Univariate Time Series Analysis

basic notions

- **stochastic process**: a family $\{y_t \colon t \in \mathcal{T}\}$ of random variables
- **weak-sense stationary** (WSS) stochastic process: a stochastic process $\{y_t \colon t \in \mathcal{T}\}$ satisfying
  1. there is $\mu < +\infty$ st $\mathrm{E}[y_t] = \mu$ for $t \in \mathcal{T}$,
  2. $\mathrm{cov}(y_t, y_s) = \mathrm{cov}(y_0, y_{|t-s|})$ for $t, s \in \mathcal{T}$,
  3. there is $v < +\infty$ st $\mathrm{Var}[y_t] = v$ for $t \in \mathcal{T}$
- **time series**: a realization $y_{1:T} \colon = \{y_1, \ldots, y_T\}$ of a stochastic process

autocovariance & autocorrelation

- **autocovariance function**
  - $\gamma(t,s) := \mathrm{cov}(y_t, y_s)$
  - $\gamma(h) := \mathrm{cov}(y_t, y_{t+h})$ //WSS SP
  - $\hat{\gamma}(h) := \frac{1}{T} \sum_{t=1}^{T-h} (y_{t+h} - \overline{y})(y_t - \overline{y})$ //sample
- **autocorrelation function** (**ACF**)
  - $\rho(t,s) := \frac{\gamma(t,s)}{\sqrt{\gamma(t,t)}\sqrt{\gamma(s,s)}}$
  - $\rho(h) := \frac{\gamma(h)}{\gamma(0)}$ //WSS SP
  - $\hat{\rho}(h) := \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}$ //sample

Partial autocorrelation

The **partial autocorrelation function** (**PACF**) of a zero-mean WSS process $\{y_t\}$ at lag h is defined as

$$\phi(h,h) = \begin{cases} \mathrm{corr}(y_{t+1}, y_t) = \rho(1) & \text{for } h = 1, \\ \mathrm{corr}(y_{t+h} - \hat{y}_{t+h}, y_t - \hat{y}_t) & \text{for } h \geq 2, \end{cases}$$

where $\hat{y}_t = \beta_1 y_{t-1} + \ldots + \beta_{h-1} y_{t-h+1}$ is the linear combination of $\{y_{t-1}, \ldots, y_{t-h+1}\}$ minimizing $\mathrm{E}[(y_t - \hat{y}_t)^2]$.

Time series transformations

- **reducing variance**: take $\log y_t, \sqrt{y_t}$ or $y_t^{1/3}$; alternatively, use a Box-Cox transformation

  $$y_t' = \begin{cases} \log y_t, \text{ if } \lambda = 0 \\ (y_t^\lambda - 1)/\lambda \text{ otherwise} \end{cases}$$

  with λ that makes the size of the seasonal variation about the same across the whole series
- **differencing**: difference operator D is defined by $\mathrm{D}\, y_t = y_t - y_{t-1}$, $\mathrm{D}^d\, y_t = \mathrm{D}(\mathrm{D}^{d-1}\, y_t)$ and can be used for detrending; taking $y_t' = \mathrm{D}^d\, y_t$ with $d \in \{1, 2\}$ should be sufficient
- **seasonal differencing**: to remove seasonal effect for data with periodicity m, take $y_t' = y_t - y_{t-m}$
- **moving average smoothing**: replacing original observation with a weighted average of values around it: $y_t' := \sum_{j=-q}^{p} a_j y_{t+j}$, with the weights summing up to one. Reduces periodicity, exposes trend. Usually, is
  - **centered**: $p = q$, and
  - **symmetric**: $a_j = a_{-j}$.

  To remove periodicity in data with period m, use centered and symmetric, with
  - $a_p = a_q = \frac{1}{2m}, a_j = \frac{1}{m}$ for $j \notin \{-q, q\}$, if $m = 2q$,
  - $a_j = \frac{1}{m}$ for every j, if $m = 2q + 1$.

