```
#building the time series data
library(readx1)

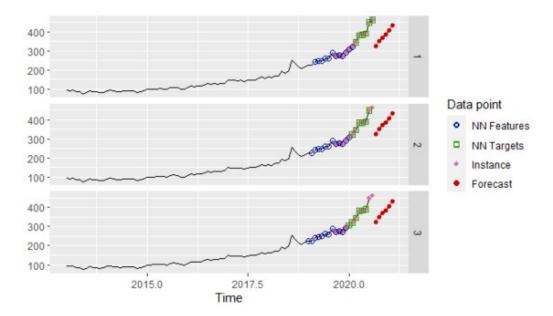
df_xautry <- read_excel("xau_try.xlsx")
xautry_ts <- ts(df_xautry$price, start = c(2013,1), frequency = 12)</pre>
```

KNN Regression

We are going to use **tsfknn** package which can be used to forecast time series in R programming language. KNN regression process consists of **instance**, **features**, **and targets** components. Below is an example to understand the components and the process.

library(tsfknn)

```
pred <- knn_forecasting(xautry_ts, h = 6, lags = 1:12,k=3)
autoplot(pred, highlight = "neighbors", faceting = TRUE)</pre>
```



The **lags** parameter indicates the lagged values of the time series data. The lagged values are used as features or explanatory variables. In this example, because our time series data is monthly, we set the parameters to 1:12. The last 12 observations of the data build the **instance**, which is shown by purple points on the graph.

This **instance** is used as a reference vector to find **features** that are the closest vectors to that instance. The relevant distance metric is calculated by the Euclidean formula as shown below:

$$\sqrt{\sum_{x=1}^{n} (f_x^i - q_x)^2}$$

 q_x denotes the instance and f_x^i indicates the features that are ranked in order by the distance metric. The k parameter determines the number of k closest features vectors which are called **k nearest neighbors**.

nearest neighbors function shows the instance, k nearest neighbors, and the targets.

```
nearest_neighbors(pred)
#$instance
#Lag 12 Lag 11 Lag 10 Lag 9 Lag 8 Lag 7 Lag 6 Lag 5 Lag 4 Lag 3 Lag 2
#272.79 277.55 272.91 291.12 306.76 322.53 345.28 382.02 384.06 389.36 448.28
# Lag 1
```

```
#$nneighbors

# Lag 12 Lag 11 Lag 10 Lag 9 Lag 8 Lag 7 Lag 6 Lag 5 Lag 4 Lag 3 Lag 2

#1 240.87 245.78 248.24 260.94 258.68 288.16 272.79 277.55 272.91 291.12 306.76

#2 225.74 240.87 245.78 248.24 260.94 258.68 288.16 272.79 277.55 272.91 291.12

#3 223.97 225.74 240.87 245.78 248.24 260.94 258.68 288.16 272.79 277.55 272.91 291.12

# Lag 1 H1 H2 H3 H4 H5 H6

#1 322.53 345.28 382.02 384.06 389.36 448.28 462.59

#2 306.76 322.53 345.28 382.02 384.06 389.36 448.28

#3 291.12 306.76 322.53 345.28 382.02 384.06 389.36 389.36
```

Targets are the time-series data that come right after the nearest neighbors and their number is the value of the **h parameter**. The targets of the nearest neighbors are averaged to forecast the future h periods.

$$\sum_{i=1}^{k} \frac{t^{i}}{k}$$

As you can see from the above plotting, features or targets might overlap the instance. This is because the time series data has no seasonality and is in a specific uptrend. This process we mentioned so far is called **MIMO**(multiple-input-multiple-output) strategy that is a forecasting method used as a default with KNN.

Decomposing and analyzing the time series data

Before we mention the model, we first analyze the time series data on whether there is seasonality. The decomposition analysis is used to calculate the strength of seasonality which is described as shown below:

$$F_S = \max\left(0, 1 - rac{ ext{Var}(R_t)}{ ext{Var}(S_t + R_t)}
ight)$$

$$F_T = \max\left(0, 1 - rac{ ext{Var}(R_t)}{ ext{Var}(T_t + R_t)}
ight)$$

```
#Seasonality and trend measurements
library(fpp2)

fit <- stl(xautry_ts,s.window = "periodic",t.window = 13,robust = TRUE)
seasonality <- fit %>% seasonal()
trend <- fit %>% trendcycle()
remain <- fit %>% remainder()

#Trend
1-var(remain)/var(trend+remain)
#[1] 0.990609
#Seasonality
1-var(remain)/var(seasonality+remain)
#[1] 0.2624522
```

The **stl** function is a decomposing time series method. STL is short for seasonal and trend decomposition using loess, which loess is a method for estimating nonlinear relationships. The **t.window**(trend window) is the number of consecutive observations to be used for estimating the trend and should be odd numbers. The s.window(seasonal window) is the number of consecutive years to estimate each value in the seasonal component, and in this example, is set to '**periodic**' to be the same for all years. The **robust parameter** is set to '**TRUE**' which means that the outliers won't affect the estimations of trend and seasonal components.

When we examine the results from the above code chunk, it is seen that there is a strong uptrend with 0.99,

weak seasonality strength with 0.26, because that any value less than 0.4 is accepted as a negligible seasonal effect. Because of that, we will prefer the non-seasonal ARIMA model.

Non-seasonal ARIMA

This model consists of differencing with autoregression and moving average. Let's explain each part of the model.

