Tutorial 2

Naïve forecasting can be easily done in Excel. However, if you want to try programming, go for it. You'll be proud of yourselves, I am sure 😊 At the end of the file you will find a complete program with highlights and explanations of what you should do with your program when you are writing it for naïve forecasting.

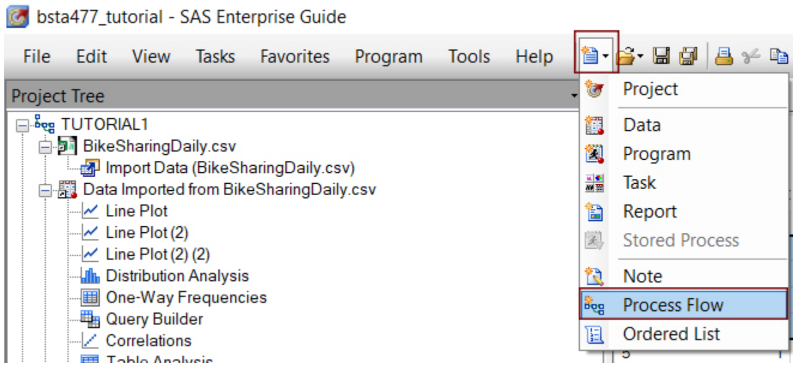
Before you start: How to create a Process Flow

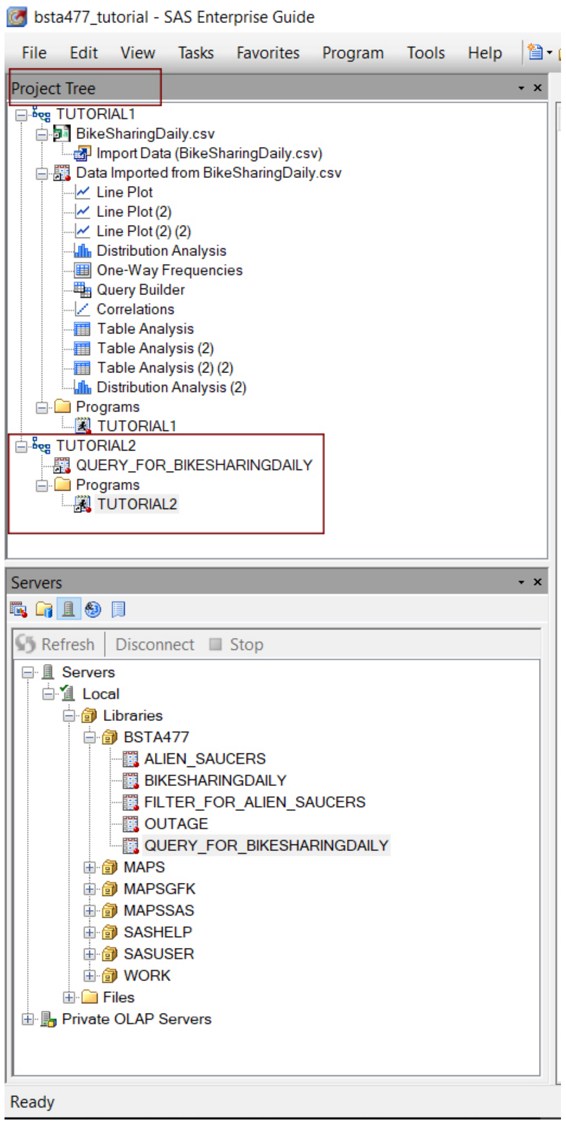
This tutorial assumes that you have already created a project. You need to open it. For the next tutorial we will create a new Process Flow, which will let us separate tutorials from each other and keep our project more organized.

Click on the icon that looks like a sheet of paper → Process Flow → Right click on the icon Process Flow once you see in the pane on the left Process Flow **→ Call it tutorial2.**

When you work with the materials for Tutorial2, make sure you left click on that Process Flow so that all the new datasets you open and create, get opened under that tutorial.

For this tutorial we are going to use QUERY_FOR_BIKESHARINGDAILY, that we created in the last tutorial.





Run a libname statement

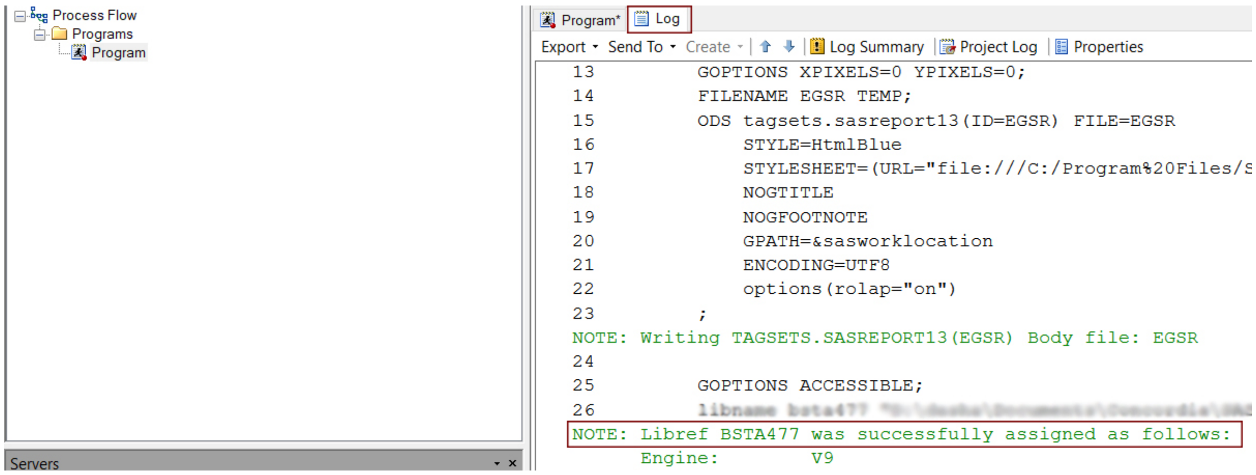
ALWAYS START A NEW SESSION WITH THIS STEP!!! NEVER SKIP IT!

Make sure that you are in the TUTORIAL2 Process Workflow.

Click on the icon that looks like a sheet of paper → Choose program → Type the libname statement (as shown below)

```
libname bsta477 "C:\bsta477";
```

Remember to always check the log. It will show whether the library has been successfully assigned or not!



```
          GOPTIONS XPIXELS=0 YPIXELS=0;
          FILENAME EGSR TEMP;
          ODS tagsets.sasreport13(ID=EGSR) FILE=EGSR
              STYLE=HtmlBlue
              STYLESHEET=(URL="file:///C:/Program%20Files/S
              NOGTITLE
              NOGFOOTNOTE
              GPATH=&sasworklocation
              ENCODING=UTF8
              options(rolap="on")
          ;
NOTE: Writing TAGSETS.SASREPORT13(EGSR) Body file: EGSR

          GOPTIONS ACCESSIBLE;
          libname bsta477 [illegible]
NOTE: Libref BSTA477 was successfully assigned as follows:
      Engine:        V9
```

About Naïve Forecasting

In Naïve forecasting, that is typically considered a benchmark against which all other models are going to be compared, we use data from the previous month to forecast the following month.