Time series modelling and forecasting: selected methods

- AR(p), autoregressive process of order p
- MA(q), moving average process of order q
- ARMA(p, q), autoregressive moving average model
- ARIMA(p, d, q), integrated ARMA(p, q)
- ARIMA$(p, d, q)(P, D, Q)_m$, seasonal ARIMA
- ARIMAX$(p, d, q)(P, D, Q)_m$, seasonal ARIMA with exogenous variables
- exponential smoothing
- Bayesian models, e.g., normal dynamic linear models (NDLMs)
- other, e.g., gradient boosting on regression trees, neural networks (deep, recurrent, long short-term memory), TBATS, ...

AR(p)

In **autoregressive process** AR(p) of order p every value is a linear combination of previous p values:

$$y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, v).$$

Suitable for stationary time series.
PACF(k) ≈ 0 for AR(p) and $k > p$.

MA(q)

In **moving average** MA(q) process of order q every values is modelled as the sum of time series mean and a linear combination of white noise error terms:

$$y_t = \mu + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}, \quad \epsilon_{t-i} \sim \mathcal{N}(0, v).$$

Suitable for stationary time series.
ACF(k) ≈ 0 for MA(q) and $k > q$.

ARMA(p, q)

A combination of an autoregressive process of order p and a moving average process of order q:

$$y_t = \mu + \epsilon_t + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}, \quad \epsilon_{t-i} \sim \mathcal{N}(0, v).$$

Suitable for stationary time series.

ARIMA(p, d, q)

Arises by applying ARIMA(p, q) to time series $y'_t = D^d y_t$, i.e., to the original time series differenced (detrended, "integrated") d times:

$$y'_t = \mu' + \epsilon_t + \sum_{i=1}^{p} \phi_i y'_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}, \quad \epsilon_{t-i} \sim \mathcal{N}(0, v).$$

Suitable for time series exhibiting trend but lacking seasonal variations.

ARIMA(p, d, q)(P, D, Q)$_m$

Seasonal ARIMA (also called SARIMA) tries to capture seasonal behaviour by adding to ARIMA a linear dependence of y_t on $y_{t-m}, \ldots, y_{t-Pm}$ and on $\epsilon_{t-m}, \ldots, \epsilon_{t-Qm}$.
Suitable for time series exhibiting trend or seasonal variations with period m.
Usually, $P, Q \in \{0, 1, 2\}$ and $D \in \{0, 1\}$.
Warning: very long model fitting time even for a couple thousands of observations.

ARIMAX(p, d, q)(P, D, Q)$_m$

Seasonal ARIMA with exogenous variables - seasonal ARIMA that additionally includes a linear regression term on some explanatory variables.
Suitable for time series exhibiting trend or seasonal variations with period m.

Simple exponential smoothing (SES)

In SES forecasted values are linear combinations of all the values observed so far, with exponentially decaying coefficients, controlled by a single parameter $\alpha \in (0, 1)$. The most recent observations contribute the most; the closer α is to one, the bigger their contributions when compared with the contributions of more distant observations.

$$\hat{y}_{t+1|t} = \alpha y_t + \alpha(1-\alpha) y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + \ldots,$$

or, in a weighted average form,

$$\hat{y}_{t+1|t} = \alpha y_t + (1-\alpha)\hat{y}_{t|t-1}.$$

Suitable for stationary time series.
Flat forecast: $\hat{y}_{T+h|T} = \hat{y}_{T+1|T}$.

Double exponential smoothing (DES)

Or **Holt's linear trend method**. Modification of SES that allows for modelling linear trend behaviour:

$\hat{y}_{t+h|t} = l_t + h b_t$ //forecast equation
$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1})$ //level equation
$b_t = \beta(l_t - l_{t-1}) + (1-\beta) b_{t-1}$ //trend equation

Here, l_t and b_t are level and trend slope estimates at time t, and α and β are smoothing parameters for level and trend, respectively. The level l_t is a weighted average of observed y_t and one-step-ahead training forecast, and b_t is a weighted average of the estimated trend at time t and the previous estimate of trend, b_{t-1}.
Suitable for time series exhibiting linear trend but lacking seasonal variations.

damped trend smoothing

A modification of DES, obtained by adding a new parameter that dampens the trend to a constant in the long run.

