Aplicaciones de análisis

Práctica 3 - Predicción de Energías Renovables como Series Temporales

José Manuel Bustos Muñoz

Una serie de tiempo o serie temporal es una secuencia de datos, observaciones o valores, medidos en determinados momentos y ordenados cronológicamente. Los datos pueden estar espaciados a intervalos iguales o desiguales.

Lo primero que hacemos es cargar los datos y quedarnos con la columna que queremos analizar.

```
# cargamos los datos cogiendo la columna deseada de Wind Energy
df_eolica_completa = pd.read_csv('Eolica.csv', sep=',', header=0, usecols=[0,3], parse_dates=[0], dtype=np.float64,\
                     dayfirst=True, index_col=0)

print("Columnas:\n", list(df_eolica_completa.columns))
```

```
Columnas:
 ['Wind Energy']
```

Una vez cargados los datos, debemos comprobarlos. Pintamos los 10 primeros registros y los 10 últimos, así como el número de registros cargados. Es importante para la posterior manipulación de la ST que cada punto tenga asociado un índice que indique su ordenación temporal.

```
print(df_eolica_completa.head(10))
print("\n")
print(df_eolica_completa.tail(10))
print("\n")
print("num_rows: %d\tColumnas: %d\n" % (df_eolica_completa.shape[0], df_eolica_completa.shape[1]) )
```

```
            Wind Energy
Date
2016-01-01    280606.59
2016-01-02    181981.12
2016-01-03    329470.87
2016-01-04    216707.38
2016-01-05    204859.99
2016-01-06    278101.80
2016-01-07    247007.30
2016-01-08          NaN
2016-01-09          NaN
2016-01-10          NaN


            Wind Energy
Date
2018-12-22     95918.79
2018-12-23     43276.21
2018-12-24     13585.14
2018-12-25     18349.89
2018-12-26     21548.11
2018-12-27     10872.37
2018-12-28     18077.52
2018-12-29     85855.44
2018-12-30     63911.85
2018-12-31     31660.28


num_rows: 1096  Columnas: 1
```

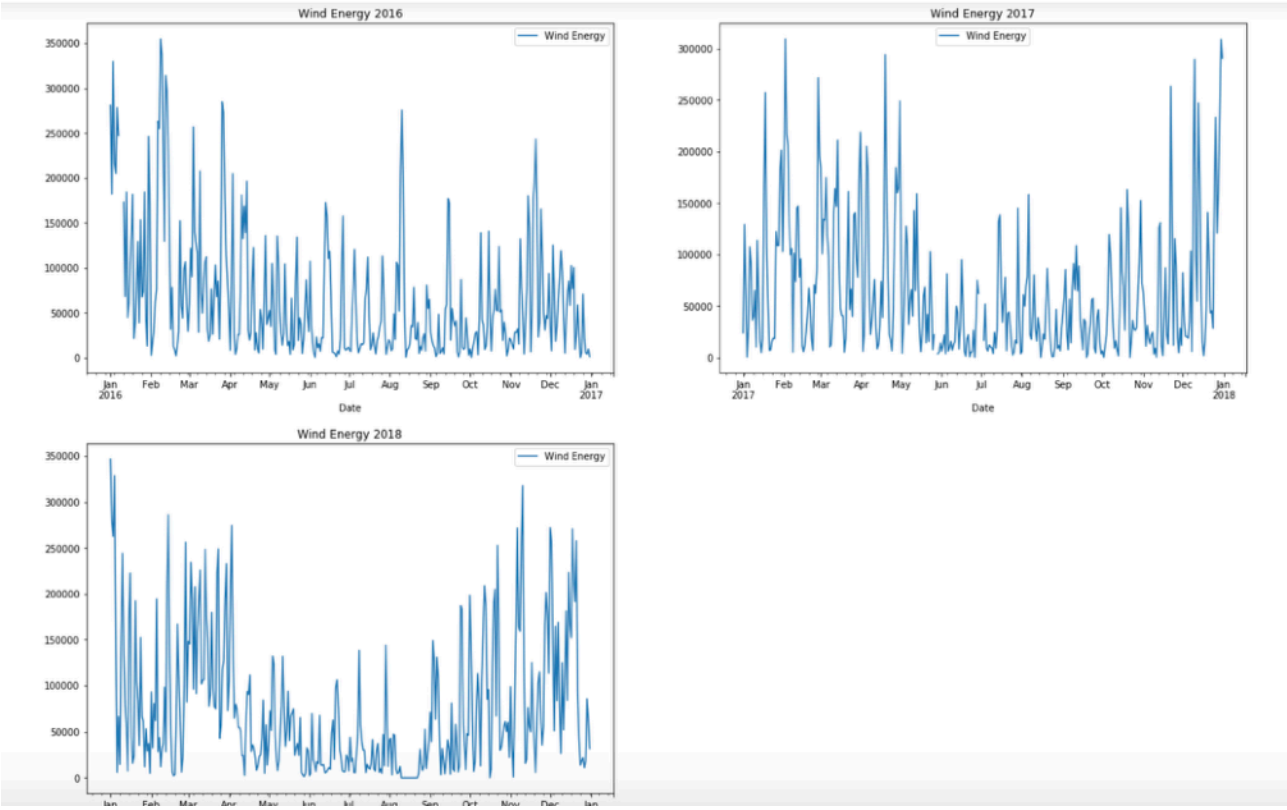
También podemos utilizar “describe” para ver algunos valores estadísticos de los datos cargados.

