Chapter 1 : Introduction to Time Series

Ravi Mummigatti

Features of Time Series Data :

Time Series is a “Sequence” of information which attaches a “Time Period” to each value.

The “Interval” of time between recording one point of the set and the next is called a “Time Period.”

“How often” values of the data set are recorded is referred to as the “Frequency” of the data set which helps us analyse the time series in a meaningful way.

All time periods must be equal and clearly defined, which would result in a constant frequency.

The frequency is a measurement of time and could range from a few milliseconds to several decades.

Patterns observed in Time series are expected to persist in the future.

In Time Series Analysis , we try to predict the future by analyzing past recorded values.

Examples of Time Series Analysis :

Predicting Weather by analyzing historical weather patterns.

Forecasting Sales , Profits , Stock Prices.

A common topic in Time series analysis is determining the stability of financial markets and the efficiency portfolios.

Time series data can suffer from seasonality.Some values like rain or temperature vary depending on the time of day in the season of the year.Since it’s a repeating cycle, we can anticipate these changes and account for them when making our predictions.Seasonality as a trait is not observed in regular data, especially when there is no chronological order.

Notations of Time Series Data

X : Values of the data point example “Daily Stock Prices in the year 2008” over an entire period
T : Total Time Period example “One Year”
t : Value at a single time period

Xt : Value of the data at a single point in time example “Stock Price on Sunday 30th Jan 2008”
$Xt-1$: Value of the data at one period before time period t example “Stock Price on Saturday 29th Jan 2008”
$Xt+1$: Value of the data at one period after time period t example “Stock Price on Monday 31st Jan 2 008”

Peculiarities of Time Series data :

The intervals between observations need to be identical.

Frequency of the Time Series dataset can be adjusted depending on the values we are interested in example Daily Stock Price , Weekly Average Stock Price , Monthly Average Stock Price and so on. By aggregating the data over the frequency we desire example “Mean of Daily Closing Stock Price Over 30 Days we arrive at Monthly Average Closing Stock Price”.

Increasing the Frequency Leads to increasing the “number of time periods” within the intervals... Monthly>Weekly>Daily

Most important of all , unlike regular tabular data, ***Time Series data requires chronological order*** from a machine learning perspective.This is inconvenient because we cannot freely shuffle the data before separating it into a training and a testing data set.What we do instead is pick a cut off point.The period before the cutoff point is the training set, the period after the cutoff point is the testing set. example Training Set = 01-Jan-2008 to30-Sep-2008 while Testing Set = 01-Oct-2008 to 31-Dec-2008

Time Series data does not follow any of the standard distributions we are familiar with from statistics and probability.example Gaussian Normal Distribution.Time series data assumes that past patterns in the variable will continue unchanged into the future.Using this approach, we can forecast our predictions about future values.

Packages for Time Series Modeling

```
# importing libraries for Time Series Analysis and Modelling
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.graphics.tsaplots as sgt
import statsmodels.tsa.stattools as sts
from statsmodels.tsa.seasonal import seasonal_decompose
import seaborn as sns
sns.set()

# disable warnings
import warnings
warnings.filterwarnings('ignore')
```

Load & Examine Data

We will examine the Index Prices of selected stock exchanges.Data is downloaded from Yahoo Finance for the period 07-Jan-1994 to 29-Jan-2018

for the S&P500 , DAX30 , FTSE and Nikkei.

Load Data

```
# load the data
raw_data = pd.read_csv("index_2018.csv")
```

Examine Data

Numerical Summaries

1. **Let us examine the top 5 rows applying the head() method on the dataframe**

```
# examine top 5 rows
df = raw_data.copy()
df.head()
```

```
##          date     spx      dax     ftse    nikkei
## 0    7/1/1994  469.90  2224.95  3445.98  18124.01
## 1   10/1/1994  475.27  2225.00  3440.58  18443.44
## 2   11/1/1994  474.13  2228.10  3413.77  18485.25
## 3   12/1/1994  474.17  2182.06  3372.02  18793.88
## 4  13/01/1994  472.47  2142.37  3360.01  18577.26
```

- date : date on which the index values were recorded
- spx : Dow Jone S&P Index Value {S&P : 500}
- dax : Frankfurt DAX Index Value {DAX 30}
- ftse : London Stock Exchange Index Value {Footsie 100}
- nikkei : Tokyo Stock Exchange Index Value {Nikkei 225}

Each market index is a portfolio of the most traded public companies on the respective stock exchange markets.