$\hat{y}_{t+h|t} = l_t + (\phi + \phi^2 + \ldots + \phi^h)b_t$ //forecast equation
$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + \phi b_{t-1})$ //level equation
$b_t = \beta(l_t - l_{t-1}) + (1-\beta)\phi b_{t-1}$ //trend equation

Equivalent to SES for $\phi = 0$ and to DES for $\phi = 1$; with $\phi \in [0, 1]$, the forecast tends to $l_t + \phi/(1-\phi)b_t$ as h grows. In practice, $\phi \in [0.8, 0.98]$.
Suitable for time series exhibiting linear trend but lacking seasonal variations.

triple exponential smoothing (tes)

Or **Holt-Winters Exponential Smoothing**.
Modification of DES that allows for modelling seasonal behaviour. The form below is intended for **additive seasonal effect** (i.e., roughly constant through the series), and k is the integer part of $(h-1)/m$.

$\hat{y}_{t+h|t} = l_t + hb_t + s_{t+h-m(k+1)}$ //forecast equation
$l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1})$ //level equation
$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$ //trend equation
$s_t = \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$ //seasonal equation

The seasonal component s_t is a weighted average of the current seasonal contribution and the seasonal contribution m time periods ago.
Suitable for time series exhibiting trend or seasonal variations with period m, with additive seasonal effect.

TES with multiplicative seasonal effect

When data suggests multiplicative seasonal effect, the following form of TES should be used.

$\hat{y}_{t+h|t} = (l_t + hb_t)s_{t+h-m(k+1)}$ //forecast equation
$l_t = \alpha y_t/s_{t-m} + (1-\alpha)(l_{t-1} + b_{t-1})$ //level equation
$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$ //trend equation
$s_t = \gamma y_t/(l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$ //seasonal equation

As previously, k in the above is the integer part of $(h-1)/m$.
Suitable for time series exhibiting trend or seasonal variations with period m, with multiplicative seasonal effect.

normal dynamic linear models (NDLMs)

In their most general form NDLMs are specified as follows.

$y_t = F_t^T\theta_t + v_t, v_t \overset{\text{iid}}{\sim} \mathcal{N}(0, V_t)$ //observation equation
$\theta_t = G_t\theta_{t-1} + w_t, w_t \overset{\text{iid}}{\sim} \mathcal{N}(0, W_t)$ //evolution equation
$(\theta_0|D_0) \sim \mathcal{N}(m_0, C_0)$ //conjugate prior

Here, $F_t^T \in \mathbb{R}^k$ is the design matrix (of known values of independent variables), $\theta_t \in \mathbb{R}^k$ is the state/system vector, $G_t^T \in \mathbb{R}^{k\times k}$ is the evolution matrix (known), $V_t \in \mathbb{R}$ is the observational error and $W_t \in \mathbb{R}$ is the system/evolution error. Forecast is computed as the expected value of the forecast distribution, namely

$$\hat{y}_{t+h|t} = \mathbb{E}[y_{t+h}|D_t] = F_{t+h}^T G_{t+h} \ldots G_{t+1}\mathbb{E}[\theta_t|D_t],$$

with $D_t = \{D_0, y_{1:t}\}$ being the information available at time t ($D_0 = \{m_0, C_0\}$).
Suitable for all time series. Admits multiple seasonal patterns as well as explanatory variables. Model fitting is generally extremely fast, as the parameters of distributions of interest are computed using simple recurrence relations.

simple forecasting methods

Näive method: $\hat{y}_{t+h|t} = y_t$
Näive seasonal method: $\hat{y}_{t+h|t} = y_{t+h-m(k+1)}$, with k being the integer part of $(h-1)/m$
Simple average method: $\hat{y}_{t+h|t} = (y_1 + \ldots + y_t)/t$
Moving average method: $\hat{y}_{t+h|t} = (y_{t-p} + \ldots + y_t)/p$
Weighted moving average method:
$\hat{y}_{t+h|t} = a_p y_{t-p+1} + \ldots + a_0 y_t$, with the weights summing up to one

Neural networks

If there is enough training data, can use deep neural networks, in particular networks using recurrent or LSTM layers.

