**Introduction to Time Series Analysis in Python**

*Data that is updated in real-time requires additional handling and special care to prepare it for machine learning models. The important Python library, Pandas, can be used for most of this work, and this tutorial guides you through this process for analyzing time-series data.*

*A****time series****is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus it is a sequence of discrete-time data. Examples of time series are heights of ocean tides, counts of sunspots, and the daily closing value of the Dow Jones Industrial Average.*

So any dataset in which is taken at successive equally spaced points in time. For example, we can see [this](https://fred.stlouisfed.org/series/UMTMVS) data set that is **Value of Manufacturers’ Shipments for All Manufacturing Industries.**

We will see some important points that can help us in analyzing any time-series dataset. These are:

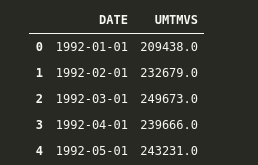
* **Loading time series dataset correctly in Pandas**
* **Indexing in Time-Series Data**
* **Time-Resampling using Pandas**
* **Rolling Time Series**
* **Plotting Time-series Data using Pandas**

**Loading time series dataset correctly in Pandas**

Let’s load the dataset mentioned above in pandas.

df = pd.read\_csv('Data/UMTMVS.csv')

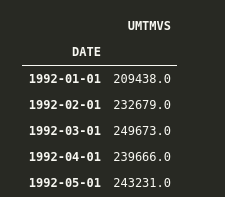
df.head()



Since we want our “DATE” column as our index, but simply by reading, it is not doing it, so we have to add some extra parameters.

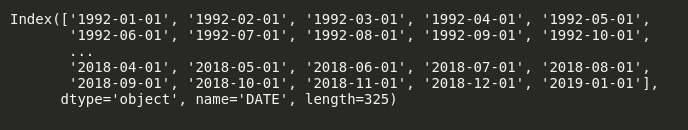
df = pd.read\_csv(‘Data/UMTMVS.csv’, index\_col=’DATE’)

df.head()



Great, now we have added our DATE column as the index, but let’s check it’s data type to know that if pandas is dealing with the index as simple objects or pandas built-in DateTime datatype.

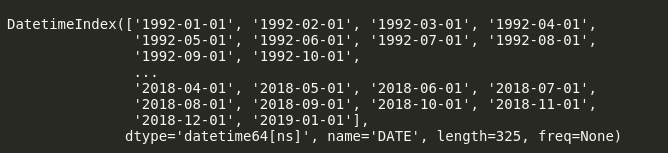
df.index



Here we can see that Pandas is dealing with our Index column as a simple object, so let’s convert it into DateTime. We can do it as follows:

df.index = pd.to\_datetime(df.index)

df.index

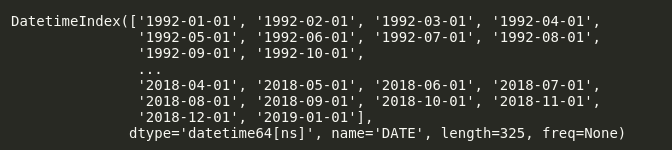


Now we can see that *dtype* of our dataset is *datetime64[ns]*. This “[ns]” shows that it is precise in nanoseconds. We can change it to “Days” or “Months” if we want.

Alternatively, to avoid all this fuss, we can load data in single line of code using Pandas as follows.

df = pd.read\_csv(‘Data/UMTMVS.csv’, index\_col=’DATE’, parse\_dates=True)

df.index

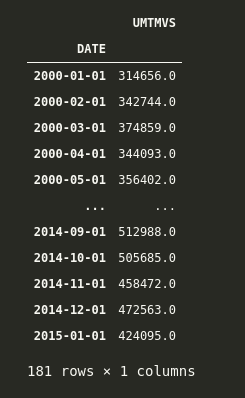


Here we have added *parse\_dates=True*, so it will automatically use our *index*as dates.

**Indexing in Time-Series Data**

Let’s say I want to get all the data from *2000-01-01* till *2015-05-01*. In order to do this, we can simply use indexing in Pandas like this.

df.loc['2000-01-01':'2015-01-01']



Here we have data for all the months from *2000-01-01* till *2015-01-01*.

