

ML TEMPS RÉEL SUR FLUX DE DONNÉES

ML TEMPS RÉEL SUR FLUX DE DONNÉES

Présenté par :

- **RAHMOUN Hayat:** FST Marrakech, Modélisation Calcul Scientifique pour L'ingénierie Mathématique
- **AGOUMI Achraf:** ENSA Al Hoceima, Ingénierie des données
- **EL BOUGRINI Nassim:** EMI, Génie industriel.
- **NACHOUR Ilham:** ENSA Al Hoceima, Ingénierie des données
- **HAFSI Siham :** ENSA KHOURIBGA , Informatique et ingénierie des données
- **CHAHIDI Khadija:** ENSA Al Hoceima, Ingénierie des données

Encadré par :

- **Thierry BERTIN**



PLAN

1

INTRODUCTION

2

DATA STREAMING

3

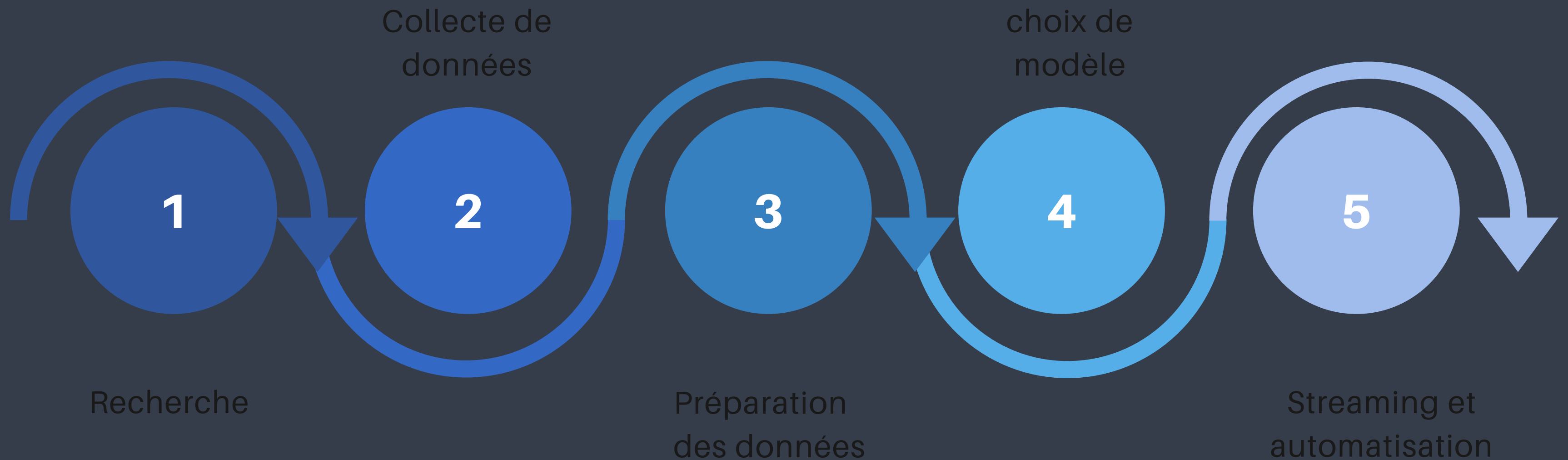
AUTOMATISATION

4

CONCLUSION

CHRONOLOGIE DU PROJECT

De l'Idée à l'Automatisation



Collecte des données

Le site "*alphavantage*" présente plusieurs avantages:

- Présentation d'un flux de données en temps réel.
- Choix du type de données suivant la société et les intervalles temporelles.





Préparation des données

Réorganisation de la DataFrame : Afin de garantir que les données sont classées de manière chronologique, nous inversons l'ordre des lignes dans la DataFrame.

Nettoyage des Étiquettes de Colonnes : Nous effectuons un nettoyage des étiquettes de colonnes en supprimant les préfixes numériques ('1. open', '2. high', etc.) pour améliorer la lisibilité.

Gestion des Valeurs Manquantes : Nous vérifions la présence de valeurs manquantes dans la DataFrame et nous assurons qu'il n'y a pas de données manquantes.