SAS EG has a special function for naïve forecasting: lag(). So, if we want to use naïve forecasting trying to predict next month's total rides – both member and non-member - (the ***cnt*** variable), we should use **lag(cnt)**. How does the function work? As shown in the table below, when we create a lagged variable (called ***Naïve_Cnt***) we take the value of the previous day, i.e. 1/1/2011 and use it to project member rides for 1/2/2011 and so on. With this approach we cannot predict the number of member rides for 1/1/2011 since the information for the previous period is not available.

instant	dteday	cnt	Naive_Cnt
1	1/1/2011	985	.
2	1/2/2011	801	985
3	1/3/2011	1349	801
4	1/4/2011	1562	1349

What we are going to do in this tutorial

In this tutorial we are going to:

A. Forecast member rides and calculate residuals
B. plot our actual, forecasted and residual data on a line plot
C. create a needle plot of residuals
D. calculate error terms, i.e. MAPE, MAD, RMSE, MSE and so on in order to assess the accuracy of the model

This tutorial is very programming oriented. You can easily do the same operations in Excel. However, if you want to have some hands-on programming experience, you are welcome to give it a try! Are you excited? We're about to build our first model 😊

HOW TO FORECAST AND PLOT THE RESULTS

A. Forecast total rides

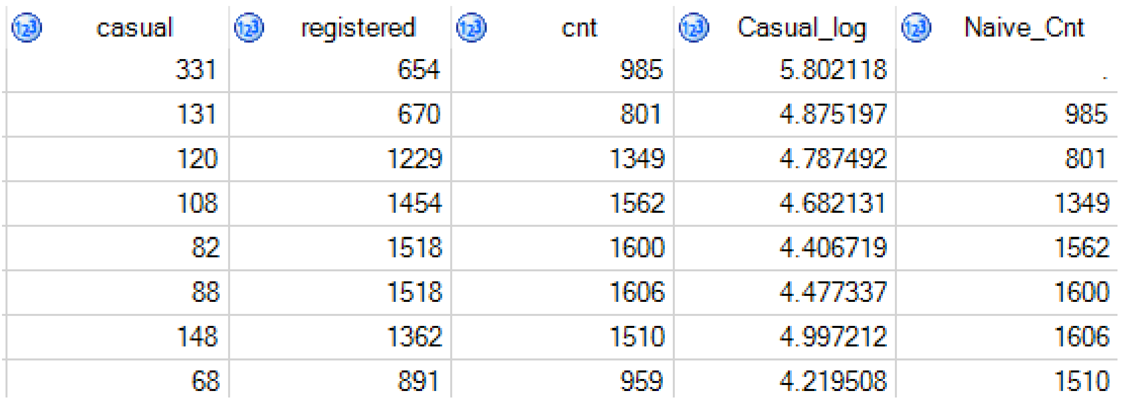
All of this is done in a few lines 😊 Here's the program (for the full program, check out the SAS file accompanying this tutorial for annotations):

```
/*FORECASTING TOTAL RIDES USING THE SIMPLE NAIVE MODEL*/
data bsta477.naive_bikedata;
    set bsta477.bikedata1;
    Naive_Cnt=lag(cnt);
run;
```

Let's analyze it:

- **data** → used to indicate that you are going to create a new file
    - bsta477.naive_bikedata:
        - bsta477 – the permanent library used to store our new file
        - naïve_bikedata – the name of the new file we are creating
- **set** → used to indicate which original dataset we are going to use to create a new file
    - bsta477.bikedata1:
        - bsta477 – the permanent library used to store our new file
        - bikedata – the name of the existing file we are using
- **Naïve_cnt** → the name of a new variable that we are creating, and it equals a lagged variable or is, basically, our naïve forecast.
- **run** → tells SAS to execute the command.

As a result, a new dataset will be created. It will be called Naïve_bike and will contain the two newly created variables: Naïve_reg and Residual.



casual	registered	cnt	Casual_log	Naive_Cnt
331	654	985	5.802118	.
131	670	801	4.875197	985
120	1229	1349	4.787492	801
108	1454	1562	4.682131	1349
82	1518	1600	4.406719	1562
88	1518	1606	4.477337	1600
148	1362	1510	4.997212	1606
68	891	959	4.219508	1510

B. How to partition the datasets

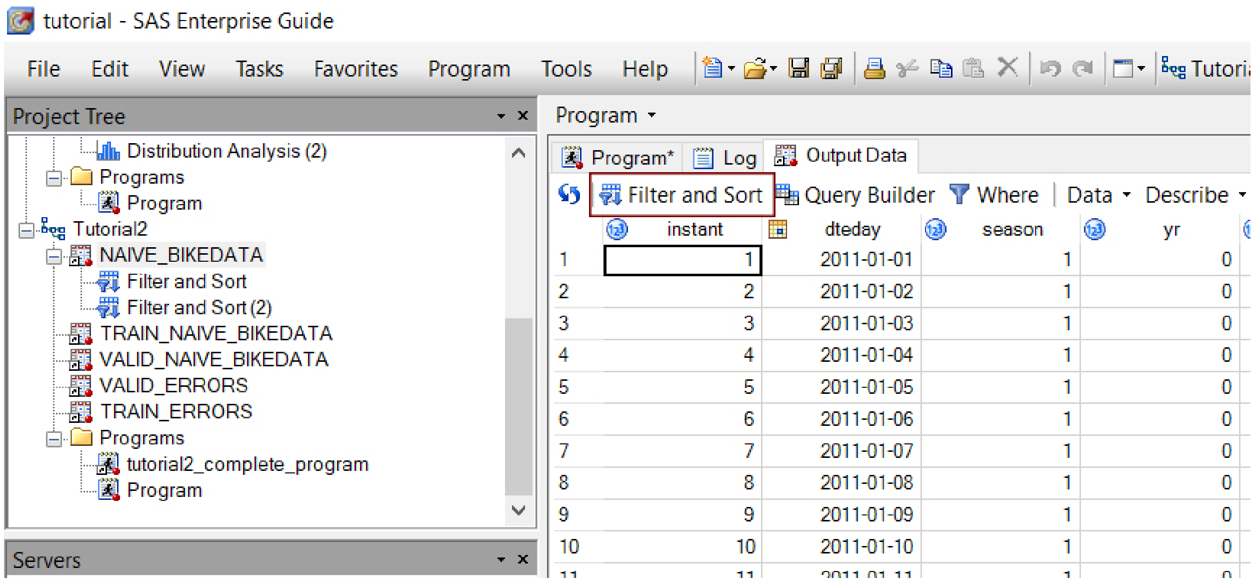
Now that we have our forecast, we will actually, have to partition the dataset into two parts – a train and a validation sets. After that we will need to calculate error terms for each of the datasets. After that we will compare the errors terms of the train and validation data with each other, and we will be able to identify whether the Naïve model is good or not and whether it overfits the data.

Here is how to do it in SAS:

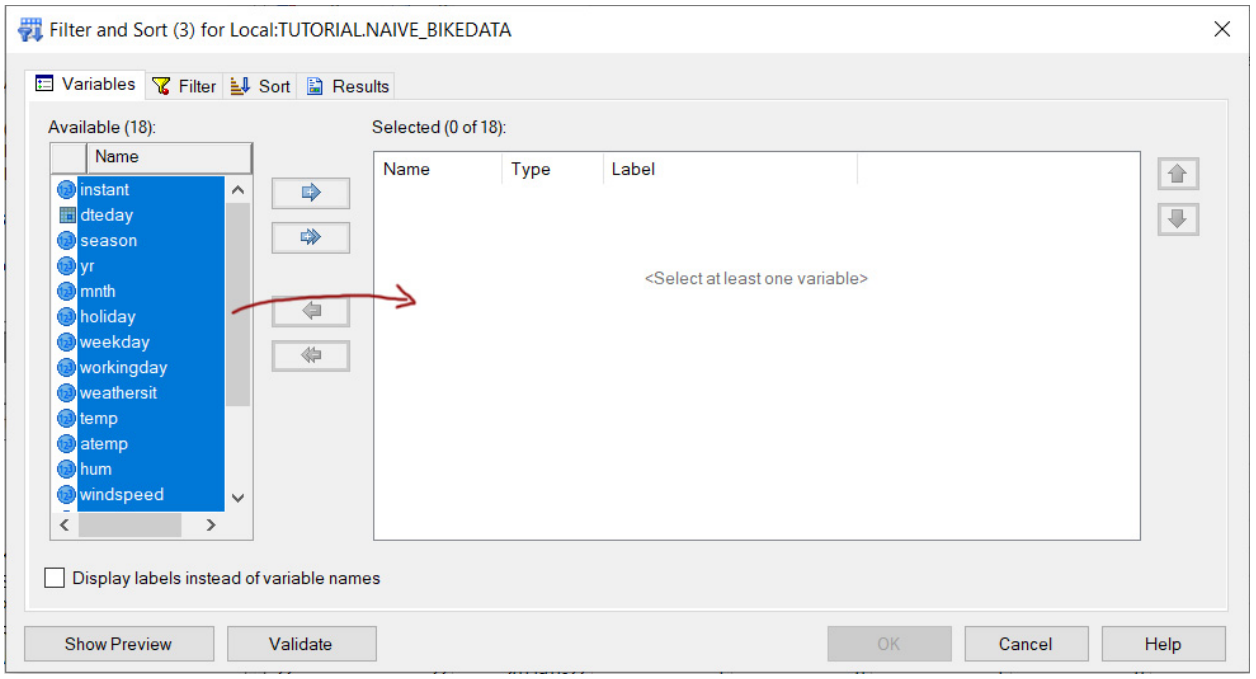
Open the dataset → Filter and Sort → Drag to the right the variables you want to see in your dataset → Filter → Choose the variable (in our case ***dtedate)*** → Choose the condition (in our case ***less than)*** → Click on the ellipsis button → More values (if you do not see the values you need) → Click on the value of interest → OK → Results → Change → Double click on the Servers → Double click on the Local → Double click on the BSTA477 library → Change the name of the dataset → Save → Ok

This lets you create a training subset of the original dataset. Then create the validation one.