Let’s say we want to get all the data of all the first months from *1992-01-01* to *2000-01-01*. We can simply do it by adding another argument that is similar to when we slice the list in python, and we add a step argument in the end.

The syntax for this in Pandas is *['starting date':'ending date':step].* Now, if we observe our dataset, it is in months format, so we want data every 12 months, from 1992 till 2000. We can do it as follows.

df.loc['1992-01-01':'2000-01-01':12]



And here, we can see that we can get the values of the first month of every year.

**Time-Resampling using Pandas**

Think of resampling as *groupby()* where we group by based on any column and then apply an aggregate function to check our results. Whereas in the Time-Series index, we can resample based on any *rule*in which we specify whether we want to resample based on “Years” or “Months” or “Days or anything else.

Some important rules for which we resample our time series index are:

* M = Month End
* A = Year-End
* MS = Month Start
* AS = Year Start

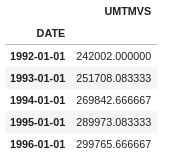
and so on. You can check the detailed aliases in the [official documentation](https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases).

Let’s apply this to our dataset.

Let’s say we want to calculate the mean value of shipment at the start of every year. We can do this by calling resample at *rule='AS'* for Year Start and then calling the aggregate function *mean*on it.

We can see the *head*of it as follows.

df.resample(rule='AS').mean().head()



Here we have resampled the index based on starting of every year(remember what “AS” does), then applied the *mean*function on it, and now we have the mean of Shipping at the start of every year.

We can even use our own custom functions with *resample*. Let’s say we want to calculate the sum of every year with a custom function. We can do that as follows.

def sum\_of\_year(year\_val):

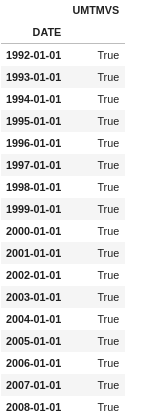
return year\_val.sum()

And then we can apply it via resampling as follows.

df.resample(rule='AS').apply(year\_val)

We can confirm that it is working correctly by comparing it to

df.resample(rule='AS').apply(my\_own\_custom) == df.resample(rule='AS').sum()



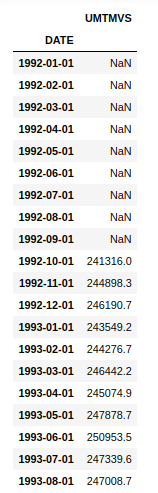
And they both are equal.

**Rolling Time Series**

Rolling is also similar to Time Resampling, but in Rolling, we take a window of any size and perform any function on it. In simple words, we can say that a rolling window of size *k* means *k* consecutive values.

Let’s see an example. If we want to calculate the rolling average of 10 days, we can do it as follows.

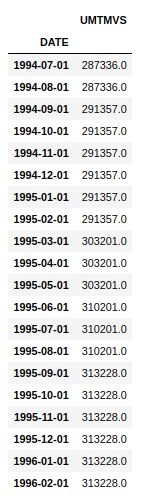
df.rolling(window=10).mean().head(20) # head to see first 20 values



Now here, we can see that the first 10 values are *NaN*because there are not enough values to calculate the rolling mean for the first 10 values. It starts calculating the mean from the 11th value and goes on.

Similarly, we can check out the maximum value from a window of 30 days as follows.

df.rolling(window=30).max()[30:].head(20) # head is just to check top 20 values



Note that here I have added*[30:]*just because the first 30 entries, i.e., the first window, do not have values to calculate the *max*function, so they are *NaN*, and for adding a screenshot, to show the first 20 values, I just skipped the first 30 rows, but you do not need to do it in practice.

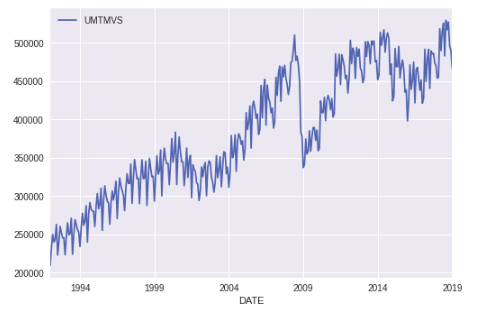
And here, we can see that we have maximum values over a rolling window of 30 days.

