Demo Steps: Automatic Forecasting Using the Python API

The United States Department of Transportation (USDOT) through its Bureau of Transportation Statistics publishes information that is related to airline transportation. Transportation data sets and related statistics are available from USDOT supported websites. The data set **airpass** contains monthly passenger totals for scheduled flights of domestic US carriers from January 1990 through February 2017. The data set exhibits classical trend and seasonality structure that is well suited for classical time series techniques.

Name	Model Role	Measurement Level	Description
DATE	Date	Date	Number of days since January 1, 1960
MONTH	Input	Nominal	1 = January, 2 = February, … 12 = December
YEAR	Input	Interval	Years 1990 – 2017
PASSENGERS	Target	Interval	Number of flight passengers

1. From Jupyter Lab, select **File Browser** > **Home** > **Courses** > **EVMLOPRC** > **Notebooks** and select the **Python_Time_Series_Demo.ipynb** notebook.

2. Load the os, sys, SWAT, pandas, and matplotlib packages. Use the %MATPLOTLIB inline option to print graphics inside the notebook.
```
import os
import sys
import swat
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
swat.options.cas.print_messages = True
```
3. Connect to CAS.
```
conn = swat.CAS(os.environ.get("CASHOST"), os.environ.get("CASPORT"), None, os.environ.get("SAS_VIYA_TOKEN"))
```
4. Upload the **airpass.sas7bdat** file and name it **airpass**. Find the dimension and print the first few observations.
```
castbl = conn.read_sas(os.environ.get("HOME")+"/Courses/EVMLOPRC/DATA/airpass.sas7bdat",
                       casout = dict(name="airpass", replace=True))

indata = 'airpass'

display(castbl.shape)
castbl.head()
```
5. Download the data table to the client and plot the time series.
```
df = castbl.to_frame()
plt.figure(figsize=(8,8))
plt.plot(df['Passengers'], color='blue')
plt.title('1973-2017', fontsize=20)
plt.xlabel('Month', fontsize=15)
plt.ylabel('Number of Passengers', fontsize=15)
plt.show()
```
   This time series exhibits the standard trend and seasonality structure of common time data. Instead of applying a complex model, you can use the timeData action set to build simply structured time series models.

6. Use a DATA step to partition the data. Let data before 2012 be training data and use the remaining data for validation.
```
conn.loadActionSet('dataStep')
actions = conn.builtins.help(actionSet='dataStep')

conn.dataStep.runCode(code=
    '''
    data train;
        set airpass;
        if year < 2012 then output train;
    run;
    '''
)
```
7. Use the forecast action from the timeData action set to enable SAS to choose the best time series model for the data.
```
conn.loadActionSet('timeData')
actions = conn.builtins.help(actionSet='timeData')

tsout = conn.timeData.forecast(
    table = 'train',
    timeid = 'Date',
    interval = 'Month',
    tstart = 'Jan 1, 1990',
    tend = 'Dec 1, 2011',
    dependents = dict(name='Passengers'),
    lead = 62,
    mode = 'automodel',
    forOut = dict(name='forecast_out', replace=True),
    alpha = .05,
    infoOut = dict(name='info_out', replace=True),
    selectOut = dict(name='select_out', replace=True),
    specOut = dict(name='spec_out', replace=True)
)

display(conn.table.fetch(table='select_out'))
conn.table.fetch(table='info_out')
)
```

The forecast action applies automatic time series model identification and forecasting to data and fits the following models:

- regression model with autoregressive integrated moving average (ARIMA) time series errors
- ARIMA with explanatory variable (ARIMAX) model
- simple exponential smoothing

Selected arguments:

Argument	Description
timeid	specifies the timestamp variable.
interval	specifies the time interval (or frequency).
tstart	specifies the start of the time window.
tend	when set to *true*, generates a partition indicator column in the output table.
dependents	specifies the dependent variables for the forecasting service.
lead	specifies the forecast horizon (or lead).
mode	specifies the mode for the forecasting service (AUTOMODEL, DYNAMIC, SIMPLE).
foreOut	specifies the output data table for the forecasts.
alpha	specifies the significance for forecast confidence intervals.
infoOut	specifies the output data table for the forecast information.
selectOut	specifies the output data table for the model selection information.
specOut	specifies the output data table for the generated model specifications.

