# Prévoir l’évolution de la valeur des cryptos monnaies

Paul Fayard, Jean Pachebat, Tom Reppelin

- Mars 2022

# Our github :

https://github.com/paulfyd/Datastream-Project

# Obtention des données en live avec Kafka

API Binance :

Requête toutes les minutes


**Response:**

```
[
  [
    1499040000000,      // Open time
    "0.01634790",       // Open
    "0.80000000",       // High
    "0.01575800",       // Low
    "0.01577100",       // Close
    "148976.11427815",  // Volume
    1499644799999,      // Close time
    "2434.19055334",    // Quote asset volume
    308,                // Number of trades
    "1756.87402397",    // Taker buy base asset volume
    "28.46694368",      // Taker buy quote asset volume
    "17928899.62484339" // Ignore.
  ]
]
```

# Producer.py

- producer.py des requetes sur l'API Binance toute les minutes

- puis l'envoie dans un topic "binance_topic"

```
producer = KafkaProducer(bootstrap_servers="localhost:9092",
    value_serializer=lambda v: json.dumps(v).encode('utf-8'))
topic_name = "binance_topic"

request = 'https://api.binance.com/api/v3/klines'
params = dict({
    'symbol':'SOLUSDT',
    'interval':'1m',
    'limit':'100'
})

while 1:
    response = rep = requests.get(request, params=params)
    data = rep.json()
    print(data)
    producer.send(topic_name, data)
    time.sleep(60)
```

# Consumer.py

```python
topic_name = "binance_topic"
consumer = KafkaConsumer(topic_name, bootstrap_servers="localhost:9092")

model = (
    river.preprocessing.StandardScaler() |
    river.tree.HoeffdingTreeRegressor(
        grace_period=200,
        leaf_prediction='adaptive',
        model_selector_decay=0.9
       )
    )
metric_mae = river.metrics.MAE()
metric_mape = river.metrics.SMAPE()

df_results = pd.DataFrame()
mape = 0
name = 'long_test'

for message in consumer:
    res = json.loads(message.value.decode('utf-8'))
    columns = ['open_time', 'open', 'high', 'low', 'close', 'volume',
       'close_time', 'quote_asset_volume', 'n_trades',
       'Taker buy base asset volume', 'Taker buy quote asset volume', 'Ignore']
    df = pd.DataFrame(res, columns=columns)
```

```python
    for col in columns:
        df[col] = df[col].astype(float)

    final_row = df.iloc[-1]

    features = []
    for t in [5, 20, 60]:
        open_moy = df.iloc[-t:-1].open.mean()
        open_std = df.iloc[-t:-1].open.std()
        n_trades_sum = df.iloc[-t:-1].n_trades.sum()
        delta = df.iloc[-t].close - df.iloc[-2].close
        feats = np.array([open_moy, open_std, n_trades_sum, delta])
        features.append(feats)
    features = np.array(features).flatten()

    times = ['5m', '20m', '60m']
    col_x = [[f'moy_{t}', f'std_{t}', f'n_trades_sum_{t}', f'delta_{t}'] for t in times]
    col_x = np.array(col_x).flatten()
    x = {col_x[i] : features[i] for i in range(len(features))}

    y = final_row.close
    y_pred = model.predict_one(x)
```

# Consumer.py (Suite)

```
    timestamp = final_row.open_time
    date = time.strftime('%A, %Y-%m-%d %H:%M:%S', time.localtime(timestamp/1000))
    print(f'Prevision at {date}')
    print(f'Predicted value : {y_pred:.2f}')
    print(f'Real value : {y}')
    mae = np.abs(y-y_pred)
    print(f'MAE : {mae:.2f}')
    if y_pred != 0:
        mape = 100*mean_absolute_percentage_error([y], [y_pred])
        print(f'MAPE : {mape:.2f}%')
    new_mae = metric_mae.update(y, y_pred).get()
    print(f'Current MAE : {new_mae:.2f}')
    new_smape = metric_mape.update(y, y_pred).get()
    print(f'Current SMAPE : {new_smape:.3f}%')

    results = [timestamp, y, y_pred, mae, mape, new_mae, new_smape] + list(features)
    print(results)
    cols_res = ['timestamp', 'y', 'y_pred', 'mae', 'mape', 'new_mae', 'new_smape']
    + list(col_x)
    print(cols_res)
    df_tmp = pd.DataFrame([results], columns=cols_res)
    df_results = pd.concat((df_results, df_tmp))
```

```
    df_results.to_csv(f'df_results_{name}_sol.csv', sep=';', index=False)

    model = model.learn_one(x, y)
    print()
    print('------------------------------------------------------------------------')
    print()

time.sleep(1)
```

# Online learning model

```
    model = (
        river.preprocessing.StandardScaler() |
        river.tree.HoeffdingTreeRegressor(
            grace_period=200,
            leaf_prediction='adaptive',
            model_selector_decay=0.9
            )
        )
```

