

lab1

May 19, 2022

0.1 Importing the needed libraries

```
[ ]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
```

0.1.1 Set seed

1. For comparable results set the random seed in both numpy and tensorflow to (123). You can use this code:

```
[ ]: np.random.seed(123)
tf.random.set_seed(123)
```

2. If you don't work in google colab, import the appropriate metric that will be used for optimization.
3. To download the csv file from github use `!wget -nc https://lazyprogrammer.me/course_files/airline_passengers.csv`

0.2 Importing the dataset

4. Create your data frame. To ensure that all works fine index the 'Month' column and parse the dates.

```
[ ]: #Because time series are ordered sequences we need to indicate the column
    ↳ according to which the rows will be indexed
    #parse_date will put the column with dates in the year/month/day format
df = pd.read_csv('airline_passengers.csv', index_col='Month', parse_dates=True)
```

Check how the dataset looks

```
[ ]: df.head()
```

```
[ ]:
      Passengers
Month
1949-01-01      112
1949-02-01      118
1949-03-01      132
1949-04-01      129
```

5. Plot the data and check for stationarity. You can use the Dickey Fuller test to do this. Or if you would rather rely just on visual inspection, argument your conclusion on whether data are stationary or not. Remember, for stationarity you will want a constant variance (i.e. variance is not increasing/decreasing with time) and no trend in your data. If data proved to be non-stationary correct for this problem. (to correct for variance issues you can log transform the data, for trend, you can take the first difference).

0.2.1 Dickey Fuller test

```
[ ]: # Dickey Fuller test
# The null hypothesis of the Dickey Fuller test is that there is a unit root in
  ↳ the time series.
# The alternative hypothesis is that there is no unit root.
# The test statistic is the largest eigenvalue of the covariance matrix of the
  ↳ series.
# The p-value is the probability of obtaining a test statistic as large as the
  ↳ one that was actually observed,
# assuming that the true distribution of the test statistic is normal.
# The test statistic is approximately normally distributed if the p-value is
  ↳ small.
```

```
[ ]: from statsmodels.tsa.stattools import adfuller

def test_stationarity(timeseries):
    # rolling statistics
    rolmean = timeseries.rolling(window=12).mean()
    rolstd = timeseries.rolling(window=12).std()

    # plot rolling statistics:
    orig = plt.plot(timeseries, color='blue', label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    # Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value',
  ↳ '#Lags Used', 'Number of Observations Used'])
    for key, value in dftest[4].items():
        dfoutput['Critical Value (%)' % key] = value
```

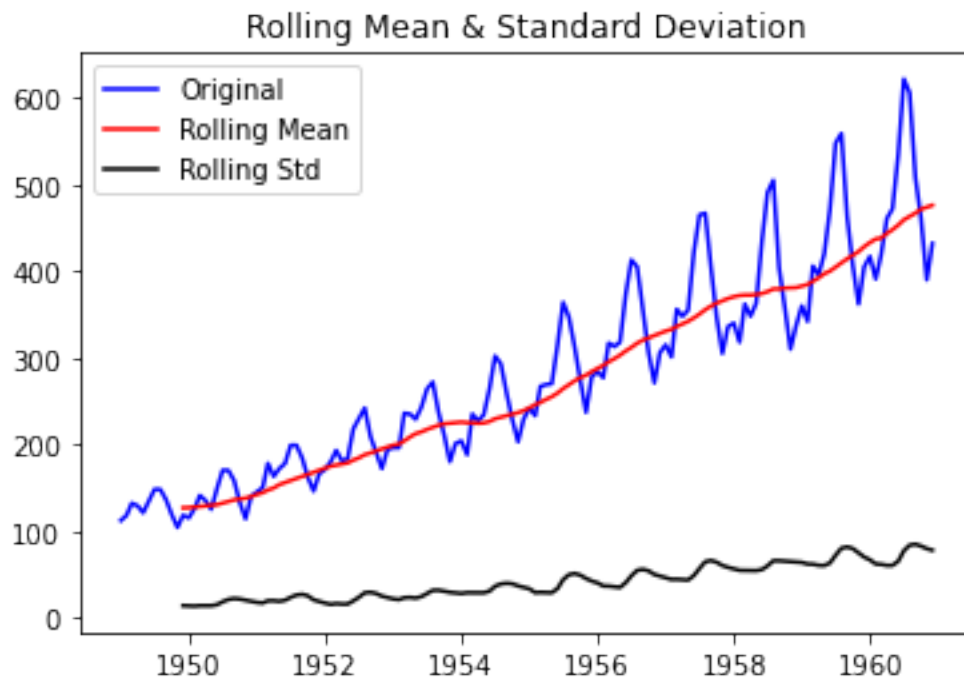
```

    return dfoutput

test_result = test_stationarity(df['Passengers'])
print(test_result)

if test_result['p-value'] < 0.05:
    print('The data is stationary')
else:
    print('The data is not stationary')

```



Results of Dickey-Fuller Test:

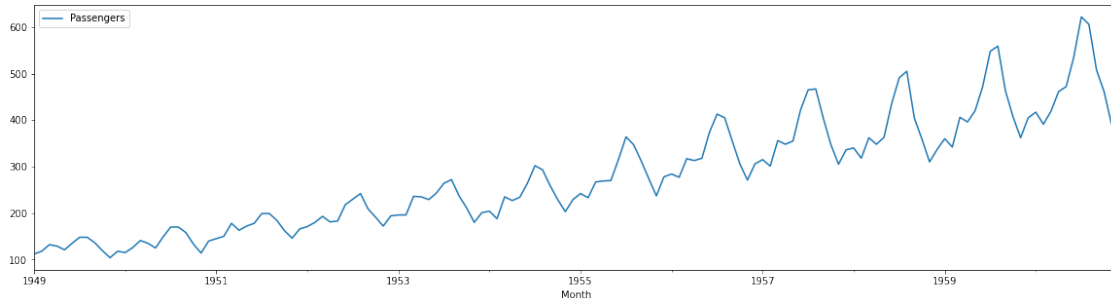
Test Statistic	0.815369
p-value	0.991880
#Lags Used	13.000000
Number of Observations Used	130.000000
Critical Value (1%)	-3.481682
Critical Value (5%)	-2.884042
Critical Value (10%)	-2.578770
dtype: float64	
The data is not stationary	

As we can see, the data are not stationary because p-value is greater than 0.05.