	count	mean	std	min	25%	50%	75%	max
Wind Energy	1088.0	68592.571756	70408.721701	0.0	15139.6	43866.7	100261.4475	354637.16

Ya en alguna visualización y viendo el count que arroja describe, podemos decir que hay valores nulos o vacíos que más tarde veremos como eliminar o sustituir.

Viendo la media obtenida, el mínimo y máximo, y además viendo los percentiles, podemos decir que la distribución de los datos no es uniforme. Se ve por dichos valores como hay mayor número de datos en valores bajos, y habría menos datos en los valores cercanos al valor máximo. El 75% de los datos están por debajo de '100.261', y el 25% restante iría desde ese valor hasta '354.637'.

Podemos visualizar la serie de datos de bastantes maneras, por ejemplo mostrando la gráfica por año. Pintamos tres gráficas, una para 2016, otra para 2017, y finalmente otra para 2018. También al pintar los datos así se aprecia algún hueco donde hay datos faltantes.



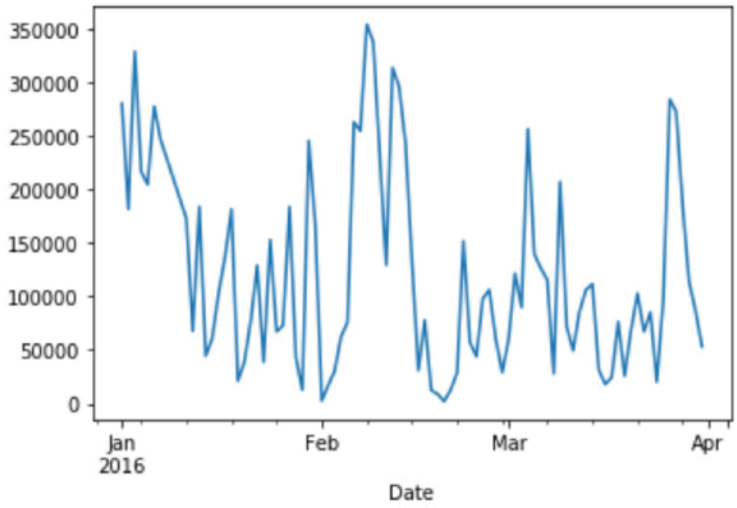
Se aprecia como las gráficas son similares año tras año, y viendo la forma y los valores que alcanza en cada mes, se aprecia que los primeros meses y los últimos de cada año tienen unos valores más altos para la energía, y que en los meses centrales de verano hay valores más bajos.

Como se han visto huecos en los datos, se van a interpolar usando un método existente y así rellenamos todos los huecos y tenemos los datos de la serie completos.

