Introduction

In this notebook, we will learn how to work with and predict time series. Time series are a collection of **time dependent data** points. That means that each data point is associated with a time stamp. Time series can be univariate or multivariate. In this notebook we will work with univariate time series, meaning that we are only looking at one variable. Time series data can be **quite problematic to the point where it makes predictions about the future.** The most popular example is probably stock price prediction. Other examples are sales of seasonal clothing or weather forecasting. Because of its dependent property, time series are the hardest data predi

```
DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Spain,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Germany,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Italy,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_France,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_United_States_of_America,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Brazil,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Mexico,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Peru,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Chile,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Argentina,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Colombia,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Ecuador,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Bolivia,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Paraguay,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Uruguay,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Venezuela,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Cuba,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Dominican_Republic,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Guatemala,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Honduras,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Puerto_Rico,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Panama,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Costa_Rica,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_El_Salvador,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Haiti,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Nicaragua,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Bahamas,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Jamaica,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Trinidad_and_Tobago,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Guadeloupe,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Martinique,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_French_Guiana,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Reunion,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Mayotte,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Guadeloupe,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Martinique,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_French_Guiana,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Reunion,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Mayotte
```

```
DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Spain,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Germany,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Italy,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_France,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_United_States_of_America,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Brazil,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Mexico,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Peru,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Chile,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Argentina,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Colombia,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Ecuador,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Bolivia,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Paraguay,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Uruguay,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Venezuela,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Cuba,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Dominican_Republic,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Guatemala,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Honduras,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Puerto_Rico,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Panama,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Costa_Rica,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_El_Salvador,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Haiti,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Nicaragua,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Bahamas,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Jamaica,DEATHS_COVID_19_PER_10_MILLION_INHABITANTS_LAST_7_DAYS_Trinidad_and_Tobago
```

Cross Validation

For time series data, you can use the Time Series Split library or the Time Forecasting Bootcamp Analysis, but this depth about different types of time series problems and their cross validation methods.

```
from sklearn.model_selection import TimeSeriesSplit

N_SPLITS = 3
df = df.sort_values('date')
X = df.drop(['target', 'date'], axis=1)
y = df['target']

folds = TimeSeriesSplit(n_splits=N_SPLITS)

lr = LR(l1_ratio=l1_ratio[0], random_state=42)
mae_scores = []

for n, (train_idx, val_idx) in enumerate(folds.split(X, y)):
    print('Training on fold {} ...'.format(n+1))
    X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]
    lr.fit(X_train, y_train)
    mae_coefficients = np.abs(lr.coef_)[:-1]
    print('Coefficients: {}'.format(np.round(mae_coefficients, 5)))
    print('Intercept: {}'.format(np.round(lr.intercept_, 5)))
    mae = MAE(y_val, lr.predict(X_val))
    mae_scores.append(mae)
    print('MAE: {}\n'.format(round(mae, 5)))

for n in range(len(mae_scores)):
    print('Training on fold {} finished with error: {}'.format(n+1, round(mae_scores[n], 5)))
print('Average training error across folds: {}'.format(round(np.mean(mae_scores), 5)))

fig = plt.figure(figure_size=(20, 5))
for n, (train_idx, val_idx) in enumerate(folds.split(X, y)):
    ax = fig.add_subplot(1, N_SPLITS, n+1)
    ax.plot(df.iloc[train_idx]['date'], y[train_idx])
    ax.plot(df.iloc[val_idx]['date'], y[val_idx])
    ax.set_title('Training on Fold {}'.format(n+1), fontsize='15')
    ax.set_ylabel('Target', fontsize='15')
    ax.set_xlabel('Date', fontsize='15')
plt.show()
```

Models

Time series can be either univariate or multivariate.

- **Univariate** time series only has a single time-dependent variable.
- **Multivariate** time series has multiple time-dependent variables.

