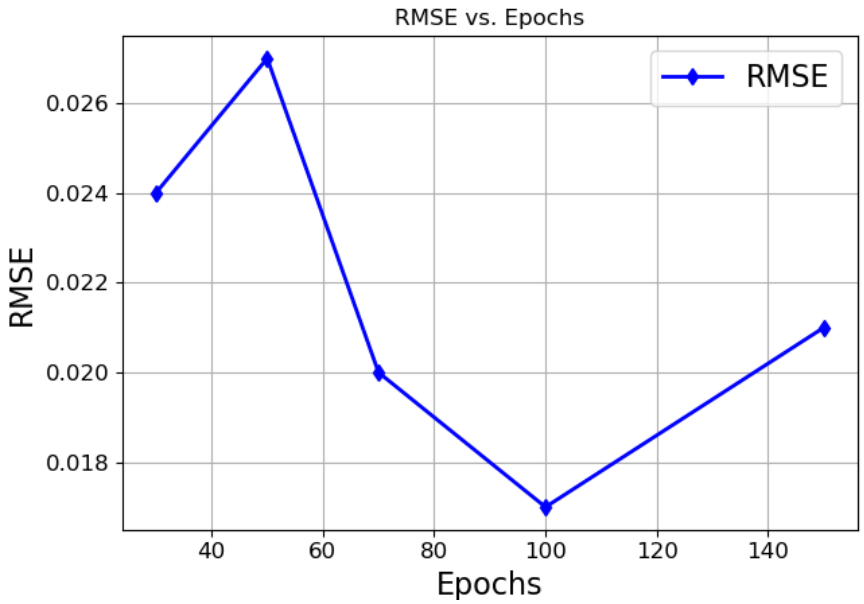
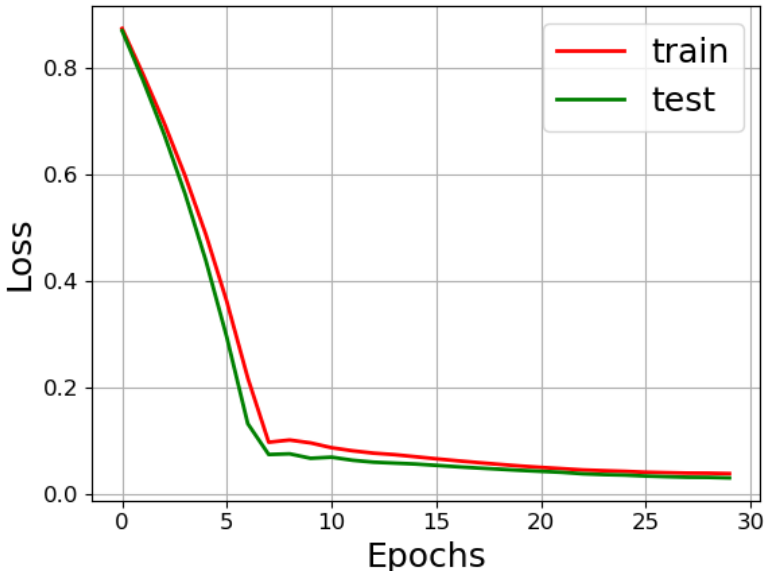
private note 2 views