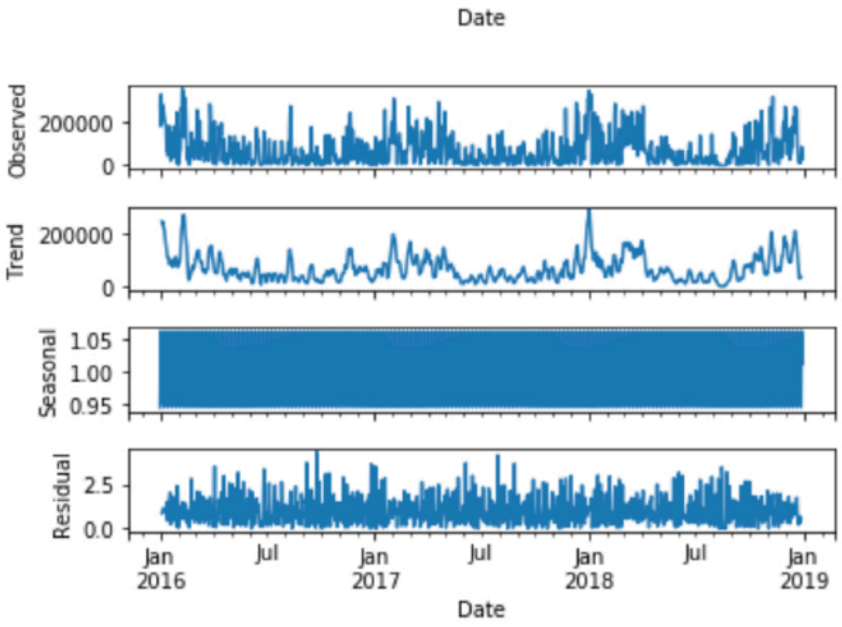
```
# como se han visto huecos, se usa el método interpolate para rellenar huecos.
df_eolica_int = df_eolica_completa.interpolate(method='linear', axis=0, limit=None, inplace=False,\
                        limit_direction='forward', limit_area=None, downcast=None)

print("num_rows: %d\tColumnas: %d\n" % (df_eolica_int.shape[0], df_eolica_int.shape[1]) )
```

```
num_rows: 1096  Columnas: 1
```

Tras el uso de interpolate volvemos a pintar un trozo de la serie donde había huecos y ya se aprecia que están todos los datos completos.

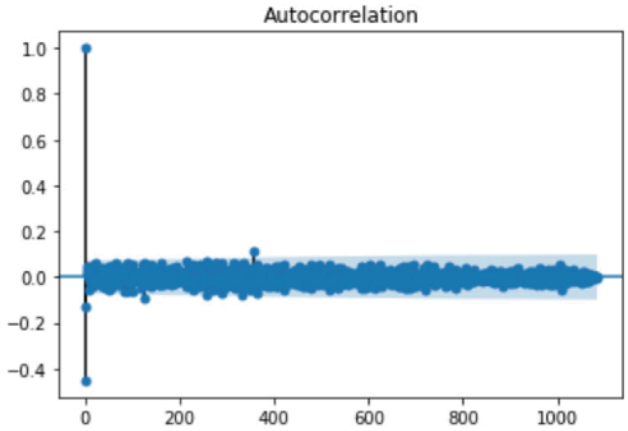


También podemos ver posibles tendencias y estacionalidad en la serie, pero viendo los gráficos que se obtienen parece que en estos datos no habría tendencia apreciable ni estacionalidad.

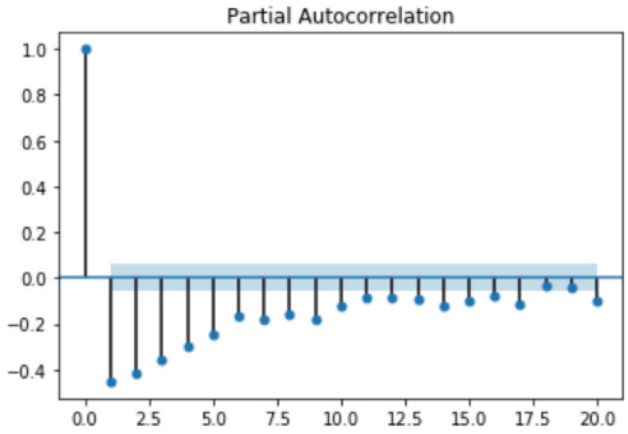


Como para aplicar modelos a las series temporales lo mejor son series estacionarias, aplicando el método “diff” la hacemos estacionaria, y la guardamos en un csv para utilizarla después con los modelos.