For example weather is a multivariate time series because for the medical features that are of time-dependent. However, by only looking at the target variable (Sports Car Demand), we can convert it to a univariate time series.

Univariate time series **predicts** the future values of a single variable **using its own history**. We will only **look at univariate time series** in this section.

We will evaluate the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE) of the models. For metrics see better the smaller they are.

Models for Univariate Time Series

- Moving Average
- Exponential Smoothing
- Moving Average
- Simple Exponential Smoothing
- ARIMA
- SARIMA
- Deep Learning
  - Vanilla LSTM
  - Stacked LSTM
  - Bidirectional LSTM
  - GRU

```
df['sports_car'] = df.sports_car/1000 * df.quarter.dt.days/91.25625

target = 'Sports Car Demand'
features = ['Medical Price Index', 'Medical CPI', 'Medical Consumption', 'Medical Spending', 'Medical Employment', 'Medical Productivity', 'Medical Wage', 'Medical Capital', 'Medical Debt', 'Medical Profit', 'Medical Revenue']

# split quarterly data
X = df[features].values
y = df[target].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42, shuffle=False)

lr = LR(l1_ratio=l1_ratio[0], random_state=42)
lr.fit(X_train, y_train)
mae_coefficients = np.abs(lr.coef_)[:-1]
print('Coefficients: {}'.format(np.round(mae_coefficients, 5)))
print('Intercept: {}'.format(np.round(lr.intercept_, 5)))
mae = MAE(y_test, lr.predict(X_test))
print('MAE: {}'.format(round(mae, 5)))

fig = plt.figure(figure_size=(20, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(df['date'], y)
ax.set_title('Sports Car Demand', fontsize='15')
ax.set_ylabel('Sports Car Demand', fontsize='15')
ax.set_xlabel('Date', fontsize='15')
plt.show()
```

```
def plot_exponential_smoothing(series, plot_intervals, plot_scaling, plot_ma=False):
    '''
    Accepts a pandas series, a boolean for whether to plot the moving average,
    a scaling factor for the confidence intervals, and a boolean for whether to plot the moving average.
    '''
    ma = sum(series[-ma_window:]) / ma_window
    mae = MAE(series[-ma_window:], [ma for i in range(ma_window)])
    deviation = np.sqrt(mae)
    scale = plot_scaling
    lower = series - abs(series) * deviation * scale
    upper = series + abs(series) * deviation * scale
    print("Last moving average: %s" % ma, "\nLast MAE: %s" % mae, "\nLast deviation: %s" % deviation)

    plt.rcParams['figure.figsize'] = [15, 7]
    plot_series(series, legend=['Sports Car Demand'])
    if plot_ma:
        plot_series(series.rolling(ma_window).mean(), legend=['Moving Average'])
    if plot_intervals:
        plot_series(upper, legend=['Upper Bound'])
        plot_series(lower, legend=['Lower Bound'])
    plt.title('Exponential Smoothing Forecast')
    plt.ylabel('Sports Car Demand')
    plt.xlabel('Date')
    plt.show()

plot_exponential_smoothing(df.sports_car, True, True, True, True)
```

Naive Approach

$$y_t = y_{t-1}$$

```
# Naive approach
NAIVE = df.sports_car[:-1]
print("Naive MAE: %s" % round(MAE(df.sports_car[1:], NAIVE), 5))
print("Naive MAPE: %s" % round(MAPE(df.sports_car[1:], NAIVE), 5))
print("Naive RMSE: %s" % round(RMSE(df.sports_car[1:], NAIVE), 5))

# Prediction
NAIVE = pd.concat([df.sports_car[-1:], NAIVE], ignore_index=True)
NAIVE.index = df.index
print("Naive Forecast: %s" % NAIVE[-1:].values[0])

# Plot
plot_series(df.sports_car, legend=['Sports Car Demand'])
plot_series(NAIVE, legend=['Naive Forecast'])
plt.title('Naive Forecast')
plt.ylabel('Sports Car Demand')
plt.xlabel('Date')
plt.show()
```