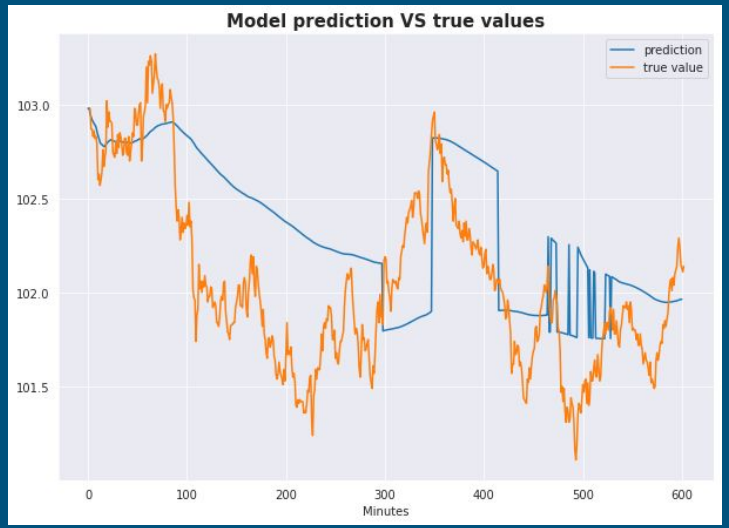
Les points arrivent au fur et à mesure et permettent d'apprendre les nouvelles feuilles, qui peuvent à terme devenir des noeuds internes de l'arbre.

Suivi des metrics dans notre terminal et animation en live

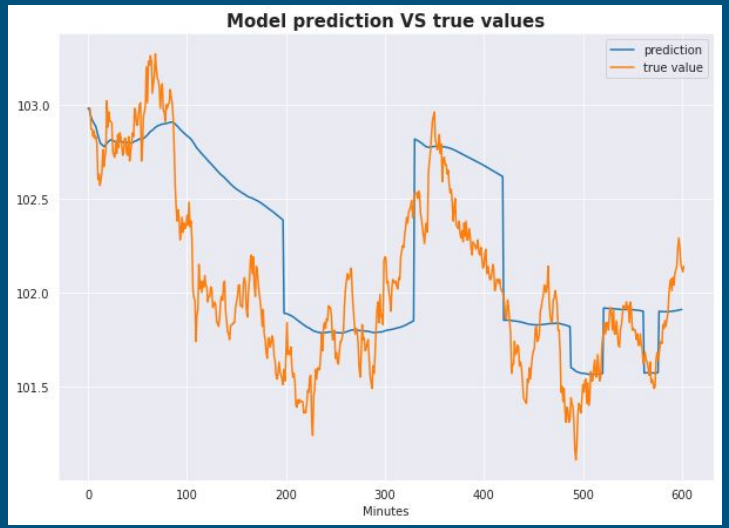
# Expériences a posteriori

On fait varier le paramètre grace_period de notre modèle : nombre de sample qu'une feuille doit observer avant de tenter un split : default=200

Grace_period = 100




Grace_period = 200



# Animation.py

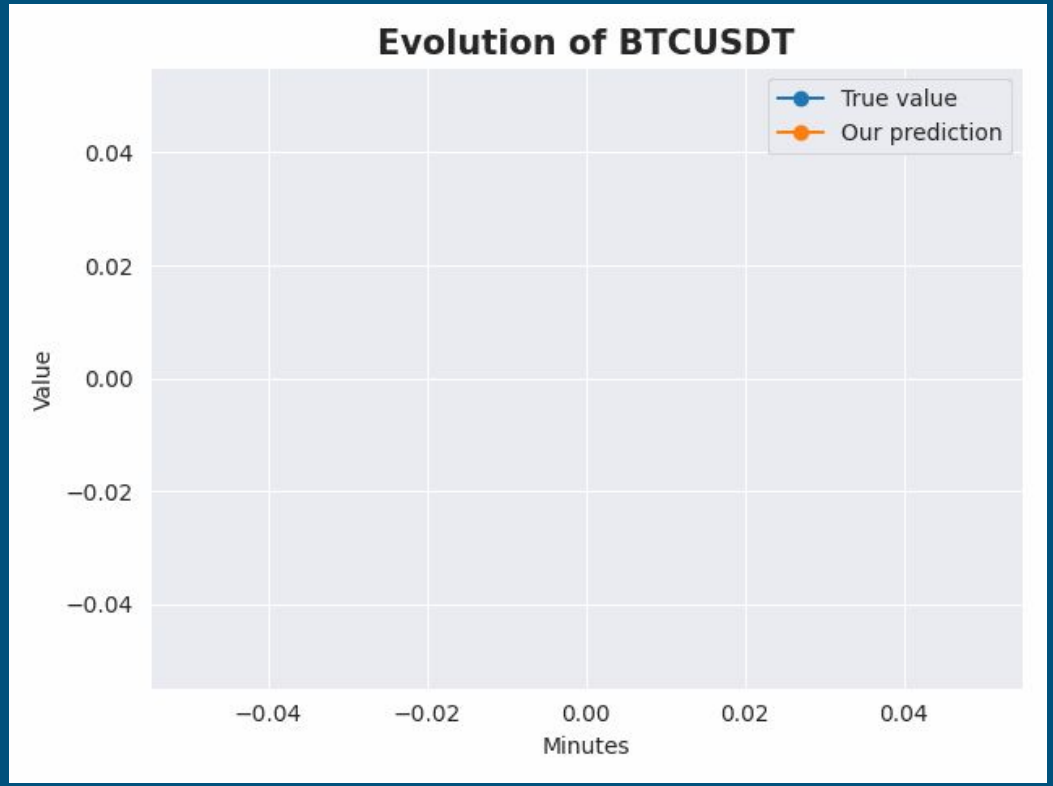
- Ce script nous permet ensuite d’afficher les animations de l’ évolution du BTCUSDT/ SOLUSDT/ ETHUSDT en fonction du temps:

```
sns.set_style('darkgrid')
name = sys.argv[1]
coin = sys.argv[2]

def animate(i):
    data = pd.read_csv(f'./data/df_results_{name}.csv', sep=';').iloc[2:]
    data = data.iloc[2:]

    if i < len(data):
        data = data.iloc[:i]


    x = np.arange(len(data))
    y1 = data['y']
    y2 = data['y_pred']

    plt.cla()
    plt.rcParams["figure.figsize"] = (20,10)
    plt.xlabel('Minutes')
    plt.ylabel('Value')
    plt.plot(x, y1, label='True value', marker='o')
    plt.plot(x, y2, label='Our prediction', marker='o')
    plt.title(f'Evolution of {coin}', fontsize=15, fontweight='bold')
    plt.legend()
    plt.tight_layout()


frames = np.arange(250)

ani = FuncAnimation(plt.gcf(), animate, frames=frames, interval=100)
ani.save(f'./visu/animation_{name}.gif')

plt.tight_layout()
plt.show()
```