También pueden usarse gráficos de caja y similares para ver la distribución de los datos, o analizarse en las series las correlaciones y correlaciones parciales, generando gráficos como los de la imagen:




```
<Figure size 432x288 with 0 Axes>
```



Como dijimos anteriormente, ahora cargamos en otro notebook los datos almacenados anteriormente, tras analizar la serie y simular la estacionalidad. Para tener valores más manejables dividimos los datos entre la potencia instalada.

```
# cargamos los datos generados en el notebook de análisis
df_eolica_stacionary = pd.read_csv('series_stacionary.csv', sep=',', header=0, parse_dates=[0], dtype=np.float64,\
                       dayfirst=True, index_col=0)

potInstal = 17560
# Para tener valores más manejables dividimos los datos entre la potencia instalada del parque
df_eolica_stacionary = df_eolica_stacionary/potInstal

print("Columnas:\n", list(df_eolica_stacionary.columns))
```

```
Columnas:
 ['Wind Energy']
```

Con los datos cargados los dividimos en dos conjuntos, los dos primeros años en el conjunto de entrenamiento, y el tercer año para el conjunto de test o validación.

Viendo los conjuntos con “describe” se ve como más o menos los valores estadísticos son similares, o no muy distintos, lo que hace pensar que los conjuntos son relativamente homogéneos.

```
# Definimos la partición de train y test
series_train = df_eolica_stacionary['2016':'2017']
series_test = df_eolica_stacionary['2018']
```

```
series_train.describe()
```

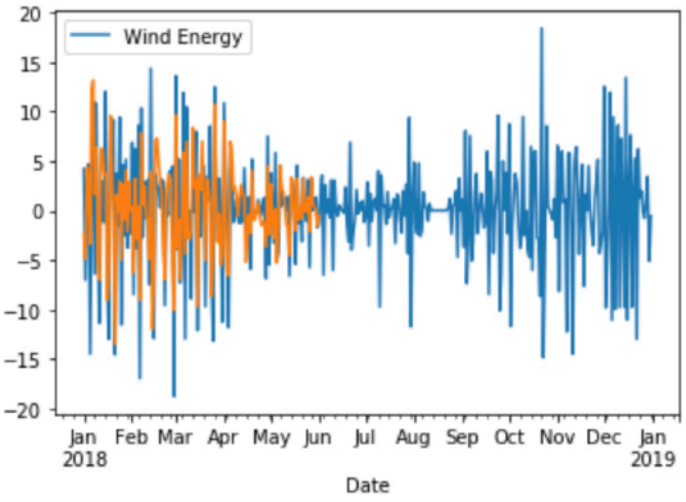
	Wind Energy
count	719.000000
mean	0.006880
std	5.013499
min	-18.342683
25%	-1.988804
50%	0.426924
75%	2.654971
max	16.375174

```
series_test.describe()
```

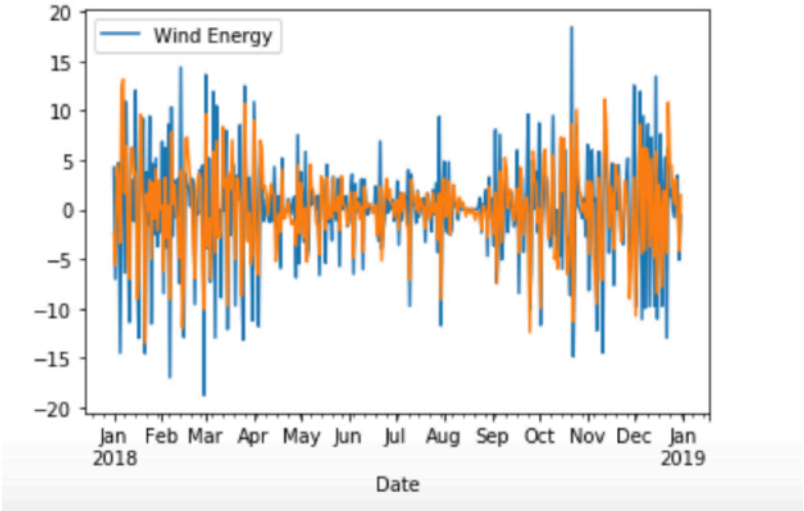
	Wind Energy
count	365.000000
mean	-0.002165
std	5.650685
min	-18.805856
25%	-2.539528
50%	0.114929

Hemos probado con la serie los siguientes tipos de modelos: AR, ARMA, ARIMA, SARIMA y de suavizado exponencial.

