Capítulo 1

Introducción a ciencia de datos

1.1. -

1.2. -

1.3. -

Correlación

Antes de revisar los métodos para las series de tiempo, veamos cómo puede determinarse la correlación entre variables aleatorias. Es importante mencionar que la correlación sólo explica relación lineal y no implica necesariamente causalidad.

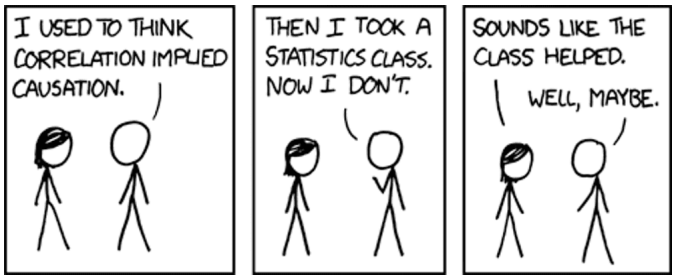


Figura 1.1: https://xkcd.com/552/

Los dos coeficientes más utilizados para determinar correlación son el de Pearson y el de Spearman; cada uno depende del tipo de variables con las que se está trabajando:

♢ Pearson:

- variables cuantitativas con distribución normal
- se calcula como:

$$r = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \sum (y - \bar{y})^2}}$$

De acuerdo al resultado, la correlación entre las variables x, y será:

Rango		Correlación
± 0.8	± 1.0	Muy buena
± 0.6	± 0.8	Buena
± 0.4	± 0.6	Moderada
± 0.2	± 0.4	Escasa
± 0.0	± 0.2	Nula

♢ Spearman:

- variables cuantitativas que no tienen distribución normal
- cuando se tienen valores atípicos (*outliers*)
- variables ordinales
- se calcula como:

$$\rho = 1 - \frac{6 \sum d^2}{n^3 - n}$$

Donde n es el total de valores en la muestra. Para calcular d se ordenan ambas variables de forma ascendente, anotando el índice que ocupa cada valor ordenado; finalmente d es la diferencia de dichos índices en el orden original; un ejemplo sencillo del cálculo de d:

x	y	$pos - x$	$pos - y$	d
46.8	31.7	5	2	3
43.2	30.6	1	1	0
44.1	38.9	2	5	-3
44.7	34.0	3	4	-1
45.4	32.4	4	3	1

x	pos
43.2	1
44.1	2
44.7	3
45.4	4
46.8	5

y	pos
30.6	1
31.7	2
32.4	3
34.0	4
38.9	5

Para verificar la normalidad de una variable aleatoria es posible utilizar alguna de las siguientes técnicas:

♢ realizar un histograma y ver si *parece normal*

♢ hacer una gráfica *Quantile-Quantile Normal*

♢ realizar un *contraste de normalidad* Shapiro-Wilks

♢ hacer un diagrama de dispersión por pares de variables (*PairPlot*)

Ejemplos con Python

Importando bibliotecas útiles:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Datos:

```
cont = pd.read_csv('https://bit.ly/31B56KB')
cont.info()
```

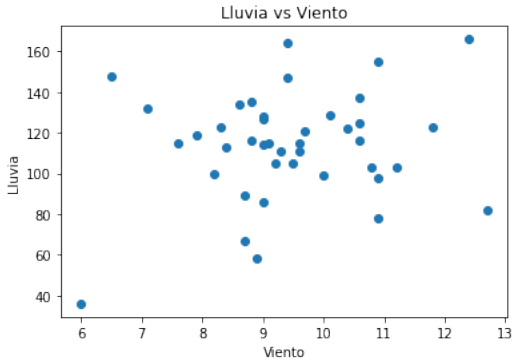
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41 entries, 0 to 40
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Contaminacion_SO2   41 non-null     int64
 1   Temperatura         41 non-null     float64
 2   Fabricas            41 non-null     int64
 3   Habitantes          41 non-null     int64
 4   Velocidad_viento    41 non-null     float64
 5   Lluvia              41 non-null     float64
 6   Dias_Lluvia         41 non-null     int64
dtypes: float64(3), int64(4)
memory usage: 2.4 KB
```

```
cont.head()
```

	Contaminacion_SO2	Temperatura	Fabricas	Habitantes
0	10	70.3	213	582	6.0
1	13	61.0	91	132	8.2
2	12	56.7	453	716	8.7
3	17	51.9	454	515	9.0
4	56	49.1	412	158	9.0

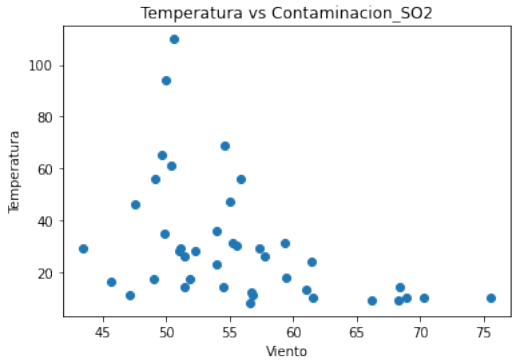
Se puede probar cómo se comportan pares de variables:

```
plt.scatter(cont.Velocidad_viento, cont.Dias_Lluvia)
plt.title('Lluvia vs Viento')
plt.xlabel('Viento')
plt.ylabel('Lluvia')
plt.show()
```



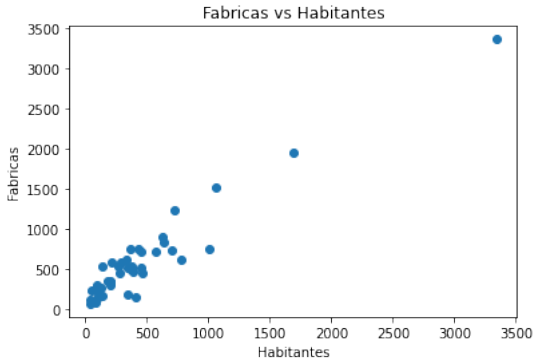
No parece mostrar evidencia de correlación lineal.

```
plt.scatter(cont.Temperatura, cont.Contaminacion_SO2)
plt.title('Temperatura vs Contaminacion_SO2')
plt.xlabel('Viento')
plt.ylabel('Temperatura')
plt.show()
```



Parece existir correlación.

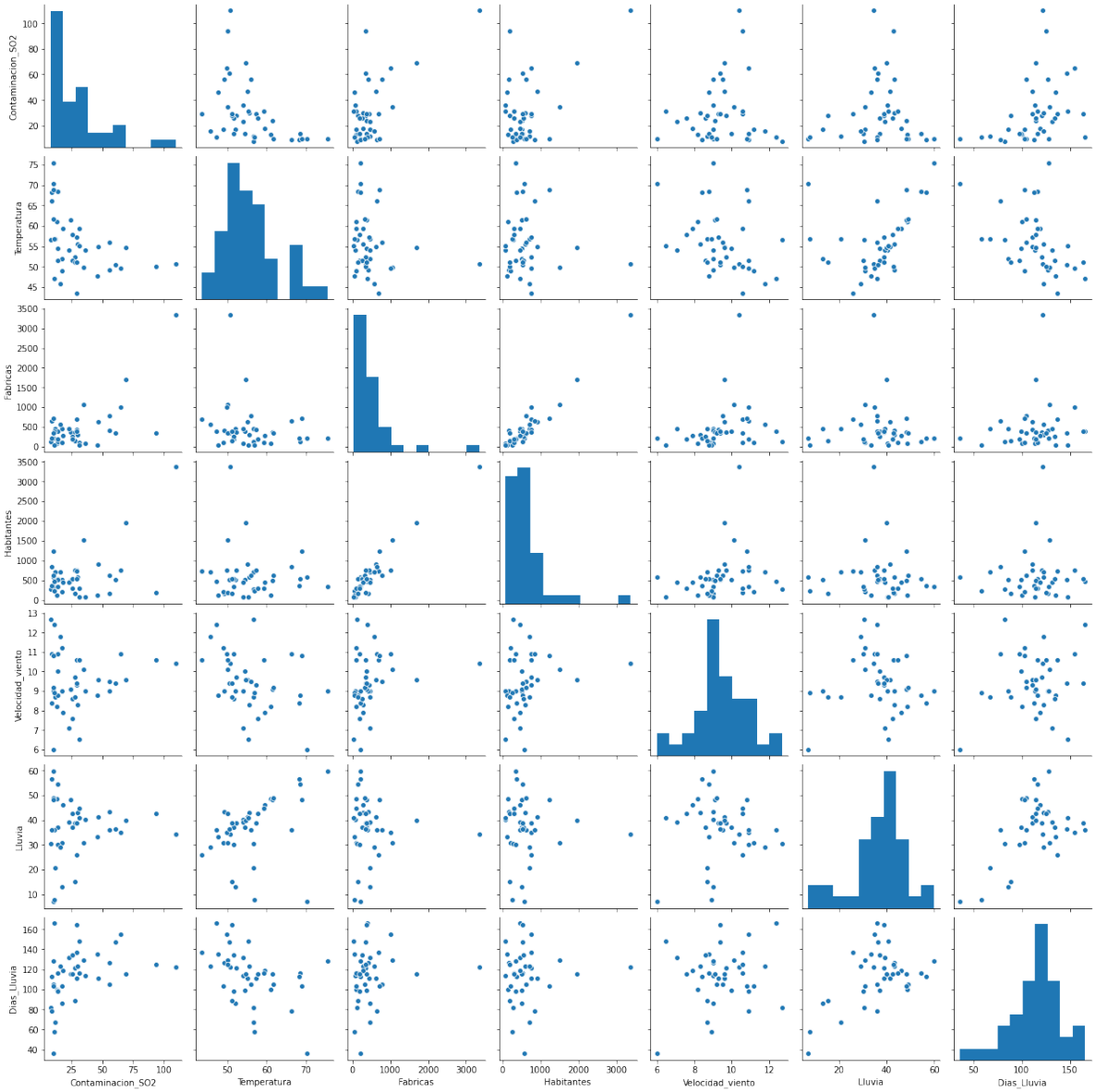
```
plt.scatter(cont.Fabricas, cont.Habitantes) # tiene 'outliers'
plt.title('Fabricas vs Habitantes')
plt.xlabel('Habitantes')
plt.ylabel('Fabricas')
plt.show()
```



Sin lugar a dudas existe correlación entre este par de variables, y contiene valores atípicos.

Para ver todas las variables juntos, podemos realizar una diagrama de dispersión por pares de variables:

```
sns.pairplot(cont)
```



En la diagonal, este diagrama muestra el histograma de cada variable, con esto podemos ver si tiene distribución normal.

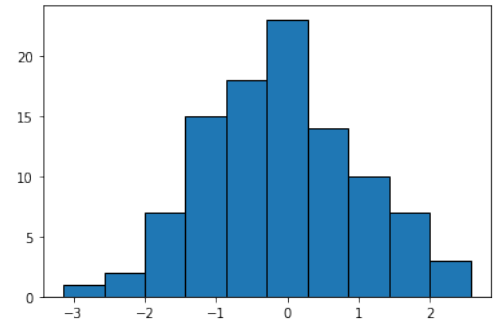
Prueba de normalidad

Para verificar si una variable tiene distribución normal, primero mostremos el resultado con una ficticia creada para este propçosito:

```
import numpy as np
data = np.random.normal(0,1,100)
```

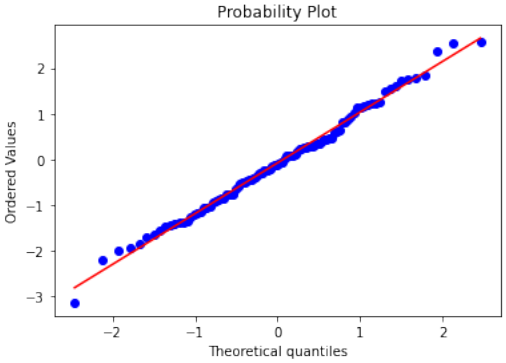
Veamos su histograma:

```
plt.hist(data, edgecolor='black', linewidth=1)
```



Y usemos una gráfica *Quantile-Quantile Normal* con *stats.probplot*:

```
import pylab
import scipy.stats as stats
stats.probplot(data, dist='norm', plot=pylab)
pylab.show()
```



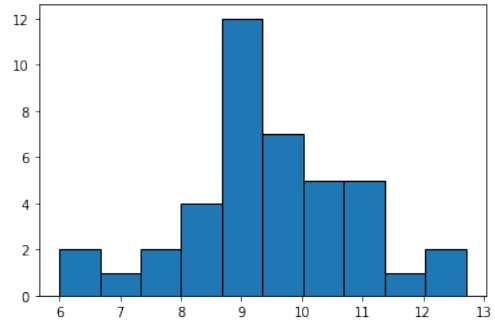
Esta gráfica indica que la distribución es normal si los puntos está distribuídos cerca de la recta. Además, podemos aplicar un contraste de normalidad con Shapiro:

```
from scipy.stats import shapiro
estad, p_value = shapiro(data)
print("Estadístico = %.3f, p_value=%.3f" % (estad, p_value))
# p_value > 0.05 => distribución normal
```

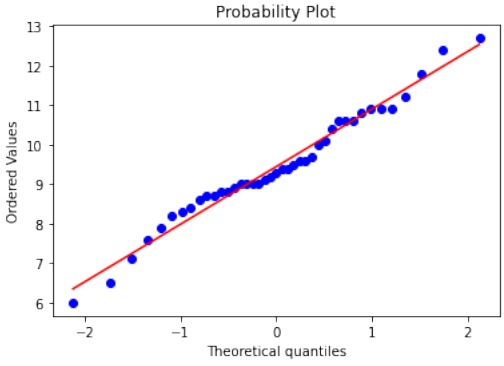
```
Estadístico = 0.993, p_value=0.887
```

Como el *p_ value* es mayor a 0.05, esta prueba indica que la variable tiene distribución normal. Regresando al conjunto de datos de contaminación:

```
plt.hist(cont.Velocidad_viento, edgecolor='black', linewidth=1)
```




```
import pylab
import scipy.stats as stats
stats.probplot(cont.Velocidad_viento, dist='norm', plot=pylab)
pylab.show()
```

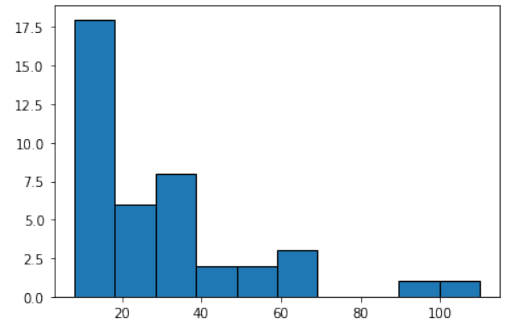



```
from scipy.stats import shapiro
estad, p_value = shapiro(cont.Velocidad_viento)
print("Estadístico = %.3f, p_value=%.3f" % (estad, p_value))
```

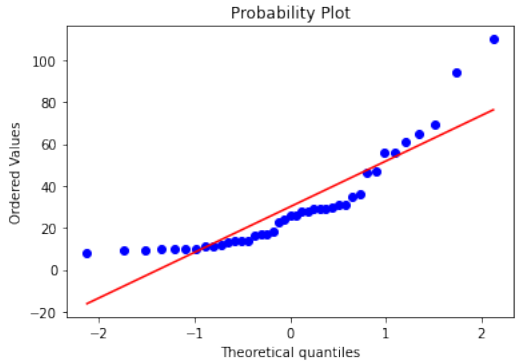
```
Estadístico = 0.981, p_value=0.697
```

Con estos resultados podemos concluir que la variable *Velocidad_ viento* tiene distribución normal.

```
plt.hist(cont.Contaminacion_SO2, edgecolor='black', linewidth=1)
```




```
import pylab
import scipy.stats as stats
stats.probplot(cont.Contaminacion_SO2, dist='norm', plot=pylab)
pylab.show()
```




```
from scipy.stats
import shapiro estad, p_value = shapiro(cont.Contaminacion_SO2)
print("Estadístico = %.3f, p_value=%.3f" % (estad, p_value))
```

```
Estadístico = 0.812, p_value=0.000
```

Con estos resultados podemos concluir que la variable *Contaminacion_ SO2* **no** tiene distribución normal.

Correlación y significancia

Ahora revisemos la correlación entre todos los pares de variables; como vimos anteriormente, no todas las variables tienen distribución normal, por tanto debemos utilizar el coeficiente de

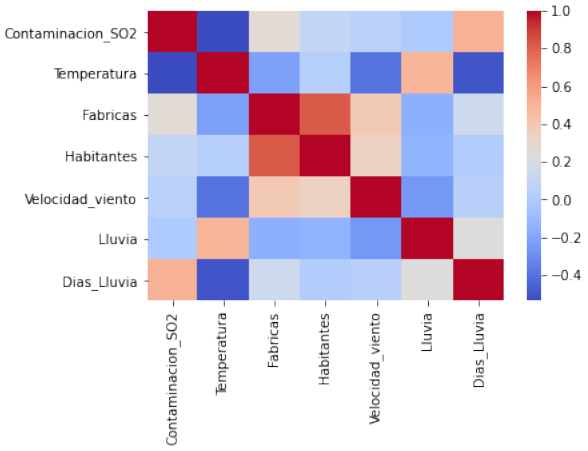
Spearman:

```
cont_corr = cont.corr(method='spearman') # Valor pequeño => baja correlación
cont_corr
```

	Contaminacion_SO2	Temperatura	Fabricas	Habitan
Contaminacion_SO2	1.000000	-0.538831	0.264051	0.08947
Fabricas	0.264051	-0.225629	1.000000	0.82299
Habitantes	0.089470	0.031362	0.822997	1.00000
Velocidad_viento	0.047309	-0.398282	0.394193	0.33743
Lluvia	-0.002616	0.498650	-0.151568	-0.1303
Dias_Lluvia	0.517709	-0.505730	0.145266	0.01010

El mayor valor se encuentra entre las variables *Fabricas* y *Habitantes*. Pero las correlaciones son mucho más fácilmente identificables en un diagrama de calor (*heatmap*):

```
sns.heatmap(cont_corr,
            xticklabels=cont_corr.columns,
            yticklabels=cont_corr.columns,
            cmap = 'coolwarm')
```



Con ayuda de este gráfico es más sencillo notar que la correlación entre fabricas y habitantes es la más intensa, pero también podemos notar otras como la existente entre *Temperatura* y *Contaminacion_SO2*.

Finalmente, es necesario verificar que las correlaciones no se deben simplemente al azar esto se realiza revisando la *significancia* de la correlación; para esto usaremos un módulo llamado *pingouin*:

```
!pip install pingouin # para instalarlo
```

Se realiza con el método *pairwise_ corr* (https://bit.ly/39ik21W):

```
import pingouin as pg
corr = pg.pairwise_corr(cont, method='spearman')
```

Ordenamos los valores de la relación de acuerdo a la significancia *p-unc* y seleccionamos las columnas deseadas:

```
corr.sort_values(by=['p-unc'])[['X','Y','n','r','p-unc']]
# p-unc > 0.05 => que posiblemente la correlación se debe al azar
```

	X	Y	r	p-unc
11	Fabricas	Habitantes	0.822997	4.020507e-11
0	Contaminacion_SO2	Temperatura	-0.538831	2.784226e-04
5	Contaminacion_SO2	Dias_Lluvia	0.517709	5.270914e-04
10	Temperatura	Dias_Lluvia	-0.505730	7.434618e-04
9	Temperatura	Lluvia	0.498650	9.057623e-04
8	Temperatura	Velocidad_viento	-0.398282	9.906609e-03
12	Fabricas	Velocidad_viento	0.394193	1.076594e-02
15	Habitantes	Velocidad_viento	0.337431	3.096515e-02
1	Contaminacion_SO2	Fabricas	0.264051	9.527256e-02
18	Velocidad_viento	Lluvia	-0.256605	1.053381e-01
20	Lluvia	Dias_Lluvia	0.242342	1.268559e-01
6	Temperatura	Fabricas	-0.225629	1.560681e-01
13	Fabricas	Lluvia	-0.151568	3.441610e-01
14	Fabricas	Dias_Lluvia	0.145266	3.648243e-01
16	Habitantes	Lluvia	-0.130314	4.167452e-01
2	Contaminacion_SO2	Habitantes	0.089470	5.780124e-01
3	Contaminacion_SO2	Velocidad_viento	0.047309	7.689686e-01
19	Velocidad_viento	Dias_Lluvia	0.034454	8.306599e-01
7	Temperatura	Habitantes	0.031362	8.456670e-01
17	Habitantes	Dias_Lluvia	0.010108	9.499849e-01
4	Contaminacion_SO2	Lluvia	-0.002616	9.870484e-01

Aquellos pares en los que el valor *p-unc* es mayor a 0.05 tiene alta probabilidad de deberse al azar.

Tarea:

♢ Escribir un código que obtenga los coeficientes de Pearson y de Spearman

♢ Realizar las mismas pruebas al conjunto de datos usando R

*

1.4. Series de tiempo y pronósticos

Los *pronósticos*, *previsiones* (*forecasting*) son importantes en diversas áreas, p.e. salud, negocios, climatología, etc. Por tanto, es una habilidad esencial de la ciencia de datos; los algoritmos de pronósticos no son tan distintas de las utilizadas en los análisis predictivos. Sin embargo existen aspectos relacionados con el tiempo que deben entenderse para poder explotar el potencial de las previsiones. *Predicción* es un término más general para obtener un valor no observado que puede o no tener un componente temporal. En términos generales, las funciones de pronósticos se construyen estimando el valor promedio de un proceso para un *tiempo futuro*; como la media estimada se convierte en el pronóstico, el objetivo central es estimar el valor esperado de la variable de interés para un momento en el futuro. Para el análisis, el tiempo presente es n, de tal forma que el valor estimado $\widehat{\mu}_{n+\tau}$ es la previsión para τ pasos en el futuro.

Los datos de interés se llaman *series de tiempo* porque el proceso que los genera posee un atributo cronológico; como resultado de esto, los datos observados se encuentran ordenados conológicamente, y este ordenamiento puede resultar útil para entender el proceso y realizar previsiones. Los algoritmos efectivos para extraer información de series de tiempo deben ser capaces de explotar el ordenamiento temporal siempre que las observaciones cercanas esten *autocorrelacionadas*[1] y no sean independientes; si existe autocorrelación, los datos observados más recientemente serán más útiles para hacer pronósticos y la función de predicción debería dar mayor peso a las observaciones recientes.

Se asume que los datos *están a la deriva* (*drift*); es decir, el valor de la media (y posiblemente el de la varianza) no son constantes sino cambian con el tiempo; y su cambio es predominantemente hacia valores mayores o menores, pero no de forma alternada. La dirección de la *tendencia* (*trend*) y el ritmo del cambio también puede variar.

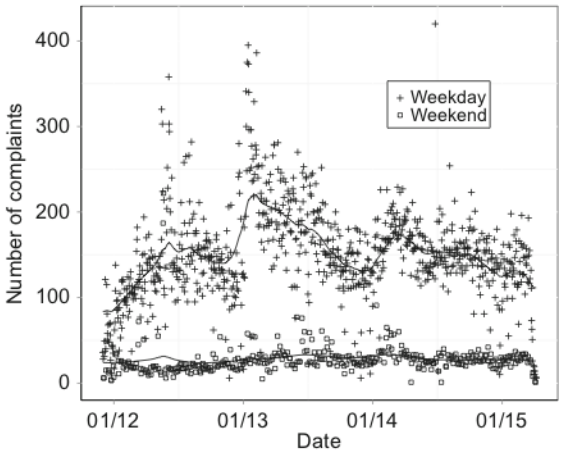


Figura 1.2: Número de quejas de hipotecas por fecha (*the U.S. Consumer Financial Protection Bureau*)

La figura 1.2 muestra un ejemplo de quejas de hipotecas y las fechas en las que se recibieron;

[1]la autocorrelación se refiere al grado de correlación entre los valores de las mismas variables entre diferentes observaciones de los datos

como se observa se realizan muchas más reclamaciones entre semana comparados con los fines de semana. Los fines de semana no se observan tendencias interesantes mientras que los demás días se tienen espigas seguidas con tendencias a la baja el resto del año.

La evidencia visual de tendencias entre semana implican que observaciones pasadas pueden ser usadas para predecir observaciones futuras y que los datos son *serialmente correlacionados* (autocorrelacionados); por tanto, pretender que los datos son independientes sería una afirmación indefendible y resultaría poco productivo analizar los datos como si no estuvieran relacionados.

Si existen tendencias en la serie de datos, la media de todas las observaciones puede resultar informativa sobre el proceso que genera la serie; si el objetivo es predecir observaciones futuras conforme se obtienen más datos, entonces un enfoque sensible es actualizar el valor incorporando los datos nuevos, de esta forma la importancia de observaciones pasadas se reduce y las recientes dominarán la previsión.

1.4.1. Métodos analíticos

1.4.1.1. Notación

Por el momento, se considerarán datos univariados; los n datos ordenados observados se denotan como:

$$D_n = (y_1, \ldots, y_t, \ldots, y_n)$$

donde y_t es una instancia de la variable aleatoria Y_t; los índices ordenan cronológicamente las observaciones, de modo que y_t se obtuvo después de y_{t-1} para $1 \leq t \leq n$; no se asume que los tiempos de llegada son constantes.

1.4.1.2. Estimación de la media y la varianza

Si los datos fuesen generados por un proceso estacionario (i.e. no a la deriva), entonces se utilizarían todas las observaciones para estimar μ, σ^2 y cualquier otro parámetro relevante; la única preocupación sería el almacenamiento de los datos que se siguen generando. Sin embargo, almacenar gran cantidad de observaciones no es práctico ni necesario, dado que la estadística asociativa se puede actualizar al momento que lleguen nuevos datos utilizando los métodos vistos anteriormente.

Sin embargo, si los datos no poseen media constante y están a la deriva, las observaciones antiguas deben omitirse o tener menor peso al estimar la media, dado que contienen menos información acerca del nivel actual. Un enfoque elemental es utilizar una *ventana móvil* para realizar estimaciones. Una ventana es un conjunto compuesto por las m observaciones más recientes; esta ventana se mueve con cada observación nueva y se elimina a la más vieja en el conjunto; por lo tanto, se tiene un número constante de observaciones dentro de la ventana. Una estimación de μ_n obtenida de una ventana móvil se conoce como *promedio móvil* (*moving average*) y se calcula como:

$$\widehat{\mu}_n^{MA} = m^{-1} \sum_{t=n-m+1}^{n} y_t$$

Existen limitaciones para la ventana móvil; si m es muy pequeña, la media será sensible a desviaciones aleatorias: como herramienta de previsión será imprecisa. Si m es muy grande, algunos cambios en la media podrían no detectarse hasta que hayan pasado muchos pasis temporales. El

promedio simple de las m observaciones trata cada obervación como igual, en lugar de asignar mayor peso a las observaciones recientes; una premisa importante es que el valor informativo de y_t decrece conforme n aumenta; por tanto, el sistema de dos pesos ($1/m$ ó 0) no es muy recomendable.

1.4.1.3. Autocorrelación

Los algoritmos para extraer información útil de series de tiempo deben explotar la tendencia en las observaciones que estén serialmente correlacionadas; el término estadístico para este tipo de observaciones es dependencia; la media con pesos exponenciales (más abajo) es un ejemplo de uso de la correlación serial. Sin embargo, no se obtienen ganancias al hacer previsiones si las observaciones no son dependientes; incluso habrá pérdida de precisión, dado que las observaciones antiguas se pierden en el cálculo del estimador. Por tanto, es deseable poder asegurar cierto grado de correlación en las series de datos ordenados $(y_1, \ldots, y_n)$.

La autocorrelación puede cuantificarse por un conjunto de parámetros poblacionales $\rho_0, \ldots, \rho_\tau$ donde τ es un entero positivo; el parámetro ρ_τ se conoce como el *coeficiente de correlación de retraso−τ* (*lag−τ*); mide el grado de asociación lineal entre un conjunto de observaciones que se dan después de otro conjunto τ pasos después; una definición es:

$$\rho_\tau = \frac{E\left[\left(Y_t - \mu\right)\left(Y_{t-\tau} - \mu\right)\right]}{\sigma^2}$$

donde $\sigma^2 = E\left[\left(Y_t - \mu\right)^2\right]$ es la varianza de Y_t. Lo anterior supone que el proceso es estacionario dado que sólo hay una media poblacional μ y una varianza σ^2 en lugar de una seria de ellas; además, nótese que se implica que $\rho_0 = 1$.

Un estimador para el coeficiente de correlación de retraso−τ es:

$$\widehat{\rho}_\tau = \frac{\sum_{t=1}^{n-\tau}\left(y_t - \widehat{\mu}\right)\left(y_{t+\tau} - \widehat{\mu}\right)}{\left(n - \tau\right)\sigma^2}$$

$\widehat{\rho}_\tau$ proporciona buenas estimaciones de ρ_τ si el nivel de la media y la varianza son constantes; i.e. el proceso es constante y no está a la deriva. Este estimador es el coeficiente de correlación de Pearson (Karl Pearson https://bit.ly/2mbbf8U) calculado para los $n - \tau$ pares de datos

$$\left\{\left(y_1, y_{\tau+1}\right), \left(y_2, y_{\tau+2}\right), \ldots, \left(y_{n-\tau}, y_n\right)\right\}$$

Si el conjunto de estimaciones $(\rho_1, \ldots, \rho_\tau)$ que son mayormente grandes sugieren que el proceso es predecible, en el sentido de que los valores recientemente observados serán muy similares en magnitud a los valores futuros siempre que los valores futuros no estén muy lejos de los más recientes. Un proceso a la deriva presentará autocorrelación, debido a que cuando y_t es substancialmente mayor (o menor) que μ, entonces las observaciones cercanas son también probablemente mayores (menores) que la media y más parecidas que una selección aleatoria de observaciones. Típicamente, los coeficientes de autocorrelación disminuyen lentamente hacia cero conforme τ tiende a valores mayores.

La figura 1.3 muestra dos conjuntos de coeficientes de autocorrelación calculados para el número de quejas de hipotecas; nótese que los coeficientes de autocorrelación calculados tanto para los días entre y fin de semana revelan autocorrelación persistente. Es también notorio que los valores mayores para $\widehat{\rho}$ están asociados con los retrasos de $5, 10, 15$ y 20 en los días de la semana; esto

implica que el número de quejas observadas en un día particular (p.e. lunes) son más parecidos a las recibidas en otro día de la semana. Por tanto, se puede concluir que el número de quejas depende del día de la semana.

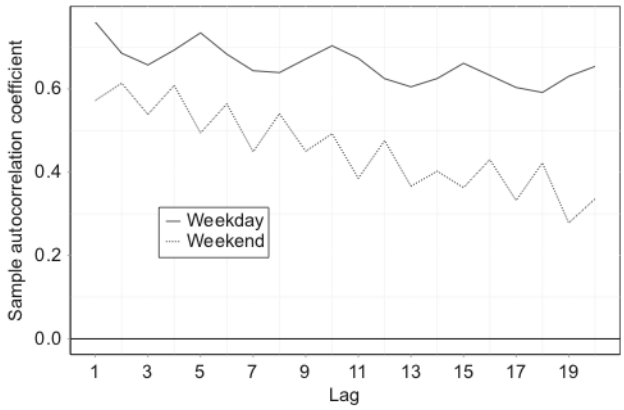


Figura 1.3: Estimaciones de los coeficientes de autocorrelación para diferentes retrasos (*lags*) del número de quejas (*the U.S. Consumer Financial Protection Bureau*)

Ejemplo con Python, media móvil:

Provee múltiples funciones para trabajar con series de tiempo; bibliotecas auxiliares:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
# registrar el convertidor datetime para la gráficas
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
# estadística
import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt
import statsmodels.api as sm
import scipy.stats as scs
```