Choix de modèle

Random Forest

ARIMA

XGBoost



Adaptabilité aux Données
Temporelles

faible MSE



Data **streaming**

Confluent Cloud Kafka est une plateforme de streaming de données gérée proposée par Confluent, la société derrière Apache Kafka. Cette plateforme cloud permet aux entreprises de tirer parti de la puissance de Kafka sans avoir à gérer l'infrastructure sous-jacente. Elle offre une solution évolutive, hautement disponible et sécurisée pour la gestion des flux de données en temps réel.

Kafka Producer

- Les producteurs sont responsables de l'envoi des données vers les sujets Kafka.
- Ils publient des messages dans les sujets Kafka, qui sont ensuite disponibles pour les consommateurs.

```
producer_conf = {  
    "bootstrap.servers": "pkc-921jm.us-east-2.aws.confluent.cloud:9092",  
    "sasl.mechanisms": "PLAIN",  
    "security.protocol": "SASL_SSL",  
    "sasl.username": "EZ2JWRQPD2TDKYQQ",  
    "sasl.password": "zMCP7SYUEv+XkX8+fqgro1AbKmCGhsMD3be320IS7JXslH06Iyv+b3KBcwYT+VxP",  
    "broker.version.fallback": "0.10.0.0",  
    "api.version.fallback.ms": 0  
}
```

```
# Topic to produce messages to  
topic = '3D_topic'
```

Kafka Producer

cette fonction `delivery_report` est utilisée pour gérer les rapports de livraison des messages Kafka. Si un message est livré avec succès, elle affiche des informations sur le sujet et la partition. Si la livraison échoue, elle affiche des informations sur l'erreur qui s'est produite

```
# Callback function to check delivery status
def delivery_report(err, msg):
    if err is not None:
        print('Message delivery failed: {}'.format(err))
    else:
        print('Message delivered to {} [{}]'.format(msg.topic(), msg.partition()))
```

Kafka Producer

cette fonction prend des données JSON depuis l' URL, les transforme en un DataFrame Pandas, effectue diverses opérations de nettoyage et de transformation sur les données, puis renvoie le DataFrame résultant prêt à être utilisé .

```
def cleaned_data(url):  
    r = requests.get(url)  
    data = r.json()  
  
    df = pd.DataFrame.from_dict(data['Time Series (Daily)']).T  
  
    df = df.reindex(index=df.index[::-1])  
    df.columns = [col.split('.')[1] for col in df.columns]  
    df.reset_index(inplace=True)  
    df.rename(columns={'index': 'date'}, inplace=True)  
  
    df['date'] = pd.to_datetime(df['date'])  
    df['open'] = pd.to_numeric(df['open'])  
    df['high'] = pd.to_numeric(df['high'])  
    df['low'] = pd.to_numeric(df['low'])  
    df['close'] = pd.to_numeric(df['close'])  
    df['volume'] = pd.to_numeric(df['volume'])  
  
    df = df.drop(["open", "high", "low", "volume", 'date'], axis = 1)  
    df = df.reset_index(drop=True)  
  
    return df
```

Kafka Producer

ce code produit un message contenant des données JSON dans un sujet Kafka en utilisant un producteur Kafka. Il s'assure également que le message est livré avec succès en utilisant une fonction de rappel pour gérer les rapports de livraison, puis affiche un message de confirmation une fois la livraison réussie.

```
# Create a Kafka producer instance
producer = Producer(producer_conf)

# Produce a message to the topic
df = cleaned_data(url)
df_json = df.to_json(date_format='iso', orient='split')

producer.produce(topic, df_json.encode('utf-8'), callback=delivery_report)

# Wait for any outstanding messages to be delivered and delivery reports to be received
producer.flush()

print("Message produced successfully.")
```

Kafka consumer


- Consommer des données depuis Apache Kafka implique de lire des messages ou des événements à partir des sujets Kafka en utilisant un consommateur Kafka.
- Les consommateurs Kafka sont des applications ou des processus qui s'abonnent à un ou plusieurs sujets Kafka et récupèrent des données en temps réel ou à un rythme spécifié.

```
consumer_conf = {  
    "bootstrap.servers": "pkc-921jm.us-east-2.aws.confluent.cloud:9092",  
    "sasl.mechanisms": "PLAIN",  
    "security.protocol": "SASL_SSL",  
    "sasl.username": "EZ2JWRQPD2TDKYQQ",  
    "sasl.password": "zMCP7SYUEv+XkX8+fqgro1AbKmCGhsMD3be320IS7JXs1H06Iyv+b3KBcwYT+VxP",  
    "group.id": "spring-boot-avro-consumer-id",  
    "auto.offset.reset": "earliest"    # Start consuming from the beginning of the topic  
}
```