# 3 animations :

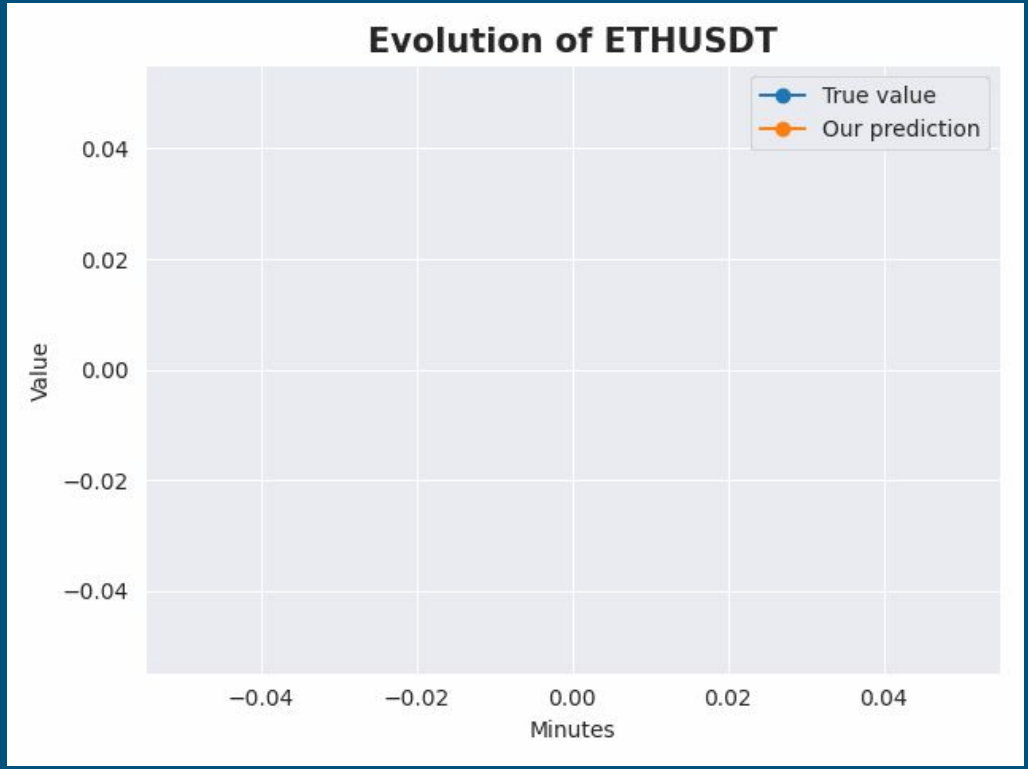
**Bitcoin Usdt:**

- **Nous arrivons, sur le long terme, récupérer la tendance globale, mais on voit que l'on est pas très précis:**