0.2.2 Visualizing the data to detect non-stationarity, variance and trend

```
[ ]: df.plot(figsize=(20, 5))
```

```
[ ]: <AxesSubplot:xlabel='Month'>
```

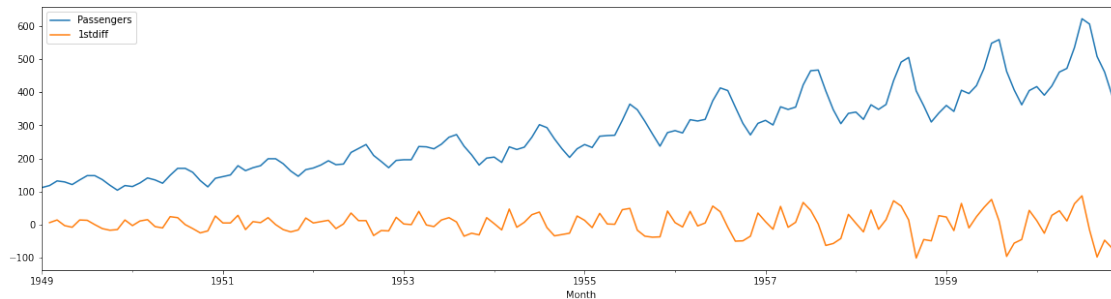


From the graph we can see that the data is not stationary it also has a positive trend. There is also seasonality and variance. The variance is increasing in time which is called heteroskedasticity.

Fixing the data We will first take the first difference to see if it fixes any of the problems mentioned above.

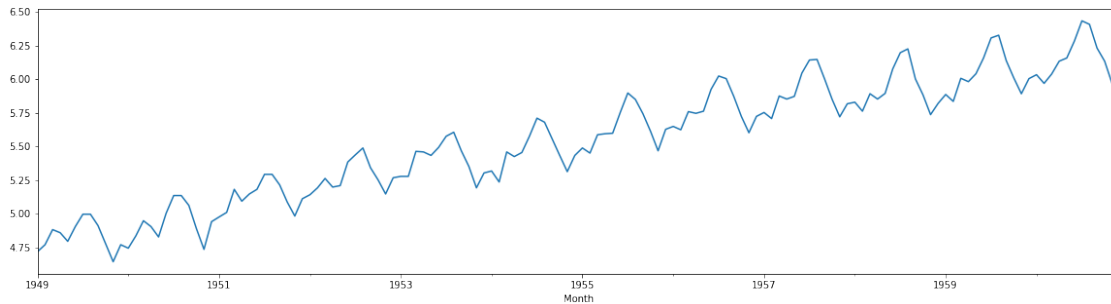
```
[ ]: df['1stdiff'] = df['Passengers'].diff()
```

```
[ ]: df.plot(figsize=(20, 5));
```



After taking the first difference we can see that the trend was removed. But there is still some seasonality and a variance that increases with time. Since it is common to use log function together with the first difference in order to solve the non-stationarity issues, next will be used the log transformation and the results will be analyzed.

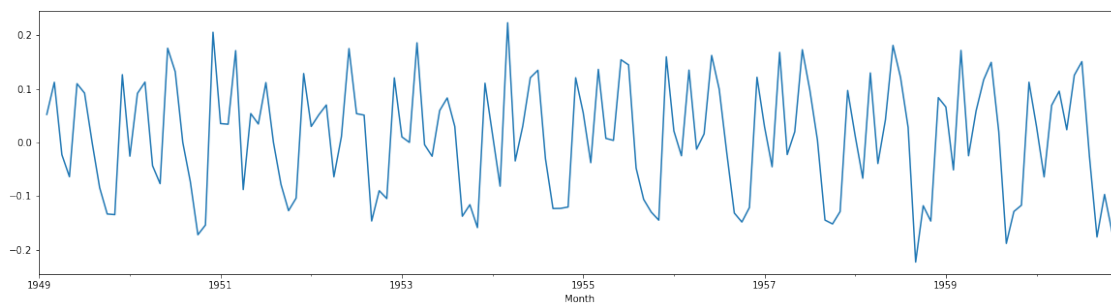
```
[ ]: df['LogPassengers'] = np.log(df['Passengers'])  
df['LogPassengers'].plot(figsize=(20, 5));
```



We can see that the log function helped to remove the variance with time. The log transformation did not remove the trend, but the differenciation did, because of this we will use the two functions combined in the next step. The log transformation did not solve the problem with seasonality and heteroskedasticity.

Use Log transformation in combination with the first difference

```
[ ]: df['Log1stDiff'] = df['LogPassengers'].diff()
      df['Log1stDiff'].plot(figsize=(20, 5));
```



This time we have got a much better result. The result is not perfect because there is some cyclicity, with this should work for our model.

0.2.3 Create the train and test sets

Because time series are sequences, we'll take the y values for the last 12 months and assign them to the test set, and the rest will go into the training set.

```
[ ]: df.index.freq = 'MS'

Ntest = 12
train = df.iloc[:-Ntest]
test = df.iloc[-Ntest:]
```

7. Create the index arrays for both train and test datasets. If you don't remember how to do this from our ML class here is one example: `train_idx = df.index <= train.index[-1]`, so just

do the same for the test set.

```
[ ]: train_idx = df.index <= train.index[-1]
test_idx = df.index > train.index[-1]
```

```
[ ]: # taking a look at the dataset
print(df)

# get first value of Passengers, needed for reverse differencing
first_passengers_value = df.loc[train.index[0], 'Passengers']
```

	Passengers	1stdiff	LogPassengers	Log1stDiff
Month				
1949-01-01	112	NaN	4.718499	NaN
1949-02-01	118	6.0	4.770685	0.052186
1949-03-01	132	14.0	4.882802	0.112117
1949-04-01	129	-3.0	4.859812	-0.022990
1949-05-01	121	-8.0	4.795791	-0.064022
...
1960-08-01	606	-16.0	6.406880	-0.026060
1960-09-01	508	-98.0	6.230481	-0.176399
1960-10-01	461	-47.0	6.133398	-0.097083
1960-11-01	390	-71.0	5.966147	-0.167251
1960-12-01	432	42.0	6.068426	0.102279

[144 rows x 4 columns]

8. Create the appropriate data structure for a time series analysis where the past 10 datapoints will be used to make predictions of the next 1 datapoint.

```
[ ]: # prepare dataset for RNN
def prepare_dataset(df, look_back=1):
    X, Y = [], []
    for i in range(len(df) - look_back):
        X.append(df[i:(i+look_back)])
        Y.append(df[i+look_back])
    return np.array(X), np.array(Y)

look_back = 10
n_features = 1

# using Log1stDiff
df = df.fillna(0)
X, Y = prepare_dataset(df['Log1stDiff'], look_back)

for i in range(len(X)):
    print(X[i], Y[i])
```