The S&P, DAX, Footsie and Nikkei measure the stability of the US stock exchange,the German stock exchange, the London Stock Exchange and the Japanese stock exchanges respectively.

We analyze Time Series in consecutive chunks of data. It is therefore necessary to define the index column of our dataframe which in our case will be the “date column”. Date column is always the Index for Time Series.

2. **To understand the data frame in more detail, we can use the describe() method to summarize statistics.**

```
# summary statistics
df.describe()
```

```
##                spx           dax         ftse        nikkei
## count  6269.000000   6269.000000  6269.000000   6269.000000
## mean   1288.127542   6080.063363  5422.713545  14597.055700
## std     487.586473   2754.361032  1145.572428   4043.122953
## min     438.920000   1911.700000  2876.600000   7054.980000
## 25%     990.671905   4069.350000  4486.100000  10709.290000
## 50%    1233.420000   5773.340000  5662.430000  15028.170000
## 75%    1459.987747   7443.070000  6304.250000  17860.470000
## max    2872.867839  13559.600000  7778.637689  24124.150000
```

Notice that we only get summarized information about the four market indices.This is because date is not numeric and the default setting of the describe() method omits such variables.

count : This shows the total number of observations in our data which in our case is 6269 observations.These are observations only for the dates when the indices were recorded...

mean : This shows the average value of the indices over the complete time period...

Mean Price for S&P is far lower than the Minimum Price of FTSE and NIKKEI.We need to take into account this difference in magnitude when we are analyzing this dataset further...Mean price for DAX and FTSE look comparable.

We can infer that the values for the DAX and FTSE are similar, while the S&P and the Nikkei are a lot smaller and larger, respectively.

3. **Let us look at missing values in our dataset using the isna().sum() method applied to the dataframe**

```
# examine missing values
df.isna().sum()
```

```
## date      0
## spx       0
## dax       0
## ftse      0
## nikkei    0
## dtype: int64
```

We can infer that our dataset does not have any missing values

Visual Summaries

Looking at the statistical summaries gives us a general idea of the spread of the data.This however does not give us insights into the evolution / trend of these indices over time. We turn to seaborn to visualize our data to examine possible trends

1. **Let us look at the trend of S&P over time**

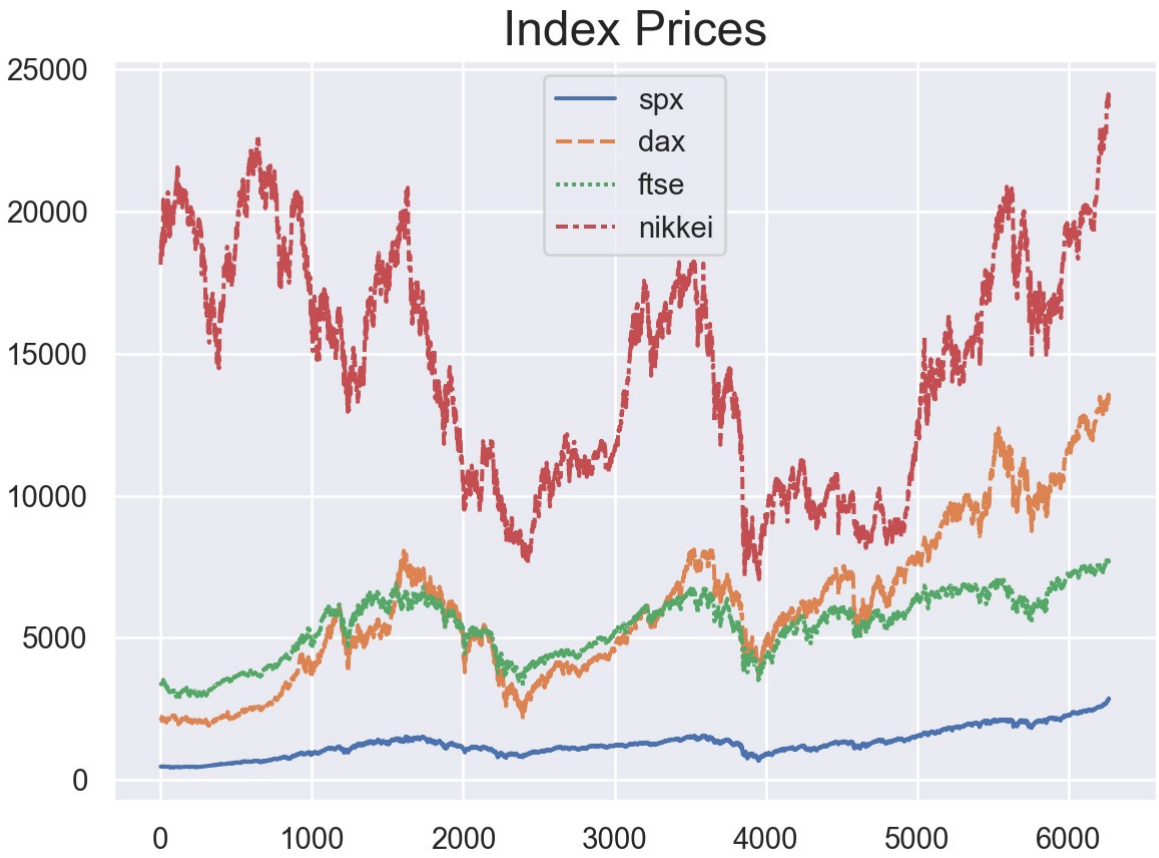
```
## create the line chart by extracting only the spx values
sns.lineplot(data = df.spx);
plt.title("S&P500 Prices" , size = 18);
plt.show()
```