Moving Average

```
# Moving Average
window = 5
NAIVE = df.sports_car[:-1].rolling(window).mean()
print("Naive MAE: %s" % round(MAE(df.sports_car[1:], NAIVE), 5))
print("Naive MAPE: %s" % round(MAPE(df.sports_car[1:], NAIVE), 5))
print("Naive RMSE: %s" % round(RMSE(df.sports_car[1:], NAIVE), 5))

# Prediction
NAIVE = pd.concat([df.sports_car[-1:], NAIVE], ignore_index=True)
NAIVE.index = df.index
print("Naive Forecast: %s" % NAIVE[-1:].values[0])

# Plot
plot_series(df.sports_car, legend=['Sports Car Demand'])
plot_series(NAIVE, legend=['Naive Forecast'])
plt.title('Naive Forecast')
plt.ylabel('Sports Car Demand')
plt.xlabel('Date')
plt.show()
```


Notice the Naive Approach and the Moving Average Approach are pretty good results for our example. Usually, these approaches serve as a benchmark rather than the method of choice.

ARIMA

The Auto Regressive Integrated Moving Average (ARIMA) model describes the **autocorrelations** in the data. The model assumes that the time series is **stationary**. It consists of three parts:

- **Auto Regressive (AR) term** (long-term):

$$y_t = \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \alpha_3 y_{t-3} + \ldots + \alpha_n y_{t-n} + \epsilon_t$$

- **Integrated (I) term**:

$$y_t = y_t - y_{t-d}$$

- **Moving Average (MA) the** (short-term):

$$y_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \theta_3 \epsilon_{t-3} + \ldots + \theta_q \epsilon_{t-q}$$

ARIMA (p, d, q) corresponds to AR (p), I (d), MA (q) respectively or corresponds to:

- **Autocorrelation** (AR term): y_t depends on y_{t-1} and y_{t-2}
- **Stationarity** (I term): y_t depends on $y_t - y_{t-1}$
- **Shock** (MA term): y_t depends on ϵ_t

Seasonal ARIMA (SARIMA) extends ARIMA by adding a linear combination of seasonal past values and forecast errors:

$$SARIMA(p, d, q)(P, D, Q)_s$$

- p is number of autoregressive terms (AR)
- d is number of differencing (or difference order) (Integration or Stationarity)
- q is order of moving average (or errors) terms (MA)
- P, D, Q are the seasonal components of AR, I, and MA
- s is the seasonality period (12 for monthly, 4 for quarterly, etc.)

Stationarity is the key assumption. Stationarity means that the time series has no trend, no seasonality, and no other dependencies. Usually, time series data is not stationary. Most of the time series data has trend and seasonality. Differencing is a method to transform a non-stationary time series into a stationary time series. In this transformation, we do not lose any time series information. Sometimes, we need to take the difference more than once. Usually, second order differencing is enough to make the time series stationary.

Quick Explanation:

- Wikipedia
- Forecasting: Principles and Practice (Section 8.5)

```
from statsmodels.tsa.arima.model import ARIMA

window = 5
order = (1, 1, 0)
NAIVE = []
for n in range(len(df.sports_car[:-1])):
    if (n < window):
        model = ARIMA(df.sports_car[:n], order=order)
        model = model.fit(disp=0)
        NAIVE.append(model.forecast()[0])
    else:
        model = ARIMA(df.sports_car[(n-window):n], order=order)
        model = model.fit(disp=0)
        NAIVE.append(model.forecast()[0])
NAIVE = np.array(NAIVE)
print("Naive MAE: %s" % round(MAE(df.sports_car[1:], NAIVE), 5))
print("Naive MAPE: %s" % round(MAPE(df.sports_car[1:], NAIVE), 5))
print("Naive RMSE: %s" % round(RMSE(df.sports_car[1:], NAIVE), 5))

# Prediction
model = ARIMA(df.sports_car[(-window):], order=order)
model = model.fit(disp=0)
print("Naive Forecast: %s" % model.forecast()[0])

# Plot
plot_series(df.sports_car, legend=['Sports Car Demand'])
plot_series(NAIVE, legend=['Naive Forecast'])
plt.title('Naive Forecast')
plt.ylabel('Sports Car Demand')
plt.xlabel('Date')
plt.show
```