Univariate Time Series Forecasting: Python Code, 1/2

Imports

```
import os
import numpy as np  #==1.24.4
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
# stationarity test
from statsmodels.tsa.stattools import adfuller
# models
from statsmodels.tsa.arima.model import ARIMA
import pydlm # for NDLMs
import xgboost as xgb
import tensorflow as tf  #==2.7.0
from statsmodels.tsa.holtwinters import
ExponentialSmoothing
```

Data preparation

```
# monthly champagne sales, taken from Kaggle
input_path = os.path.join("data",
"perrin-freres-monthly-champagne.csv")
df = pd.read_csv(input_path)
df = df.rename(
columns={"Month": "month",
  "Perrin Freres monthly champagne sales\
  millions ?64-?72": "sales"})
df = df[:-2]  # drop last two rows
df["month"] = pd.to_datetime(df["month"])
df = df.set_index("month")
p = 12  # periodicity
h = 24  # h for horizon
y_train = df["sales"][:-h]
y_test = df["sales"][-h:]
```

Simple methods

```
# 1.1 naive seasonal method
df["naive_seasonal_forecast"] = np.nan
df["naive_seasonal_forecast"].iloc[-2*p:-p] = \
y_train[-p:].values
df["naive_seasonal_forecast"].iloc[-p:] = \
y_train[-p:].values
# RMSE: 1014

# 1.2 weighted moving average
n_months = 12
weights = np.array(
[0.1**i for i in range(1, n_months + 1)])
weights = weights / weights.sum()
forecast_series = y_train[-n_months:].values
for t in range(h):
  forecasted_value = \
  forecast_series[-n_months:].dot(weights)
  forecast_series = \
  np.append(forecast_series, forecasted_value)
forecast = forecast_series[-h:]
df["wma_forecast"] = np.nan
df["wma_forecast"].iloc[-h:] = forecast
# RMSE: 1020
```

ARIMAX$(p,d,q)(P,D,Q)_m$

```
# 2. SARIMA
model = ARIMA(
y_train,
order=(0, 1, 1),  # p, d, q
seasonal_order=(1, 1, 1, 12),  # P, D, Q, m
freq="MS", # month start
)
model.fit()
sarima_forecast = model.predict(
  start=y_test.index.min(),
  end=y_test.index.max()
)
df["sarima_forecast"] = np.nan
df["sarima_forecast"].iloc[-h:] = \
sarima_forecast
# RMSE: 1517
```

Holt-Winters exponential smoothing

```
# 3. Holt-Winters smoothing,
# multiplicative seasonality
model = ExponentialSmoothing(
  y_train, damped_trend=False,
  seasonal="mul", seasonal_periods=p
)
model.fit()
exp_smoothing_forecast = model.predict(
  params=model.params,
  start=y_test.index.min(),
  end=y_test.index.max()
)
df["exp_smoothing_forecast"] = np.nan
df["exp_smoothing_forecast"].iloc[-h:] = \
exp_smoothing_forecast
# RMSE: 936
```

NDLM

```
# 4. normal dynamic linear model
model = pydlm.dlm(y_train)
# add linear trend component
model = model + pydlm.trend(degree=1,
discount=0.96, name="trend", w=10.0)
# add seasonal component
model = model + pydlm.seasonality(period=12,
discount=0.95, name="seas", w=10.0)
model.fit()
ndlm_mean_forecast, _ = model.predictN(
  N=h,
  date=model.n - 1)
df["ndlm_forecast"] = np.nan
df["ndlm_forecast"].iloc[-h:] = \
ndlm_mean_forecast
# RMSE: 419
```

Univariate Time Series Forecasting: Python Code, 2/2

XGBOOST

```
# 5. xgboost
model = xgb.XGBRegressor(
learning_rate=0.2,
max_depth=1,
n_estimators=50,
n_jobs=multiprocessing.cpu_count() // 2,
tree_method="hist",
objective="reg:squarederror",
)
# use the value of sales 12 months ago as
# the only predictor
df["sales_i-12"] = df["sales"].shift(12)
reg_cols = ["sales_i-12"]
model.fit(df[reg_cols].iloc[12:-h],
y_train[12:])
xgbt_forecast = model.predict(
  df[reg_cols].iloc[-2 * p : -p]
  )
xgbt_forecast = np.append(
xgbt_forecast,
model.predict(xgbt_forecast)
)
df["xgbt_forecast"] = np.nan
df["xgbt_forecast"].iloc[-h:] = xgbt_forecast
# RMSE: 924
```

