Central Bank communication and asset prices

Auteur : Sougnez, Elia
Promoteur(s) : Hambuckers, Julien
Faculté : HEC-Ecole de gestion de l'Université de Liège
Diplôme : Master en sciences de gestion, à finalité spécialisée en Banking and Asset Management
Année académique : 2024-2025
URI/URL : http://hdl.handle.net/2268.2/24273

Code

Scripts python

Script de web scraping

```
#Importation des librairies
import os
import time
import requests
import io
import logging as lg
import re
import pandas as pd
from bs4 import BeautifulSoup
from PyPDF2 import PdfReader
from selenium import webdriver
from selenium.webdriver.firefox.options import Options
from selenium.webdriver.firefox.service import Service

lg.basicConfig(level=lg.INFO)

#Configuration
GECKO_PATH = "/usr/local/bin/geckodriver"
OUTPUT_FILE = "data/ecb_speeches_aligned_final.csv"

#Webdriver
def create_webdriver(headless=True):
    options = Options()
    if headless:
        options.add_argument("--headless")
    service = Service(GECKO_PATH)
    driver = webdriver.Firefox(service=service, options=options)
    return driver

def scroll_page(driver, speed=50):
    last_height = driver.execute_script("return document.body.scrollHeight")
    for i in range(0, last_height, speed):
        driver.execute_script(f"window.scrollTo(0, {i});")
    time.sleep(1)

#Scraper la page principale
def extract_index_page(url):
    driver = create_webdriver(headless=True)
    driver.get(url)
    scroll_page(driver)
    page = driver.page_source
    driver.quit()
```

```
    soup = BeautifulSoup(page, "lxml")
    base_url = url.split("/press")[0]

    dates = [dt['isodate'] for dt in soup.find_all("dt", isodate=True)]
    divs  = soup.find_all("div", class_="title")
    langs = soup.find_all("div", class_="ecb-langSelector")

    links = [base_url + div.find("a")["href"] if div.find("a") else None for div in divs]
    langs = [lang.find("a")["lang"] if lang.find("a") else None for lang in langs]

    return list(zip(dates, links, langs))

#Télécharger le contenu du lien
def extract_speech_from_url(url, date=None):
    try:
        if "shared/pdf" in url:
            response = requests.get(url, timeout=10)
            pdf = PdfReader(io.BytesIO(response.content))
            text = " ".join(page.extract_text() for page in pdf.pages if page.extract_text())
        else:
            response = requests.get(url, timeout=10)
            soup = BeautifulSoup(response.text, "lxml")
            main = soup.select_one("main")
            if not main:
                return ""
            qa_div = main.find(id="qa")
            if qa_div:
                main_str = str(main)
                qa_str = str(qa_div)
                main_cut = BeautifulSoup(main_str.split(qa_str)[0], "lxml")
                paragraphs = main_cut.find_all("p")
            else:
                paragraphs = main.find_all("p")

            text = " ".join(p.get_text(strip=True) for p in paragraphs)

    #Gestion des coupures du texte (enlever la session de questions-réponses)
            cut_markers = [
                "We are now at your disposal for questions.",
                "We are now ready to take your questions.",
                "We stand ready to take your questions.",
                "Before we take your questions",
                "I am now at your disposal for questions.",
                "I am now open to questions.",
                " * * * "
            ]
            if date == "2011-10-06":
                start_marker = "But now, Mr. President, the floor is yours."
                start_idx = text.find(start_marker)
                end_idx = -1
                for marker in cut_markers:
                    if marker in text:
```

```
                end_idx = text.find(marker)
                break
        if start_idx != -1:
            text = text[start_idx + len(start_marker): end_idx if end_idx != -1 else None]

        for marker in cut_markers:
            if marker in text:
                text = text.split(marker)[0]
                break

#Nettoyage des espaces manquants
    text = re.sub(r'([a-z])([A-Z])', r'\1 \2', text)
    return text.strip()

  except Exception as e:
    lg.warning(f"Erreur pour {url} : {e}")
    return ""

#Pipeline principal
def scrap_speeches(lang_filter=["en"]):
  base_url = "https://www.ecb.europa.eu/press/pressconf/html/index.en.html"
  index_data = extract_index_page(base_url)

  records = []
  for date, url, lang in index_data:
    if not date or not url or not lang:
      continue
    if lang not in lang_filter:
      continue

    year = int(date[:4])
    speech = extract_speech_from_url(url, date)

    if speech:
      records.append({
        "date": date,
        "url": url,
        "language": lang,
        "year": year,
        "speech": speech
      })
      lg.info(f"{date} récupéré")
    else:
      lg.warning(f"{date} vide ou échoué")

  df = pd.DataFrame(records)
  os.makedirs("data", exist_ok=True)
  df.to_csv(OUTPUT_FILE, index=False)
  lg.info(f"{len(df)} discours enregistrés dans {OUTPUT_FILE}")

if __name__ == "__main__":
  scrap_speeches(lang_filter=["en"])
```