In our case we created a training dataset where the dates were before December 1st, 2012 (the condition was less than). For the validation dataset it was all the dates including the first of December (the condition was greater than or equal to).


tutorial - SAS Enterprise Guide

	instant	dteday	season	yr
1	1	2011-01-01	1	0
2	2	2011-01-02	1	0
3	3	2011-01-03	1	0
4	4	2011-01-04	1	0
5	5	2011-01-05	1	0
6	6	2011-01-06	1	0
7	7	2011-01-07	1	0
8	8	2011-01-08	1	0
9	9	2011-01-09	1	0
10	10	2011-01-10	1	0

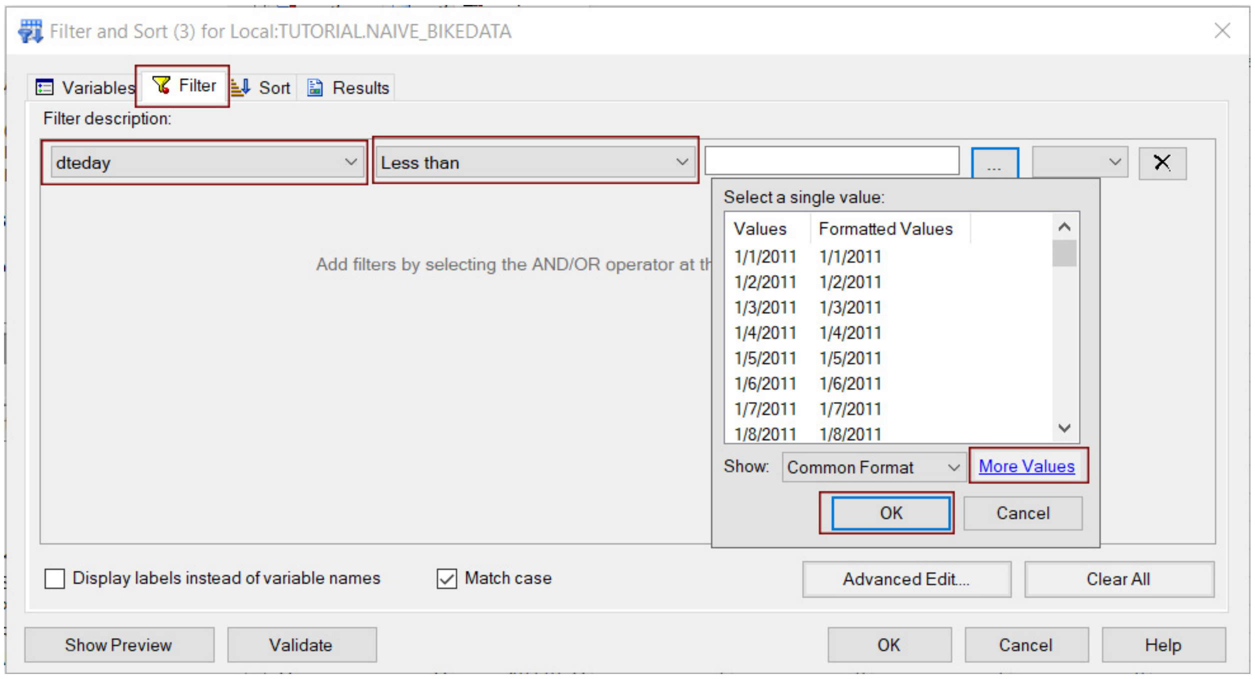

Filter and Sort (3) for Local:TUTORIAL.NAIVE_BIKEDATA

Available (18):

Selected (0 of 18):

Name	Type	Label

<Select at least one variable>


Filter and Sort (3) for Local:TUTORIAL.NAIVE_BIKEDATA

Filter description:

dteday | Less than

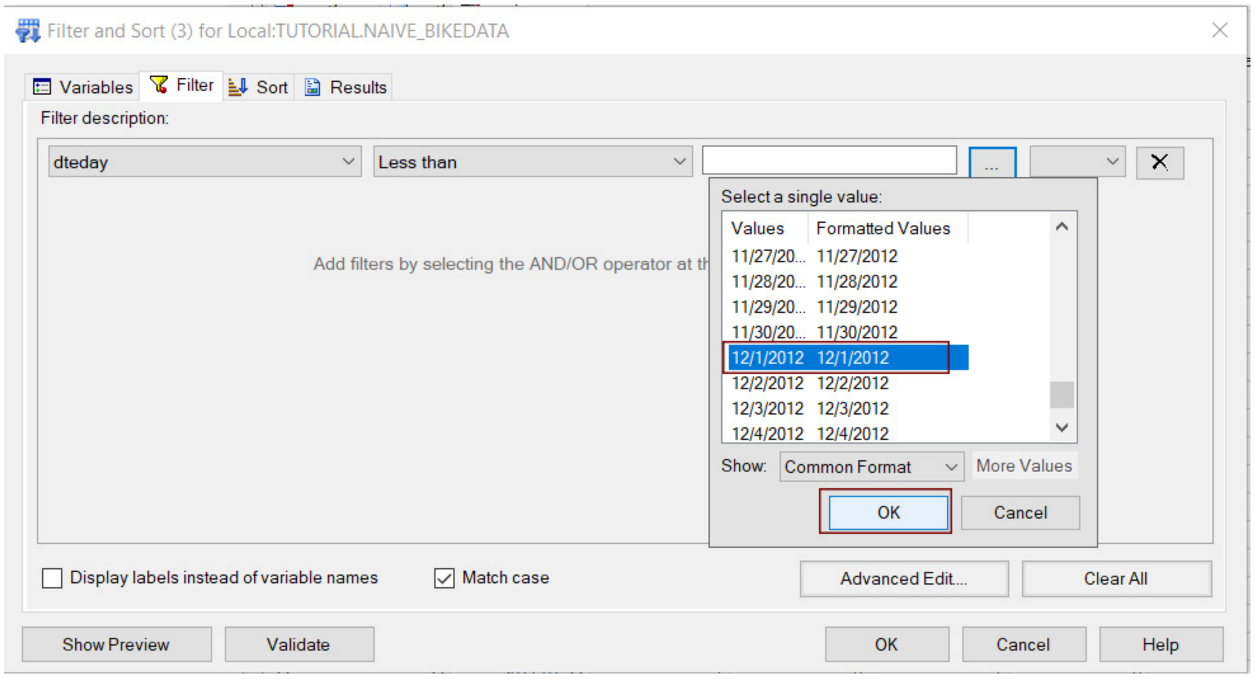
Add filters by selecting the AND/OR operator at th

Select a single value:

Values	Formatted Values
1/1/2011	1/1/2011
1/2/2011	1/2/2011
1/3/2011	1/3/2011
1/4/2011	1/4/2011
1/5/2011	1/5/2011
1/6/2011	1/6/2011
1/7/2011	1/7/2011
1/8/2011	1/8/2011