En el modelo **AR** se usan 2 horizontes, uno con 6 meses y otro con todo el año 2018 y se pintan los datos de test y las predicciones:




```
<Figure size 432x288 with 0 Axes>
```

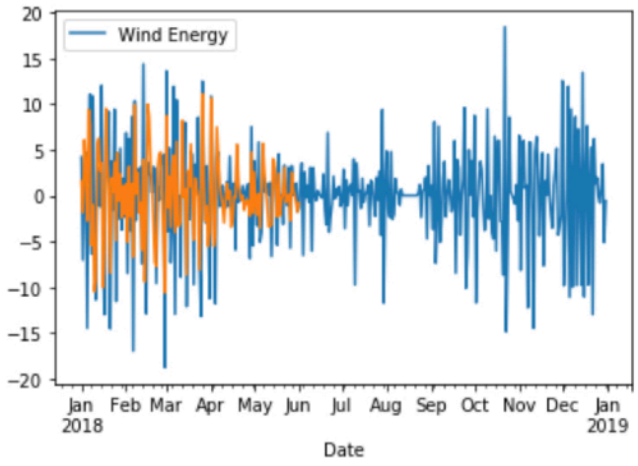


Los valores obtenidos de error RMSE para el modelo inicial, y tras aplicar ambos horizontes:

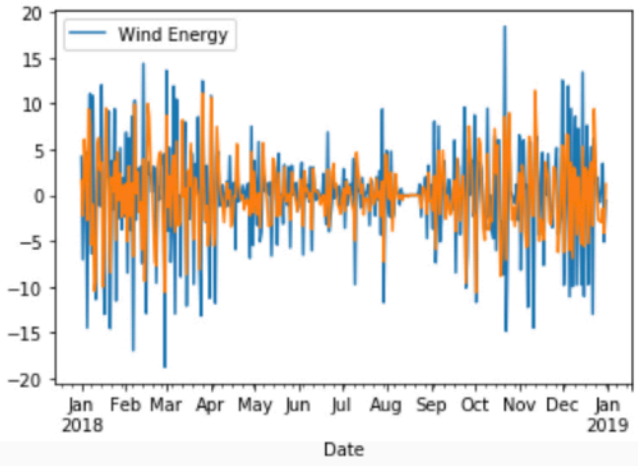
```
# ver valor error RMSE
mse_y = mean_squared_error(series_test, y_ar)
mse_yr = mean_squared_error(series_test['2018-01-01':'2018-06-01'], y_r)
mse_yr_2 = mean_squared_error(series_test, y_r_2)

print(mse_y);
print(mse_yr);
print(mse_yr_2);
```

```
31.719903291736742
15.59796699680534
12.310996071897664
```

En el modelo **ARMA** volvemos a aplicar los mismos horizontes anteriores:




```
<Figure size 432x288 with 0 Axes>
```



Con un bucle vamos viendo los valores de p y q si va mejorando el modelo o no. Se ha probado a ir aumentado p de 1 a 5, y se aprecia como a medida que crece se tienen mejores resultados, así que nos quedamos con p=5. Nos basamos en la minimización de la métrica AIC.

Valores del error RMSE obtenidos con ARMA y los distintos horizontes:

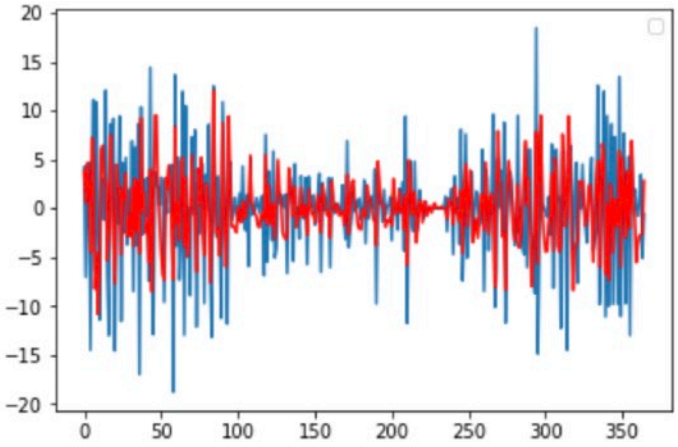
```
mse_y = mean_squared_error(series_test, y_arma)
mse_yr = mean_squared_error(series_test['2018-01-01':'2018-06-01'], y_r)
mse_yr_2 = mean_squared_error(series_test, y_r_2)

print(mse_y);
print(mse_yr);
print(mse_yr_2);
```

```
31.73794691279809
18.670325580036774
15.153813500506947
```

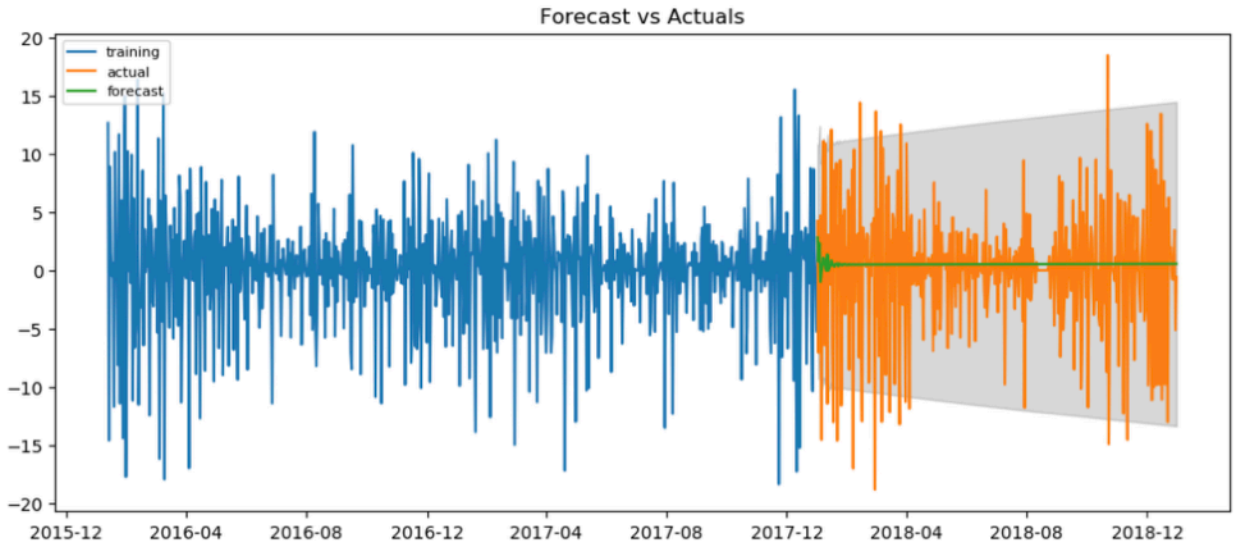
Con el modelo ARIMA realizamos el mismo ejercicio, de ir probando aumentando el valor del parámetro p para ver la minimización de AIC. Nos quedamos con la configuración ARIMA(9, 1, 0).

Se obtiene para el test el siguiente MSE de la imagen, y además pintamos los datos actuales y los de la predicción:

```
Test MSE: 19.130
```