[0. 0.05218575 0.1121173 -0.02298952 -0.06402186 0.10948423
0.0919375 0. -0.08455739 -0.13353139] -0.13473259397015713
[0.05218575 0.1121173 -0.02298952 -0.06402186 0.10948423 0.0919375
0. -0.08455739 -0.13353139 -0.13473259] 0.12629372532429262
[0.1121173 -0.02298952 -0.06402186 0.10948423 0.0919375 0.
-0.08455739 -0.13353139 -0.13473259 0.12629373] -0.025752496102414923
[-0.02298952 -0.06402186 0.10948423 0.0919375 0. -0.08455739
-0.13353139 -0.13473259 0.12629373 -0.0257525] 0.09134977858822779
[-0.06402186 0.10948423 0.0919375 0. -0.08455739 -0.13353139
-0.13473259 0.12629373 -0.0257525 0.09134978] 0.11247798342669046
[0.10948423 0.0919375 0. -0.08455739 -0.13353139 -0.13473259
0.12629373 -0.0257525 0.09134978 0.11247798] -0.04348511193973881
[0.0919375 0. -0.08455739 -0.13353139 -0.13473259 0.12629373
-0.0257525 0.09134978 0.11247798 -0.04348511] -0.07696104113612812
[0. -0.08455739 -0.13353139 -0.13473259 0.12629373 -0.0257525
0.09134978 0.11247798 -0.04348511 -0.07696104] 0.17563256864315768
[-0.08455739 -0.13353139 -0.13473259 0.12629373 -0.0257525 0.09134978
0.11247798 -0.04348511 -0.07696104 0.17563257] 0.13185213110480287
[-0.13353139 -0.13473259 0.12629373 -0.0257525 0.09134978 0.11247798
-0.04348511 -0.07696104 0.17563257 0.13185213] 0.0
[-0.13473259 0.12629373 -0.0257525 0.09134978 0.11247798 -0.04348511
-0.07696104 0.17563257 0.13185213 0.] -0.07320340402329517
[0.12629373 -0.0257525 0.09134978 0.11247798 -0.04348511 -0.07696104
0.17563257 0.13185213 0. -0.0732034] -0.17224590480521318
[-0.0257525 0.09134978 0.11247798 -0.04348511 -0.07696104 0.17563257
0.13185213 0. -0.0732034 -0.1722459] -0.15415067982725805
[0.09134978 0.11247798 -0.04348511 -0.07696104 0.17563257 0.13185213
0. -0.0732034 -0.1722459 -0.15415068] 0.2054439742148082
[0.11247798 -0.04348511 -0.07696104 0.17563257 0.13185213 0.
-0.0732034 -0.1722459 -0.15415068 0.20544397] 0.03509131981127034
[-0.04348511 -0.07696104 0.17563257 0.13185213 0. -0.0732034
-0.1722459 -0.15415068 0.20544397 0.03509132] 0.03390155167568132
[-0.07696104 0.17563257 0.13185213 0. -0.0732034 -0.1722459
-0.15415068 0.20544397 0.03509132 0.03390155] 0.17114825619582952
[0.17563257 0.13185213 0. -0.0732034 -0.1722459 -0.15415068
0.20544397 0.03509132 0.03390155 0.17114826] -0.08803334948532271
[0.13185213 0. -0.0732034 -0.1722459 -0.15415068 0.20544397
0.03509132 0.03390155 0.17114826 -0.08803335] 0.053744276006690406
[0. -0.0732034 -0.1722459 -0.15415068 0.20544397 0.03509132
0.03390155 0.17114826 -0.08803335 0.05374428] 0.034289073478632304
[-0.0732034 -0.1722459 -0.15415068 0.20544397 0.03509132 0.03390155
0.17114826 -0.08803335 0.05374428 0.03428907] 0.11152127443240722
[-0.1722459 -0.15415068 0.20544397 0.03509132 0.03390155 0.17114826
-0.08803335 0.05374428 0.03428907 0.11152127] 0.0
[-0.15415068 0.20544397 0.03509132 0.03390155 0.17114826 -0.08803335
0.05374428 0.03428907 0.11152127 0.] -0.0783690671155064
[0.20544397 0.03509132 0.03390155 0.17114826 -0.08803335 0.05374428
0.03428907 0.11152127 0. -0.07836907] -0.12733942237660223

[0.03509132 0.03390155 0.17114826 -0.08803335 0.05374428 0.03428907
0.11152127 0. -0.07836907 -0.12733942] -0.10398971352404729
[0.03390155 0.17114826 -0.08803335 0.05374428 0.03428907 0.11152127
0. -0.07836907 -0.12733942 -0.10398971] 0.1283811666482073
[0.17114826 -0.08803335 0.05374428 0.03428907 0.11152127 0.
-0.07836907 -0.12733942 -0.10398971 0.12838117] 0.02967576814611661
[-0.08803335 0.05374428 0.03428907 0.11152127 0. -0.07836907
-0.12733942 -0.10398971 0.12838117 0.02967577] 0.05129329438755015
[0.05374428 0.03428907 0.11152127 0. -0.07836907 -0.12733942
-0.10398971 0.12838117 0.02967577 0.05129329] 0.06973333801467518
[0.03428907 0.11152127 0. -0.07836907 -0.12733942 -0.10398971
0.12838117 0.02967577 0.05129329 0.06973334] -0.06419315763905953
[0.11152127 0. -0.07836907 -0.12733942 -0.10398971 0.12838117
0.02967577 0.05129329 0.06973334 -0.06419316] 0.010989121575595284
[0. -0.07836907 -0.12733942 -0.10398971 0.12838117 0.02967577
0.05129329 0.06973334 -0.06419316 0.01098912] 0.1750089099476675
[-0.07836907 -0.12733942 -0.10398971 0.12838117 0.02967577 0.05129329
0.06973334 -0.06419316 0.01098912 0.17500891] 0.05358424613410673
[-0.12733942 -0.10398971 0.12838117 0.02967577 0.05129329 0.06973334
-0.06419316 0.01098912 0.17500891 0.05358425] 0.05085841723349116
[-0.10398971 0.12838117 0.02967577 0.05129329 0.06973334 -0.06419316
0.01098912 0.17500891 0.05358425 0.05085842] -0.1466034741918758
[0.12838117 0.02967577 0.05129329 0.06973334 -0.06419316 0.01098912
0.17500891 0.05358425 0.05085842 -0.14660347] -0.09006082391818104
[0.02967577 0.05129329 0.06973334 -0.06419316 0.01098912 0.17500891
0.05358425 0.05085842 -0.14660347 -0.09006082] -0.10477895123317715
[0.05129329 0.06973334 -0.06419316 0.01098912 0.17500891 0.05358425
0.05085842 -0.14660347 -0.09006082 -0.10477895] 0.12036368224987548
[0.06973334 -0.06419316 0.01098912 0.17500891 0.05358425 0.05085842
-0.14660347 -0.09006082 -0.10477895 0.12036368] 0.010256500167188598
[-0.06419316 0.01098912 0.17500891 0.05358425 0.05085842 -0.14660347
-0.09006082 -0.10477895 0.12036368 0.0102565] 0.0
[0.01098912 0.17500891 0.05358425 0.05085842 -0.14660347 -0.09006082
-0.10477895 0.12036368 0.0102565 0.] 0.18571714579509369
[0.17500891 0.05358425 0.05085842 -0.14660347 -0.09006082 -0.10477895
0.12036368 0.0102565 0. 0.18571715] -0.004246290881451564
[0.05358425 0.05085842 -0.14660347 -0.09006082 -0.10477895 0.12036368
0.0102565 0. 0.18571715 -0.00424629] -0.02586351058991898
[0.05085842 -0.14660347 -0.09006082 -0.10477895 0.12036368 0.0102565
0. 0.18571715 -0.00424629 -0.02586351] 0.05933943978630829
[-0.14660347 -0.09006082 -0.10477895 0.12036368 0.0102565 0.
0.18571715 -0.00424629 -0.02586351 0.05933944] 0.08288765980576773
[-0.09006082 -0.10477895 0.12036368 0.0102565 0. 0.18571715
-0.00424629 -0.02586351 0.05933944 0.08288766] 0.029852963149681777
[-0.10477895 0.12036368 0.0102565 0. 0.18571715 -0.00424629
-0.02586351 0.05933944 0.08288766 0.02985296] -0.13774192516086625
[0.12036368 0.0102565 0. 0.18571715 -0.00424629 -0.02586351
0.05933944 0.08288766 0.02985296 -0.13774193] -0.11620200765906485