Datos:

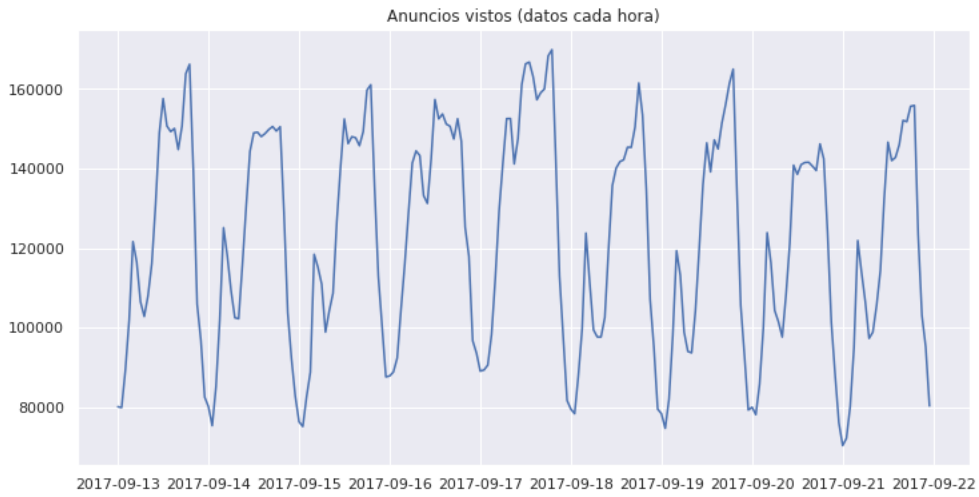
♢ *index_ col* indica explícitamente que "*Time*" sirve como índice del *df*

♢ *parse_ dates* convierte a tipo *date_ time* la columna

```
# http://dicyg.fi-c.unam.mx:8080/lalo/pypcd/presentaciones/ads.csv
ads = pd.read_csv('https://bit.ly/2l9BdhA',
                  index_col=['Time'], parse_dates=['Time'])
print(ads.head())
# http://dicyg.fi-c.unam.mx:8080/lalo/pypcd/presentaciones/currency.csv
cur = pd.read_csv('https://bit.ly/2lLao3B',
                  index_col=['Time'], parse_dates=['Time'])
print(cur.head())
```

Graficar los anuncios (*ads*):

```
plt.figure(figsize=(12, 6))
plt.plot(ads.Ads)
plt.title('Anuncios vistos (datos cada hora)')
plt.grid(True)
plt.show()
```

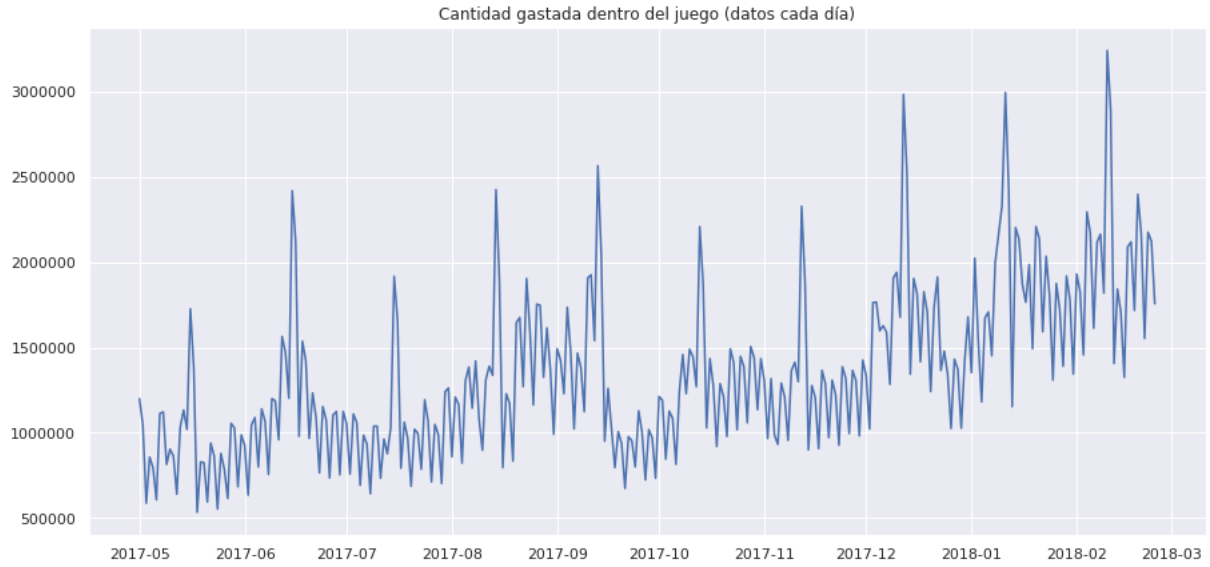


Autocorrelación (*ads*) buena explicación de la interpretación en https://bit.ly/2ZRw6UU:

```
# Autocorrelation Function (acf)
from statsmodels.graphics.tsaplots import plot_acf
fig = plot_acf(ads.Ads, lags=range(1,30), alpha=0.05)
fig = plot_acf(ads.Ads, lags=range(1,75), alpha=0.05)
```

Cantidades gastadas (*cur*):

```
plt.figure(figsize=(15, 7))
plt.plot(cur.GEMS_GEMS_SPENT)
plt.title('Cantidad gastada dentro del juego (datos cada día)')
plt.grid(True)
plt.show()
```



Autocorrelación (*cur*):

```
from statsmodels.graphics.tsaplots import plot_acf
fig = plot_acf(cur.GEMS_GEMS_SPENT, lags=range(1,30), alpha=0.05)
fig = plot_acf(cur.GEMS_GEMS_SPENT, lags=range(1,50), alpha=0.05)
```

Métricas:

```
# from sklearn.metrics import mean_absolute_error

def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

Media móvil:

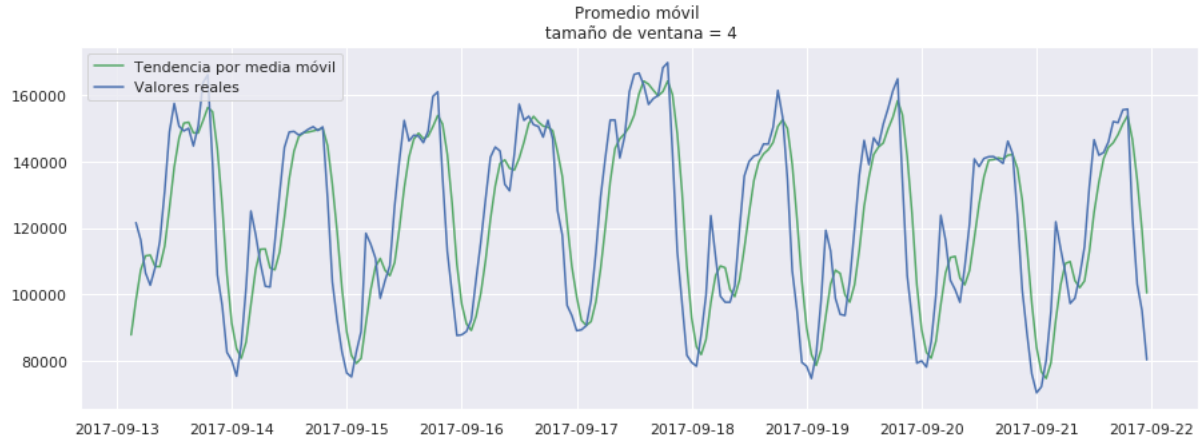
```
def moving_average(series, n):
    return np.average(series[-n:])