**Plotting Time-series Data using Pandas**

 Interestingly, Pandas offer a good set of built-in visualization tools and tricks which can help you in visualizing any kind of data.

A basic line plot can be obtained just by calling *.plot* function over the dataframe.

df.plot()

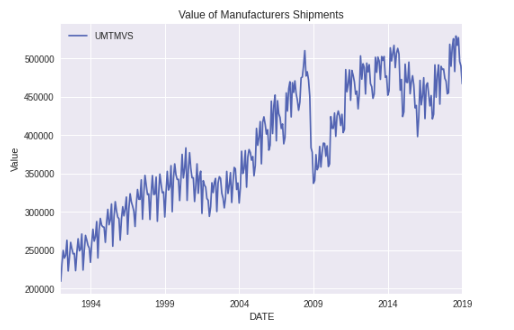


And here, we can see the value of Manufactures Shipment over time. Notice that how nicely Pandas has handled our x-axis, which is our Time Series Index.

We can further modify it by adding a title, and y-label by using *.set* on our plot.

ax = df.plot()

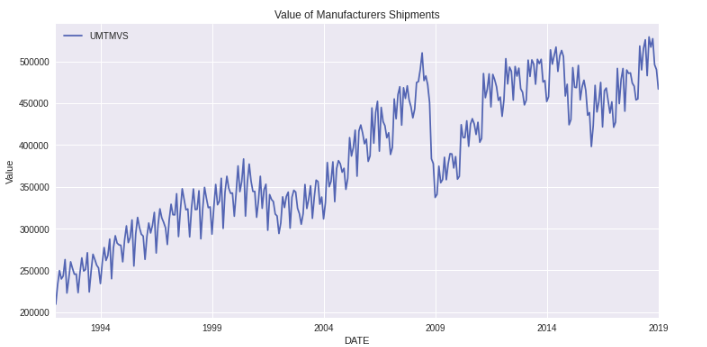
ax.set(title='Value of Manufacturers Shipments', ylabel='Value')



Similarly, we can change the plot size via *figsize*parameter in *.plot*.

ax = df.plot(figsize=(12,6))

ax.set(title='Value of Manufacturers Shipments', ylabel='Value')



Let’s now Plot the mean of the starting value of every year. We can do it via calling *.plot* after resampling with the rule ‘AS’ as ‘AS’ is the rule for the starting of the year.

ax = df.resample(rule='AS').mean().plot(figsize=(12,6))

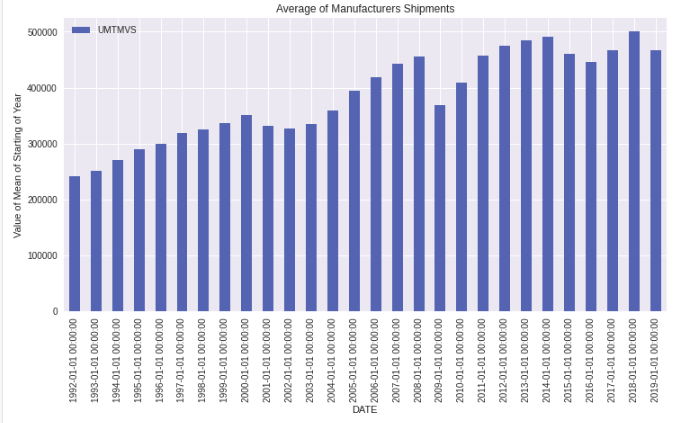
ax.set(title='Average of Manufacturers Shipments', ylabel='Value of Mean of Starting of Year')



We can also do the bar plot for the mean of starting of every year by calling *.bar* on top of *.plot*.

ax = df.resample(rule='AS').mean().plot.bar(figsize=(12,6))

ax.set(title='Average of Manufacturers Shipments', ylabel='Value of Mean of Starting of Year');



Similarly, we can plot the rolling mean and normal mean for the starting of the month as follows.