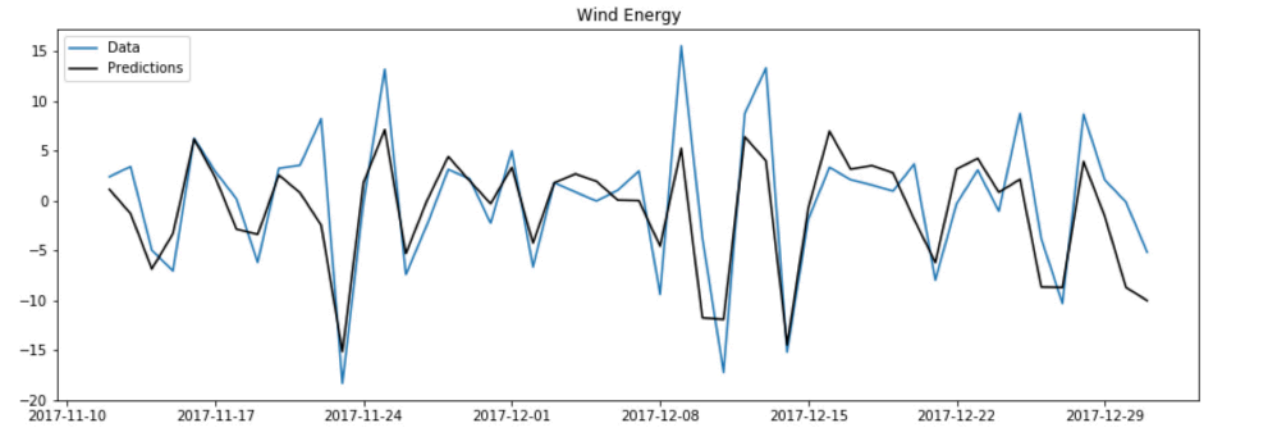
Además se usa Forecast para las predicciones y vemos otro tipo de gráfica donde se pintan los datos actuales o de test, los de entrenamiento, y el margen de las predicciones:

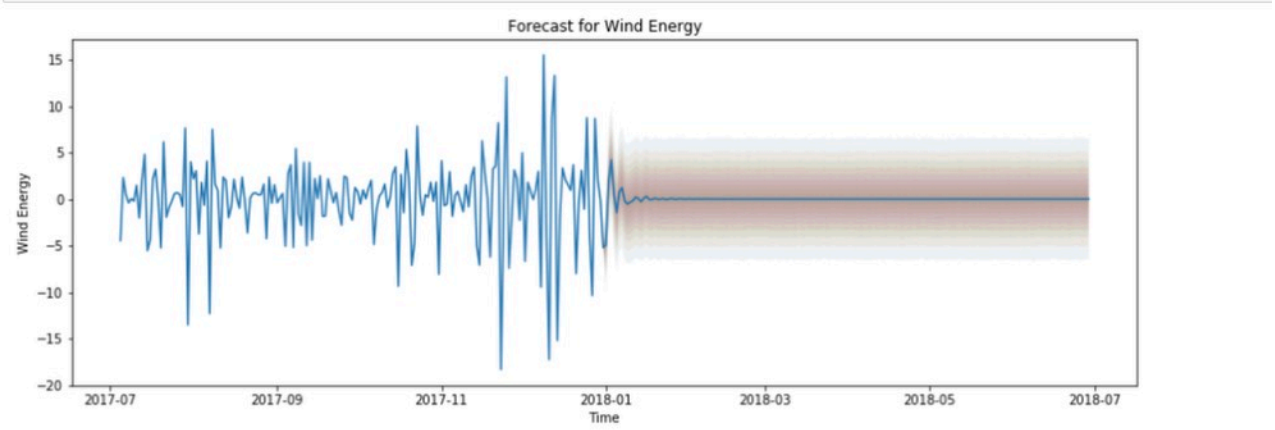


También se puede instalar y usar la librería “pyflux” y usarla para crear y entrenar el modelo ARIMA.

Tiene otros gráficos como por ejemplo los siguientes de datos de entrenamiento vs predicciones, y la introducción del forecast:

```
# pintamos prediciones
model_arima_pf.plot_predict_is(50, fit_once=False, figsize=(15,5))
```





También utilizamos modelos de suavizado exponencial, con el modelo ExponentialSmoothing, poniendo en la configuración “trend” y “seasonal” con el valor “add” y probando tanto con “damped” igual a False como a True.