Show: Common Format — More Values

☐ Display labels instead of variable names ☑ Match case


Filter and Sort (3) for Local:TUTORIAL.NAIVE_BIKEDATA

Filter description:

dteday | Less than

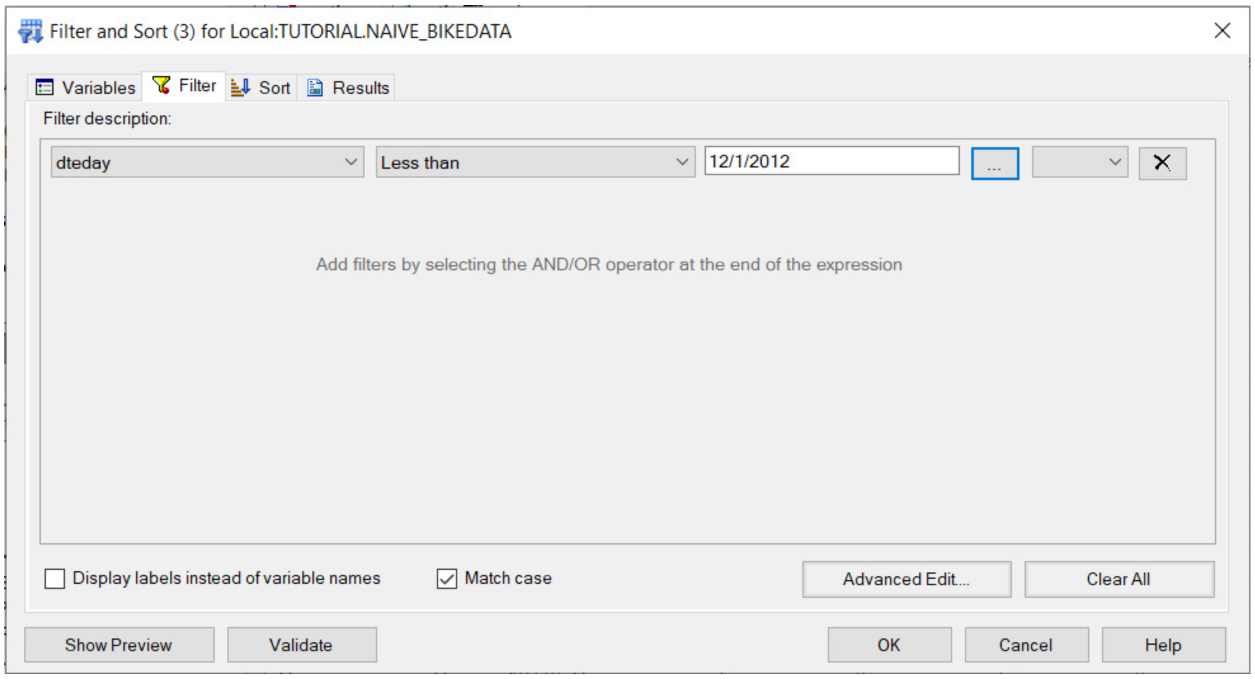
Add filters by selecting the AND/OR operator at th

Select a single value:

Values	Formatted Values
11/27/20...	11/27/2012
11/28/20...	11/28/2012
11/29/20...	11/29/2012
11/30/20...	11/30/2012
12/1/2012	12/1/2012
12/2/2012	12/2/2012
12/3/2012	12/3/2012
12/4/2012	12/4/2012

Show: Common Format — More Values

☐ Display labels instead of variable names ☑ Match case

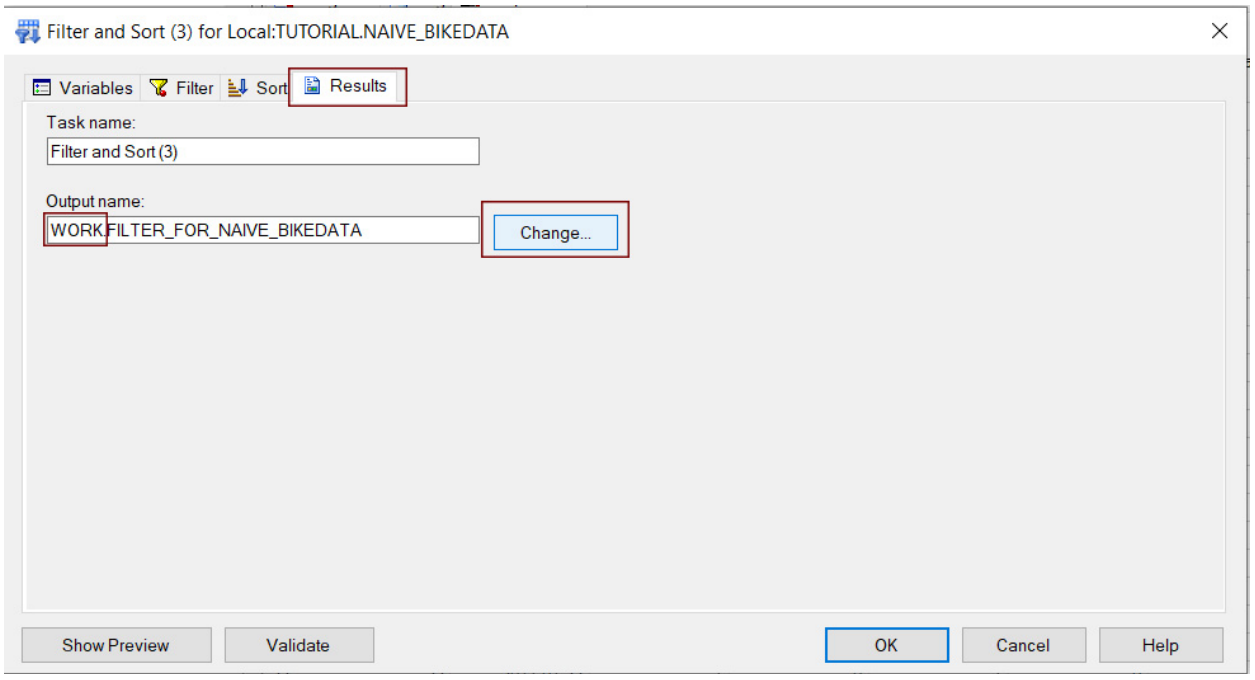

Filter and Sort (3) for Local:TUTORIAL.NAIVE_BIKEDATA

Filter description:

dteday | Less than | 12/1/2012

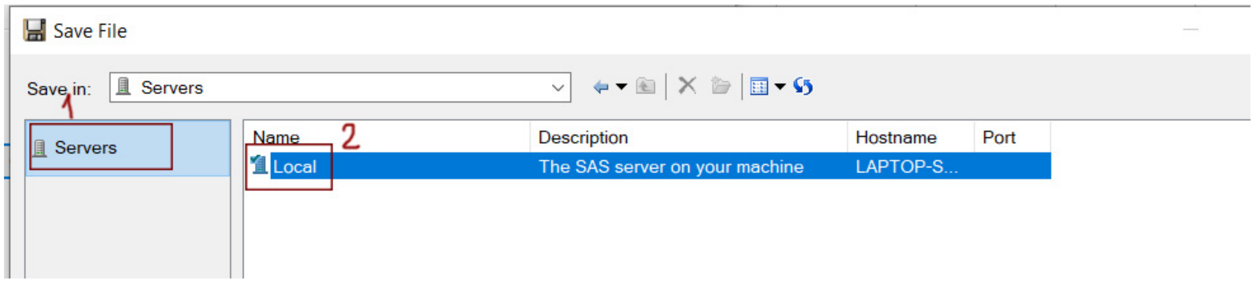
Add filters by selecting the AND/OR operator at the end of the expression

☐ Display labels instead of variable names ☑ Match case


Filter and Sort (3) for Local:TUTORIAL.NAIVE_BIKEDATA

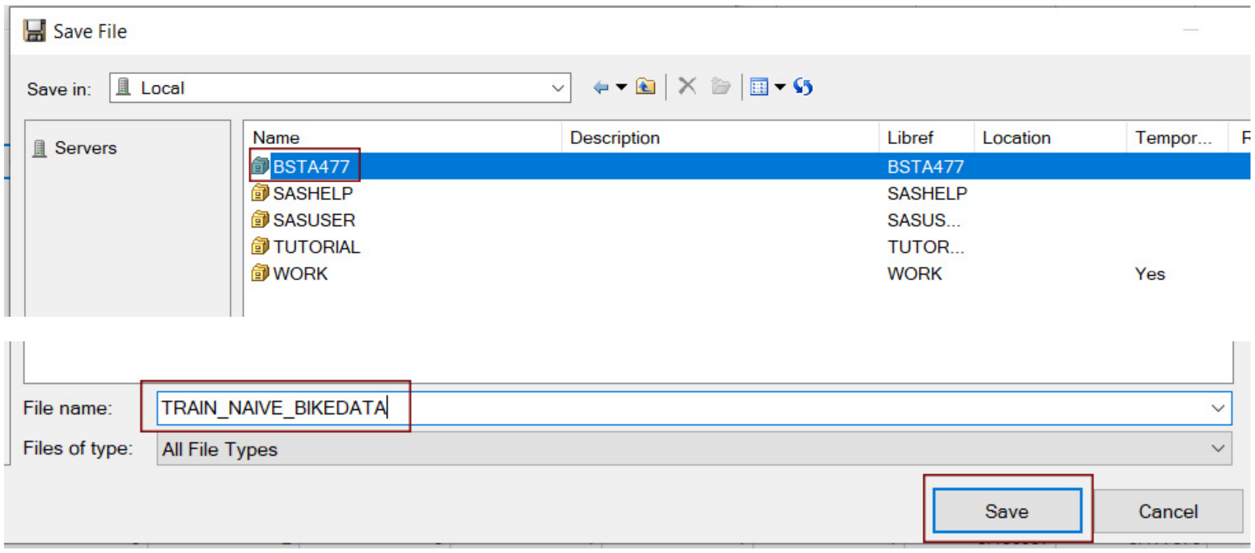
Task name: Filter and Sort (3)

Output name: WORK.FILTER_FOR_NAIVE_BIKEDATA


Save File

Save in: Servers

Name	Description	Hostname	Port
Local	The SAS server on your machine	LAPTOP-S...	