# predicción basada en el último día (24h)
moving_average(ads, 24)

# Pandas tiene una implementación:
#   https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rolling.html
#   https://pandas.pydata.org/docs/reference/api/pandas.Series.rolling.html
```

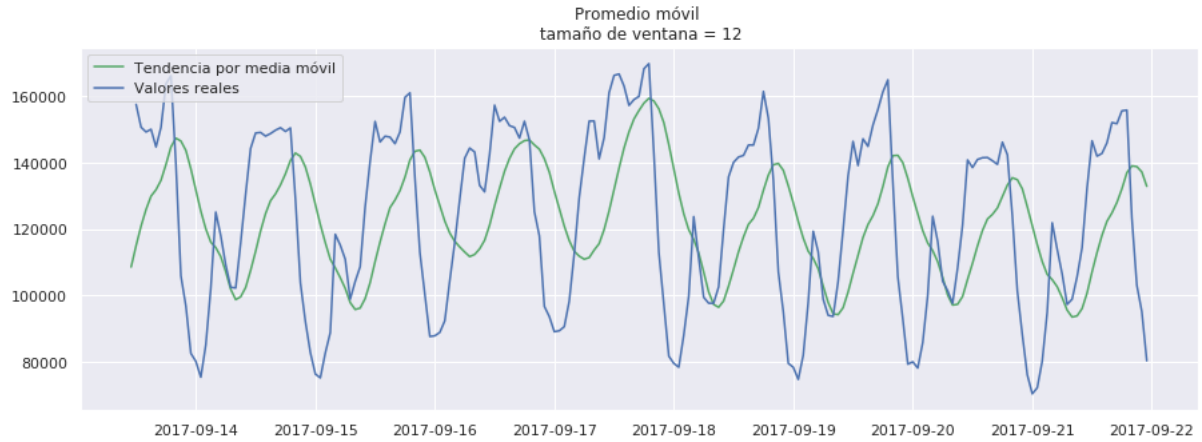
Función auxiliar para graficar:

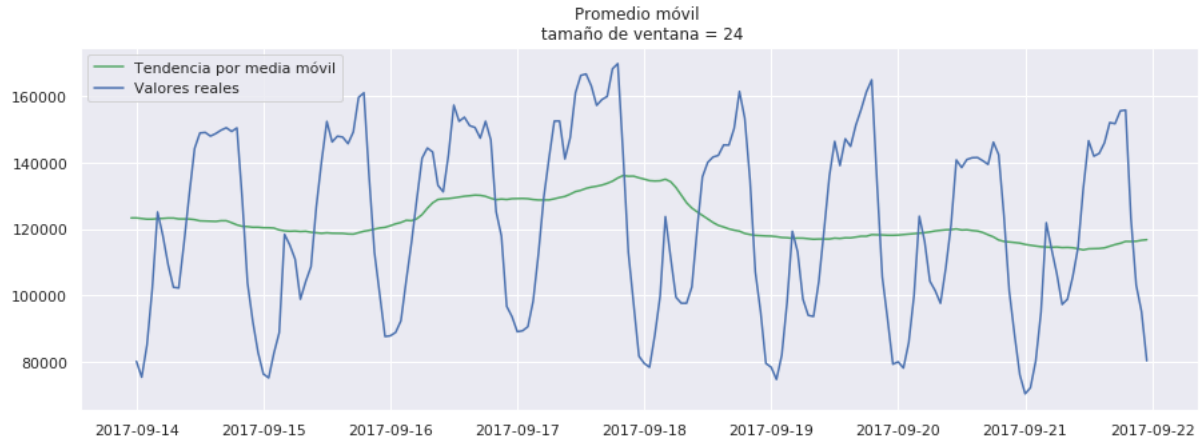
```
def plotMovingAverage(series, window, plot_intervals=False,
                      scale=1.96, plot_anomalies=False):
  """ series - dataframe con la serie de tiempo
      window - tamaño de la ventana
      plot_intervals - mostrar intervalos de confianza
      plot_anomalies - mostrar anolmalías  """
  rolling_mean = series.rolling(window=window).mean() # Pandas
  plt.figure(figsize=(15,5))
  plt.title("Promedio móvil \n tamaño de ventana = {}".format(window))
  plt.plot(rolling_mean, "g", label="Tendencia por media móvil")
  # Intervalos de confianza para los valores suavizados
  if plot_intervals:
    mae = mean_absolute_error(series[window:], rolling_mean[window:])
    deviation = np.std(series[window:] - rolling_mean[window:])
    lower_bond = rolling_mean - (mae + scale * deviation)
    upper_bond = rolling_mean + (mae + scale * deviation)
    plt.plot(upper_bond, "r--", label="Límites superior / inferior")
    plt.plot(lower_bond, "r--")
    # Buscar valores anormales Having the intervals, find abnormal values
    if plot_anomalies:
      anomalies = pd.DataFrame(index=series.index, columns=series.columns)
      anomalies[series<lower_bond] = series[series<lower_bond]
      anomalies[series>upper_bond] = series[series>upper_bond]
      plt.plot(anomalies, "ro", markersize=10)
  plt.plot(series[window:], label="Valores reales")
  plt.legend(loc="upper left")
  plt.grid(True)
```

Graficando para 4, 12 y 24 horas (la última es por día):

```
plotMovingAverage(ads, 4)
plotMovingAverage(ads, 12)
plotMovingAverage(ads, 24)
```

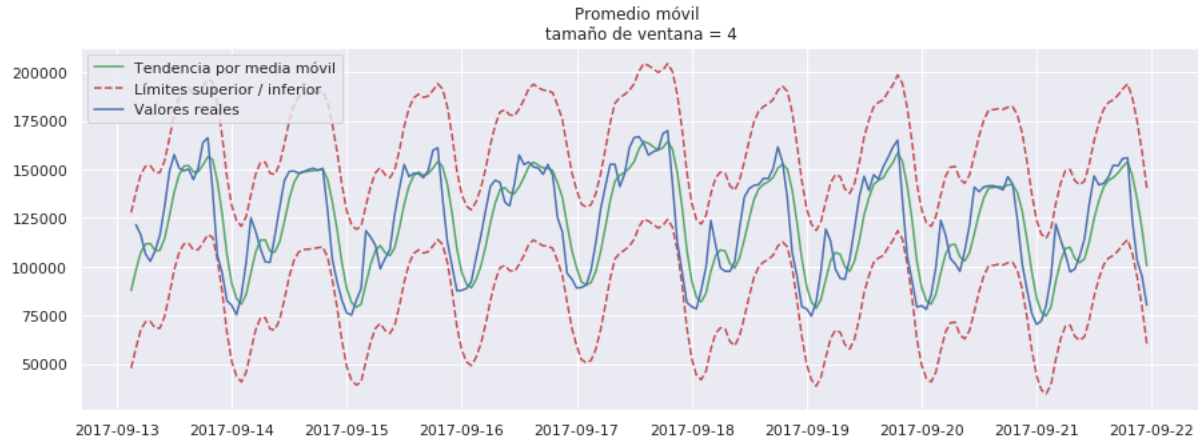






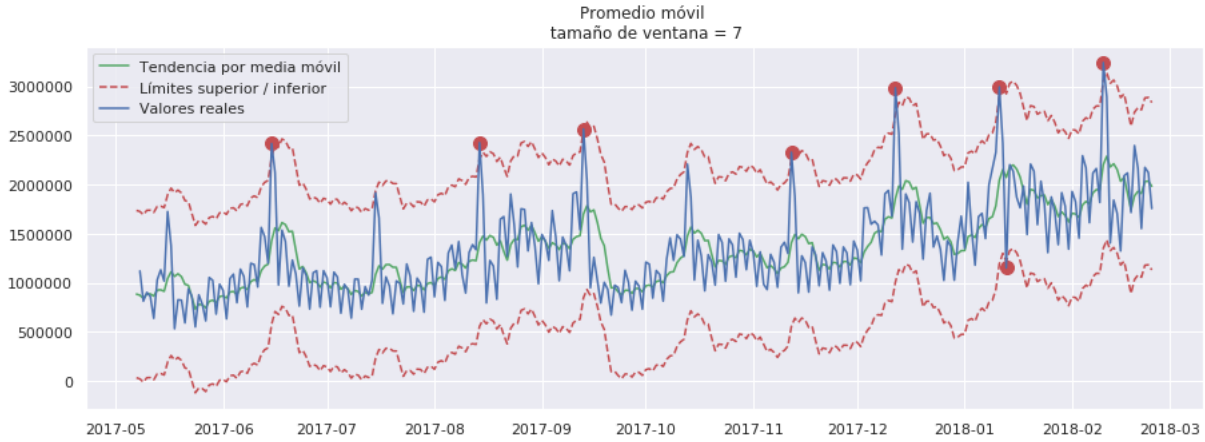
Agregando intervalos de confianza:

```
plotMovingAverage(ads, 4, plot_intervals=True)
```



Mostrando las anomalías en el conjunto de gastos, varios caen fuera, se debe al modelo tan simple, es mejor ver con los exponenciales:

```
plotMovingAverage(cur, 7, plot_intervals=True, plot_anomalies=True)
```



1.4.1.4. Previsiones exponenciales

Es posible utilizar todo el conjunto de observaciones $y_n, \ldots, y_n$ y asignar a cada valor un peso que dependa del paso temporal en el que se presentó; para esto se puede utilizar la idea de los pesos exponenciales vista en la función de predicción de $k-$vecinos más proximos. El estimador de media con pesos exponenciales de $E\left(Y_n|y_1, \ldots, y_n\right)$ es:

$$\widehat{\mu}_n = \sum_{t=1}^{n} w_t y_t$$

donde $0 \leq w_t \leq 1$ y $\sum_{t=1}^{n} w_t = 1$; los pesos se definen de acuerdo con:

$$w_t = \alpha\,(1-\alpha)^{n-t}, t \in \{1,\ldots,n\}$$

como $0 < \alpha < 1$, los pesos tienden a α conforme t incrementa su valor de 1 a n; si se presenta el caso en que $\sum_{t=1}^{n} w_t < 1$, entonces deben utilizarse pesos escalados: $v_t = \dfrac{w_t}{\sum_{t=1}^{n} w_t}, t = 1,\ldots,n$.

El efecto de los valores de α se observan en la figura 1.4: valores de α mayores a 0.25 producen estimadores que reflejan el comportamiento de las observaciones más recientes dado que los pesos disminuyen rapidamente cuando t retrocede de n a 0.; por el contrario, valores menores a 0.05 producen estimadores que reflejan valores de un conjunto más grande y antiguo de observaciones.

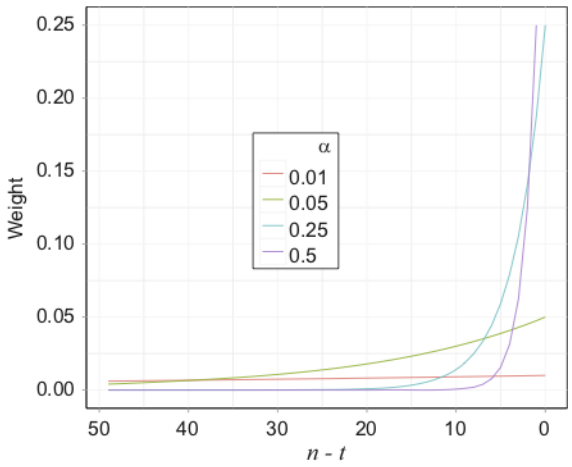


Figura 1.4: Pesos exponenciales para diferentes valores de α

Si reemplazamos w_t con $\alpha\,(1-\alpha)^{n-t}$ en el cálculo del estimador $\widehat{\mu}_n$, se observa que es una combinación lineal de la estimación en el paso anterior $\widehat{\mu}_{n-1}$ y la observación más reciente y_n:

$$\widehat{\mu}_n = (1-\alpha)^{n-t}\,\widehat{\mu}_{n-1} + \alpha y_n$$

En esta última forma es más clara la importancia en la elección de α; además, nos indica que el valor actual puede utilizar para obtener la nueva estimación y, por tanto, no se necesita almacenar los valores pasados. Otra ayuda para elegir el valor de α es determinar la autocorrelación de los datos de interés.

Los estimadores con pesos exponenciales son usados para representar visualmente la tendencia en series de datos; este proceso se conoce como *suavizado exponencial* (*exponencial smoothing*). Este estimador también puede usarse para prever el valor futuro de $Y_{n+\tau}$ donde τ es el número de pasos posteriores al $n-$ésimo paso; en este caso se utiliza el término *previsión exponencial* (*exponencial forecasting*).

Tarea: _____

♢ Verificar que $\widehat{\mu}_n = (1-\alpha)^{n-t}\,\widehat{\mu}_{n-1} + \alpha y_n$ se obtiene a partir de $\widehat{\mu}_n = \sum_{t=1}^{n} w_t y_t$

*

1.4.2. Previsiones exponenciales Holt-Winters

El método Holt-Winters (Charles C. Holt https://bit.ly/2kF2RTu y Peter R. Winters) es una extensión de los pesos ponderados exponencialmente que incluye un término adicional para la tendencia: $\widehat{y}_{n+\tau}$ actualiza el nivel de la media introduciendo un parámetro de ritmo de cambio β_n:

$$\widehat{\mu}_n = (1-\alpha_h)\left(\widehat{\mu}_{n-1} + \widehat{\beta}_n\right) + \alpha_h y_n$$

donde $0 < \alpha_h < 1$ es una constante de suavizado análoga a α para la previsión exponencial; la diferencia entre la previsión exponencial y la previsión Holt-Winters es la inclusión del coeficiente $\widehat{\beta}_n$, este valor es el ritmo de cambio de la media entre los tiempos $n-1$ y n; puede decirse que $\widehat{\beta}_n$ es la estimación de la variación en la pendiente en el tiempo. El pronóstico de $Y_{n+\tau}$ hecho en el tiempo n para τ pasos en el futuro, es:

$$\widehat{y}_n = \widehat{\mu}_n + \widehat{\beta}_n \tau$$

Este versión de la previsión tiene la forma de una regresión lineal $E(Y|x) = \beta_0 + \beta_1 x$; sin embargo, en la regresión la pendiente es fija mientras que el modelo de Holt-Winters permite que varíe en el tiempo.

Para determinar los valores de $\widehat{\mu}_1, \ldots, \widehat{\mu}_n$ se utiliza la expresión del inicio de la sección; ahora es necesario determinar una forma de calcular los coeficientes $\widehat{\beta}_1, \ldots, \widehat{\beta}_n$: una vez más, se utilizan pesos ponderados para el cálculo; en este caso, $\widehat{\beta}_n$ será una combinación lineal del valor estimado $\widehat{\beta}_{n-1}$ y $\widehat{\mu}_{n-1}$; se requiere otra constante de ajuste para controlar el ritmo de cambio en los coeficientes de la pendiente: $0 < \alpha_b < 1$, y la actualización de los coeficientes de la pendiente es:

$$\widehat{\beta}_n = (1-\alpha_b)\,\widehat{\beta}_{n-1} + \alpha_b\,(\widehat{\mu}_n - \widehat{\mu}_{n-1})$$

El último término incorpora el cambio observado en el nivel de la media $\widehat{\mu}_n - \widehat{\mu}_{n-1}$ que puede ser una estimación elemental para el ritmo de cambio actual en el valor de la media.

De esta forma, para cualquier tiempo n se puede obtener el valores de $\widehat{\mu}_n$ y posteriormente el de $\widehat{\beta}_n$; valores razonables para el paso inicial son $\widehat{\mu}_1 = y_1$ y $\widehat{\beta}_1 = 0$.

La figura 1.5 ilustra las previsiones exponencial y de Holt-Winters aplicadas a una serie de precios de productos de Apple hacia 2013. Las previsiones se calculan para $\tau = 20$ pasos en el futuro; para la previsión exponencial se utiliza $\alpha = 0.02$ mientras que para la de Holt-Winters $\alpha_h = 0.02$ y $\alpha_b = 0.015$. La previsión parece estar por delante del valor observado; sin embargo si se presta atención alrededor del paso $26,600$ el valor observado es mayor que el pronosticado, esto es debido a que se realizó el cálculo 20 pasos previos y el valor observado en ese momento era menor; si se utilizan valores mayores para las constantes α, las previsiones cambiarán más rápidamente junto con la serie de datos. sin embargo, cambios rápidos en los datos inducirían cambios muy grandes y comúnmente inadecuados en los pronósticos. También se observa que hay poca diferencia entre la previsión exponencial y la de Holt-Winters.

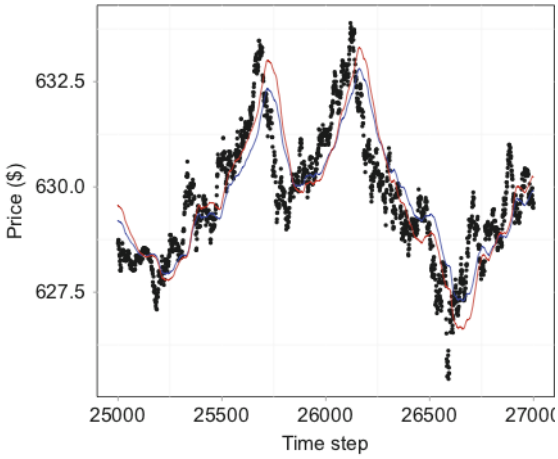


Figura 1.5: Previsiones obtenidas: exponencial en azul y Holt-Winters en rojo

1.4.2.1. Error en la previsión

El error en el pronóstico se estima comparando observaciones con las previsiones obtenidas; cuando se tiene una nueva observación, la diferencia entre la previsión y el valor real provee información para hacer una estimación del error. La precisión del pronóstico se ve afectada por el número de pasos en el futuro elegido y nuestro estimador define explícitamente el valor del error cuadrático medio como función de τ:

$$\mathrm{Err}\,(\tau) = \frac{\sum_{n=\tau+2}^{N} (y_n - \widehat{y}_n)^2}{N - \tau - 1}$$

el denominador es el número de términos de la sumatoria; en esta formulación, $\widehat{y}_n$ es el pronóstico de y_n realizado en el paso $n - \tau$, desde otra perspectiva, $\widehat{y}_{n+\tau}$ es un pronóstico de $y_{n+\tau}$ hecho en el paso n. Es muy probable que los primeros pronósticos sean poco precisos debido a que el contenido de información de pocas observaciones es muy poco. En ocasiones es mejor el error estándar ($\sqrt{\mathrm{Err}\,(\tau)}$) dado que su valor estará en las mismas unidades que las observaciones.

Ejemplo con Python, suavizado exponencial:

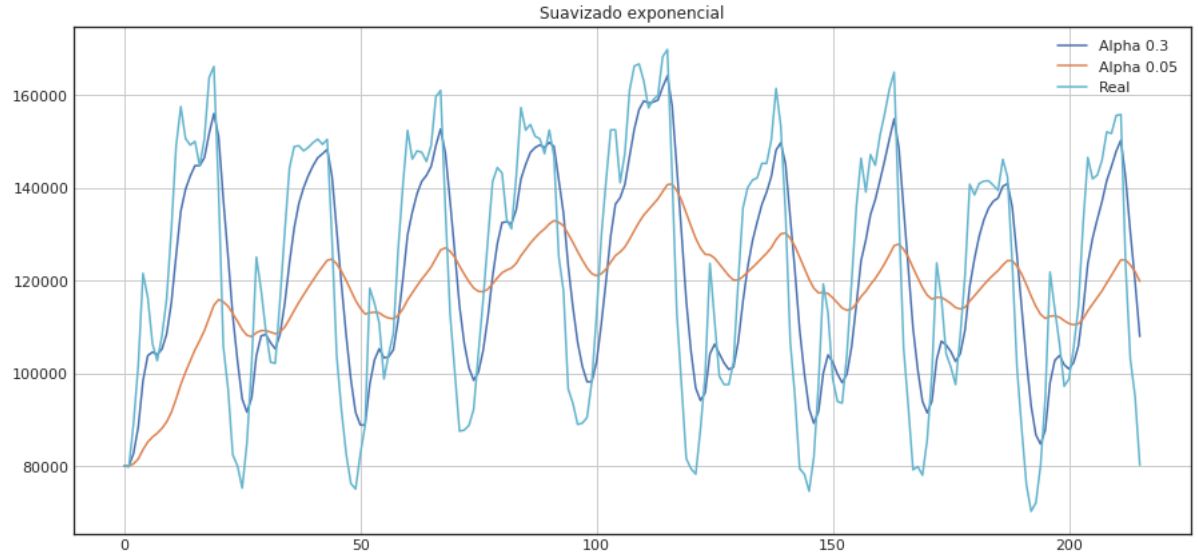
```
def exponential_smoothing(series, alpha):
  """ series - conjunto de datos con tiempos
      alpha  - parámetro de suavizado [0.0, 1.0] """
  result = [series[0]] # el primer valor es el mismo de la serie
  for n in range(1, len(series)):
      result.append(alpha * series[n] + (1 - alpha) * result[n-1])
  return result

# Dentro de Python se encuentra en:
#   from statsmodels.tsa.api import SimpleExpSmoothing
```

Función auxiliar para graficar:

```
def plotExponentialSmoothing(series, alphas):
  """ Muestra el suavizado exponencial para varios valores de alpha
    series - serie de datos con tiempos
    alphas - lista de valores del parámetro """
  with plt.style.context('seaborn-white'):
    plt.figure(figsize=(15, 7))
    for alpha in alphas:
      plt.plot(exponential_smoothing(series, alpha),
               label="Alpha {}".format(alpha))
    plt.plot(series.values, "c", label = "Real")
    plt.legend(loc="best")
    plt.axis('tight')
    plt.title("Suavizado")
    plt.grid(True);
```

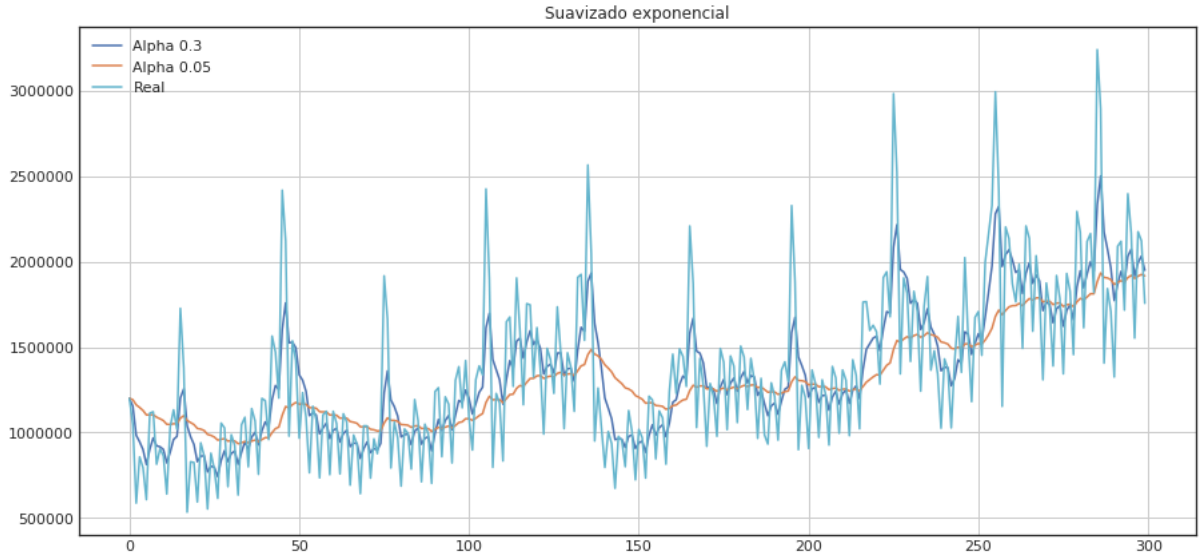
Probando con la serie de anuncios:

```
plotExponentialSmoothing(ads.Ads, [0.3, 0.05])
```



Con la de gastos:

```
plotExponentialSmoothing(cur.GEMS_GEMS_SPENT, [0.3, 0.05])
```

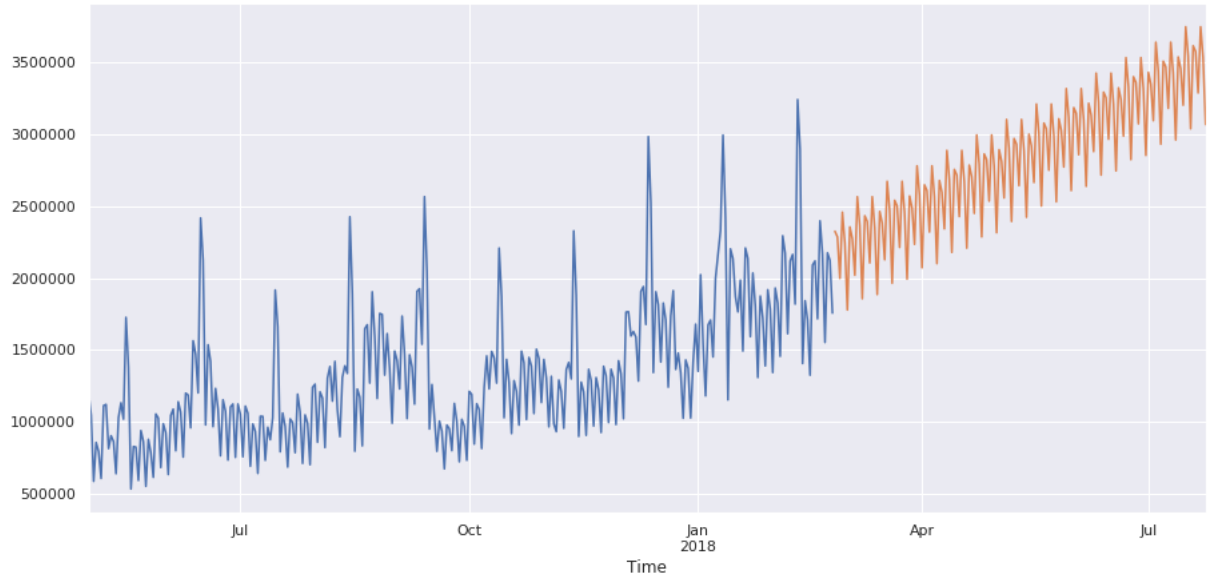


Holt-Winters:

Ahora probando el método de *ExponentialSmoothing* incluido en *statsmodels.tsa.api* y haciendo pronóstico:

```
from statsmodels.tsa.api import ExponentialSmoothing
# gráfica
def plotExpSmoothing(series):
    ses1 = ExponentialSmoothing((series), seasonal_periods=12,
trend='add', seasonal='add').fit()
    # pronóstico
    ses2 = ses1.forecast(len(series)/2)
    series.plot(kind="line",figsize=(15,7))
    ses2.plot(kind="line")
```

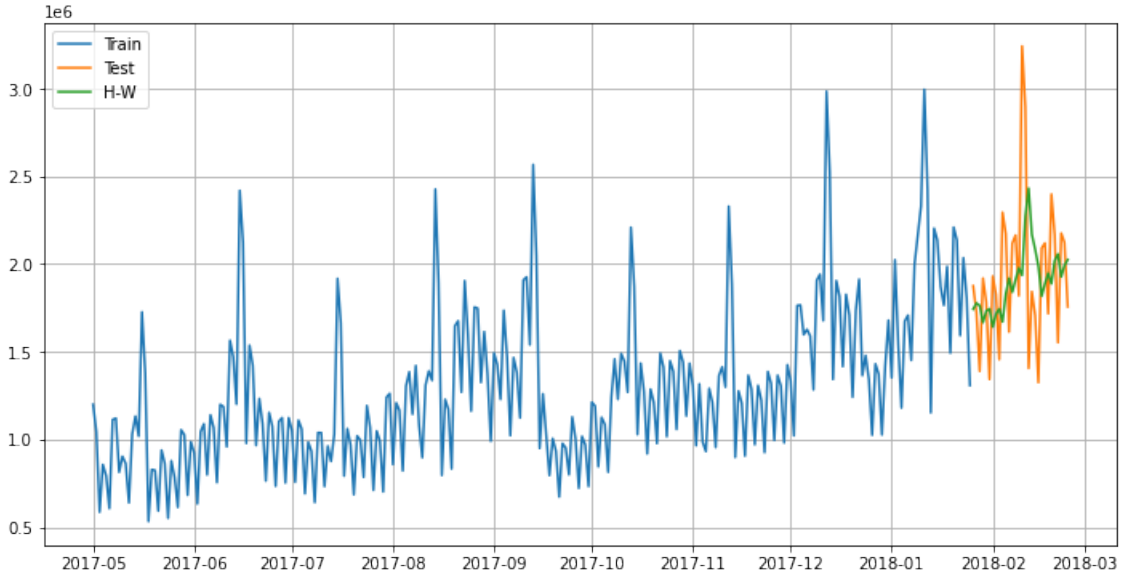
```
plotExpSmoothing(cur.GEMS_GEMS_SPENT, 0.3)
```



El método de Holt-Winters (*ExponentialSmoothing*) está incluido en *statsmodels.tsa.holtwinters*, comparando la predicción con los datos reales:

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

def plotHWSmothing(series, n=20):
  hw = ExponentialSmoothing((series),seasonal_periods=12).fit()
  # predicción
  train, test = series[:-n], series[len(series)-n:]
  hw_p = hw.predict(start=test.index[0],end=test.index[-1])
  plt.figure(figsize=(12,6))
  plt.plot(train.index,train,label='Train')
  plt.plot(test.index,test,label='Test')
  plt.plot(hw_p.index,hw_p,label='H-W')
  plt.legend(loc='best')
  plt.grid(True)
```

```
plotHWSmoothing(cur.GEMS_GEMS_SPENT, 30)
```



Ejemplo con R & ggplot:

El paquete *ggplot*2 incluye un conjunto de datos llamado *economics*:

```
library(ggplot2)
theme_set(theme_minimal())
#dentro de ggplot
head(economics)
help("economics")
```

Podemos graficar la serie con una *geom_ line* de *ggplot*:

```
ggplot(data = economics, aes(x = date, y = pop)) +
  geom_line(color = "#00AFBB", size = 2)
```

También es posible graficar sólo un subconjunto:

```
ss <- subset(economics, date > as.Date("2006-1-1"))
ggplot(data = ss, aes(x = date, y = pop)) +
   geom_line(color = "#FC4E07", size = 2)
```

Se puede modificar el tamaño de la línea de acuerdo a la proporción entre personas desempleadas y la población:

```
ggplot(data = economics, aes(x = date, y = pop)) +
  geom_line(aes(size = unemploy/pop), color = "#FC4E07")
```

La autocorrelación de *pop*, *psavert* y *uempmed*:

```
acf(economics$pop,50)
acf(economics$uempmed,30)
acf(economics$psavert,30)
```

Graficar mútiple series de datos, *psavert* y *uempmed* por fecha; primero usaremos dos bibliotecas auxiliares:

```
library(tidyr)
library(dplyr)
```

Con *select* de *tidyr* elegimos las columnas a usar y con *gather* de *dplyr* se crea una variable de agrupamiento para las columnas elegidas:

```
df <- economics %>%
  select(date, psavert, uempmed) %>%
  gather(key = "variable", value = "value", -date)
head(df, 3)
```

Gráfica con múltilpes líneas:

```
ggplot(df, aes(x = date, y = value)) +
   geom_line(aes(color = variable), size = 1) +
  scale_color_manual(values = c("#00AFBB", "#E7B800")) +
  theme_minimal()
```

Gráfica con áreas:

```
ggplot(df, aes(x = date, y = value)) +
   geom_area(aes(color = variable, fill = variable),
             alpha = 0.5, position = position_dodge(0.8)) +
  scale_color_manual(values = c("#00AFBB", "#E7B800")) +
  scale_fill_manual(values = c("#00AFBB", "#E7B800"))
```