Models for Multivariate Time Series

- Vector Auto Regression (VAR)

```
# Vector Autoregressive (VAR) Model
from statsmodels.tsa.ar_model import AR

window = 5
order = (1, 1, 0)
NAIVE = []
for n in range(len(df.sports_car[:-1])):
    if (n < window):
        model = AR(df.sports_car[:n], order=order)
        model = model.fit(disp=0)
        NAIVE.append(model.predict()[0])
    else:
        model = AR(df.sports_car[(n-window):n], order=order)
        model = model.fit(disp=0)
        NAIVE.append(model.predict()[0])
NAIVE = np.array(NAIVE)
print("Naive MAE: %s" % round(MAE(df.sports_car[1:], NAIVE), 5))
print("Naive MAPE: %s" % round(MAPE(df.sports_car[1:], NAIVE), 5))
print("Naive RMSE: %s" % round(RMSE(df.sports_car[1:], NAIVE), 5))

# Prediction
model = AR(df.sports_car[(-window):], order=order)
model = model.fit(disp=0)
print("Naive Forecast: %s" % model.predict()[0])

# Plot
plot_series(df.sports_car, legend=['Sports Car Demand'])
plot_series(NAIVE, legend=['Naive Forecast'])
plt.title('Naive Forecast')
plt.ylabel('Sports Car Demand')
plt.xlabel('Date')
plt.show
```


Additional Resources

We other textbook on time series.

- Time Series Forecasting Bootcamp Analysis

Here are some additional resources that helped me learn about time series:

- Time Series Analysis in Python - Introduction to ARIMA models
- Time Series Analysis in Python - Introduction to ARIMA
- Time Series Analysis with Pandas
- Introduction to Time Series Analysis - Hyndman Athanasopoulos
- Introduction to Time Series Forecasting
- Time Series Forecasting with Keras
- Time Series Forecasting with Keras - Part 2

Useful Libraries

- statsmodels
- Forecast: Time series data forecasting in
- Prophet: Forecasting at Scale

```
cat
cd directory/path/directory
find . -name "*.ipynb" -exec rm -f {} \;
jupyter notebook
ls
ls -lrt
ls -lrtS
ls -lrtSh
ls -lrtSh *.ipynb
jupyter nbconvert --to notebook --execute Cross-Sectional.ipynb --inplace
jupyter nbconvert --to notebook --execute --ExecutePreprocessor.timeout=-1 Cross-Sectional.ipynb --inplace
jupyter nbconvert --to notebook --execute --ExecutePreprocessor.timeout=-1 --output=Cross-Sectional.ipynb Cross-Sectional.ipynb
jupyter nbconvert --to notebook --execute --ExecutePreprocessor.timeout=-1 --output=Cross-Sectional.ipynb --output-dir=./output Cross-Sectional.ipynb
jupyter nbconvert --to notebook --execute --ExecutePreprocessor.timeout=-1 --output=Cross-Sectional.ipynb --output-dir=./output --allow-errors Cross-Sectional.ipynb
```


Copyright (c) 2017 Cristian Bravo.
The Boston Housing Data was collected by Harrison D and Rubinfeld D L and this data was first published in Hedonic housing prices and the demand for clean air. J. Environ. Economics & Management, vol.5, 81-102, 1978.
The Wine Quality Data was created by Cortez P, Cerdeira A, Almeida F, Matos T and Reis J. and published in Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.