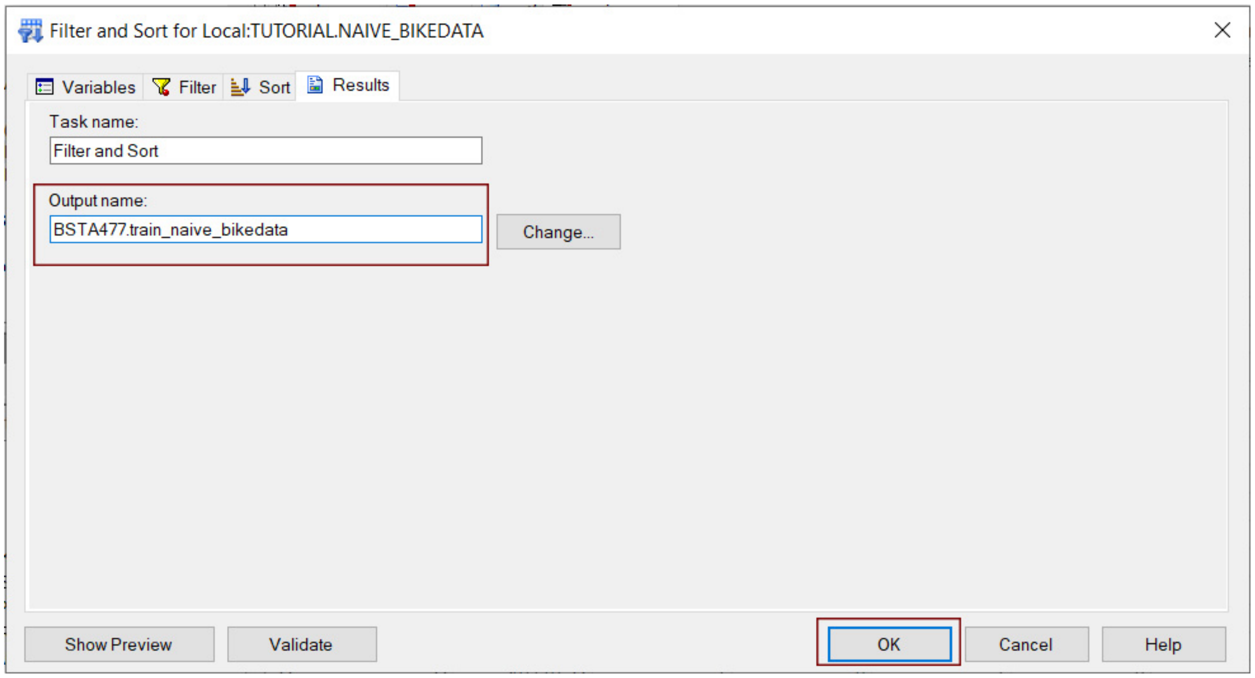

Save File

Save in: Local

Name	Description	Libref	Location	Tempor...
BSTA477		BSTA477		
SASHELP		SASHELP		
SASUSER		SASUS...		
TUTORIAL		TUTOR...		
WORK		WORK		Yes

File name: TRAIN_NAIVE_BIKEDATA

Files of type: All File Types


Filter and Sort for Local:TUTORIAL.NAIVE_BIKEDATA

Task name: Filter and Sort

Output name: BSTA477.train_naive_bikedata

C. Calculating residuals and begin to calculate errors

After having forecasted the total rides, we can now calculate the residuals and we will also create additional formulas with the residuals that will prepare our dataset for error terms calculations.

```
/*CALCULATING ERROR TERMS*/
data bsta477.train_errors;
	set bsta477.train_naive_bikedata;
	Residual=cnt-Naive_Cnt;
	Abs=abs(Residual);
	Square=Residual**2;
	Proportion=Residual/cnt;
	Abs_proportion=Abs/cnt;
run;
```

Again, when we create a new dataset we start our step with the **data** statement and in the case above we are creating a permanent dataset in the library bsta477. The dataset is called train_errors.

To indicate which dataset we should be using, we are using the **set** statement. In our case we are using the permanent dataset called train_naive_bikedata.

The new dataset (train_errors) will serve as a basis for further calculations of the error terms. So, here we are going to calculate residuals (actual – forecasted values). In addition, we are going to take the absolute value of the residuals, squared residuals and absolute residuals out of the actual values and residuals out of the actual values. We need to do it since these values are a part of the formulas of the error terms that are used to assess models' performance.

```
/*ERROR TERMS WE ARE GOING TO CALCULATE:
MAD = (sum|Actual - Forecast|)/n
MSE = (sum(Actual - Forecast)^2)/n
RMSE = Square root of MSE
MAPE = 100%*sum[|Actual - Forecast|/Actual]/n
MPE =  100%*sum[(Actual - Forecast)/Actual]/n

ACTUAL - FORECAST = RESIDUAL*/
```

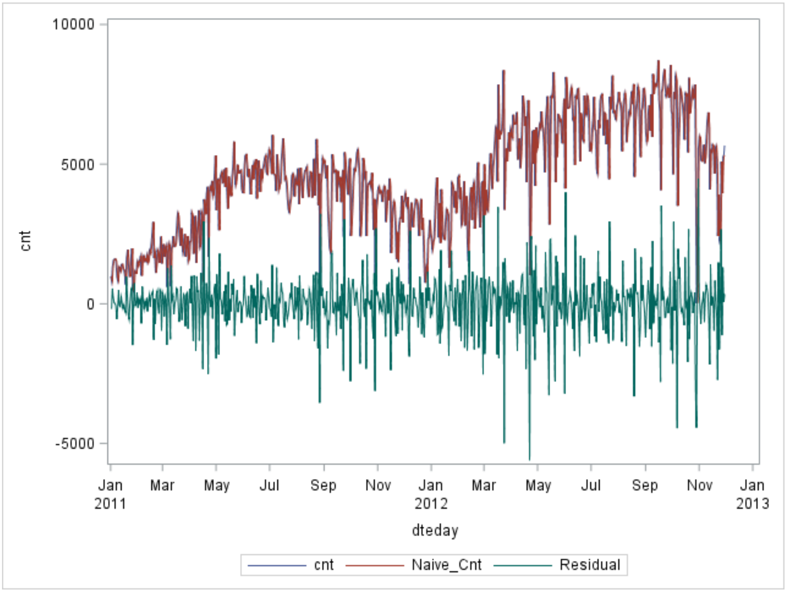
D. Plot our actual, forecasted and residual data on a line plot

It is usually helpful to visualize the data. So, now we are going to plot the actual values of the registered user rides, then our forecasted values and the residuals on the same plot.

```
/*PLOTTING THE TOTAL RIDES VS FORECASTS AND RESIDUALS*/
title "Plotting Total Rides vs Forecasts and Residuals";
proc sgplot data=bsta477.train_errors;
     series x=dteday y=cnt / lineattrs=(pattern=1);
     series x=dteday y=Naive_Cnt / lineattrs=(pattern=1);
     series x=dteday y=Residual / lineattrs=(pattern=1);
run;
```

- **proc sgplot** → gives a command to SAS to create a plot
  - **data=** → specifies a dataset to be used and its location
    - bsta477. → specifies the permanent library where the dataset is located
    - train_errors→ specifies the dataset we created in the data step (from before)
- **series** → is used for line plots and you need to specify x= and y= values for each line you are going to plot:
  - for instance, on the horizontal axis we are plotting the date values (x=dteday) and on the vertical axis we are plotting the rides and residual values (y=cnt, for example).
  - **/ lineattrs=(pattern=1)** → lets you specify the looks of the line of your line plot. Pattern=1 means that the line will be shown as a solid line
- **run** → tells SAS to execute the command.