Notice that the x axis consists of numbers.These numbers represent the periods the values come from if we are dealing with proper time series data.The X axis will show the corresponding dates for each period.For instance, period zero will express the first date of the data set, which is January 7th, 1994.This occurs because the indexes for each period are simply numeric values rather than dates.

We observe how the price of the S&P fluctuates, there are periods of booming growth followed by sharp falls. The first two large peaks followed by periods of turbulence, represent, of course, the dot-com and the housing market bubble, respectively.

2. Let us look at how all the indices trend against each other

```
# plotting all the 4 indices together
sns.lineplot(data = df);
plt.title("Index Prices" , size = 18);
plt.show()
```



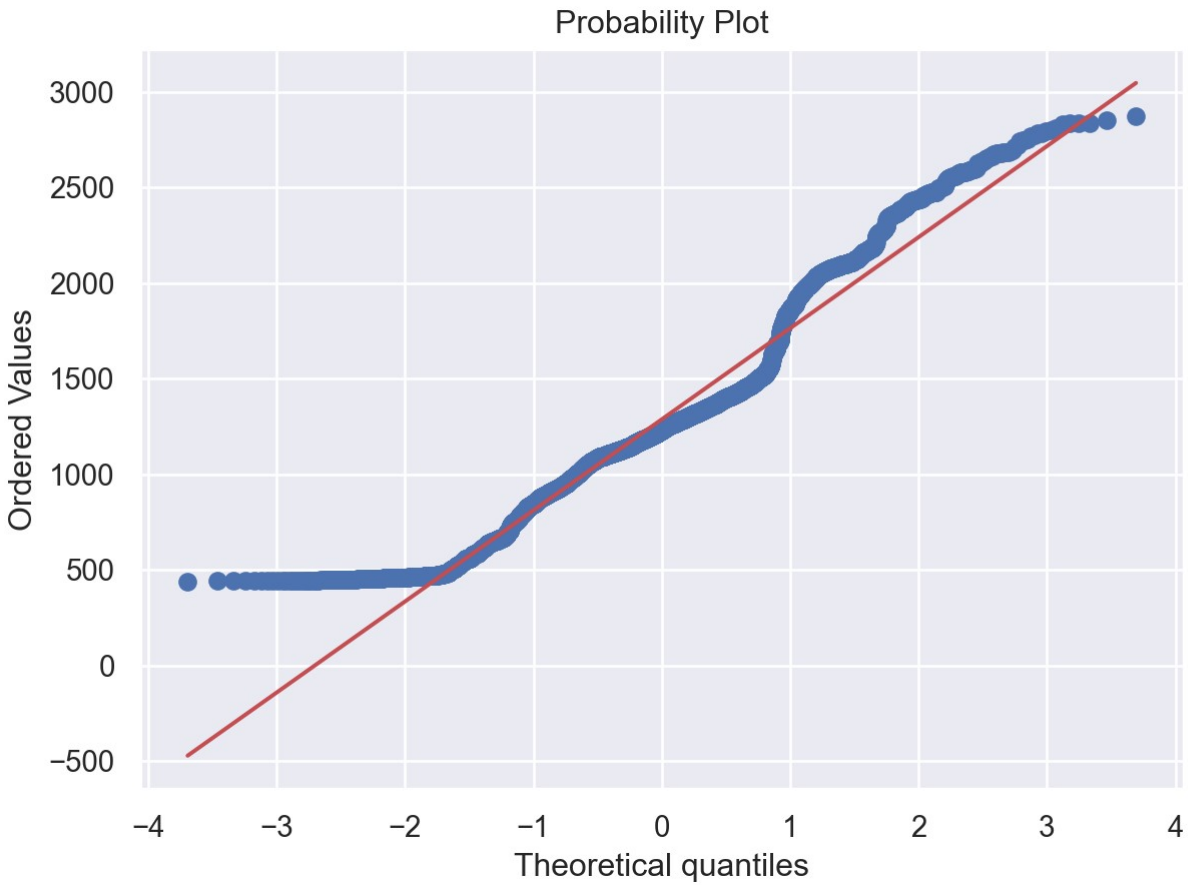
We observe that Footsie and DAX follow a similar pattern... European Markets...Nikkei shows mixed trends..