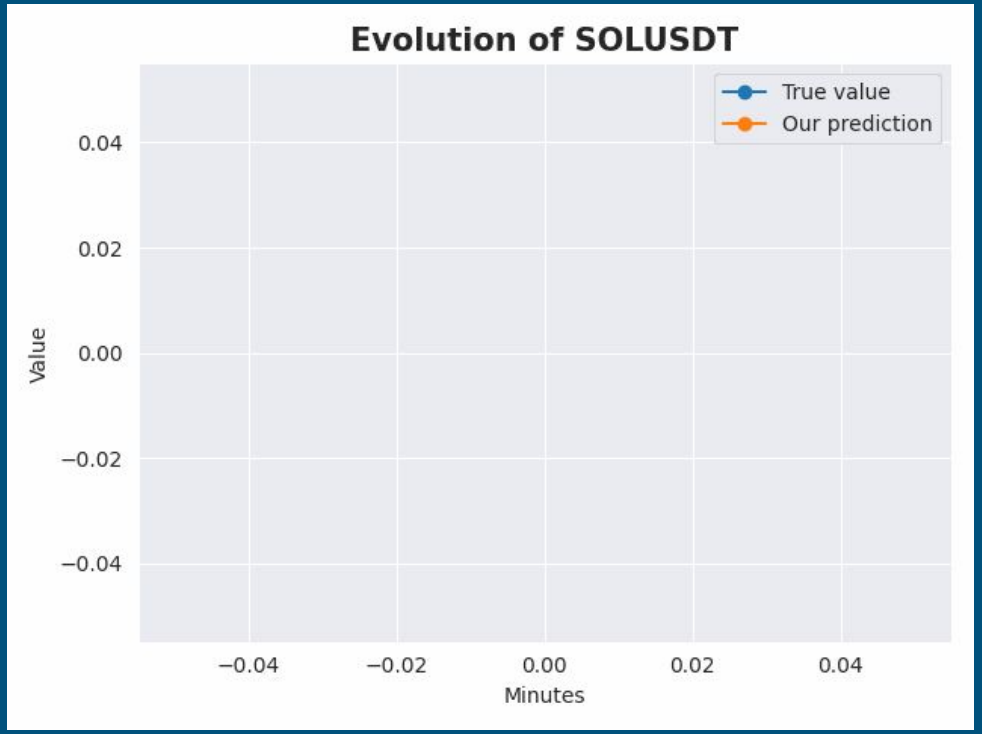
**Ethereum Usdt:**

- **Nous voyons un drift à 190 min environ,**
- **Ce modèle à tourné dans l'après midi du samedi 26 mars:**