ax = df['UMTMVS'].resample(rule='MS').mean().plot(figsize=(15,8), label='Resample MS')

ax.autoscale(tight=True)

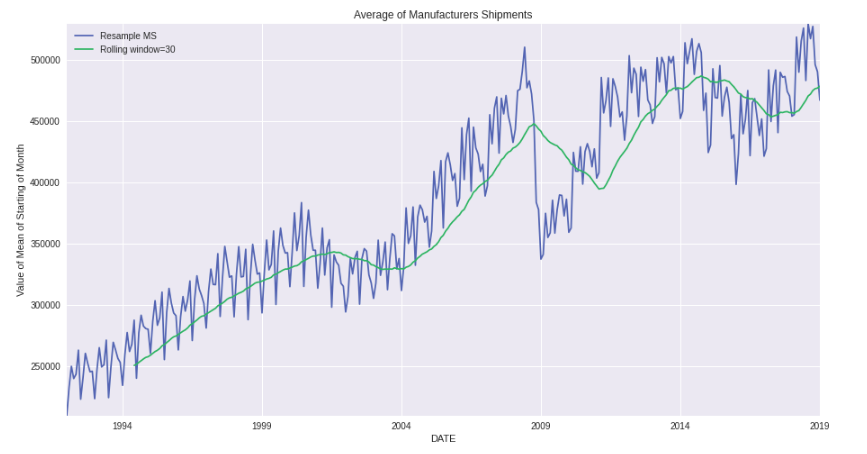
df.rolling(window=30).mean()['UMTMVS'].plot(label='Rolling window=30')

ax.set(ylabel='Value of Mean of Starting of Month',title='Average of Manufacturers Shipments')

ax.legend()

Here, first, we have plotted the mean of the starting of every month via resampling on rule = “MS” (Month start). Then we have set *autoscale(tight=True)*. This will remove the extra plot portion, which is empty. Then we have plotted the rolling mean on 30 days window. Remember that the first 30 Days are null, and you will observe this in the plot. Then we have set Label, Title, and Legend.

The output of this plot is



Notice how the first 30 days are missing in Rolling Average, and since it is rolling average, it is pretty smooth, as compared to resample one.

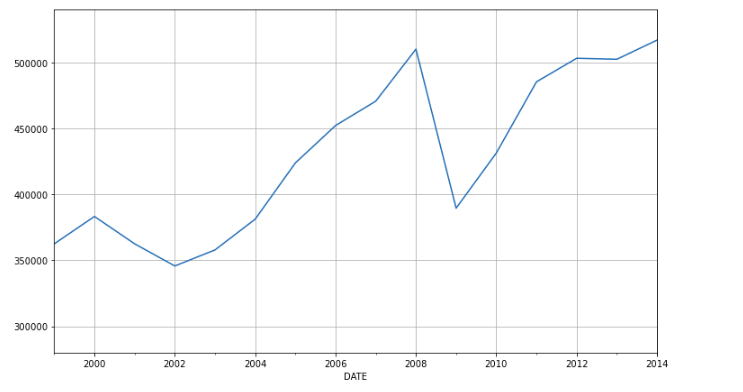
Similarly, you can plot for specific dates as per your choice. Let’s say I want to plot the maximum values for the start of every year from 1995 till 2005. I can do it as follows.

ax = df['UMTMVS'].resample(rule='AS').max().plot(xlim=["1999-01-01","2014-01-01"],ylim=[280000,540000], figsize=(12,7))

ax.yaxis.grid(True)

ax.xaxis.grid(True)

Here, we have specified the *xlim*and *ylim*. See how I have added the dates in *xlim*. The main pattern is *xlim=['starting date', 'ending date']*.



And here, you can see the output of Maximum Values at the Start of Year from 1999 till 2014.