DNN

```
# 6. deep neural network; not very deep :)
window_size = 12
batch_size = 21
shuffle_buffer_size = 100
# see next page for definition of the
# windowed_dataset function
dataset = windowed_dataset(
  y_train, window_size,
  batch_size, shuffle_buffer_size,
)
model = tf.keras.models.Sequential(
[
tf.keras.layers.Dense(
  10, input_shape=[window_size],
  activation="relu"),
tf.keras.layers.Dense(
  1, activation="relu"),
])
learning_rate = 1e-05
optimizer = tf.keras.optimizers.SGD(
learning_rate=learning_rate, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
optimizer=optimizer, metrics=["mse"])
model.fit(dataset, epochs=150)
# see next page for forecasting using NNs;
# huge variance in RMSE across 20 runs,
# most likely due to tiny training dataset
```

RNN

```
# 7. RNN, recurrent neural network
window_size = 18
batch_size = 9
shuffle_buffer_size = 100
dataset = windowed_dataset(
  y_train, window_size,
  batch_size, shuffle_buffer_size,
)
model = tf.keras.models.Sequential(
[
tf.keras.layers.Lambda(
  lambda x: tf.expand_dims(x, axis=-1),
  input_shape=[window_size]
),
tf.keras.layers.SimpleRNN(
  40, activation="relu",
  return_sequences=True),
tf.keras.layers.SimpleRNN(
  40, activation="relu"),
tf.keras.layers.Dense(1, activation="relu"),
])
learning_rate = 1e-05
optimizer = tf.keras.optimizers.SGD(
learning_rate=learning_rate, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
optimizer=optimizer, metrics=["mse"])
model.fit(dataset, epochs=150)
# see next page for forecasting using NNs;
# huge variance in RMSE across 20 runs,
# most likely due to tiny training dataset
```