Recall that FTSE Minimum Value is far higher than the Average S&P Value. Also note that Nikkei value are likely in JPY and not normalized to USD.This is one reason we should look at normalizing when we are comparing different units .

3. Let us look at distribution using Q-Q-Plot which compares data distribution against a Normal Distribution

Let us examine the density of the data. The Quanti Quintile Plot or Q-Q-Plot is a tool used in analytics to determine whether a data set is distributed a certain way.Unless specified otherwise, the Q-Q-Plot showcases how the data fits a normal distribution. We will use the stats package from scipy library to draw the Q-Q-Plot of our data

```
import scipy.stats as stats
a = df['spx']
stats.probplot(a, dist="norm", plot=plt);
plt.show()
```



Q-Q-Plot plot takes all the values a variable can take and arranges them in order.

The y axis expresses the price with the highest ones at the top and the lowest at the bottom.

The X axis represents the theoretical quantiles of the data set.(How many standard deviations away from the mean these values are.) .
The red diagonal line on the screen represents what the data points should follow if they are normally distributed in this case, we see that it is not really the case since we have more values around the500 mark than we shoul d . Therefore, the data is not normally distributed and we cannot use the elegant statistics of normal distributions to make successful forecasts.This is what we usually expect from TIME Series dat a .

Creating and Transforming Time Series

Time is the single most important piece of information when dealing with Time series, so it's essential to transform the data frame in the appropriate type.Let us look at the type of columns we have in our Stock Exchange Indices data frame

Transforming to date type

We will apply the info() method to view the data types of every column in our dataframe

```
# view data types
df.info()

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 6269 entries, 0 to 6268
## Data columns (total 5 columns):
##  #   Column  Non-Null Count  Dtype
## ---  ------  --------------  -----
##  0   date    6269 non-null   object
##  1   spx     6269 non-null   float64
##  2   dax     6269 non-null   float64
##  3   ftse    6269 non-null   float64
##  4   nikkei  6269 non-null   float64
## dtypes: float64(4), object(1)
## memory usage: 220.5+ KB

# summary statistics of data column
df.date.describe()

## count           6269
## unique          6269
## top       19/02/2014
## freq               1
## Name: date, dtype: object

# looking at the first 5 rows with date only
df.date.head()

## 0      7/1/1994
## 1     10/1/1994
## 2     11/1/1994
```

```
## 3     12/1/1994
## 4    13/01/1994
## Name: date, dtype: object
```

Date Column is stored as an “object” a.k.a. string type and not as a date type also the top date is misleading as it doesn’t hold the highest value or the date with the highest frequency

We are analyzing specific chunks of the data across time periods hence we need to transform the “date” column to “datetime type”.