Se pueden establecer límites a los ejes; gráfica base:

```
p <- ggplot(data = economics, aes(x = date, y = psavert)) +
   geom_line(color = "#00AFBB", size = 1)
p
```

Establecer los límites:

```
min <- as.Date("2002-1-1")
max <- NA
p + scale_x_date(limits = c(min, max))
```

Formato al eje de fechas:

```
p + scale_x_date(date_labels = "%b/%Y")
```

Se puede agregar una línea suavizada de tendencia; el método *loess* es una *regresión local* es un enfoque no paramétrico que ajusta la regresión en un vecindario local:

```
p + stat_smooth(
  color = "#FC4E07", fill = "#FC4E07",   method = "loess"
)
```

Y se puede usar también la media móvil:

```
library("TTR")
psr <- SMA(economics$psavert,n=8)
p + geom_line(aes(x=date,y=psr))
```

Ejemplo con R:

El conjunto de datos *kings.dat*, contiene la edad a la que fallecieron reyes sucesivos de Inglaterra, comenzando con *William the Conqueror*:

```
kings <- scan("http://robjhyndman.com/tsdldata/misc/kings.dat",skip=3)
kings
```

Para crear una serie de tiempo, se utiliza la función *ts* y se puede graficar con *plot.ts*:

```
kings_ts <- ts(kings)
kings_ts
plot.ts(kings_ts)
```

El paquete *TTS* incluye la media móvil:

```
#install.packages("TTR")
library("TTR")
kings_tsSMA8 <- SMA(kings_ts,n=8)
plot.ts(kings_tsSMA8)
```

Y el modelo de Holt-Winters está dentro del paquete *forecast*:

```
# install.packages("forecast")
library(forecast)
fit_hw <- HoltWinters(kings_ts, beta=TRUE, gamma=FALSE)
forecast(fit_hw, 3)
plot(forecast(fit_hw, 3))
```

También incluye el modelo *ets* (*error, trend, seasonality*):

```
fit_ets<- ets(kings_ts)
forecast(fit_ets, 3)
plot(forecast(fit_ets, 3))
```

1.4.3. Previsión basada en regresión

La regresión lineal provee un enfoque diferente para realizar pronósticos comparado con los métodos exponencial y Holt-Winters; en lugar de explotar tendencias recientes en el proceso, se utiliza una función predictora construida con las variables asociadas. La novedad del algoritmo es que la función de previsión se ajusta a pares de datos que están *escalonados* (*staggered*) con respecto a los pasos temporales. Por ejemplo, $(y_{n+\tau}, \mathbf{x}_n)$ donde n es un paso temporal y τ un entero positivo, es un par de datos escalonados. También puede describirse al vector predictor como *retrasado* (*lagged*) a la variable objetivo porque la función de pronóstico se entrega con objetivos observados τ pasos de tiempo después de que el vector fue observado: la función de previsión entrega predicciones del objetivo τ pasos después de que el vector se obtiene. Por ejemplo, si n es el paso actual, entonces $f\,(n|D) = \mathbf{x}_n^T\widehat{\boldsymbol{\beta}} = \widehat{y}_{n+\tau}$ es un pronóstico de $Y_{n+\tau}$; el vector $\widehat{\boldsymbol{\beta}}$ se

obtiene como una solución al problema de minimizar la suma de duferencias cuadráticas entre los pronósticos hechos en el tiempo n y los objetivos observados en el tiempo $n + \tau$.

Por tanto, el conjunto de datos se compone de pares desfazados $(y_{n+\tau}, \mathbf{x}_n), n = 1, \ldots, N - \tau$, donde $\mathbf{x}_n$ es un vector de variables medidas en el paso n. Se tiene la suposición de que el vector predictor $\mathbf{x}_n$ contiene información respecto al valor futuro de la variable objetivo. Matemáticamente esto indica que la media condicional $E(Y_{n+\tau}|\mathbf{x}_n)$ es aproximadamente igual a $\mu_{n+\tau}$; cuando esto se cumple, el enfoque basado en regresión es promisorio.

En el contexto de la regresión lineal, $\mathbf{x}_n^T\widehat{\boldsymbol{\beta}}$ es un estimador óptimo del valor futuro esperado; es decir, $E(Y_{n+\tau}|\mathbf{x}_n) = \mathbf{x}_n^T\widehat{\boldsymbol{\beta}}$; se puede especular que una combinación lineal de valores pasados puede superar algunas de las variaciones aleatorias en la serie y estimar la media del proceso con mayor precisión que el pronóstico exponencial o el basado en Holt-Winters. Aunque la regresión lineal requiere más datos y una función optimizada para minimizar los errores en las previsiones, el método de Holt-Winters implica afinar dos contantes: α_h y α_b que son un tanto difíciles de ajustar; entonces es razonable postular que la información que llevan los vactores predictores se traducen en más información para los pronósticos.

Ejemplo con Python, modelo lineal:

Primero deben crearse retrasos en la serie de tiempo:

```
# Creando una copia para realizar algunas transformaciones
data = pd.DataFrame(ads.Ads.copy())
data.columns = ["y"]
```

```
# Agregando los retrasos de la variable de 6 a 24 pasos
for i in range(6, 25):
    data["lag_{}".format(i)] = data.y.shift(i)

data.tail(7)
```

Creando el modelo de regresión:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import TimeSeriesSplit

# para cross-validation con serie de tiempo; se usan 5 pliegues
tscv = TimeSeriesSplit(n_splits=5)
```

Para generar conjuntos de entrenamiento y pruebas en series de tiempo con regresión lineal, no se realiza una selección aleatoria, se toman la última proporción para pruebas:

```
def timeseries_train_test_split(X, y, test_size):
    test_index = int(len(X)*(1-test_size))
    X_train = X.iloc[:test_index]
    y_train = y.iloc[:test_index]
    X_test = X.iloc[test_index:]
    y_test = y.iloc[test_index:]
    return X_train, X_test, y_train, y_test
```

Preprocesamiento, eliminar los que no tengan definida la variable objetivo:

```
y = data.dropna().y
X = data.dropna().drop(['y'], axis=1)
# 30% de los datos para prueba
X_train, X_test, y_train, y_test = timeseries_train_test_split(X, y,
                                                               test_size=0.3)
```

Ajuste del modelo:

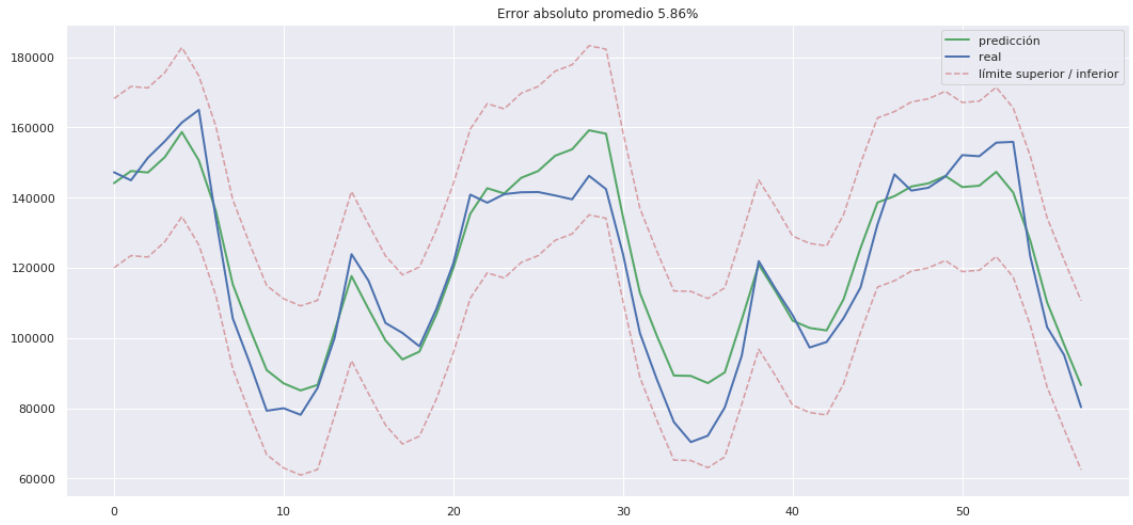
```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

Función para gráfica:

```
def plotModelResults(model, X_train, X_test, y_train, y_test,
                     plot_intervals=False, plot_anomalies=False):
  """ Gráfica del modelo vs valores reales con intervalo de predicción """
  prediction = model.predict(X_test)
  plt.figure(figsize=(15, 7))
  plt.plot(prediction, "g", label="predicción", linewidth=2.0)
  plt.plot(y_test.values, label="real", linewidth=2.0)
  # intervalos & anomalías
  if plot_intervals:
    # Aquí usamos cross_val_score
    cv = cross_val_score(model, X_train, y_train, cv=tscv,
                         scoring="neg_mean_absolute_error")
    # ...
  error = mean_absolute_percentage_error(prediction, y_test)
  plt.title("Error absoluto promedio {0:.2f}%".format(error))
  plt.legend(loc="best")
  plt.tight_layout()
  plt.grid(True);
```

Llamada:

```
plotModelResults(lr, X_train, X_test, y_train, y_test, plot_intervals=True)
```



Existen muchas posibilidades de modelos para trabajar con series de tiempo, por ejemplo:

♢ Random Forest for Time Series Forecasting (https://bit.ly/2JubNr9)

♢ Time series forecasting | TensorFlow Core(https://bit.ly/3qsSvDi)

Programa: _____

♢ Programar en R (debe graficar la serie y el resultado del modelo):

- Media móvil
- Exponencial

♢ Descargar el *urban Consumer Price Index from the US. Bureau of Labor Statistics* de https://research.stlouisfed.org/fred2/series/CPIAUCSL/downloaddata. Contiene valores mensuales del cambio porcentual del costo de alimentos, comida, ropa, habitación y combistibles en relación con una fecha inicial; cambios grandes en el *Consumer Price Index* indican inflación o deflación económica. Con ayuda de estos datos, compara el rendimiento de las predicciones obtenidas por los modelos:

- Media móvil
- Exponencial
- Holt-Winters
- Regresión lineal
  De preferencia utilizar las implementaciónes de las bibliotecas de Python

_____*