[0.0102565 0. 0.18571715 -0.00424629 -0.02586351 0.05933944
 0.08288766 0.02985296 -0.13774193 -0.11620201] -0.15890128258585623
 [0. 0.18571715 -0.00424629 -0.02586351 0.05933944 0.08288766
 0.02985296 -0.13774193 -0.11620201 -0.15890128] 0.11034805716886531
 [0.18571715 -0.00424629 -0.02586351 0.05933944 0.08288766 0.02985296
 -0.13774193 -0.11620201 -0.15890128 0.11034806] 0.014815085785140347
 [-0.00424629 -0.02586351 0.05933944 0.08288766 0.02985296 -0.13774193
 -0.11620201 -0.15890128 0.11034806 0.01481509] -0.08167803101426685
 [-0.02586351 0.05933944 0.08288766 0.02985296 -0.13774193 -0.11620201
 -0.15890128 0.11034806 0.01481509 -0.08167803] 0.2231435513142097
 [0.05933944 0.08288766 0.02985296 -0.13774193 -0.11620201 -0.15890128
 0.11034806 0.01481509 -0.08167803 0.22314355] -0.034635496662756005
 [0.08288766 0.02985296 -0.13774193 -0.11620201 -0.15890128 0.11034806
 0.01481509 -0.08167803 0.22314355 -0.0346355] 0.03037109787629877
 [0.02985296 -0.13774193 -0.11620201 -0.15890128 0.11034806 0.01481509
 -0.08167803 0.22314355 -0.0346355 0.0303711] 0.12062798778861428
 [-0.13774193 -0.11620201 -0.15890128 0.11034806 0.01481509 -0.08167803
 0.22314355 -0.0346355 0.0303711 0.12062799] 0.1344779142285537
 [-0.11620201 -0.15890128 0.11034806 0.01481509 -0.08167803 0.22314355
 -0.0346355 0.0303711 0.12062799 0.13447791] -0.03025440835780202
 [-0.15890128 0.11034806 0.01481509 -0.08167803 0.22314355 -0.0346355
 0.0303711 0.12062799 0.13447791 -0.03025441] -0.1233445473175303
 [0.11034806 0.01481509 -0.08167803 0.22314355 -0.0346355 0.0303711
 0.12062799 0.13447791 -0.03025441 -0.12334455] -0.1231060581452974
 [0.01481509 -0.08167803 0.22314355 -0.0346355 0.0303711 0.12062799
 0.13447791 -0.03025441 -0.12334455 -0.12310606] -0.1205160245124528
 [-0.08167803 0.22314355 -0.0346355 0.0303711 0.12062799 0.13447791
 -0.03025441 -0.12334455 -0.12310606 -0.12051602] 0.1205160245124528
 [0.22314355 -0.0346355 0.0303711 0.12062799 0.13447791 -0.03025441
 -0.12334455 -0.12310606 -0.12051602 0.12051602] 0.05521572260244678
 [-0.0346355 0.0303711 0.12062799 0.13447791 -0.03025441 -0.12334455
 -0.12310606 -0.12051602 0.12051602 0.05521572] -0.03789927259098658
 [0.0303711 0.12062799 0.13447791 -0.03025441 -0.12334455 -0.12310606
 -0.12051602 0.12051602 0.05521572 -0.03789927] 0.1362102048345495
 [0.12062799 0.13447791 -0.03025441 -0.12334455 -0.12310606 -0.12051602
 0.12051602 0.05521572 -0.03789927 0.1362102] 0.007462721201589417
 [0.13447791 -0.03025441 -0.12334455 -0.12310606 -0.12051602 0.12051602
 0.05521572 -0.03789927 0.1362102 0.00746272] 0.0037105793965359624
 [-0.03025441 -0.12334455 -0.12310606 -0.12051602 0.12051602 0.05521572
 -0.03789927 0.1362102 0.00746272 0.00371058] 0.15415067982725805
 [-0.12334455 -0.12310606 -0.12051602 0.12051602 0.05521572 -0.03789927
 0.1362102 0.00746272 0.00371058 0.15415068] 0.14458122881110747
 [-0.12310606 -0.12051602 0.12051602 0.05521572 -0.03789927 0.1362102
 0.00746272 0.00371058 0.15415068 0.14458123] -0.047829087689881256
 [-0.12051602 0.12051602 0.05521572 -0.03789927 0.1362102 0.00746272
 0.00371058 0.15415068 0.14458123 -0.04782909] -0.1063215921373768
 [0.12051602 0.05521572 -0.03789927 0.1362102 0.00746272 0.00371058
 0.15415068 0.14458123 -0.04782909 -0.10632159] -0.12987508142141202