# Final Project_2017310936_Md_Shirajum_Munir

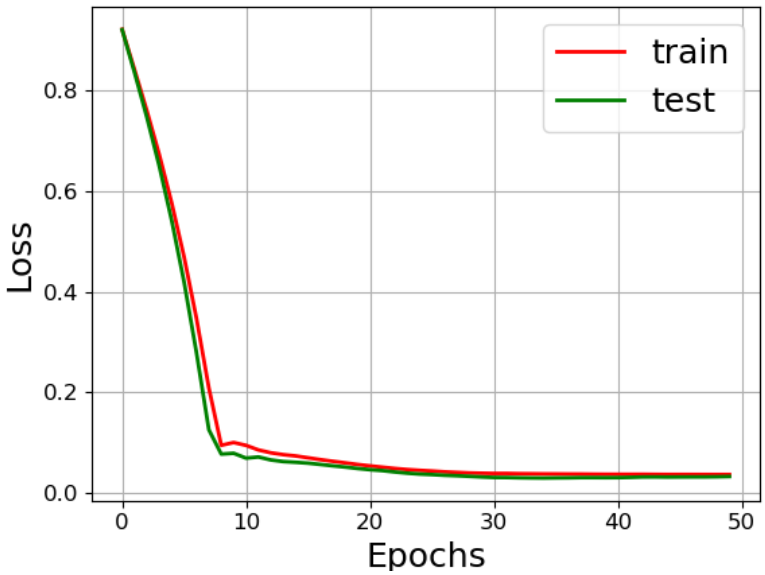
**Project Result:**
**1. RMSE :**



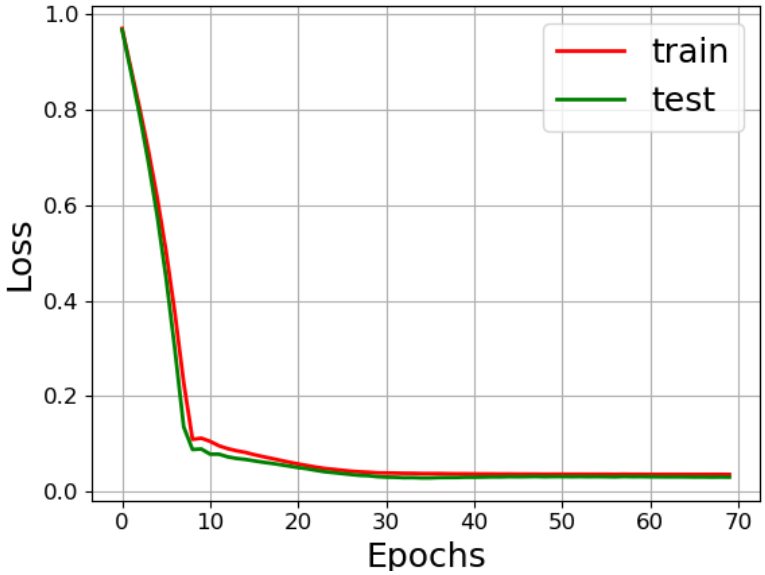

**2. Loss vs. Epochs:**
**30 Epochs:**



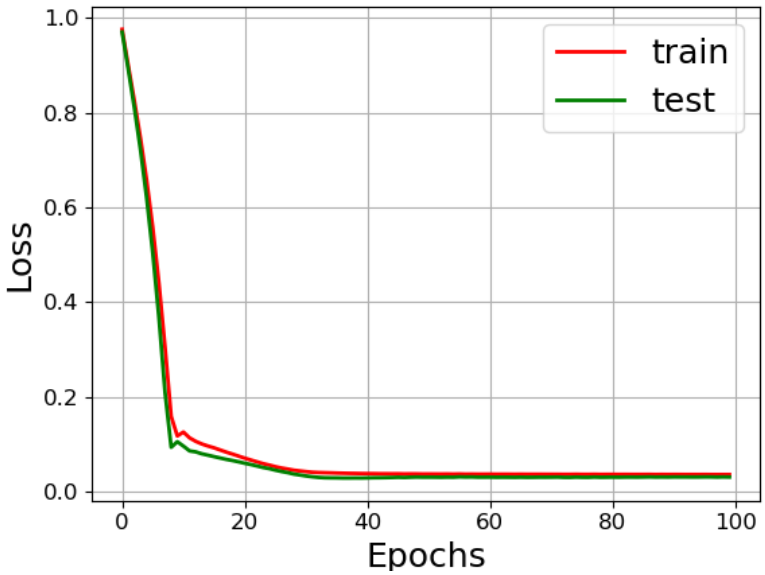

**50 Epochs:**



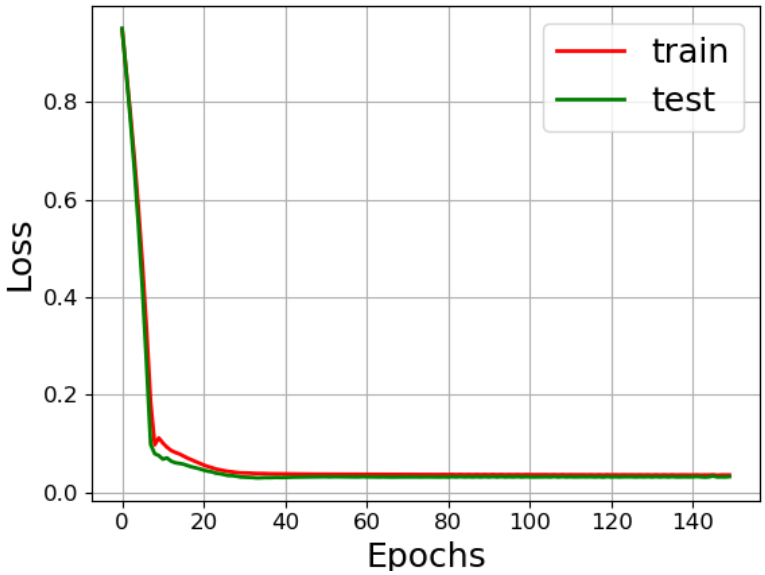

**70 Epochs:**



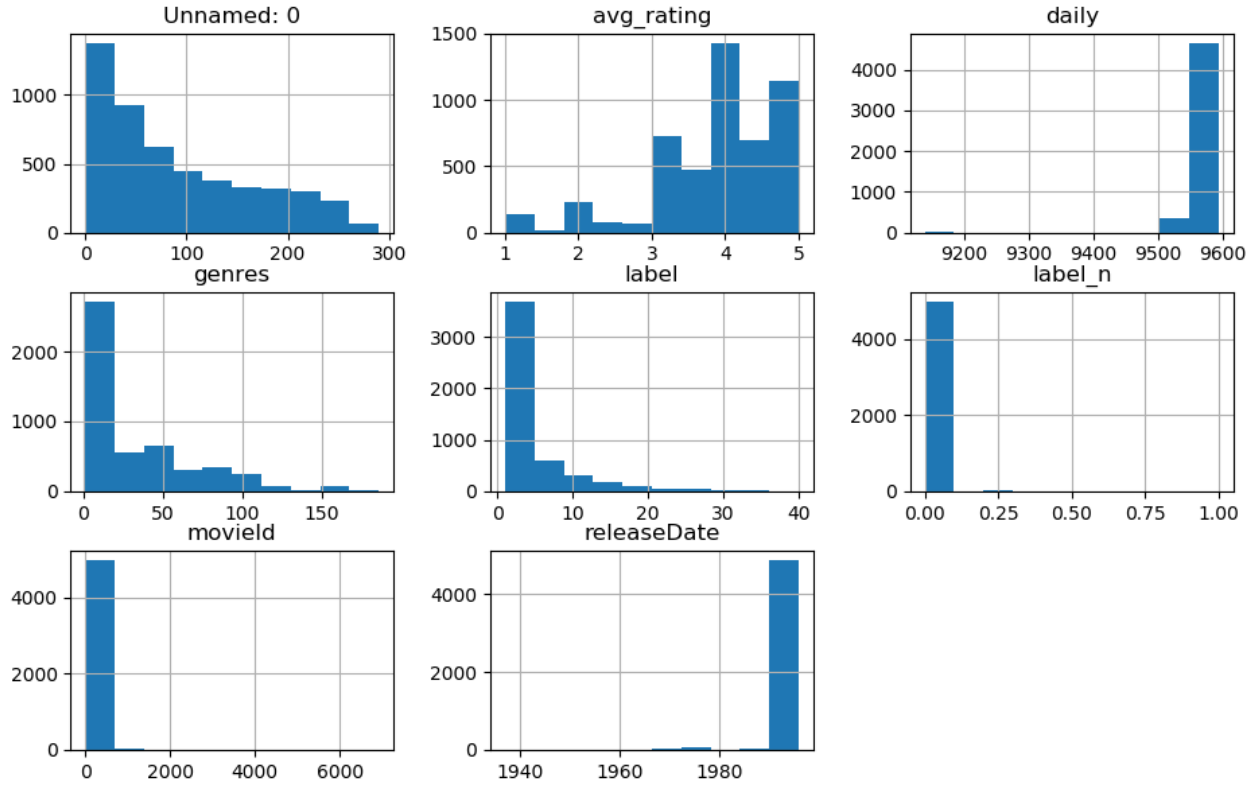

**100 Epochs:**

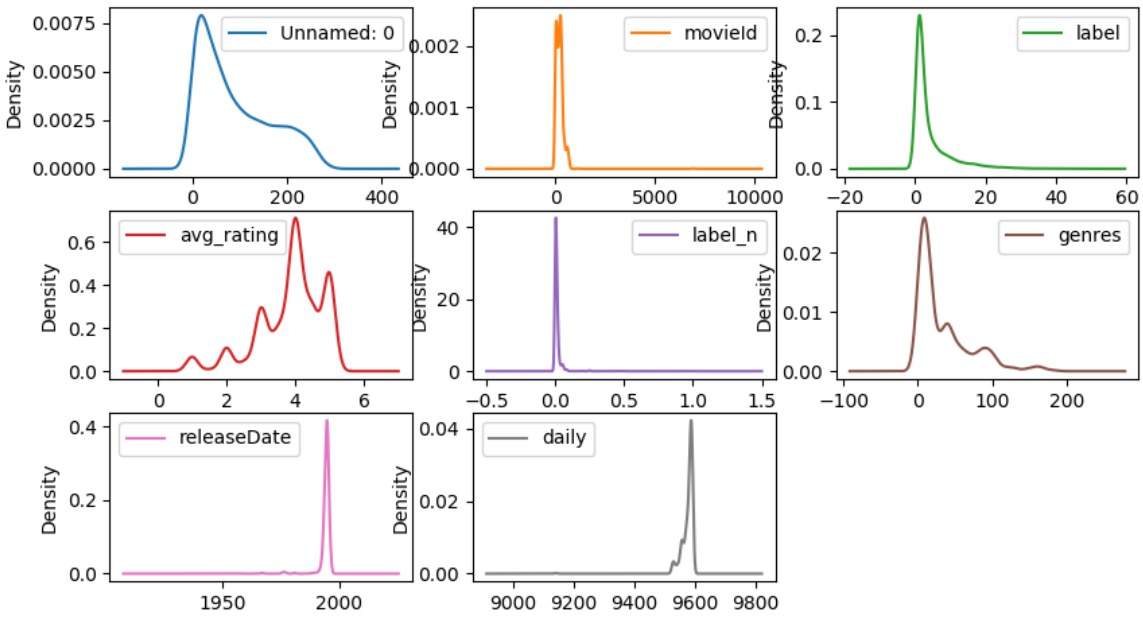


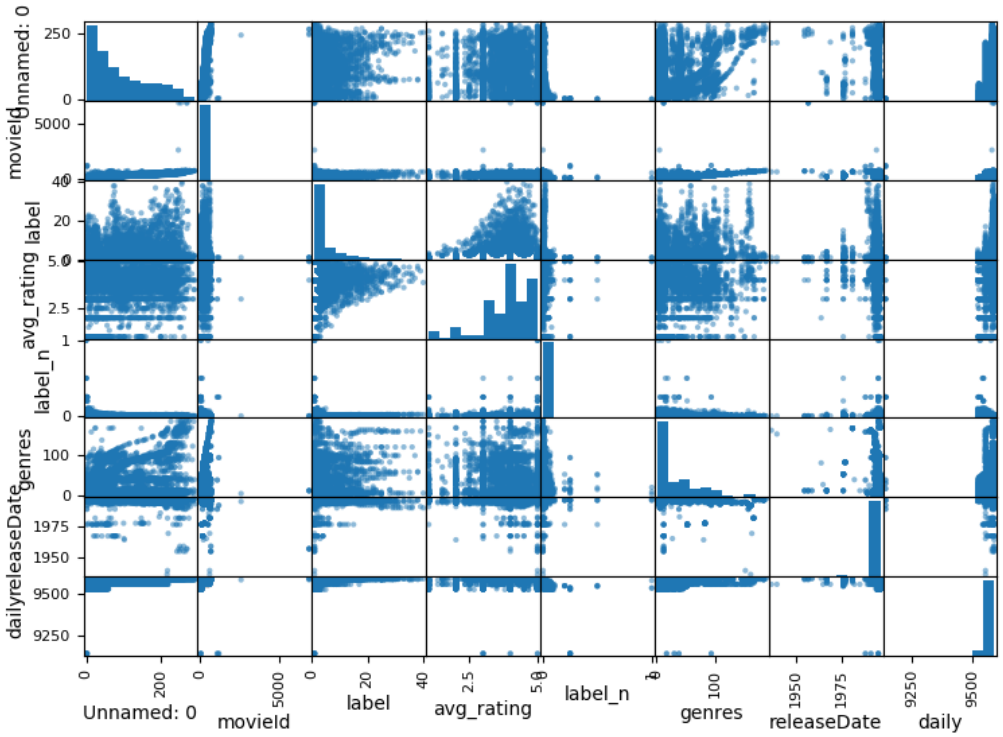