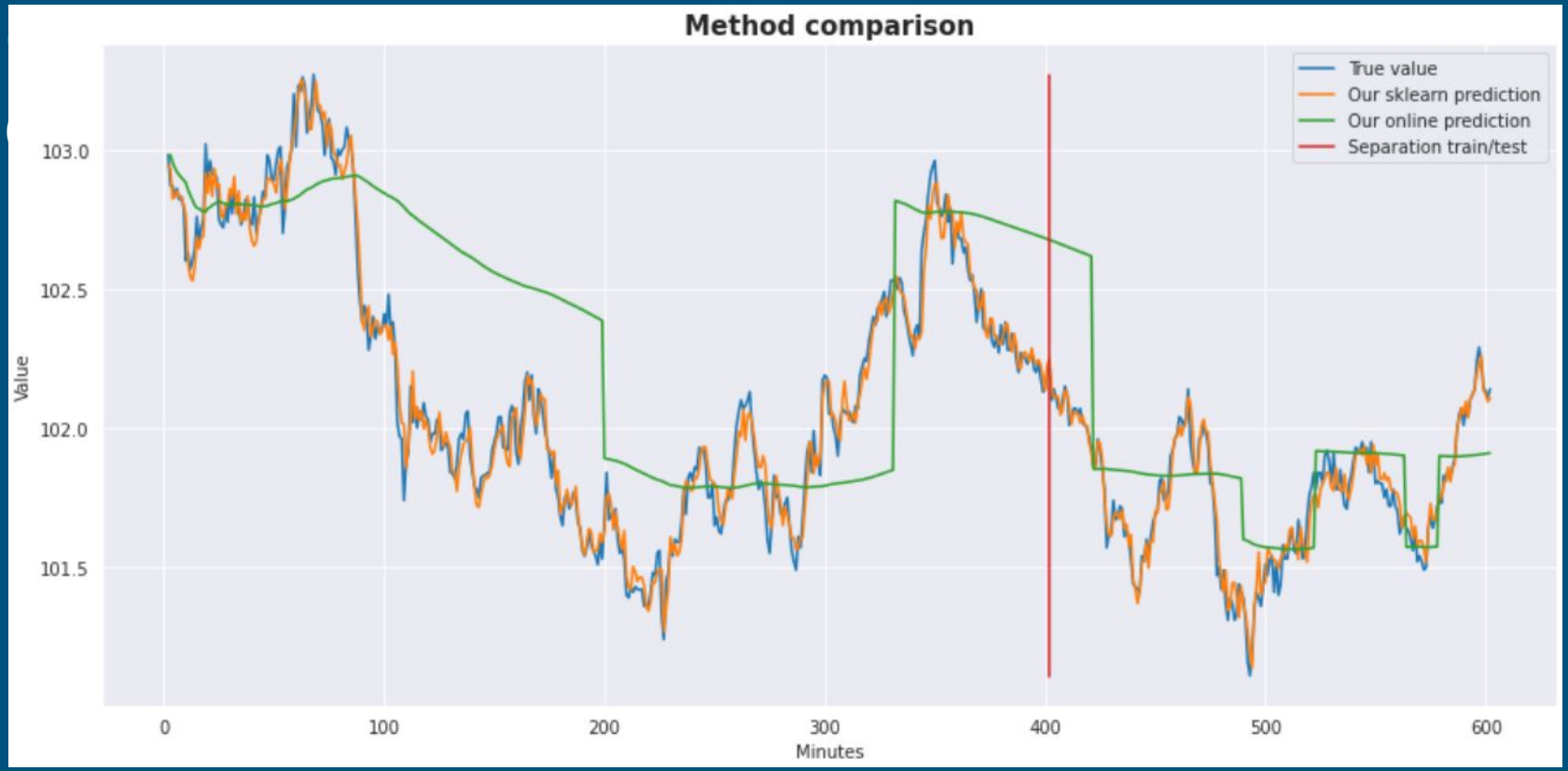
**Solana Usdt:**

- Nous voyons un drift à 195 min environ,
- Ce modèle à tourné dans la nuit du 26 mars:



# Comparaison avec du batch learning

Comparaison HoeffdingTreeRegressor (River) et RidgeRegressor (sklearn)



# Comparaison de modèles : SNARIMAX:

```
>>> model = (
...     time_series.SNARIMAX(
...         p=0,
...         d=0,
...         q=0,

...         m=12,
...         sp=3,
...         sq=6
...         )
...     )
metric =
metrics.Rolling(metrics.SMAPE(), 12)

for x, y in zip(date, y_data):

    y_pred = model.forecast(horizon=1)
    model = model.learn_one(y)
    metric = metric.update(y, y_pred[0])
```

SNARIMAX signifie:

- (S)easonal (N)on-linear (A)uto(R)egressive (I)ntegrated (M)oving-(A)verage with e(X)ogenous inputs model.

Avec SNARIMAX, il est possible de faire un modèle MA, ou ARIMA, ou ARMA, simplement en fonction des paramètres qui sont renseignés.

Ce modèle généralise de nombreux modèles de séries chronologiques établis dans une interface unique qui peut être entraînée en ligne. Il suppose que les données d'entraînement fournies sont **ordonnées dans le temps** et **uniformément espacées**. Il est composé des éléments suivants : (slide suivante)

# Comparaison de modèles : SNARIMAX:

Il est composé des éléments suivants :

S (saisonnier)

N (non linéaire) : Tout modèle de régression en ligne peut être utilisé, pas nécessairement une régression linéaire comme cela est fait dans les manuels.

AR (Autoregressive) : Les retards de la variable cible sont utilisés comme caractéristiques.

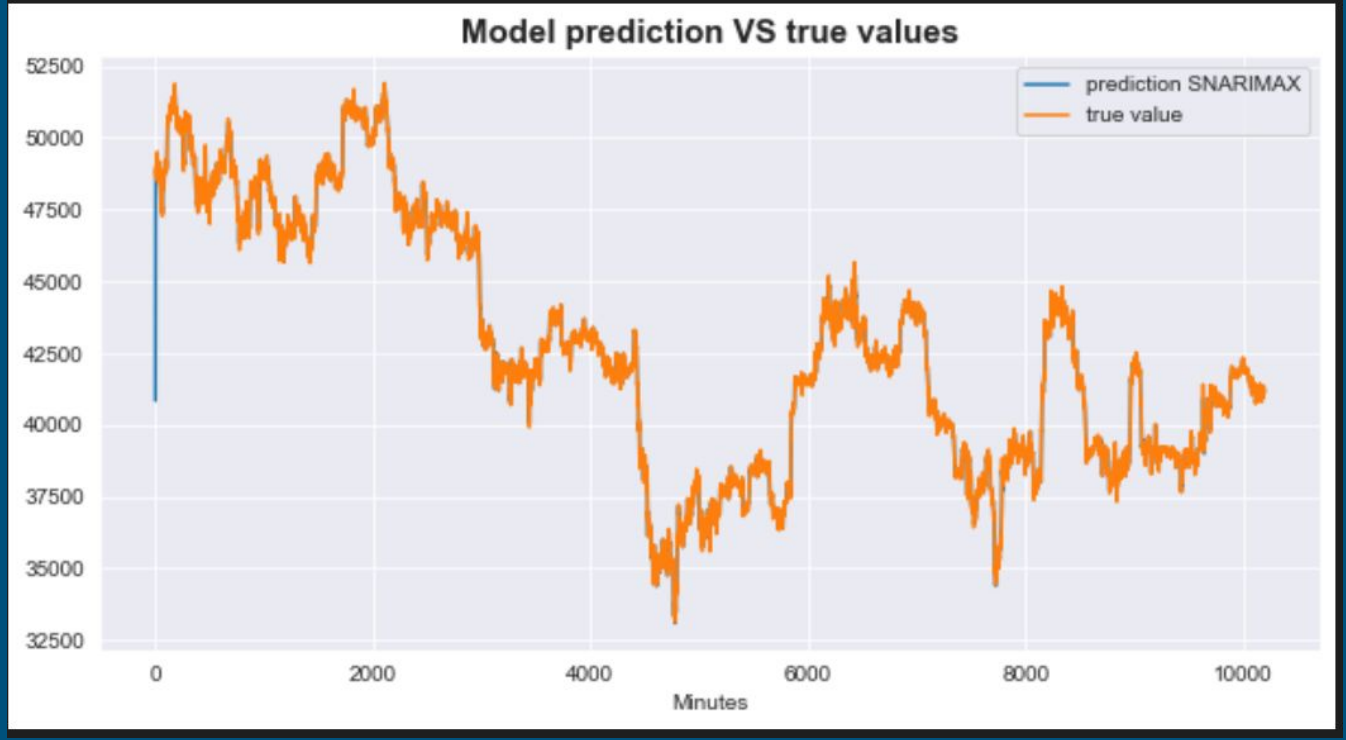
I (Integrated) : Le modèle peut être ajusté sur une version différencié d'une série temporelle. Dans ce contexte, l'intégration est l'inverse de la différenciation.