**Learning Outcomes**

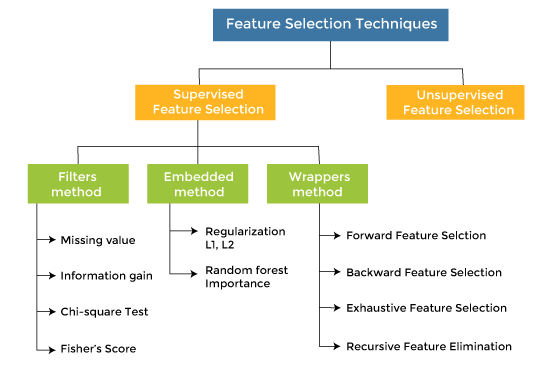
 This brings us to the end of this article. Hopefully, you are now aware of the basics of

* **Loading time series dataset correctly in Pandas**
* **Indexing in Time-Series Data**
* **Time-Resampling using Pandas**
* **Rolling Time Series**
* **Plotting Time-series Data using Pandas**

These topics correctly and can apply them in your own datasets too.

**Feature Selection Techniques in Machine Learning**

While building a machine learning model for real-life dataset, we come across a lot of features in the dataset and not all these features are important every time. Adding unnecessary features while training the model leads us to reduce the overall accuracy of the model, increase the complexity of the model and decrease the generalization capability of the model and makes the model biased. Even the saying “Sometimes less is better” goes as well for the machine learning model. Hence, **feature selection** is one of the important steps while building a machine learning model. Its goal is to find the best possible set of features for building a machine learning model.



Some popular techniques of feature selection in machine learning are:

* Filter methods
* Wrapper methods
* Embedded methods

**Filter Methods**

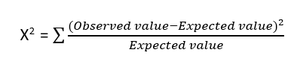
These methods are generally used while doing the pre-processing step. These methods select features from the dataset irrespective of the use of any machine learning algorithm. In terms of computation, they are very fast and inexpensive and are very good for removing duplicated, correlated, redundant features but these methods do not remove multicollinearity. Selection of feature is evaluated individually which can sometimes help when features are in isolation (don’t have a dependency on other features) but will lag when a combination of features can lead to increase in the overall performance of the model.

https://media.geeksforgeeks.org/wp-content/uploads/20201204094030/15.PNG

*Filter Methods Implementation*

Some techniques used are:

* **Information Gain –** It is defined as the amount of information provided by the feature for identifying the target value and measures reduction in the entropy values. Information gain of each attribute is calculated considering the target values for feature selection.
* **Chi-square test —** Chi-square method (X2) is generally used to test the relationship between categorical variables. It compares the observed values from different attributes of the dataset to its expected value.

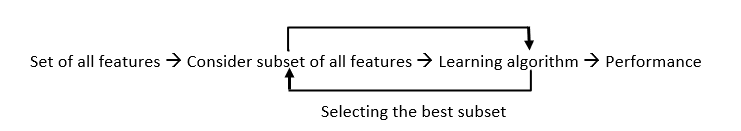


*Chi-square Formula*

* **Fisher’s Score –** Fisher’s Score selects each feature independently according to their scores under Fisher criterion leading to a suboptimal set of features. The larger the Fisher’s score is, the better is the selected feature.
* **Correlation Coefficient –** Pearson’s Correlation Coefficient is a measure of quantifying the association between the two continuous variables and the direction of the relationship with its values ranging from *-1 to 1*.
* **Variance Threshold –** It is an approach where all features are removed whose variance doesn’t meet the specific threshold. By default, this method removes features having zero variance. The assumption made using this method is higher variance features are likely to contain more information.
* **Mean Absolute Difference (MAD) –** This method is similar to variance threshold method but the difference is there is no square in MAD. This method calculates the mean absolute difference from the mean value.
* **Dispersion Ratio –** Dispersion ratio is defined as the ratio of the Arithmetic mean (AM) to that of Geometric mean (GM) for a given feature. Its value ranges from *+1 to ∞ as AM ≥ GM* for a given feature. Higher dispersion ratio implies a more relevant feature.
* **Mutual Dependence –**This method measures if two variables are mutually dependent, and thus provides the amount of information obtained for one variable on observing the other variable. Depending on the presence/absence of a feature, it measures the amount of information that feature contributes to making the target prediction.
* **Relief –** This method measures the quality of attributes by randomly sampling an instance from the dataset and updating each feature and distinguishing between instances that are near to each other based on the difference between the selected instance and two nearest instances of same and opposite classes.