Setting the mode argument to AUTOMODEL enables SAS to build different models and compare the fit. Here it built an ARIMAX and WINTERS model. The model fit statistics for the ARIMAX model are lower, and therefore, it was chosen during the model fitting process. The **info_out** table provides information about the selected model. It found the seasonality period of 12 months in the 264-observation training data set and also specifies the other ARMA parameters.

8. Download the forecast table and the original **airpass** data table to the client. Order both data sets by the **date** variable.

```
forecast_out = conn.CASTable(name='forecast_out')
df_forecast = forecast_out.to_frame()
df_forecast.sort_values(['Date'], inplace=True, ascending=True)

airpass = conn.CASTable(name=indata)
df_actual = airpass.to_frame()
df_actual.sort_values(['Date'], inplace=True, ascending=True)
```

9. Plot the original data and overlay the model on the training data to view the fit.

```
plt.figure(figsize=(8,8))
plt.plot(list(range(1,df.shape[0]+1)), df_actual['Passengers'], color='blue', linewidth=1, label='Observed')
plt.plot(list(range(1,len(df_forecast['PREDICT'][:264])+1)),
         df_forecast['PREDICT'][:264],
         color='red', linewidth=1, label='Model')
plt.xlabel('Month', fontsize=15)
plt.ylabel('Passengers', fontsize=15)
plt.ylim(25000,70000)
plt.axvline(13, color='black')
plt.axvline(264, color='black')
plt.legend(['Observed', 'Model'], fontsize=15, loc='lower right')
plt.show()
```

10. Plot the original data for the past five years of observations (that is, the data that was not used to train the model). Add the forecast to the plot and confidence limits to view the fit.

```
plt.figure(figsize=(8,8))
plt.plot(list(range(1,len(df_actual['Passengers'][264:])+1)), df_actual['Passengers'][264:],
         color='blue', linewidth=1, label='Observed')
plt.plot(list(range(1,len(df_forecast['PREDICT'][264:])+1)), df_forecast['PREDICT'][264:],
         color='red', linewidth=1, label='Predicted')
plt.plot(list(range(1,len(df_forecast['LOWER'][264:])+1)), df_forecast['LOWER'][264:],
         color='red',linewidth=1,linestyle='dashed', label='Confidence Limits')
plt.plot(list(range(1,len(df_forecast['UPPER'][264:])+1)), df_forecast['UPPER'][264:],
         color='red',linewidth=1,linestyle='dashed', label='Confidence Limits')
plt.title('5 Year Forecast', fontsize=15)
plt.xlabel('Month', fontsize=15)
plt.ylabel('Passengers', fontsize=15)
plt.ylim(15000,85000)
plt.legend(['Observed', 'Predicted','Confidence Limits'], fontsize=15, loc='lower left')
plt.show()
```

The ARIMAX model fits the data well for the first few years and then, as expected, the forecast degrades in the last few years. The confidence limits also expand farther out into the forecast period as the model uncertainty increases.

11. Print the absolute average error for the first year of the forecast and all five years of the forecast.

```
df_actual.reset_index(inplace=True)
df_forecast.reset_index(inplace=True)

print('Average absolute error for first year = ' +
      str(round(abs(df_actual['Passengers'][264:276]-
      df_forecast['PREDICT'][264:276]).mean(),0)))
print('Average absolute error for all five years = ' +
      str(round(abs(df_actual['Passengers'][264:]-
      df_forecast['PREDICT'][264:]).mean(),0)))
```

12. End the CAS session.

```
conn.session.endSession()
```