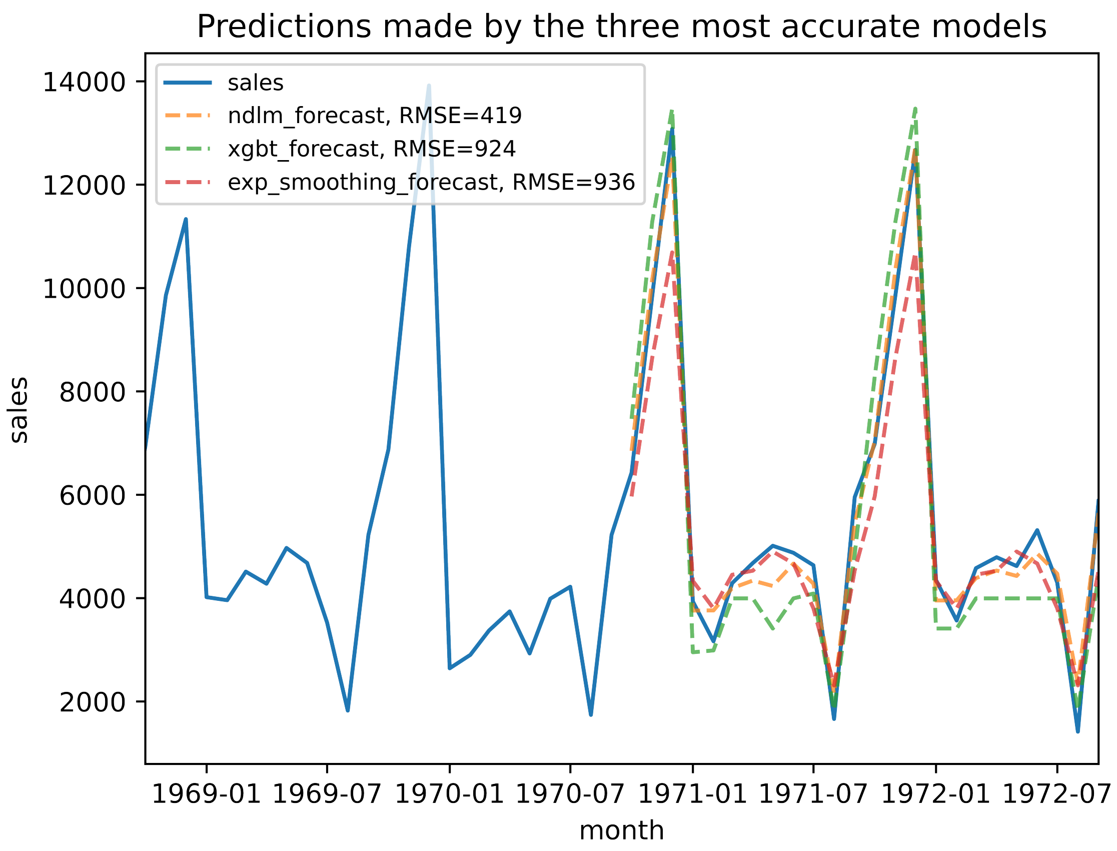
LSTM

```
# 8. LSTM, long short-term memory neural network
window_size = 13
batch_size = 32
shuffle_buffer_size = 100
dataset = windowed_dataset(
  y_train, window_size,
  batch_size, shuffle_buffer_size,
)
model = tf.keras.models.Sequential(
[
tf.keras.layers.Conv1D(
  filters=21,  kernel_size=3,
  strides=1, padding="causal",
  activation="relu", input_shape=[window_size, 1]
),
tf.keras.layers.LSTM(21, return_sequences=True,
  activation="relu"),
tf.keras.layers.LSTM(21, activation="relu"),
tf.keras.layers.Dense(1, activation="relu"),
]
)
learning_rate = 1e-05
optimizer = tf.keras.optimizers.SGD(
learning_rate=learning_rate, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
optimizer=optimizer, metrics=["mse"])
model.fit(dataset, epochs=150)
# see next page for forecasting using NNs;
# huge variance in RMSE across 20 runs,
# most likely due to tiny training dataset
```

```python
import tensorflow as tf

# the following function taken from Coursera's "Sequences, Time Series and Prediciton" course
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  """Generates dataset windows
  Args:
  series (array of float) - contains the values of the time series
  window_size (int) - the number of time steps to include in the feature
  batch_size (int) - the batch size
  shuffle_buffer(int) - buffer size to use for the shuffle method
  Returns:
  dataset (TF Dataset) - TF Dataset containing time windows
  """
  # Generate a TF Dataset from the series values
  dataset = tf.data.Dataset.from_tensor_slices(series)
  # Window the data but only take those with the specified size
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  # Flatten the windows by putting its elements in a single batch
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  # Create tuples with features and labels
  dataset = dataset.map(lambda window: (window[:-1], window[-1]))
  # Shuffle the windows
  dataset = dataset.shuffle(shuffle_buffer)
  # Create batches of windows
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset

# forecasting using trained neural network; taken from the same course
def model_forecast(model, series, window_size, batch_size):
  """Uses an input model to generate predictions on data windows
  Args:
  model (TF Keras Model) - model that accepts data windows
  series (array of float) - contains the values of the time series
  window_size (int) - the number of time steps to include in the window
  batch_size (int) - the batch size
  Returns:
  forecast (numpy array) - array containing predictions
  """
  # Generate a TF Dataset from the series values
  dataset = tf.data.Dataset.from_tensor_slices(series)
  # Window the data but only take those with the specified size
  dataset = dataset.window(window_size, shift=1, drop_remainder=True)
  # Flatten the windows by putting its elements in a single batch
  dataset = dataset.flat_map(lambda w: w.batch(window_size))
  # Create batches of windows
  dataset = dataset.batch(batch_size).prefetch(1)
  # Get predictions on the entire dataset
  forecast = model.predict(dataset)
  return forecast
```



Predictions made by the three most accurate models
sales
ndlm_forecast, RMSE=419
xgbt_forecast, RMSE=924
exp_smoothing_forecast, RMSE=936
14000
12000
10000
8000
6000
4000
2000
sales
1969-01
1969-07
1970-01
1970-07
1971-01
1971-07
1972-01
1972-07
month