The Pandas package provides a **to_datetime()** method we can call.

to_datetime() requires a single argument, the attribute/string column we wish to transform into a daytime type , the syntax for which is *pd.to_datetime(str_col_to_convert , dayfirst=?)*

By default, to_datetime() will parse string with month first format, and this arrangement is relatively unique in the US

.Pandas Assumes that the “string date” is of the format “mm/dd/yyyy”

In most of the rest of the world, the day is written first “dd/mm/yyyy”. If we want to consider day first instead of month, we need to set the argument dayfirst to True

```
# convert the date column to datetime type and overwrite the dataframe
df.date = pd.to_datetime(df.date , dayfirst = True)

# look at the first 5 rows
df.date.head()
```

```
## 0   1994-01-07
## 1   1994-01-10
## 2   1994-01-11
## 3   1994-01-12
## 4   1994-01-13
## Name: date, dtype: datetime64[ns]
```

```
# summary statistics of date column
df.date.describe()
```

```
## count                    6269
## unique                   6269
## top       1999-08-23 00:00:00
## freq                        1
## first     1994-01-07 00:00:00
## last      2018-01-29 00:00:00
## Name: date, dtype: object
```

The date column is now converted to datetime type with 1st date as “07-Jan-1994” , last date as “29-Jan-2018”

Setting Date as index

To refer to a set of values as a Time Series, each one should correspond to a time period.This is crucial for referring to values according to the day they were recorded.

We often wish to examine specific chunks of data between two concrete dates , hence we need to use the associated time periods as indices.In our case we have a data set with daily values of various Stock exchanges Index Values , therefore, each date would be a separate time period.We can therefore use the date column for the data frame as the new index column.

Pandas provides a method called data.set_index("column name , inplace=True) where data is the dataframe ,"column_name" is the name of the column which we want the dataframe to be indexed on and the inplace = True ensures that the data frame is overwritten.***Note : Column name should be in quotes" "***

```
# set date column as index
df.set_index("date" , inplace = True)

# look at the top 5 rows
df.head()
```

```
##                spx      dax     ftse    nikkei
## date
## 1994-01-07  469.90  2224.95  3445.98  18124.01
## 1994-01-10  475.27  2225.00  3440.58  18443.44
## 1994-01-11  474.13  2228.10  3413.77  18485.25
## 1994-01-12  474.17  2182.06  3372.02  18793.88
## 1994-01-13  472.47  2142.37  3360.01  18577.26
```

We observe that there is no longer a column with integer values on the left.Instead, we have the date column in its place.We can tell these are the new index values because they appear in bold once the data frame is displayed.Let us look at the data types again using the .info() method

```
## <class 'pandas.core.frame.DataFrame'>
## DatetimeIndex: 6269 entries, 1994-01-07 to 2018-01-29
## Data columns (total 4 columns):
##  #   Column  Non-Null Count  Dtype
## ---  ------  --------------  -----
##  0   spx     6269 non-null   float64
##  1   dax     6269 non-null   float64
##  2   ftse    6269 non-null   float64
##  3   nikkei  6269 non-null   float64
## dtypes: float64(4)
## memory usage: 244.9 KB
```

We notice that the Data Frame is now indexed on dates "DatetimeIndex: 6269 entries, 1994-01-07 to 2018-01-29" and not "RangeIndex: 6269 entries, 0 to 6268" as before. Also the number of columns is now reduced from 5 earlier to 4

Setting the frequency

The pandas dataframe.asfreq() function is used to convert Time Series to specified frequency. This function Optionally provides filling method to pad/backfill missing values. It Returns the original data conformed to a new index with the specified frequency.

Syntax : DataFrame.asfreq(freq = "X", method=None, fill_value=None)
Parameters : freq : DateOffset object {'M' as end of the month , 'W' for week , 'D' for day , 'B' for business days and 'H' for hours , 'A' for yearly / annual} method : Method to use for filling holes in re-indexed Series fill_value : Value to use for missing values, applied during up-sampling (note this does not fill NaNs that already were present) .

Let us now turn our data into a true Time Series Data by specifying the frequency of observations...

Our data is that of "Daily Closing Index Values" of the 4 Stock Exchanges which implies our frequency should be daily i.e. 'D'

```
# convert data to a daily frequency
df = df.asfreq('D')
# review top rows
df.head()
```

```
##                spx      dax     ftse    nikkei
## date
## 1994-01-07  469.90  2224.95  3445.98  18124.01
## 1994-01-08     NaN      NaN      NaN       NaN
## 1994-01-09     NaN      NaN      NaN       NaN
## 1994-01-10  475.27  2225.00  3440.58  18443.44
## 1994-01-11  474.13  2228.10  3413.77  18485.25
```

We see two new periods for January 8th and 9th which have missing values.That's because these dates were not included in the original set.This means we generated new periods which do not have values associated with them.Examining these particular dates i.e. 08-Jan-1994 and 09-Jan-1994 we notice that these are Saturday and Sunday which are "Non-Business-Days"... This is also evident from the fact that our data represents the closing prices of financial indices which can only be recorded during working days.