[0.05521572 -0.03789927 0.1362102 0.00746272 0.00371058 0.15415068
 0.14458123 -0.04782909 -0.10632159 -0.12987508] -0.14506796525293897
 [-0.03789927 0.1362102 0.00746272 0.00371058 0.15415068 0.14458123
 -0.04782909 -0.10632159 -0.12987508 -0.14506797] 0.1595609725555054
 [0.1362102 0.00746272 0.00371058 0.15415068 0.14458123 -0.04782909
 -0.10632159 -0.12987508 -0.14506797 0.15956097] 0.021353124470569362
 [0.00746272 0.00371058 0.15415068 0.14458123 -0.04782909 -0.10632159
 -0.12987508 -0.14506797 0.15956097 0.02135312] -0.02495673197386772
 [0.00371058 0.15415068 0.14458123 -0.04782909 -0.10632159 -0.12987508
 -0.14506797 0.15956097 0.02135312 -0.02495673] 0.13488426768994177
 [0.15415068 0.14458123 -0.04782909 -0.10632159 -0.12987508 -0.14506797
 0.15956097 0.02135312 -0.02495673 0.13488427] -0.012698583337127367
 [0.14458123 -0.04782909 -0.10632159 -0.12987508 -0.14506797 0.15956097
 0.02135312 -0.02495673 0.13488427 -0.01269858] 0.015848192240023984
 [-0.04782909 -0.10632159 -0.12987508 -0.14506797 0.15956097 0.02135312
 -0.02495673 0.13488427 -0.01269858 0.01584819] 0.1622044146343553
 [-0.10632159 -0.12987508 -0.14506797 0.15956097 0.02135312 -0.02495673
 0.13488427 -0.01269858 0.01584819 0.16220441] 0.09919179554650093
 [-0.12987508 -0.14506797 0.15956097 0.02135312 -0.02495673 0.13488427
 -0.01269858 0.01584819 0.16220441 0.0991918] -0.019560525854494415
 [-0.14506797 0.15956097 0.02135312 -0.02495673 0.13488427 -0.01269858
 0.01584819 0.16220441 0.0991918 -0.01956053] -0.13176927763112278
 [0.15956097 0.02135312 -0.02495673 0.13488427 -0.01269858 0.01584819
 0.16220441 0.0991918 -0.01956053 -0.13176928] -0.14853268752303528
 [0.02135312 -0.02495673 0.13488427 -0.01269858 0.01584819 0.16220441
 0.0991918 -0.01956053 -0.13176928 -0.14853269] -0.12146628107267965
 [-0.02495673 0.13488427 -0.01269858 0.01584819 0.16220441 0.0991918
 -0.01956053 -0.13176928 -0.14853269 -0.12146628] 0.12146628107267965
 [0.13488427 -0.01269858 0.01584819 0.16220441 0.0991918 -0.01956053
 -0.13176928 -0.14853269 -0.12146628 0.12146628] 0.028987536873252395
 [-0.01269858 0.01584819 0.16220441 0.0991918 -0.01956053 -0.13176928
 -0.14853269 -0.12146628 0.12146628 0.02898754] -0.04546237407675768
 [0.01584819 0.16220441 0.0991918 -0.01956053 -0.13176928 -0.14853269
 -0.12146628 0.12146628 0.02898754 -0.04546237] 0.16782046610315504
 [0.16220441 0.0991918 -0.01956053 -0.13176928 -0.14853269 -0.12146628
 0.12146628 0.02898754 -0.04546237 0.16782047] -0.02272825107755594
 [0.0991918 -0.01956053 -0.13176928 -0.14853269 -0.12146628 0.12146628
 0.02898754 -0.04546237 0.16782047 -0.02272825] 0.019915309700941464
 [-0.01956053 -0.13176928 -0.14853269 -0.12146628 0.12146628 0.02898754
 -0.04546237 0.16782047 -0.02272825 0.01991531] 0.17288752456059608
 [-0.13176928 -0.14853269 -0.12146628 0.12146628 0.02898754 -0.04546237
 0.16782047 -0.02272825 0.01991531 0.17288752] 0.09703209155134385
 [-0.14853269 -0.12146628 0.12146628 0.02898754 -0.04546237 0.16782047
 -0.02272825 0.01991531 0.17288752 0.09703209] 0.004291852081541592
 [-0.12146628 0.12146628 0.02898754 -0.04546237 0.16782047 -0.02272825
 0.01991531 0.17288752 0.09703209 0.00429185] -0.144914379707747
 [0.12146628 0.02898754 -0.04546237 0.16782047 -0.02272825 0.01991531
 0.17288752 0.09703209 0.00429185 -0.14491438] -0.1520900980142912

[0.02898754 -0.04546237 0.16782047 -0.02272825 0.01991531 0.17288752
 0.09703209 0.00429185 -0.14491438 -0.1520901] -0.12901300333944743
 [-0.04546237 0.16782047 -0.02272825 0.01991531 0.17288752 0.09703209
 0.00429185 -0.14491438 -0.1520901 -0.129013] 0.0967993833557923
 [0.16782047 -0.02272825 0.01991531 0.17288752 0.09703209 0.00429185
 -0.14491438 -0.1520901 -0.129013 0.09679938] 0.011834457647003305
 [-0.02272825 0.01991531 0.17288752 0.09703209 0.00429185 -0.14491438
 -0.1520901 -0.129013 0.09679938 0.01183446] -0.06689423483003054
 [0.01991531 0.17288752 0.09703209 0.00429185 -0.14491438 -0.1520901
 -0.129013 0.09679938 0.01183446 -0.06689423] 0.12959282904559455
 [0.17288752 0.09703209 0.00429185 -0.14491438 -0.1520901 -0.129013
 0.09679938 0.01183446 -0.06689423 0.12959283] -0.039441732051296974
 [0.09703209 0.00429185 -0.14491438 -0.1520901 -0.129013 0.09679938
 0.01183446 -0.06689423 0.12959283 -0.03944173] 0.04220035449037596
 [0.00429185 -0.14491438 -0.1520901 -0.129013 0.09679938 0.01183446
 -0.06689423 0.12959283 -0.03944173 0.04220035] 0.18094319682383375
 [-0.14491438 -0.1520901 -0.129013 0.09679938 0.01183446 -0.06689423
 0.12959283 -0.03944173 0.04220035 0.1809432] 0.12109809670583616
 [-0.1520901 -0.129013 0.09679938 0.01183446 -0.06689423 0.12959283
 -0.03944173 0.04220035 0.1809432 0.1210981] 0.02811430148083982
 [-0.129013 0.09679938 0.01183446 -0.06689423 0.12959283 -0.03944173
 0.04220035 0.1809432 0.1210981 0.0281143] -0.2231435513142097
 [0.09679938 0.01183446 -0.06689423 0.12959283 -0.03944173 0.04220035
 0.1809432 0.1210981 0.0281143 -0.22314355] -0.11809248947287188
 [0.01183446 -0.06689423 0.12959283 -0.03944173 0.04220035 0.1809432
 0.1210981 0.0281143 -0.22314355 -0.11809249] -0.14675009100908643
 [-0.06689423 0.12959283 -0.03944173 0.04220035 0.1809432 0.1210981
 0.0281143 -0.22314355 -0.11809249 -0.14675009] 0.0835106328731694
 [0.12959283 -0.03944173 0.04220035 0.1809432 0.1210981 0.0281143
 -0.22314355 -0.11809249 -0.14675009 0.08351063] 0.06602110109779424
 [-0.03944173 0.04220035 0.1809432 0.1210981 0.0281143 -0.22314355
 -0.11809249 -0.14675009 0.08351063 0.0660211] -0.051293294387551036
 [0.04220035 0.1809432 0.1210981 0.0281143 -0.22314355 -0.11809249
 -0.14675009 0.08351063 0.0660211 -0.05129329] 0.17154242253912777
 [0.1809432 0.1210981 0.0281143 -0.22314355 -0.11809249 -0.14675009
 0.08351063 0.0660211 -0.05129329 0.17154242] -0.024938948347251966
 [0.1210981 0.0281143 -0.22314355 -0.11809249 -0.14675009 0.08351063
 0.0660211 -0.05129329 0.17154242 -0.02493895] 0.058840500022933284
 [0.0281143 -0.22314355 -0.11809249 -0.14675009 0.08351063 0.0660211
 -0.05129329 0.17154242 -0.02493895 0.0588405] 0.11672427430814114
 [-0.22314355 -0.11809249 -0.14675009 0.08351063 0.0660211 -0.05129329
 0.17154242 -0.02493895 0.0588405 0.11672427] 0.14929630136246086
 [-0.11809249 -0.14675009 0.08351063 0.0660211 -0.05129329 0.17154242
 -0.02493895 0.0588405 0.11672427 0.1492963] 0.019874186207083255
 [-0.14675009 0.08351063 0.0660211 -0.05129329 0.17154242 -0.02493895
 0.0588405 0.11672427 0.1492963 0.01987419] -0.18842241906886503
 [0.08351063 0.0660211 -0.05129329 0.17154242 -0.02493895 0.0588405
 0.11672427 0.1492963 0.01987419 -0.18842242] -0.12891386864363952

```
[ 0.0660211 -0.05129329  0.17154242 -0.02493895  0.0588405  0.11672427
 0.1492963  0.01987419 -0.18842242 -0.12891387] -0.1171689736168231
[-0.05129329  0.17154242 -0.02493895  0.0588405  0.11672427  0.1492963
 0.01987419 -0.18842242 -0.12891387 -0.11716897] 0.11224285528076727
[ 0.17154242 -0.02493895  0.0588405  0.11672427  0.1492963  0.01987419
-0.18842242 -0.12891387 -0.11716897  0.11224286] 0.029199154692262752
[-0.02493895  0.0588405  0.11672427  0.1492963  0.01987419 -0.18842242
-0.12891387 -0.11716897  0.11224286  0.02919915] -0.06437866181343566
[ 0.0588405  0.11672427  0.1492963  0.01987419 -0.18842242 -0.12891387
-0.11716897  0.11224286  0.02919915 -0.06437866] 0.06916335993677158
[ 0.11672427  0.1492963  0.01987419 -0.18842242 -0.12891387 -0.11716897
 0.11224286  0.02919915 -0.06437866  0.06916336] 0.09552712307451117
[ 0.1492963  0.01987419 -0.18842242 -0.12891387 -0.11716897  0.11224286
 0.02919915 -0.06437866  0.06916336  0.09552712] 0.023580942588906417
[ 0.01987419 -0.18842242 -0.12891387 -0.11716897  0.11224286  0.02919915
-0.06437866  0.06916336  0.09552712  0.02358094] 0.12528776131045127
[-0.18842242 -0.12891387 -0.11716897  0.11224286  0.02919915 -0.06437866
 0.06916336  0.09552712  0.02358094  0.12528776] 0.15067334584317305
[-0.12891387 -0.11716897  0.11224286  0.02919915 -0.06437866  0.06916336
 0.09552712  0.02358094  0.12528776  0.15067335] -0.026060106669865135
[-0.11716897  0.11224286  0.02919915 -0.06437866  0.06916336  0.09552712
 0.02358094  0.12528776  0.15067335 -0.02606011] -0.1763985384908322
[ 0.11224286  0.02919915 -0.06437866  0.06916336  0.09552712  0.02358094
 0.12528776  0.15067335 -0.02606011 -0.17639854] -0.0970834045818334
[ 0.02919915 -0.06437866  0.06916336  0.09552712  0.02358094  0.12528776
 0.15067335 -0.02606011 -0.17639854 -0.0970834 ] -0.1672513038729564
[-0.06437866  0.06916336  0.09552712  0.02358094  0.12528776  0.15067335
-0.02606011 -0.17639854 -0.0970834 -0.1672513 ] 0.10227884912041851
```