MA (Moving average) : Les retards des erreurs sont utilisés comme caractéristiques.

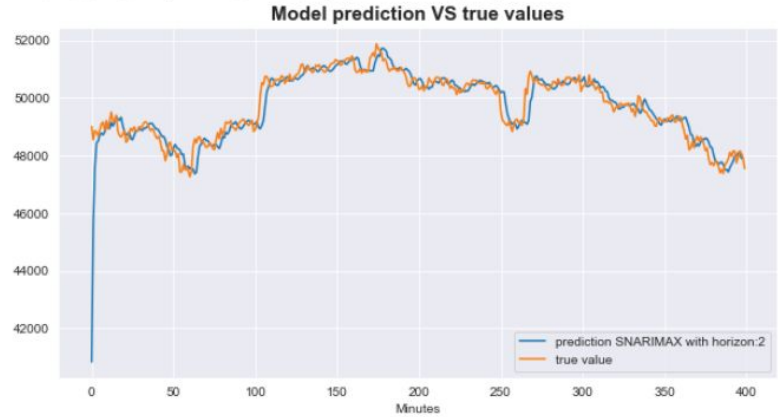
X (Exogène) : Les utilisateurs peuvent fournir des caractéristiques supplémentaires. Il faut veiller à inclure des caractéristiques qui seront disponibles à la fois au moment de la formation et de la prédiction.

# BTCUSDT

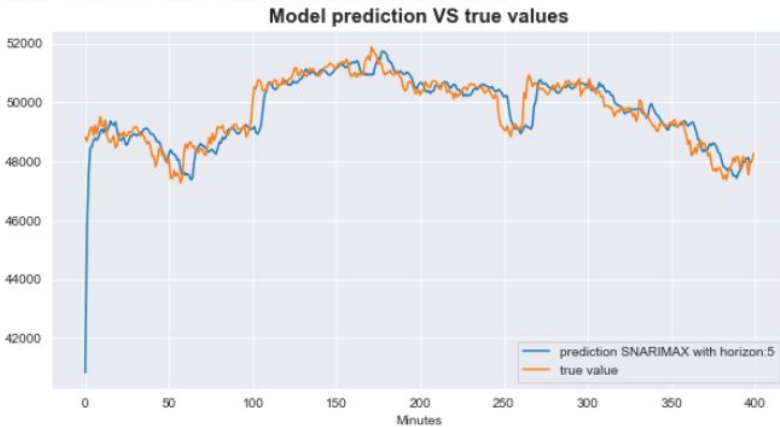
```
Final scores : (MAE: 116.090027, SMAPE: 0.)
```



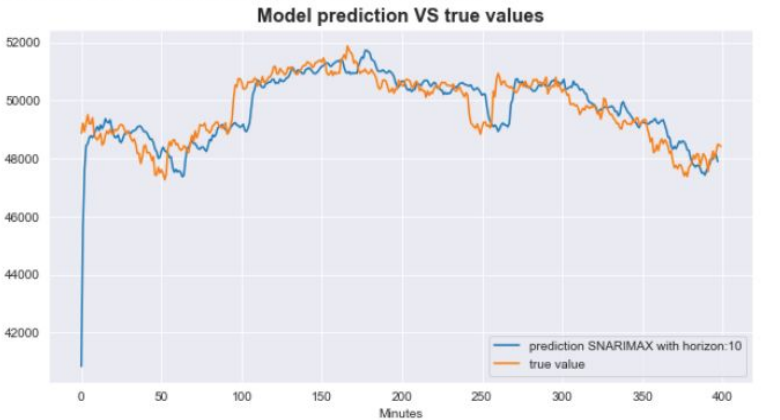
Final scores : (MAE: 435.769824, SMAPE: 1.163478)



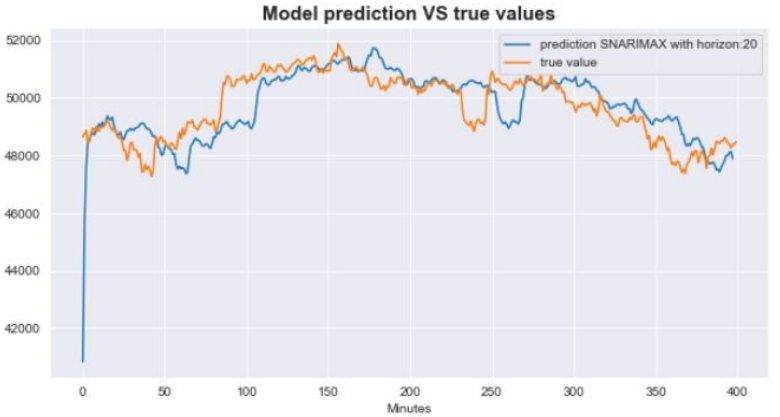

Final scores : (MAE: 522.376423, SMAPE: 1.339885)