Differencing: First of all, we have to explain **stationary** data. If data doesn't contain information pattern like trend or seasonality in other words is white noise that data is stationary. **White noise** time series has no autocorrelation at all.

Differencing is a simple arithmetic operation that extracts the difference between two consecutive observations to make that data stationary.

$$y_t' = y_t - y_{t-1}$$

The above equation shows the first differences that difference at lag 1. Sometimes, the first difference is not enough to obtain stationary data, hence, we might have to do differencing of the time series data one more time(second-order differencing).

$$y''_{t} = y'_{t} - y'_{t-1}$$

$$= (y_{t} - y_{t-1}) - (y_{t-1} - y_{t-2})$$

$$= y_{t} - 2y_{t-1} + y_{t-2}$$

In **autoregressive models**, our target variable is a linear combination of its own lagged variables. This means the explanatory variables of the target variable are past values of that target variable. The **AR(p)** notation denotes the autoregressive model of order \mathbf{p} and the $\boldsymbol{\epsilon}_t$ denotes the white noise.

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \epsilon_t$$

Moving average models, unlike autoregressive models, they use past error(white noise) values for predictor variables. The MA(q) notation denotes the autoregressive model of order q.

$$y_t = c + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q}$$

If we integrate differencing with autoregression and the moving average model, we obtain a non-seasonal ARIMA model which is short for the autoregressive integrated moving average.

$$y_t' = c + \phi_1 y_{t-1}' + \dots + \phi_p y_{t-p}' + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

 y_t is the differenced data and we must remember it may have been first and second order. The explanatory variables are both lagged values of y_t and past forecast erros. This is denoted as **ARIMA(p,d,q)** where **p**; the order of the autoregressive; **d**, degree of first differencing; **q**, the order of the moving average.

Modeling with non-seasonal ARIMA

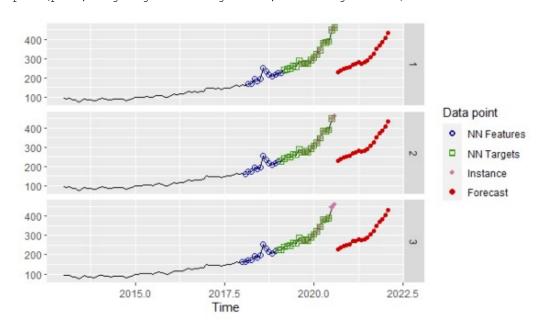
Before we model the data, first we split the data as train and test to calculate accuracy for the ARIMA model.

As seen above code chunk, stepwise=FALSE, approximation=FALSE parameters are used to amplify the searching for all possible model options. The **drift** component indicates the constant **c** which is the average change in the historical data. From the results above, we can see that there is no autoregressive part of the model, but a second-order moving average with the first differencing.

Modeling with KNN

```
#Modeling and forecasting
library(tsfknn)
pred <- knn forecasting(xautry_ts, h = 18, lags = 1:12, k=3)</pre>
```

```
#Forecasting plotting for KNN
autoplot(pred, highlight = "neighbors", faceting = TRUE)
```



Forecasting and accuracy comparison between the models

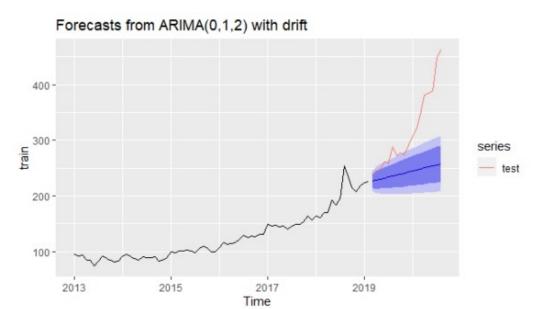
```
#ARIMA accuracy
f_arima<- fit_arima %>% forecast(h =18) %>%
    accuracy(test)

f_arima[,c("RMSE","MAE","MAPE")]

# RMSE MAE MAPE
#Training set 9.045488 5.529203 4.283023
```

For forecasting accuracy, we take the results of the test set shown above.

```
#Forecasting plot for ARIMA
fit arima %>% forecast(h=18) %>% autoplot()+ autolayer(test)
```



```
#KNN Accuracy
ro <- rolling_origin(pred, h = 18,rolling = FALSE)
ro$global accu</pre>
```

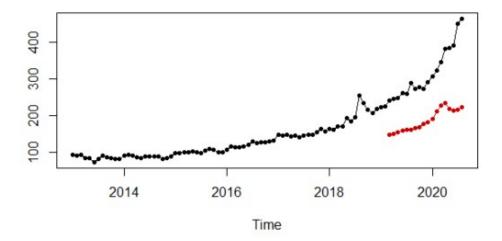
RMSE MAE MAPE #137.12465 129.77352 40.22795

The **rolling_origin** function is used to evaluate the accuracy based on rolling origin. The **rolling** parameter should be set to **FALSE** which makes the last 18 observations as the test set and the remaining as the training set; just like we did for ARIMA modeling before. The test set would not be a constant vector if we had set the rolling parameter to its default value of TRUE. Below, there is an example for h=6 that rolling_origin parameter set to TRUE. You can see the test set dynamically changed from 6 to 1 and they eventually build as a matrix, not a constant vector.



Training (blue) and test (red) sets for rolling origin evaluation with 6 observations.

#Accuracy plot for KNN
plot(ro)



When we compare the results of the accuracy measurements like RMSE or MAPE, we can easily see that the ARIMA model is much better than the KNN model for our non-seasonal time series data.