9. Reshape your created dataframe into the 3D format that RNN expects.

```
[ ]: # reshape input to be [samples, time steps, features]
X = np.reshape(X, (X.shape[0], X.shape[1], n_features))
```

10. Create the X_train, y_train, X_test and y_test.

```
[ ]: # split into train and test sets
train_size = int(len(X) * 0.8)
test_size = len(X) - train_size
trainX, testX = X[0:train_size, :, :], X[train_size:len(X), :, :]
trainY, testY = Y[0:train_size], Y[train_size:len(Y)]
```

```
[ ]: print(len(trainY))
```

107

11. Create a one-layer RNN with 25 units (neurons). Train it for 100 epochs with the default mini batch size of 32

12. Compile and fit the model. When compiling, specify the 'sgd' optimizer, and choose an

accuracy metric (e.g. mse, mae, mape, etc.) to judge the goodness of your predictions. To compare further on the prediction results with the true values in the test set, specify inside the fit function your validation data: `validation_data = (X_test, y_test)`. Save the history of the loss and the accuracy results in each epoch as the model is being trained, so that you can later on plot them and compare.

```
[ ]: from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

model = Sequential()
model.add(LSTM(25, activation='relu', input_shape=(look_back, n_features)))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')
history = model.fit(trainX, trainY, epochs=100, batch_size=32,
                    validation_data=(testX, testY))
```

```
Epoch 1/100
4/4 [=====] - 1s 39ms/step - loss: 0.0111 - val_loss: 0.0122
Epoch 2/100
4/4 [=====] - 0s 7ms/step - loss: 0.0109 - val_loss: 0.0121
Epoch 3/100
4/4 [=====] - 0s 6ms/step - loss: 0.0109 - val_loss: 0.0122
Epoch 4/100
4/4 [=====] - 0s 6ms/step - loss: 0.0108 - val_loss: 0.0121
Epoch 5/100
4/4 [=====] - 0s 6ms/step - loss: 0.0107 - val_loss: 0.0120
Epoch 6/100
4/4 [=====] - 0s 8ms/step - loss: 0.0107 - val_loss: 0.0118
Epoch 7/100
4/4 [=====] - 0s 7ms/step - loss: 0.0105 - val_loss: 0.0116
Epoch 8/100
4/4 [=====] - 0s 7ms/step - loss: 0.0104 - val_loss: 0.0115
Epoch 9/100
4/4 [=====] - 0s 6ms/step - loss: 0.0104 - val_loss: 0.0113
Epoch 10/100
4/4 [=====] - 0s 7ms/step - loss: 0.0103 - val_loss: 0.0112
```

Epoch 11/100
4/4 [=====] - 0s 6ms/step - loss: 0.0101 - val_loss: 0.0111
Epoch 12/100
4/4 [=====] - 0s 6ms/step - loss: 0.0100 - val_loss: 0.0109
Epoch 13/100
4/4 [=====] - 0s 7ms/step - loss: 0.0098 - val_loss: 0.0106
Epoch 14/100
4/4 [=====] - 0s 6ms/step - loss: 0.0096 - val_loss: 0.0102
Epoch 15/100
4/4 [=====] - 0s 6ms/step - loss: 0.0093 - val_loss: 0.0097
Epoch 16/100
4/4 [=====] - 0s 6ms/step - loss: 0.0090 - val_loss: 0.0092
Epoch 17/100
4/4 [=====] - 0s 6ms/step - loss: 0.0086 - val_loss: 0.0087
Epoch 18/100
4/4 [=====] - 0s 6ms/step - loss: 0.0082 - val_loss: 0.0081
Epoch 19/100
4/4 [=====] - 0s 6ms/step - loss: 0.0078 - val_loss: 0.0072
Epoch 20/100
4/4 [=====] - 0s 6ms/step - loss: 0.0072 - val_loss: 0.0067
Epoch 21/100
4/4 [=====] - 0s 6ms/step - loss: 0.0067 - val_loss: 0.0060
Epoch 22/100
4/4 [=====] - 0s 6ms/step - loss: 0.0064 - val_loss: 0.0055
Epoch 23/100
4/4 [=====] - 0s 8ms/step - loss: 0.0062 - val_loss: 0.0054
Epoch 24/100
4/4 [=====] - 0s 7ms/step - loss: 0.0060 - val_loss: 0.0053
Epoch 25/100
4/4 [=====] - 0s 7ms/step - loss: 0.0059 - val_loss: 0.0052
Epoch 26/100
4/4 [=====] - 0s 6ms/step - loss: 0.0059 - val_loss: 0.0052