Final scores : (MAE: 648.727101, SMAPE: 1.592027)

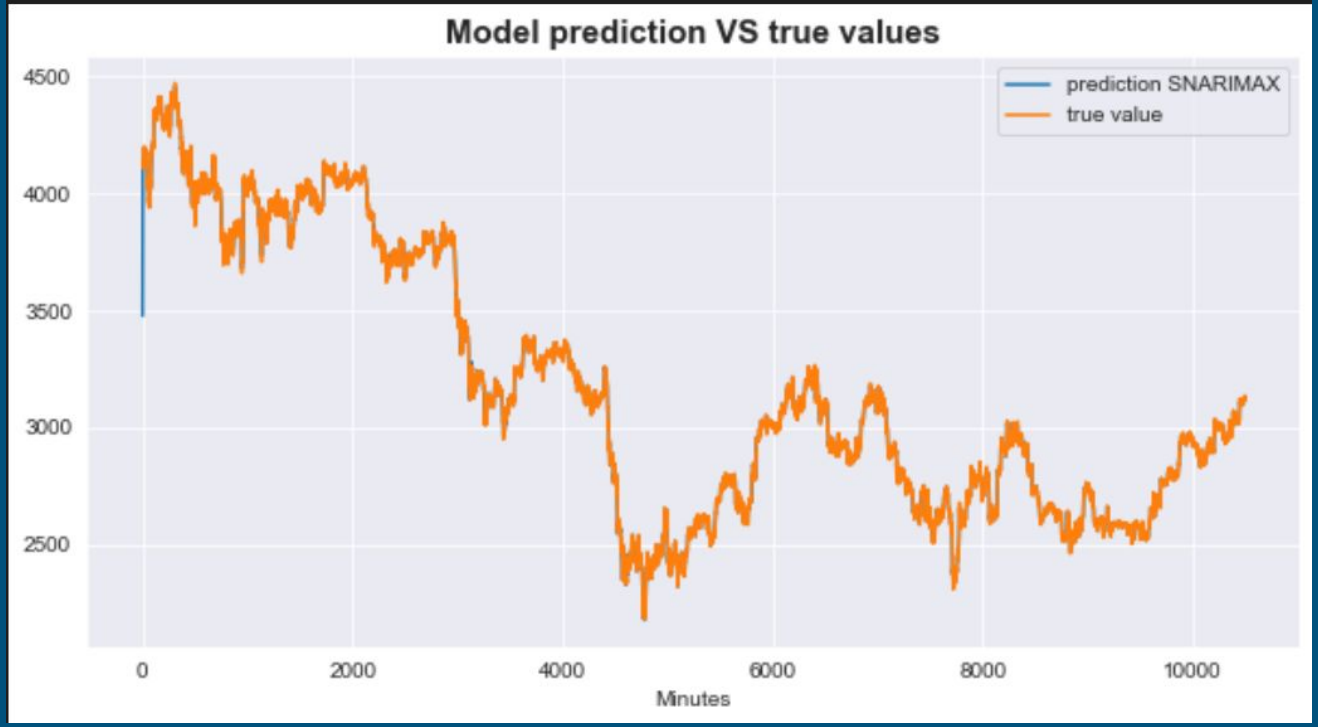



Final scores : (MAE: 780.050023, SMAPE: 1.859757)



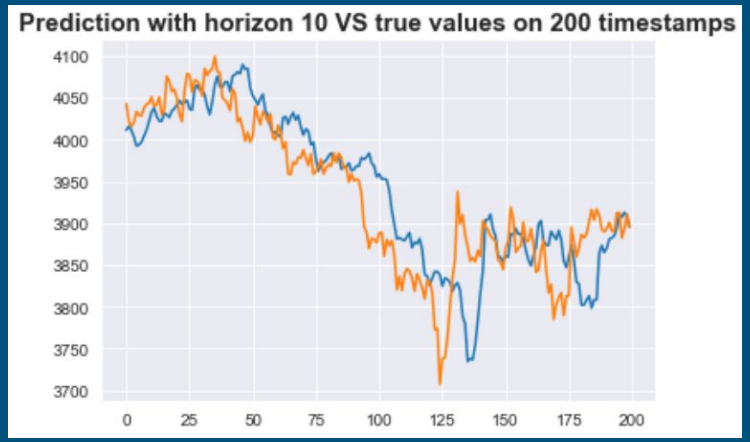
# ETHUSDT

```
Final scores : (MAE: 10.481028, SMAPE: 0.)
```