As a result, the following plot will be created. The plot of the residuals is represented by the green line. The registered users are represented by the blue line and the red line represents the forecasted registered user rides. As you can see so far, the residuals seem to be close to zero and the forecasted values are pretty close to the actual ones. It may indicate that the forecast is not so bad.



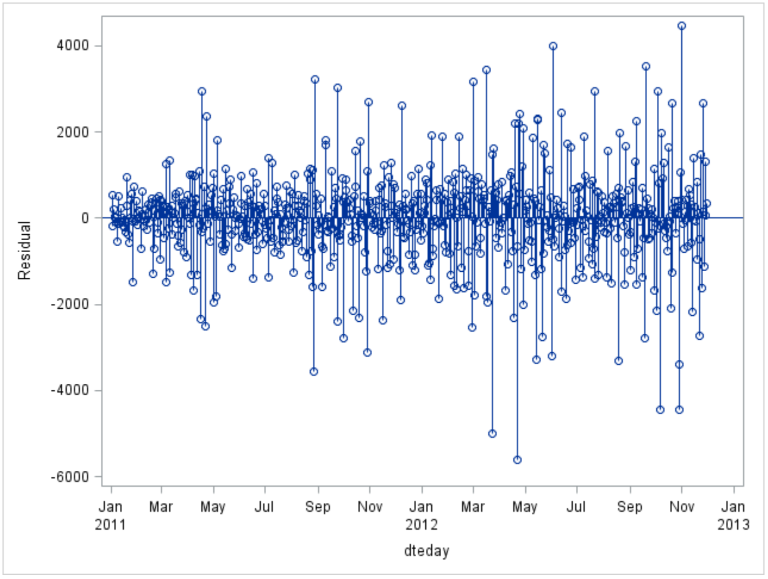
E. Create a needle plot of residuals

To look deeper into the residuals, we may create a needle plot. Let's have a look at how it is created.

```
/*PLOTTING RESIDUALS*/
title "Residual Plot";
proc sgplot data=bsta477.train_errors;
      needle x=dteday y=Residual / markers;
run;
```

- **proc sgplot** → gives a command to SAS to create a plot
  - **data=** → specifies a dataset to be used and its location
    - bsta477. → specifies the permanent library where the dataset is located
    - train_errors → specifies the dataset we created in the data step (from before)
- **needle** → is used for a needle plot and you need to specify x= and y= values:
  - on the horizontal axis we are plotting the date values (x=dteday) and on the vertical axis we are plotting the residual values (y=residual).
  - **/ markers** → makes SAS add a marker at the end of each needle.
- **run** → tells SAS to execute the command.

As a result, the following plot will be created. The plot of the residuals is represented by the needles. Again, as you can see so far, the residuals seem to be close to zero and the forecasted values are pretty close to the actual ones. We can also see outliers with this plot, for instance, in May 2012 we can see that the residuals are at around 6000, which means that we were pretty off with our forecast for this month.



F. Calculate error terms, i.e. MAPE, MAD, RMSE, MSE and MPE in order to assess the accuracy of the model

To calculate the error measurements for the naïve forecast, we will be following the regular formulas for MAPE, MAD, RMSE, MSE, and MPE

- Mean absolute deviation: $MAD = \frac{1}{n}\sum_{t=1}^{n}\left|Y_t - \widehat{Y}_t\right|$
- Mean squared error: $MSE = \frac{1}{n}\sum_{t=1}^{n}\left(Y_t - \widehat{Y}_t\right)^2$
- Root mean squared error: $RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(Y_t - \widehat{Y}_t\right)^2}$
- Mean absolute percentage error: $MAPE = \frac{1}{n}\sum_{t=1}^{n}\frac{|Y_t - \widehat{Y}_t|}{|Y_t|}$
- Mean percentage error: $MPE = \frac{1}{n}\sum_{t=1}^{n}\frac{(Y_t - \widehat{Y}_t)}{Y_t}$

In SAS, we will need to go through a few steps when we calculate errors (trust me, it's much faster in Excel!). Here's the logic that we are going to follow:

- We will need to calculate residuals, the absolute value of individual residuals, squared values of individual residuals, proportion of the residuals vs the forecast as well as the absolute proportion of the residual vs the absolute forecasted value.
- After that we will need to sum the proportions, absolute proportions, individual residuals, absolute residuals and squared values of the residuals.
- Then we will have all the values needed to be used in the formulas and we will use the values from above to create needed calculations.

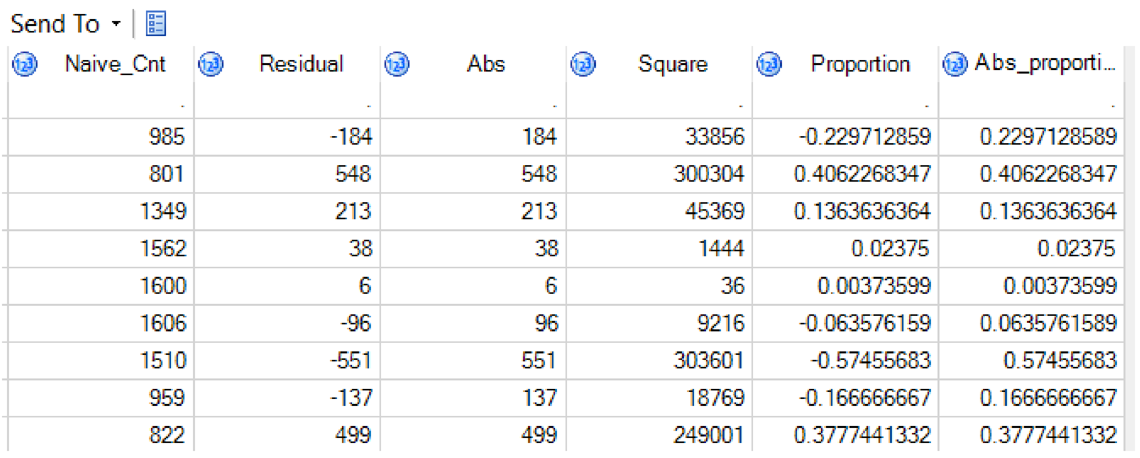
STEP1: Create a dataset with residuals, absolute, squared residuals and proportions and absolute proportions.
Below is the program that we will use to do that. We will start with a data step like before, but we will add a few more new variables in order to expand the naïve_bike dataset we created before. This program will overwrite the dataset we have created.

```
/*CALCULATING ERROR TERMS*/
data bsta477.train_errors;
     set bsta477.train_naive_bikedata;
     Residual=cnt-Naive_Cnt;
     Abs=abs(Residual);
     Square=Residual**2;
     Proportion=Residual/cnt;
     Abs_proportion=Abs/cnt;
run;
```

Let’s analyze it:

- **data** → used to indicate that you are going to create a new file
    - bsta477.train_errors:
        - bsta477 – the permanent library used to store our new file
        - naïve_bike – the name of the new file we are creating
- **set** → used to indicate which original dataset we are going to use to create a new file
    - bsta477.train_naive_bikedata:
        - bsta477 – the permanent library used to store our new file
        - train_naive_bikedata – the name of the new file we are creating
- **Residual** → the name of a new variable we are creating, and it is basically represented by a formula “actual value minus forecasted value”
- **Abs** → the name of a new variable that we are creating and that equals an absolute value of a certain residual
- **Square** → the name of a new variable that we are creating and that is represented by a squared residual
- **Proportion** → the name of a new variable that we are creating and that equals a ratio of a residual out of a forecasted value
- **Abs_proportion** → the name of a new variable that we are creating and that equals the absolute value of a proportion
- **run** → tells SAS to execute the command.

As a result, a new dataset will be created. It will be called Naïve_bike and will contain the six newly created variables: Naïve_reg (our forecast), Residual (residuals), Abs (absolute residuals), Square (squared residuals), Proportion (a proportion of residuals out of a forecasted value) and Abs_proportion (an absolute value of a proportion).