Se entrena el modelo y se muestran los valores de errores MSE y RMSE, así como los parámetros:

```
# uso de ExponentialSmoothing con damped = True
model_damped = ExponentialSmoothing(series_train, trend='add', damped=True, seasonal='add',\
                                    seasonal_periods=12).fit()

mostrar_summary(model_damped);
```

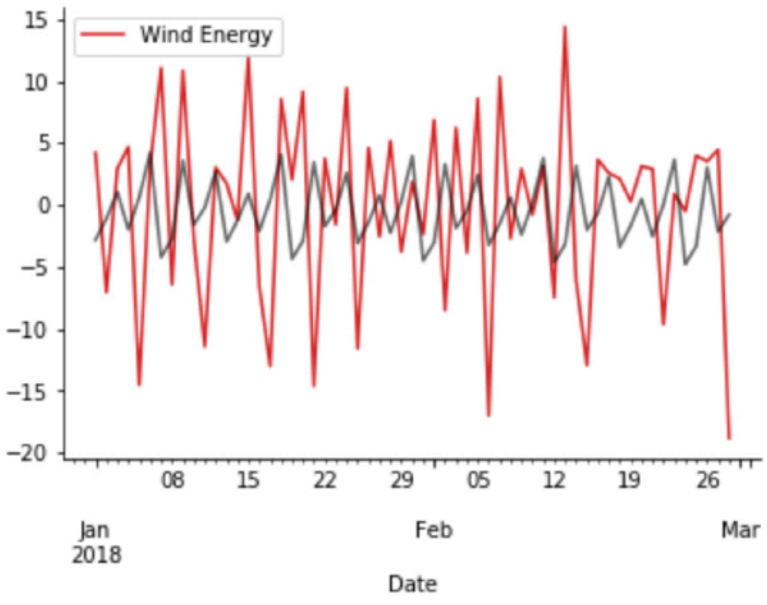
```
Smoothing parameter: alpha  0.053
Smoothing parameter: betha  0.053
Smoothing parameter: delta  0.211
In-sample fit:
MSE               31.086
RMSE               5.576
```

Después se predicen los datos para el año 2018, y con los datos de test se calcula el RMSE para las predicciones:

```
# predecir
y_damped = model_damped.predict(start='2018-01-01', end='2018-12-31')
print('\n')
print(mean_squared_error(series_test['2018-01':'2018-12'], y_damped))
```

```
40.13121026313839
```

Por último pintamos los valores actuales y las predicciones:



Por último usamos el modelo SARIMA, por ejemplo hacemos una primera prueba de modelo con los parámetros de la imagen:

```
# prueba modelo sarimax
model_fit_sarima = SARIMAX(series_train, order=(1,1,0), seasonal_order=(0,1,0,12), trend=None).fit()
model_fit_sarima.summary()
```

Statespace Model Results

Dep. Variable:	Wind Energy	**No. Observations:**	719
Model:	SARIMAX(1, 1, 0)x(0, 1, 0, 12)	**Log Likelihood**	-2606.030
Date:	Sun, 21 Apr 2019	**AIC**	5216.059
Time:	19:47:52	**BIC**	5225.179
Sample:	01-13-2016	**HQIC**	5219.583
	- 12-31-2017		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.5802	0.024	-23.701	0.000	-0.628	-0.53
sigma2	94.0727	3.810	24.690	0.000	86.605	101.540

Ljung-Box (Q):	556.98	**Jarque-Bera (JB):**	64.55
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.62	**Skew:**	0.12
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.46

STPrediccion_josemanuel

Usamos bucles for para recorrer con distintos valores de los parámetros, finalmente el menor valor de AIC que encontramos es con la siguiente configuración: order(9,0,9) seasonal_order(0,1,0,12).

```
# ahora probamos distintos valores del primer y último parámetro
for i in range(9,10):
    for j in range(6,10):
        model_sarima = SARIMAX(series_train, order=(i,0,j), seasonal_order=(0,1,0,12), trend=None).fit()
        print("SARIMA order(",i,",0,",j,"):", model_sarima.aic)
        print('\n');
```

```
SARIMA order( 9 ,0, 6 ): 3948.1179754536042


SARIMA order( 9 ,0, 7 ): 3921.045799973854


SARIMA order( 9 ,0, 8 ): 3930.6051965988436


SARIMA order( 9 ,0, 9 ): 3871.623473794375
```

STPrediccion_josema

Se predicen los datos para el año 2018, y se calcula el error MSE:

```
pred = model_sarimax.predict('2017-12-31','2018-12-31')[1:]
print('SARIMA model MSE:{}'.format(mean_squared_error(series_test,pred)))
```

```
SARIMA model MSE:41.26367007603677
```

```
plt.figure();series_test.plot(); pred.plot();
```

```
<Figure size 432x288 with 0 Axes>
```