# ETHUSDT

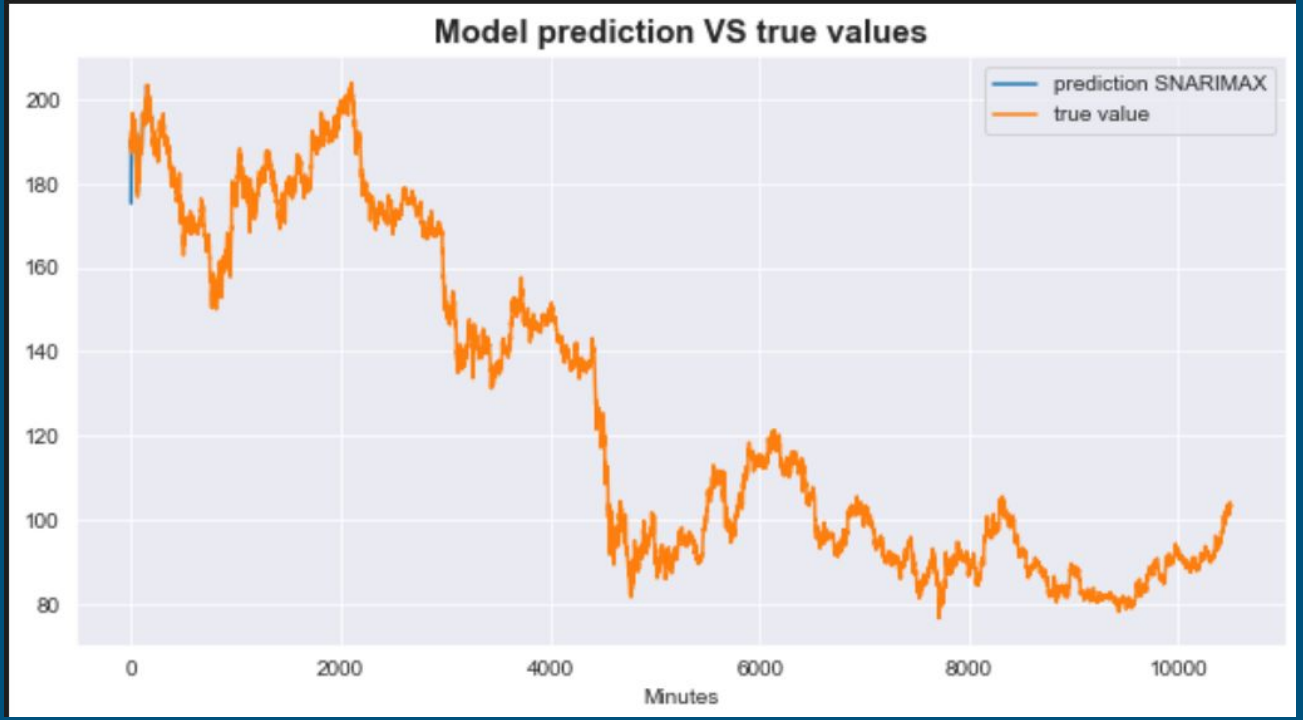

Prediction with horizon 5 VS true values on 200 timestamps
4100
4050
4000
3950
3900
3850
3800
3750
3700
0
25
50
75
100
125
150
175
200


Prediction with horizon 10 VS true values on 200 timestamps
4100
4050
4000
3950
3900
3850
3800
3750
3700
0
25
50
75
100
125
150
175
200


Prediction with horizon 20 VS true values on 200 tin
4100
4050
4000
3950
3900
3850
3800
3750
3700
0
25
50
75
100
125
150
175
200

# SOLUSDT

```
Final scores : (MAE: 0.570131, SMAPE: 0.)
```



# SOLUSDT


Prediction with horizon 5 VS true values on 200 timestamps
187.5
185.0
182.5
180.0
177.5
175.0
172.5
170.0
167.5
0
25
50
75
100
125
150
175
200


Prediction with horizon 10 VS true values on 200 timestamps
187.5
185.0
182.5
180.0
177.5
175.0
172.5
170.0
167.5
0
25
50
75
100
125
150
175
200


Prediction with horizon 20 VS true values on 200 timestamps
187.5
185.0
182.5
180.0
177.5
175.0
172.5
170.0
167.5
0
25
50
75
100
125
150
175
200

# Conclusion

# Annexe :
# Petit point informatif: Qu'en est-il du HOLTWINTER ?

**Level initialization**

$$l = \frac{1}{k}\sum_{i=1} ky_i$$

**Trend initialization**

$$t = \frac{1}{k-1}\sum_{i=2} ky_i - y_{i-1}$$

**Trend initialization**

$$s_i = \frac{y_i}{k}$$

Prévision par le modèle Holt-Winters.

- Il s'agit d'une implémentation standard de la méthode de prévision de Holt-Winters. Certains paramètres donnent lieu à des cas particuliers, comme le lissage exponentiel simple.

- Nous avons essayé d'implémenter ce modèle, cependant c'est un modèle qui fonctionne pour les saisonnalités.

C'est une idée que l'on pourrait mettre en place: sur une grande période.

Pour nous, pas spécialement intéressant, mais c'est une piste de développement pour voir le traitement avec les saisonalités.