Epoch 27/100
4/4 [=====] - 0s 7ms/step - loss: 0.0060 - val_loss: 0.0051
Epoch 28/100
4/4 [=====] - 0s 6ms/step - loss: 0.0057 - val_loss: 0.0048
Epoch 29/100
4/4 [=====] - 0s 8ms/step - loss: 0.0056 - val_loss: 0.0048
Epoch 30/100
4/4 [=====] - 0s 6ms/step - loss: 0.0055 - val_loss: 0.0048
Epoch 31/100
4/4 [=====] - 0s 6ms/step - loss: 0.0054 - val_loss: 0.0047
Epoch 32/100
4/4 [=====] - 0s 6ms/step - loss: 0.0055 - val_loss: 0.0046
Epoch 33/100
4/4 [=====] - 0s 6ms/step - loss: 0.0053 - val_loss: 0.0048
Epoch 34/100
4/4 [=====] - 0s 6ms/step - loss: 0.0054 - val_loss: 0.0045
Epoch 35/100
4/4 [=====] - 0s 6ms/step - loss: 0.0052 - val_loss: 0.0044
Epoch 36/100
4/4 [=====] - 0s 6ms/step - loss: 0.0052 - val_loss: 0.0048
Epoch 37/100
4/4 [=====] - 0s 6ms/step - loss: 0.0054 - val_loss: 0.0045
Epoch 38/100
4/4 [=====] - 0s 7ms/step - loss: 0.0052 - val_loss: 0.0043
Epoch 39/100
4/4 [=====] - 0s 7ms/step - loss: 0.0052 - val_loss: 0.0047
Epoch 40/100
4/4 [=====] - 0s 7ms/step - loss: 0.0052 - val_loss: 0.0046
Epoch 41/100
4/4 [=====] - 0s 7ms/step - loss: 0.0050 - val_loss: 0.0042
Epoch 42/100
4/4 [=====] - 0s 7ms/step - loss: 0.0053 - val_loss: 0.0042

Epoch 43/100
4/4 [=====] - 0s 7ms/step - loss: 0.0050 - val_loss: 0.0050
Epoch 44/100
4/4 [=====] - 0s 8ms/step - loss: 0.0052 - val_loss: 0.0042
Epoch 45/100
4/4 [=====] - 0s 8ms/step - loss: 0.0050 - val_loss: 0.0041
Epoch 46/100
4/4 [=====] - 0s 7ms/step - loss: 0.0051 - val_loss: 0.0042
Epoch 47/100
4/4 [=====] - 0s 8ms/step - loss: 0.0050 - val_loss: 0.0046
Epoch 48/100
4/4 [=====] - 0s 7ms/step - loss: 0.0050 - val_loss: 0.0042
Epoch 49/100
4/4 [=====] - 0s 6ms/step - loss: 0.0049 - val_loss: 0.0043
Epoch 50/100
4/4 [=====] - 0s 7ms/step - loss: 0.0049 - val_loss: 0.0044
Epoch 51/100
4/4 [=====] - 0s 8ms/step - loss: 0.0050 - val_loss: 0.0045
Epoch 52/100
4/4 [=====] - 0s 7ms/step - loss: 0.0049 - val_loss: 0.0041
Epoch 53/100
4/4 [=====] - 0s 7ms/step - loss: 0.0049 - val_loss: 0.0041
Epoch 54/100
4/4 [=====] - 0s 7ms/step - loss: 0.0050 - val_loss: 0.0043
Epoch 55/100
4/4 [=====] - 0s 7ms/step - loss: 0.0048 - val_loss: 0.0047
Epoch 56/100
4/4 [=====] - 0s 7ms/step - loss: 0.0050 - val_loss: 0.0041
Epoch 57/100
4/4 [=====] - 0s 8ms/step - loss: 0.0048 - val_loss: 0.0042
Epoch 58/100
4/4 [=====] - 0s 8ms/step - loss: 0.0047 - val_loss: 0.0040

Epoch 59/100
4/4 [=====] - 0s 8ms/step - loss: 0.0048 - val_loss: 0.0041
Epoch 60/100
4/4 [=====] - 0s 8ms/step - loss: 0.0047 - val_loss: 0.0043
Epoch 61/100
4/4 [=====] - 0s 8ms/step - loss: 0.0047 - val_loss: 0.0040
Epoch 62/100
4/4 [=====] - 0s 6ms/step - loss: 0.0048 - val_loss: 0.0040
Epoch 63/100
4/4 [=====] - 0s 7ms/step - loss: 0.0046 - val_loss: 0.0046
Epoch 64/100
4/4 [=====] - 0s 6ms/step - loss: 0.0048 - val_loss: 0.0040
Epoch 65/100
4/4 [=====] - 0s 8ms/step - loss: 0.0047 - val_loss: 0.0039
Epoch 66/100
4/4 [=====] - 0s 6ms/step - loss: 0.0047 - val_loss: 0.0047
Epoch 67/100
4/4 [=====] - 0s 7ms/step - loss: 0.0050 - val_loss: 0.0043
Epoch 68/100
4/4 [=====] - 0s 6ms/step - loss: 0.0051 - val_loss: 0.0038
Epoch 69/100
4/4 [=====] - 0s 7ms/step - loss: 0.0048 - val_loss: 0.0047
Epoch 70/100
4/4 [=====] - 0s 7ms/step - loss: 0.0050 - val_loss: 0.0047
Epoch 71/100
4/4 [=====] - 0s 7ms/step - loss: 0.0046 - val_loss: 0.0038
Epoch 72/100
4/4 [=====] - 0s 7ms/step - loss: 0.0048 - val_loss: 0.0039
Epoch 73/100
4/4 [=====] - 0s 7ms/step - loss: 0.0046 - val_loss: 0.0045
Epoch 74/100
4/4 [=====] - 0s 7ms/step - loss: 0.0046 - val_loss: 0.0038