Kafka consumer

1. `Consumer(consumer_conf)` : Crée une instance de consommateur Kafka en utilisant la classe `Consumer` de `confluent-kafka-python`. Cette fonction initialise le consommateur avec la configuration définie dans `consumer_conf`.
2. `consumer.subscribe([topic])` : Utilise la méthode `subscribe` de l'instance du consommateur pour s'abonner au(x) sujet(s) Kafka spécifié(s) dans la liste `[topic]`. Cette fonction indique au consommateur les sujets auxquels il doit écouter les messages.



```
# Create a Kafka consumer instance
consumer = Consumer(consumer_conf)
# Subscribe to the topic
consumer.subscribe([topic])
```

Kafka consumer

Ce code déclare une fonction `prediction_results` qui réalise des prédictions sur des données temporelles en utilisant le modèle ARIMA. Il divise les données en ensembles d'entraînement et de test, ajuste le modèle ARIMA aux données d'entraînement, puis génère des prédictions pour les données de test. Le modèle est mis à jour à chaque étape avec les nouvelles observations, permettant ainsi de fournir une prévision pour le jour suivant. Cette fonction est utile pour les prévisions en séries temporelles en utilisant ARIMA.

```
def prediction_results(data):
    train_data, test_data = data[:int(len(data)*0.9)], data[int(len(data)*0.9):]

    training_data = train_data['close'].values
    test_data = test_data['close'].values

    history = [x for x in training_data]
    model_predictions = []
    N_test_observations = len(test_data)

    model = ARIMA(history, order=(1, 1, 0))
    model_fit = model.fit()

    for time_point in range(N_test_observations):
        true_test_value = test_data[time_point]

        history.append(true_test_value)

        model = ARIMA(history, order=(1, 1, 0))
        model_fit = model.fit()

        yhat = model_fit.forecast(steps=1)[0]

        model_predictions.append(yhat)

    next_day_prediction = model_fit.forecast(steps=1)[0]

    return next_day_prediction
```


Kafka consumer

Ce code met en place un processus de consommation et de traitement en temps réel de messages provenant d'un sujet Kafka. Il utilise une boucle continue pour surveiller les messages Kafka, gère les erreurs lors de la réception et décode les messages au format UTF-8. Ensuite, il transforme les données JSON en un DataFrame Pandas et les soumet à une fonction de prédiction. Les résultats des prédictions sont ensuite affichés. Le code est conçu pour une exécution continue, avec une gestion robuste des erreurs et une fermeture propre du consommateur Kafka une fois le traitement terminé.

```
while True:
    message = consumer.poll(1.0)
    if message is None:
        continue

    if message.error():
        print("Error:", message.error())
        continue

    try:
        value = message.value().decode('utf-8')
        print("Received message")

        dfs = pd.read_json(value, orient='split')
        print("\nDataFrame created:\n", dfs)

        predictions = prediction_results(dfs)
        print("\nPredicted Price for the Next Day:", predictions)
        break

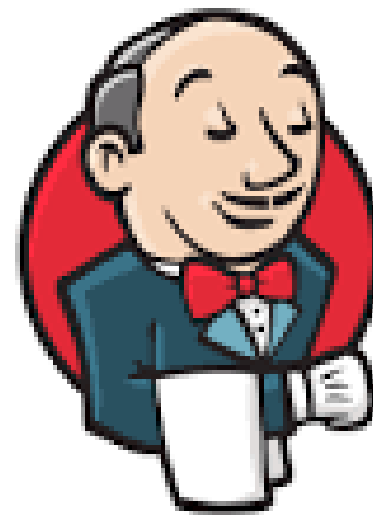
    except Exception as e:
        print("Error:", e)

consumer.close()
```