Let us convert our Time Series Data to a frequency as "Business Days" by including "B" as the argument in the as.freq() method.This will tell Python to expect missing values when encountering Saturdays and Sundays and ignore them.

```
# convert data to a daily frequency by business days
df = df.asfreq('B')

# top rows
df.head()
```

```
##                spx      dax     ftse    nikkei
## date
## 1994-01-07  469.90  2224.95  3445.98  18124.01
## 1994-01-10  475.27  2225.00  3440.58  18443.44
```

```
## 1994-01-11  474.13  2228.10  3413.77  18485.25
## 1994-01-12  474.17  2182.06  3372.02  18793.88
## 1994-01-13  472.47  2142.37  3360.01  18577.26
```

Handling Missing Values

As the 1st step after assigning the frequency to the Time Series , we need to ascertain if this has resulted in additional time periods for which data is not available i.e. look for missing values.

```
# looking for missing values
df.isna().sum()
```

```
## spx       8
## dax       8
## ftse      8
## nikkei    8
## dtype: int64
```

We see there exactly eight missing values for each of the market indexes.Recall that we had no missing values when we first examined the data set.Therefore, setting the frequency to business days must have generated eight dates for which we have no data available.

The fillna() method fills missing values according to sequence. It means that the method replaces 'nan's value with last observed non-nan value or next observed non-nan value.

- backfill - bfill : according to last observed value {if we have no data for July 7th 2004, we pass it the same value as the one we have recorded for July 8th 2004 the same year}
- forwardfill - ffill : according to next observed value {if we have no data available for the 15th of July 2004, we assign to it the value recorded on the 14th of July 2004 the same year}

FRONT FILL COPIES THE LAST KNOWN VALUE WHILE BACKFILL COPIES THE NEXT KNOWN VALUE AS PER FREQUENCY

Let us try different methodologies : ffill for S&P and DAX , bfill for FTSE and NIKKEI

```
# fill missing values in S&P  and DAX with Forward Fill method
df.spx = df.spx.fillna(method = "ffill")
df.dax = df.dax.fillna(method = "ffill")
df.ftse = df.ftse.fillna(method = "bfill")
df.nikkei = df.nikkei.fillna(method = "bfill")

# review missng values
df.isna().sum()
```

```
## spx       0
## dax       0
```

```
## ftse      0
## nikkei    0
## dtype: int64
```

Data Preparation

Selecting a single Time Series

We will be analyzing how the S&P500 performs so we can remove the columns for the FTSE, DAX and Nikkei.This will serve two purposes :

1. Lesser the data , faster we can manipulate the dataframe.In our case since we have ~6200 observations it won't be an issue.However it can significantly affect high frequency data with hundreds of thousands of observations

2. Removing unwanted columns gives us a lot of clarity and makes it easier to keep track of any new series we might add to the dataset

We will then create a new column "mkt_value_sp" and assign this column with the same values as the S&P.