Send To

Naive_Cnt	Residual	Abs	Square	Proportion	Abs_proporti...
.	.	.	.	.	.
985	-184	184	33856	-0.229712859	0.2297128589
801	548	548	300304	0.4062268347	0.4062268347
1349	213	213	45369	0.1363636364	0.1363636364
1562	38	38	1444	0.02375	0.02375
1600	6	6	36	0.00373599	0.00373599
1606	-96	96	9216	-0.063576159	0.0635761589
1510	-551	551	303601	-0.57455683	0.57455683
959	-137	137	18769	-0.166666667	0.1666666667
822	499	499	249001	0.3777441332	0.3777441332

STEP2: Summarize the variables you have created in the program above.
In the program below you will add all rows for each separate column in order just to get the total value or absolute residuals, squared residuals, proportions of residuals and absolute proportions. This program will also result in creating new variables and a completely new dataset that will be used at the next step.

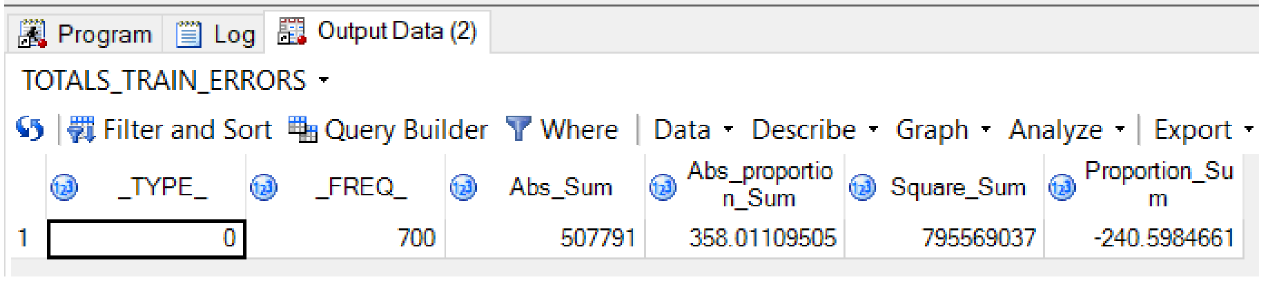
```
/*SUMMING UP THE VALUES*/
proc summary data=bsta477.train_errors;
     var Abs Abs_proportion Square Proportion;
     output out=totals_train_errors sum= / autoname;
run;
```

Let's analyze it:

- **proc summary** → gives a command to SAS to summarize the variables
    - **data=** → specifies a dataset to be used and its location
        - bsta477. → specifies the permanent library where the dataset is located
        - train_errors → specifies the dataset we created in the data step
- **var** → which variables to summarize
- **output** → specifies a name of a newly created dataset and newly created variables:
    - **out=** → specify the name of a new dataset that you want to be created once you run this program
    - **sum=** → specifies the total of the variables
    - **/ autoname** → will add a suffix _sum to every new variable it creates. For instance, once it sums all the rows for the absolute residuals, the summed variable **abs** will be called **abs_ sum.** It will be done with each variable, which makes it easy to trace them in the next step.

- **run** → tells SAS to execute the command.

As a result, a new dataset will be created. It will be called totals_naïve_bike and will contain five newly created variables: abs_sum, square_sum, proportion_sum, abs_proportion_sum. These variables will be representing total sums of all values present across each variable. What I mean is that all values will be summed up from row one and up to the last one for the abs variable, then for the square variable and so on. There is a problem with these newly created variables, because their values cannot be used in any formulas directly. That is why we have to go to step 3.



	TYPE	_FREQ_	Abs_Sum	Abs_proportion_Sum	Square_Sum	Proportion_Sum
1	0	700	507791	358.01109505	795569037	-240.5984661

STEP3: Create macro variables.
In the previous step we summed the values across all the newly created variables that we will be using to calculate MAD, MAPE and so on. However, these values are not directly available. Thus, we need to turn those numbers into macro variables that will be accessible by other programs. Only then we will be able to use those values in calculations. That's what are going to do in this step.

```
/*CREATING MACRO VARIABLES TO ACCESS THE VALUES OF THE TOTALS FROM THE
PREVIOUS STEP*/
data _null_;
set totals_train_errors;
      call symputx('abs', Abs_sum);
      call symputx('sqr', Square_sum);
      call symputx('proport', Proportion_sum);
      call symputx('abs_prop', Abs_Proportion_sum);
run;
```

Let's analyze it:

- **data** → used to indicate that you are going to create a new file
  - _null_ → creates no dataset
- **set** → used to indicate which original dataset we are going to use to create a new file
  - totals_train_errors (the temporary dataset created in the proc summary step) will be used in this case
- **call** → a function that will conduct a computation or manipulation that can be later used in other programs

  - **symputx**(*macro-variable*, *value*) → creates a macro variable
    - (**'abs'** is the name of the future macro variable created from **abs_sum**)
- **run** → tells SAS to execute the command.

As a result, no dataset will be created, but you will have macrovariables whose values you will be able to use in various computations in different programs. In this program you are creating 4 macrovariables **&abs** (from abs_sum), **&sqr** (from square_sum), **&proport** (from proportion_sum) and **&abs_prop** (from abs_proportion_sum).

If we follow a chain of transformations of one variable, we will see the following:

Residual → abs (absolute residual) → abs_sum (total sum of absolute values of all residuals across the column) → &abs (a macro variable whose numeric value will be later used in further calculations)

STEP4: Calculating the error terms. The final step!!!
In this step we will use the macro variables we created in the previous step to finally figure out the error terms that will serve as measurements of the model's accuracy.

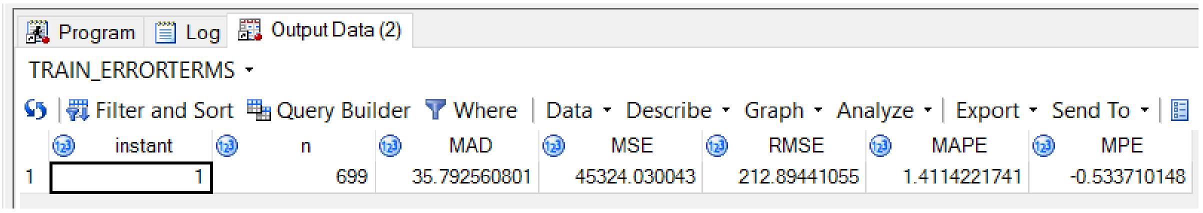
```
/*CALCULATING THE ERROR TERMS*/
data bsta477.train_errorterms (keep=INSTANT N MAD MSE RMSE MAPE MPE);
      set bsta477.train_errors;
      n=699;
      MAD=&abs/n;
      MSE=&sqr/n;
      RMSE=sqrt(MSE);
      MAPE=100*&abs_prop/n;
      MPE=100*&proport/n;
      where instant=1;
run;
```

Let's analyze it:

- **data** → used to indicate that you are going to create a new file that will contain only errors
  - bsta477.naive_bike_errors:
    - bsta477 – the permanent library used to store our new file
    - train_errorterms – the name of the new file we are creating
  - (keep= indicates that in our new dataset we only want to see the following variables: MAD = mean absolute deviation, MSE = mean squared error, RMSE = root mean squared error, MAPE = mean absolute percentage error, MPE = mean percentage error, n = number of total observations/ rows of the dataset, instant = just a number of each row, if we look at the first row, instant = 1, if we look at row 25, instant = 25)

        - Pay attention, we haven't created most of these variables yet (except for the instant), but after we have done it, only these variables will be included into the output
- **set** → used to indicate which original dataset we are going to use to create a new file
    - bsta477.query_for_bikesharingdaily:
        - bsta477 – the permanent library used to store our new file
        - train_error – the name of the new file we are creating
- **n** → the name of a new variable that we are creating, and it equals the total number of observations of the dataset (we will need this number to calculate the average), in our case it equals 699 since we have 700 rows in our dataset, but the forecast value for the first date is blank (for the 1st of January 2011) because we do not have the data to predict the rides for January 1st.
- **MAD** → the name of a new variable we are creating, and it is derived from the mean absolute deviation formula
- **MSE** → the name of a new variable that we are creating and that is calculated according to the mean squared error formula
- **RMSE** → the name of a new variable that we are creating, and it equals a square root of MSE
- **MAPE** → the name of a new variable that we are creating and that equals a value obtained from the mean absolute percentage error calculations
- **MPE** → the name of a new variable that we are creating and that equals the mean percentage error value
- **where** → is used as a filter, which, in our case, lets us limit the new dataset to only one row; otherwise we would end up having 731 rows with exactly the same values. This is not practical, and it is much better to have only one row of this kind.
- **run** → tells SAS to execute the command.