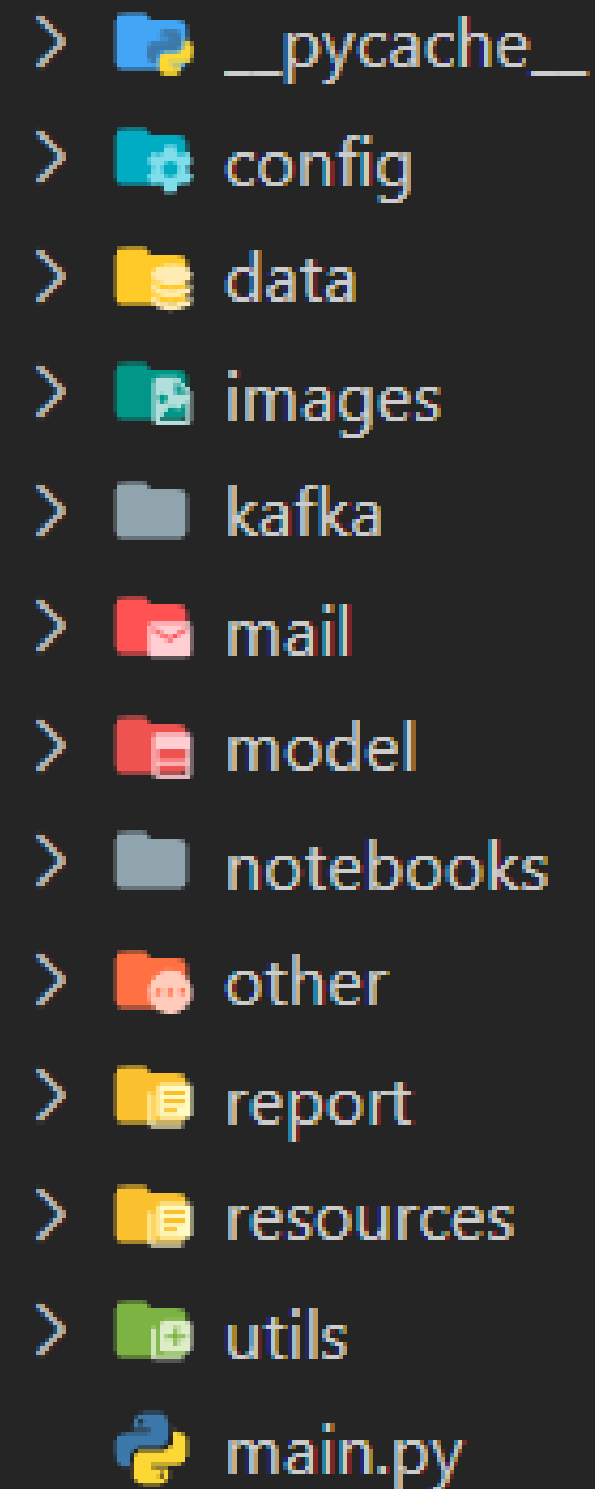
Automatisation














Choix de l'Outil d'Automatisation



Jenkins

Structure du Projet



- >  __pycache__
- >  config
- >  data
- >  images
- >  kafka
- >  mail
- >  model
- >  notebooks
- >  other
- >  report
- >  resources
- >  utils
-  main.py

Planification de l'Exécution Automatisée

☒ Construire périodiquement ?

Planning ?

TZ=Africa/Casablanca
30 15 * * *



Fonction '**send_email**'

Processus d'Automatisation

```
def main():
    # Initialize the logger
    logger = setup_logging("Main", "3d_project_log.txt")

    logger.info("Starting the main program...")
    # Load configuration from JSON files
    path = "C:/Users/a/Desktop/3D/"

    try:
        logger.info("Loading configurations.")

        with open(path+'config/data_config.json', 'r') as data_config_file:
            data_config = json.load(data_config_file)

        with open(path+'config/kafka_config.json', 'r') as kafka_config_file:
            kafka_config = json.load(kafka_config_file)

        with open(path+'config/email_config.json', 'r') as email_config_file:
            email_config = json.load(email_config_file)

        logger.info("Configurations Loaded")

    except FileNotFoundError as e:
        logger.error(f"Config file not found: {e}")

    except json.JSONDecodeError as e:
        logger.error(f"Error parsing JSON: {e}")

    producer(kafka_config["producer_config"], kafka_config["topic"], data_config)

    # Perform ARIMA prediction
    next_day_prediction = consumer(kafka_config["consumer_config"], kafka_config["topic"])

    # Send an email notification
    send_email(email_config, data_config['SYMBOL'], next_day_prediction)

    logger.info("Main script execution finished.")
    logger.info("Have a great day!")
```



Démonstration



MERCI

POUR VOTRE ATTENTION