We will then create a new dataset that contains only the S&P Market values

```
# creating a new colum "mkt_value" which is S&P values
df['mkt_value_sp'] = df.spx
df.head()
```

```
##                spx      dax     ftse    nikkei  mkt_value_sp
## date
## 1994-01-07  469.90  2224.95  3445.98  18124.01        469.90
## 1994-01-10  475.27  2225.00  3440.58  18443.44        475.27
## 1994-01-11  474.13  2228.10  3413.77  18485.25        474.13
## 1994-01-12  474.17  2182.06  3372.02  18793.88        474.17
## 1994-01-13  472.47  2142.37  3360.01  18577.26        472.47
```

```
# creating a new dataset with S&P values only by copying
df_sp_ts = df.copy()
df_sp_ts.describe()
```

```
##                spx           dax         ftse        nikkei  mkt_value_sp
## count  6277.000000   6277.000000  6277.000000   6277.000000   6277.000000
## mean   1288.642547   6083.381061  5423.690398  14597.597179   1288.642547
## std     487.868210   2755.563853  1145.568370   4043.683038    487.868210
## min     438.920000   1911.700000  2876.600000   7054.980000    438.920000
## 25%     992.715221   4070.460000  4487.880000  10701.130000    992.715221
## 50%    1233.761241   5774.260000  5663.300000  15030.510000   1233.761241
## 75%    1460.250000   7445.560000  6304.630175  17860.470000   1460.250000
## max    2872.867839  13559.600000  7778.637689  24124.150000   2872.867839
```

```
# deleting unwanted columns
```

```
# delete spx
del df_sp_ts['spx']
# delete dax
del df_sp_ts['dax']
# delete ftse
del df_sp_ts['ftse']
# delete nikkei
del df_sp_ts['nikkei']

# examine the final data frame
df_sp_ts.info()

## <class 'pandas.core.frame.DataFrame'>
## DatetimeIndex: 6277 entries, 1994-01-07 to 2018-01-29
## Freq: B
## Data columns (total 1 columns):
##  #   Column        Non-Null Count  Dtype
## ---  ------        --------------  -----
##  0   mkt_value_sp  6277 non-null   float64
## dtypes: float64(1)
## memory usage: 98.1 KB

# descriptive statistics
df_sp_ts.describe()

##        mkt_value_sp
## count   6277.000000
## mean    1288.642547
## std      487.868210
## min      438.920000
## 25%      992.715221
## 50%     1233.761241
## 75%     1460.250000
## max     2872.867839

## create the line chart by extracting only the spx values
sns.lineplot(data = df_sp_ts);
## add the title

plt.title("S&P500 Prices");
## show the plot
```

Splitting into train and test

We need to split our available data into two sets, a training set and a testing set to conduct successful machine learning.The goal is to have the option of feeding new information into the model and comparing its predictions to actual values.The closer the forecasts match, the actual values, the better our model performs.

As a standard machine learning practice , we would shuffle the data before splitting it to make both sets equally representative. However, Time series data relies on keeping the chronological order of the values within the set , which makes shuffling impossible.

Since we can’t shuffle our data, the testing and the training sets would also have to be uninterrupted sequences of values.

The training set should include all values from the beginning of the data up to a specific point in time while the testing set should include the rest.

If the training set is too large , the model will fit the training set too well and will perform poorly with the new data i.e. test set.Generally a 80-20 split is considered as a reasonable split.

To successfully implement the split , we must know where the first set finishes and the second one begins.In other words, we must determine the cutoff point between the two.

1. Total length of the dataset : Let us find the length using the len() function then extract 80% of the total length and store it in a “size” variable
2. Apply the iloc() method to subset the dataframe and extract 0 to 80% and 81 to 100% rows
3. Store 0-80 in df_train and 81-100% in df_test dataframes.

   Note : Size contains the count of 80% rows. Use this as the starting and ending indices

```
# defining the size variable with 80% of length
size = int(len(df_sp_ts)*0.8)
size

## 5021

# subsetting and creating the training and testing data sets
# create the training set by extracting indices from 0 to 80% i.e. upto size
df_train = df_sp_ts.iloc[:size]

# create the testing set by extracting indices from 81 to 100% i.e. from size
df_test  = df_sp_ts.iloc[size:]

# examine the training set last few rows
print("Bottom 6 rows of training set")

## Bottom 6 rows of training set

df_train.tail()
# examine the test set first few rows

##             mkt_value_sp
## date
## 2013-04-01   1562.173837
## 2013-04-02   1570.252238
## 2013-04-03   1553.686978
## 2013-04-04   1559.979316
## 2013-04-05   1553.278930

print("Top 6 rows of test set")

## Top 6 rows of test set

##             mkt_value_sp
```

```
## date
## 2013-04-08   1563.071269
## 2013-04-09   1568.607909
## 2013-04-10   1587.731827
## 2013-04-11   1593.369863
## 2013-04-12   1588.854623
```

Notice that the training set ends on 5th of April 2013, while the test set starts on the 8th, there is no overlap.The sixth and seventh were not business days, so they are not present.

We are now ready to perform the next phase of Time Series Analysis / Modeling which we will discuss in the next chapter

To Summarize :

- Loaded the dataset and performed Statistical and Visual Exploratory Data Analysis
- Converted the Date column from a string type to a datetime type object
- “Transformed” the data set into a Time Series Data Set with “Date as the Index” with frequency as “Business Days”
- Replaced missing values by using the appropriate padding(fill) method
- Split the data into 80% training and 20% testing sets