As a result, a new dataset with only one row/observation will be created. It will be called naïve_bike_errors and will contain the six newly created variables as well as the one that existed before: MAD, MSE, RMSE, MAPE, MPE, n, and instant. These values will let us assess the accuracy of the model and also will make the model comparable to other models when we will be trying to select the best model for further predictions.

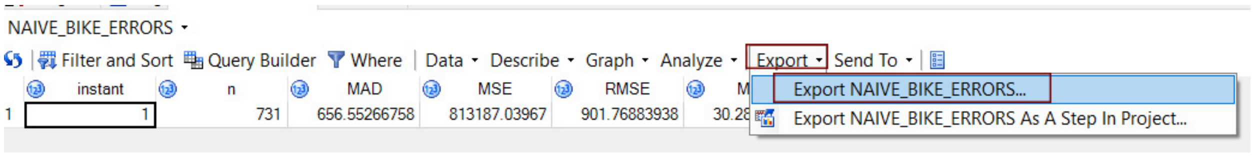


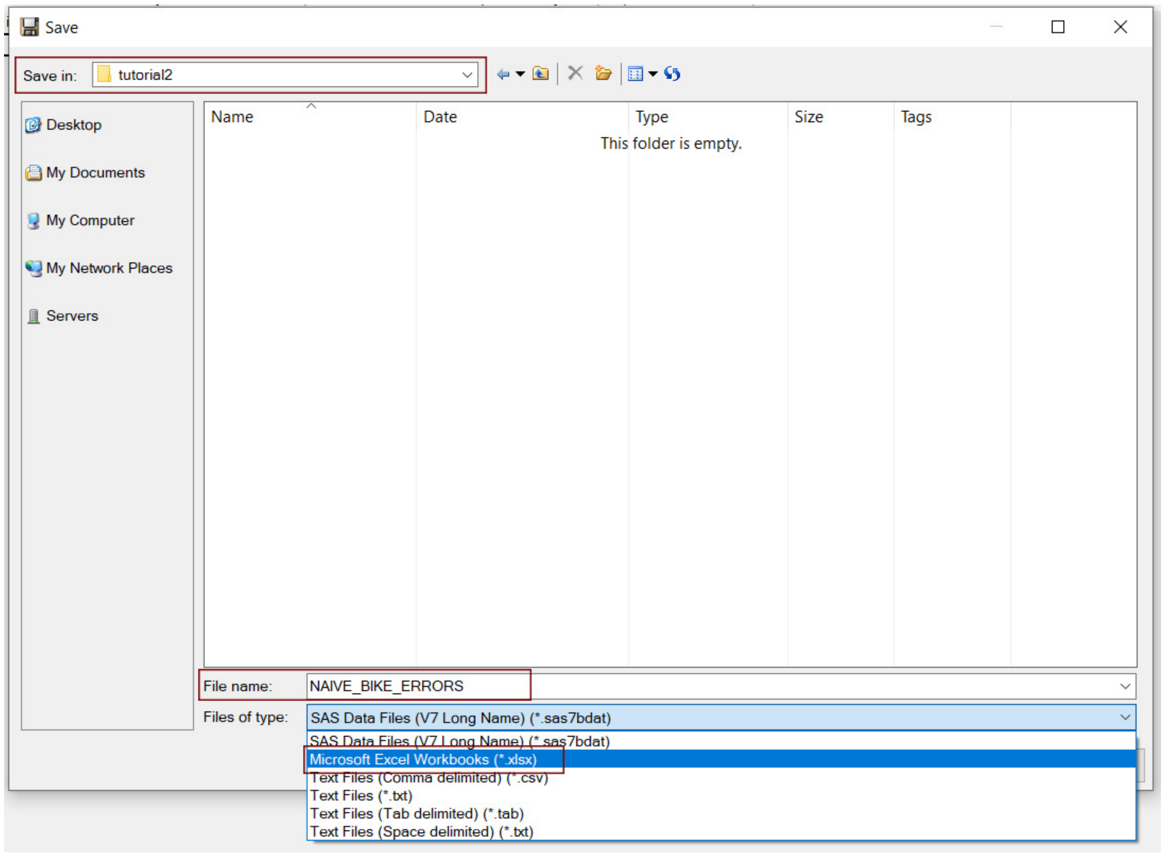
	instant	n	MAD	MSE	RMSE	MAPE	MPE
1	1	699	35.792560801	45324.030043	212.89441055	1.4114221741	-0.533710148

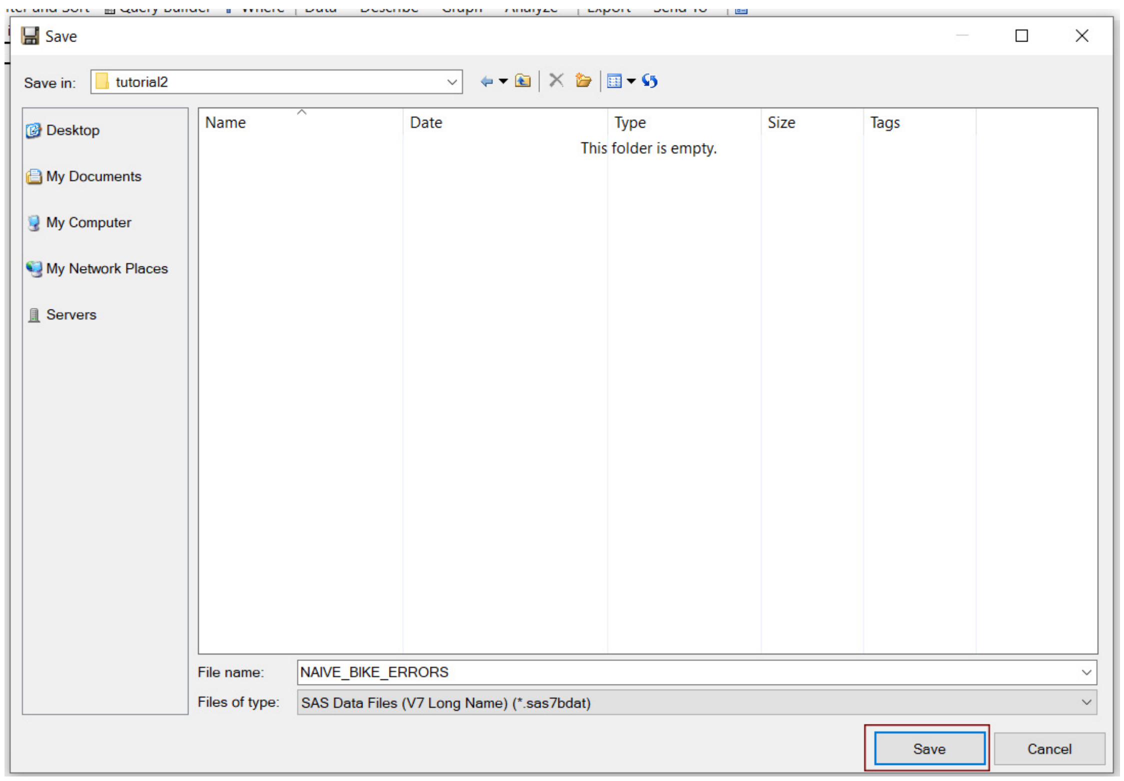
How to export data into Excel

Sometimes you may want to store your results in Excel. In SAS EG it can be done really easily by following these simple steps:

Open the SAS dataset you want to export → Export → Export (Filename) → From a dropdown menu at the bottom choose the format into which you want to export your dataset → Give your file a name and choose a location for your newly created file →




Save
Save in:
tutorial2
Desktop
My Documents
My Computer
My Network Places
Servers
Name
Date
Type
Size
Tags
This folder is empty.
File name:
NAIVE_BIKE_ERRORS
Files of type:
SAS Data Files (V7 Long Name) (*.sas7bdat)
SAS Data Files (V7 Long Name) (*.sas7bdat)
Microsoft Excel Workbooks (*.xlsx)
Text Files (Comma delimited) (*.csv)
Text Files (*.txt)
Text Files (Tab delimited) (*.tab)
Text Files (Space delimited) (*.txt)



File name: NAIVE_BIKE_ERRORS

Files of type: SAS Data Files (V7 Long Name) (*.sas7bdat)