**Wrapper methods:**

Wrapper methods, also referred to as greedy algorithms train the algorithm by using a subset of features in an iterative manner. Based on the conclusions made from training in prior to the model, addition and removal of features takes place. Stopping criteria for selecting the best subset are usually pre-defined by the person training the model such as when the performance of the model decreases or a specific number of features has been achieved. The main advantage of wrapper methods over the filter methods is that they provide an optimal set of features for training the model, thus resulting in better accuracy than the filter methods but are computationally more expensive.



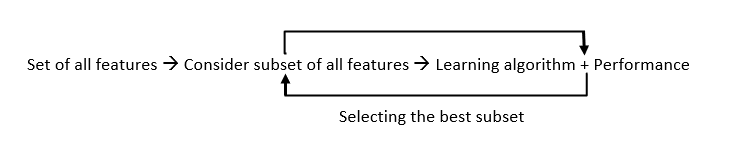
*Wrapper Methods Implementation*

Some techniques used are:

* **Forward selection –**This method is an iterative approach where we initially start with an empty set of features and keep adding a feature which best improves our model after each iteration. The stopping criterion is till the addition of a new variable does not improve the performance of the model.
* **Backward elimination –** This method is also an iterative approach where we initially start with all features and after each iteration, we remove the least significant feature. The stopping criterion is till no improvement in the performance of the model is observed after the feature is removed.
* **Bi-directional elimination –** This method uses both forward selection and backward elimination technique simultaneously to reach to one unique solution.
* **Exhaustive selection –** This technique is considered as the brute force approach for the evaluation of feature subsets. It creates all possible subsets and builds a learning algorithm for each subset and selects the subset whose model’s performance is best.
* **Recursive elimination –** This greedy optimization method selects features by recursively considering the smaller and smaller set of features. The estimator is trained on an initial set of features and their importance is obtained using feature\_importance\_attribute. The least important features are then removed from the current set of features till we are left with the required number of features.

**Embedded methods:**

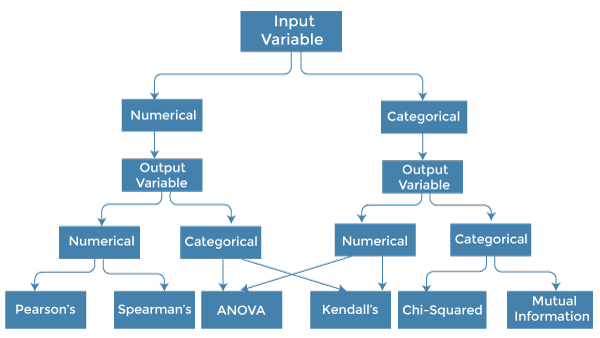
In embedded methods, the feature selection algorithm is blended as part of the learning algorithm, thus having its own built-in feature selection methods. Embedded methods encounter the drawbacks of filter and wrapper methods and merge their advantages. These methods are faster like those of filter methods and more accurate than the filter methods and take into consideration a combination of features as well.



*Embedded Methods Implementation*

Some techniques used are:

* **Regularization –** This method adds a penalty to different parameters of the machine learning model to avoid over-fitting of the model. This approach of feature selection uses Lasso (L1 regularization) and Elastic nets (L1 and L2 regularization). The penalty is applied over the coefficients, thus bringing down some coefficients to zero. The features having zero coefficient can be removed from the dataset.
* **Tree-based methods –**These methods such as Random Forest, Gradient Boosting provides us feature importance as a way to select features as well. Feature importance tells us which features are more important in making an impact on the target feature.



**Conclusion:**

Apart from the methods discussed above, there are many other methods of feature selection. Using hybrid methods for feature selection can offer a selection of best advantages from other methods, leading to reduce in the disadvantages of the algorithms. These models can provide greater accuracy and performance when compared to other methods. Dimensionality reduction techniques such as Principal Component Analysis (PCA), Heuristic Search Algorithms, etc. don’t work in the way as to feature selection techniques but can help us to reduce the number of features.

Feature selection is a wide, complicated field and a lot of studies has already been made to figure out the best methods. It depends on the machine learning engineer to combine and innovate approaches, test them and then see what works best for the given problem.