**150 Epochs:**

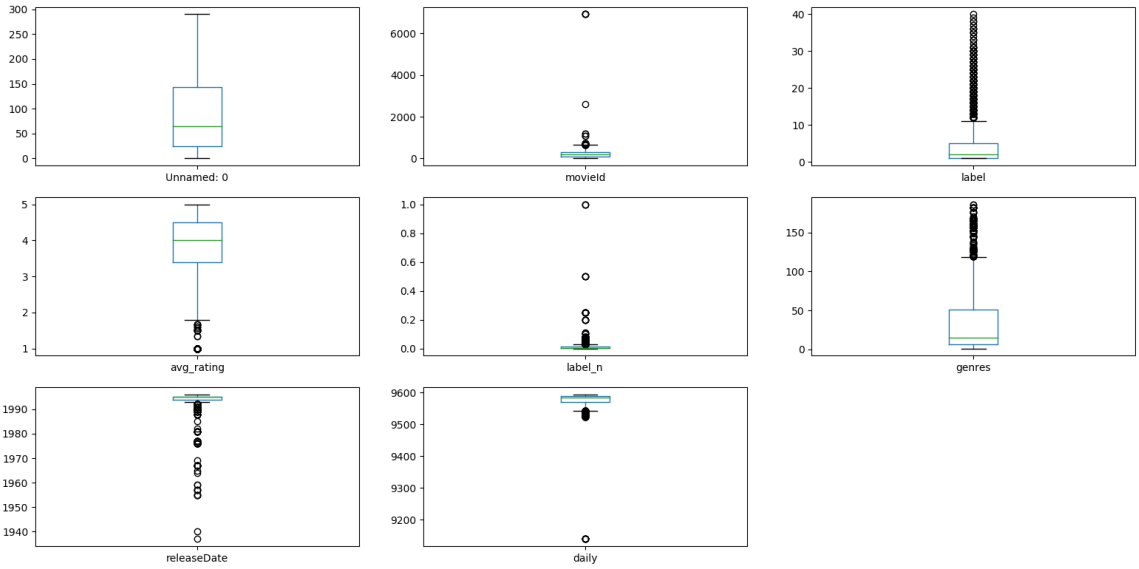


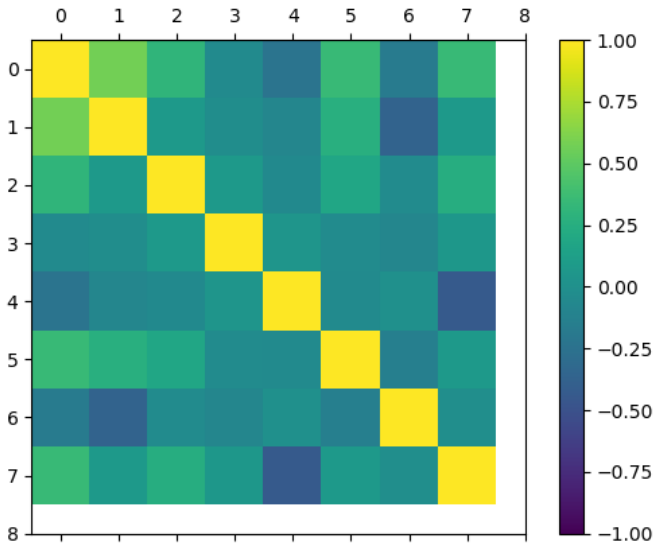

**3. EDA Result**
**All the analysis was done after the preprocessing.**


Unnamed: 0
avg_rating
daily
genres
label
label_n
movieId
releaseDate


Unnamed: 0
movieId
label
avg_rating
label_n
genres
releaseDate
daily
Density


Unnamed: 0
movieId
label
avg_rating
label_n
genres
releaseDate
daily


Unnamed: 0
movieId
label
avg_rating
label_n
genres
releaseDate
daily




**4. Source Code**

**a. LSTM Model:**

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j + 1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j + 1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j + 1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg


# load dataset
dataset = read_csv('dataset_processed/combined.csv',nrows=1000, header=0, index_col=0)
values = dataset.values

print("values ",values)
# integer encode direction
# encoder = LabelEncoder()
# values[:, 4] = encoder.fit_transform(values[:, 4])
# ensure all data is float
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
```

```
# drop columns we don't want to predict
reframed.drop(reframed.columns[[0, 1, 2, 4, 5, 6, 7]], axis=1, inplace=True)
print(reframed.head())


# split into train and test sets
values = reframed.values
n_train_hours = 700
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)


# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=150, batch_size=72, validation_data=(test_X, test_y), verbose=2, shuffle=False)
# plot history
# print( history.history.keys())

import matplotlib.pyplot as plt
import numpy as np

# pyplot.plot(history.history['loss'], label='train')
# pyplot.plot(history.history['val_loss'], label='test')
print("LOST :", history.history['loss'])
ScoreTrain = np.array(history.history['loss'])
print("ScoreTrain",ScoreTrain)
ScoreTrain1 = (100.0-ScoreTrain)
print("ScoreTrain1 ", ScoreTrain1)


t = np.arange(0, 110, 10)
fig, ax = plt.subplots()
ax.plot( history.history['loss'], color='r',  lw=2.0, label='train')
ax.plot(history.history['val_loss'], color='g', lw=2.0, label='test')
plt.ylabel('Loss', fontsize = 18)
plt.xlabel('Epochs', fontsize = 18)
plt.legend(loc='best',fontsize = 18)
ax.grid(True)
ticklines = ax.get_xticklines() + ax.get_yticklines()
gridlines = ax.get_xgridlines()
ticklabels = ax.get_xticklabels() + ax.get_yticklabels()

for line in ticklines:
    line.set_linewidth(3)

for line in gridlines:
    line.set_linestyle('-')

for line in gridlines:
    line.set_linestyle('-')

for label in ticklabels:
    label.set_color('black')
    label.set_fontsize('large')

plt.show()



# pyplot.plot(history.history['acc'], label='acc')
# pyplot.plot(history.history['val_acc'], label='val_acc')
pyplot.legend()
pyplot.show()


# make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))

print("yhat", yhat.shape)
print("test_X", test_X.shape)
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, 1:]), axis=1)
scaler = MinMaxScaler(feature_range=(0, 1)).fit(inv_yhat)
print("inv_yhat", inv_yhat)
print("inv_yhat", inv_yhat.shape)
inv_yhat = scaler.inverse_transform(inv_yhat)
print("After inv_yhat", inv_yhat)
print("After inv_yhat", inv_yhat.shape)
```

```
inv_yhat = inv_yhat[:,0]
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, 1:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

**b. EDA:**

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt

df = pd.read_csv('dataset_processed/combined.csv',nrows=5000)
# print(df.head(3))

print(df)

correlations = df.corr()
# plot correlation matrix
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
# ax.set_xticklabels(names)
# ax.set_yticklabels(names)
plt.show()



df.plot(kind='box', subplots=True, layout=(3,3), sharex=False, sharey=False)
plt.show()

df.plot(kind='density', subplots=True, layout=(3,3), sharex=False)
plt.show()

from pandas.tools.plotting import scatter_matrix
scatter_matrix(df)
plt.show()

df.hist()
plt.show()
```

**c. RMSE Graph:**

```
Output from training model :
30 : Test RMSE: 0.024
50: Test RMSE: 0.027
70: Test RMSE: 0.020
100: Test RMSE: 0.017
150: Test RMSE: 0.021
```

```
import matplotlib.pyplot as plt
import numpy as np

Epochs = [30.0,50.0,70.0,100.0,150.0]
RMSE = [0.024, 0.027, 0.020, 0.017,0.021]

t = np.arange(0, 110, 10)
fig, ax = plt.subplots()
ax.plot( Epochs, RMSE,color='b', marker='d', lw=2.0, label='RMSE')
# ax.plot(history.history['val_loss'], color='g', lw=2.0, label='test')
plt.ylabel('RMSE', fontsize = 16)
plt.xlabel('Epochs', fontsize = 16)
plt.legend(loc='best',fontsize = 16)
plt.title("RMSE vs. Epochs")
ax.grid(True)
ticklines = ax.get_xticklines() + ax.get_yticklines()
gridlines = ax.get_xgridlines()
ticklabels = ax.get_xticklabels() + ax.get_yticklabels()

for line in ticklines:
    line.set_linewidth(2)

for line in gridlines:
    line.set_linestyle('-')
```

```
for line in gridlines:
    line.set_linestyle('-')

for label in ticklabels:
    label.set_color('black')
    label.set_fontsize('large')

plt.show()
```

**d. Preprocessing according to instruction by using below class:**
It takes around 15 hours.

```
maincsv_reader.py
moviecsv_reader.py
ratingcsv_reader.py
```

project

**~ An instructor (Nguyen H. Tran) thinks this is a good note ~**
This private post is only visible to Instructors and MD SHIRAJUM MUNIR

Updated 1 hour ago by MD SHIRAJUM MUNIR

**followup discussions** *for lingering questions and comments*