Epoch 75/100
4/4 [=====] - 0s 7ms/step - loss: 0.0049 - val_loss: 0.0037
Epoch 76/100
4/4 [=====] - 0s 7ms/step - loss: 0.0046 - val_loss: 0.0046
Epoch 77/100
4/4 [=====] - 0s 7ms/step - loss: 0.0046 - val_loss: 0.0039
Epoch 78/100
4/4 [=====] - 0s 7ms/step - loss: 0.0045 - val_loss: 0.0039
Epoch 79/100
4/4 [=====] - 0s 7ms/step - loss: 0.0045 - val_loss: 0.0041
Epoch 80/100
4/4 [=====] - 0s 7ms/step - loss: 0.0045 - val_loss: 0.0039
Epoch 81/100
4/4 [=====] - 0s 8ms/step - loss: 0.0044 - val_loss: 0.0038
Epoch 82/100
4/4 [=====] - 0s 7ms/step - loss: 0.0045 - val_loss: 0.0038
Epoch 83/100
4/4 [=====] - 0s 7ms/step - loss: 0.0044 - val_loss: 0.0041
Epoch 84/100
4/4 [=====] - 0s 7ms/step - loss: 0.0044 - val_loss: 0.0037
Epoch 85/100
4/4 [=====] - 0s 8ms/step - loss: 0.0045 - val_loss: 0.0035
Epoch 86/100
4/4 [=====] - 0s 8ms/step - loss: 0.0044 - val_loss: 0.0040
Epoch 87/100
4/4 [=====] - 0s 7ms/step - loss: 0.0044 - val_loss: 0.0039
Epoch 88/100
4/4 [=====] - 0s 7ms/step - loss: 0.0043 - val_loss: 0.0035
Epoch 89/100
4/4 [=====] - 0s 6ms/step - loss: 0.0045 - val_loss: 0.0037
Epoch 90/100
4/4 [=====] - 0s 7ms/step - loss: 0.0044 - val_loss: 0.0045

```

Epoch 91/100
4/4 [=====] - 0s 8ms/step - loss: 0.0044 - val_loss: 0.0035
Epoch 92/100
4/4 [=====] - 0s 7ms/step - loss: 0.0046 - val_loss: 0.0035
Epoch 93/100
4/4 [=====] - 0s 7ms/step - loss: 0.0042 - val_loss: 0.0044
Epoch 94/100
4/4 [=====] - 0s 7ms/step - loss: 0.0045 - val_loss: 0.0041
Epoch 95/100
4/4 [=====] - 0s 7ms/step - loss: 0.0043 - val_loss: 0.0035
Epoch 96/100
4/4 [=====] - 0s 7ms/step - loss: 0.0043 - val_loss: 0.0038
Epoch 97/100
4/4 [=====] - 0s 7ms/step - loss: 0.0042 - val_loss: 0.0040
Epoch 98/100
4/4 [=====] - 0s 6ms/step - loss: 0.0043 - val_loss: 0.0036
Epoch 99/100
4/4 [=====] - 0s 6ms/step - loss: 0.0041 - val_loss: 0.0039
Epoch 100/100
4/4 [=====] - 0s 6ms/step - loss: 0.0041 - val_loss: 0.0037

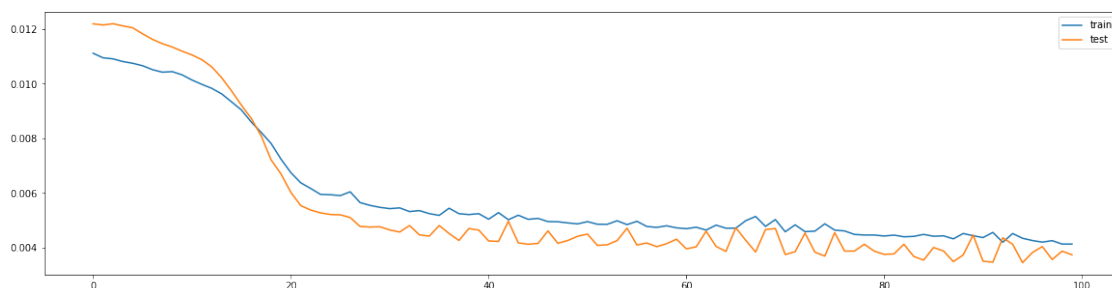
```

13. Plot the loss and accuracy in each epoch for the training set as well as the validation set. Comment on your results. Do you see signs of overfitting? If yes, explain. If no, explain.

```

[ ]: plt.figure(figsize=(20, 5))
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

```



14. Do you see signs of vanishing or exploding gradient? Explain how the two would manifest in your graph.

```
[ ]: # response
```

15. Compute the model predictions, undo the differencing and plot the results.

```
[ ]: # Compute the model predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

def rebuild_diffed(series: pd.DataFrame, first_element_original):
    cumsum = series.cumsum()
    return cumsum.fillna(0) + first_element_original

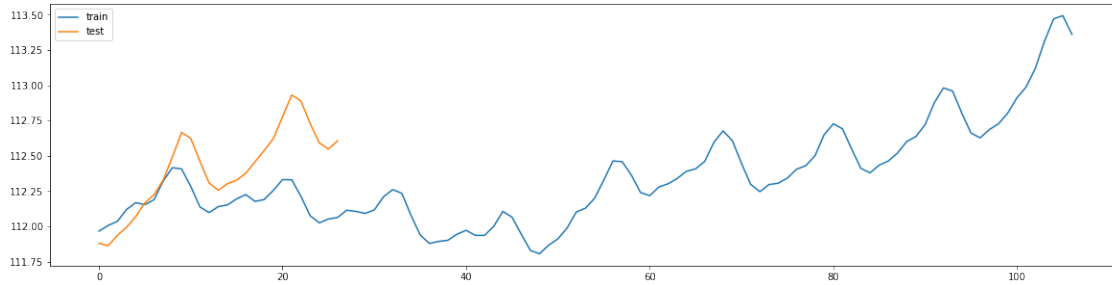
# create dataframe from the predictions
trainPredict = pd.DataFrame(trainPredict)
trainPredict.columns = ['Prediction']

# create dataframe from the predictions
testPredict = pd.DataFrame(testPredict)
testPredict.columns = ['Prediction']

print(first_passengers_value)
# undo the differencing
trainPredict = rebuild_diffed(trainPredict, first_passengers_value)
testPredict = rebuild_diffed(testPredict, first_passengers_value)

# # concatenate trainPred and testPredict
# predictions = pd.concat([trainPredict, testPredict], axis=0)
# print(predictions)
# # set the index same as the original data, but without the first 10 elements
# predictions.index = df.index[10:]

# plot the results
plt.figure(figsize=(20, 5))
plt.plot(trainPredict, label='train')
plt.plot(testPredict, label='test')
plt.legend()
plt.show()
```



16. Now try other options and see if you can beat your initial predictions:

- Train the model for 500 epochs. Are your predictions better? Should you have stopped before the end of the 500 epochs judging by your loss and accuracy plots? If yes at what epoch would you stop and why?
- Try two, three, and four layers for your RNN with 50 and 100 neurons in each of the hidden layers (you should get altogether 6 models – 2 layers & 50 neurons, 2 layers & 100 neurons, 3 layers & 50 neurons, etc.). Train each of them for 100 epochs and save and plot the accuracies and the losses for each of the 6 combinations. Do you see signs of overfitting? Which model overfits most? Explain.
- For the most severely overfitting model from the 6 options above use the Dropout in combination with the L2 method, to regularize your overfitting model. Be creative, try different options to correct for the overfit.
- Could you create a model that beats the first one (the one with 1 layer and 25 units) and is not overfitting after regularization was applied?

[]: *# response*