#Nettoyage et formatage des données

```
base_path = os.path.expanduser("~/Desktop/code2")
input_file = os.path.join(base_path, "ecb_speeches_aligned_final.csv")
output_file = os.path.join(base_path, "processed_date_data6ok.csv")

df = pd.read_csv(input_file)

# Extraction des dates à partir de l'URL
date_parts = df['url'].str.extract(r'is(\d{2})(\d{2})(\d{2})')
date_parts.columns = ['year', 'month', 'day']

# Conversion en format (année-mois-jour)
df['correct_date'] = pd.to_datetime(
   date_parts['year'] + date_parts['month'] + date_parts['day'],
   format='%y%m%d',
   errors='coerce'
)

# Mise à jour de la colonne 'year'
df['year'] = df['correct_date'].dt.year

# Suppression des doublons de date
df = df.drop_duplicates(subset='correct_date', keep='first')

# Tri chronologique
df = df.sort_values(by='correct_date').reset_index(drop=True)

#Suppression de la dernière ligne si nécessaire
df = df.iloc[:-1]

#Suppression éventuelle des colonnes inutiles
df = df.drop(columns=['date', 'language'], errors='ignore')

#Première colonne 'correct_date'
cols = ['correct_date'] + [col for col in df.columns if col != 'correct_date']
df = df[cols]

# Sauvegarde du fichier CSV final
df.to_csv(output_file, index=False)
print(f"\nFichier sauvegardé : {output_file}")

#Vérification du type de la colonne 'correct_date'
print("Type de la colonne 'correct_date' :", df['correct_date'].dtype)

# Aperçu du contenu
print(df.head())
print(df.tail())
```

Pré-traitement des données pour l'analyse avec le lexique

```
#Importation des librairies
import re
import nltk
import pandas as pd
from pathlib import Path
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.util import ngrams

# télécharger les ressources NLTK
# nltk.download('punkt')
# nltk.download('stopwords')

STOP_WORDS = set(stopwords.words('english'))
STEMMER    = PorterStemmer()

def preprocess_field_specific(text, ngram_range=(1, 10)):

 # mise en minuscules et nettoyage des caractères indésirables
   text = text.lower()
   text = re.sub(r'[^a-z\s]', ' ', text)
   text = re.sub(r'\[.*?\]|\(.*?\)', ' ', text)
   tokens = nltk.word_tokenize(text)

   # suppression des stop-words et application du stemmer
   tokens = [w for w in tokens if w not in STOP_WORDS]
   tokens = [STEMMER.stem(w) for w in tokens]

#Segmenter le texte en n-grammes
   all_ngrams = []
   min_n, max_n = ngram_range
   for n in range(min_n, max_n + 1):
      for g in ngrams(tokens, n):
         all_ngrams.append(' '.join(g))
   return all_ngrams

if __name__ == '__main__':
   base_dir    = Path.home() / 'Desktop' / 'code2'
   raw_path    = base_dir / 'processed_date_data6ok.csv'
   output_path = base_dir / 'processed_ngramsok.csv'

   # lecture et nettoyage des noms de colonnes
   df = pd.read_csv(raw_path, encoding='utf-8-sig')
   df.columns = df.columns.str.strip().str.lower()


 # renommer la colonne correct_date → date
   if 'correct_date' in df.columns:
      df.rename(columns={'correct_date': 'date'}, inplace=True)
```

```
else:
    raise KeyError("La colonne 'correct_date' est introuvable dans votre CSV")

rows = []
for _, row in df.iterrows():
    date   = row['date']
    speech = str(row['speech_clean'])
    for ng in preprocess_field_specific(speech):
        rows.append({
            'date': date,
            'taille_ngram': len(ng.split()),
            'ngram': ng
        })

# Ecriture du résultat
pd.DataFrame(rows) \
  .to_csv(output_path, index=False, encoding='utf-8-sig')

print("preprocessing done →", output_path)
```

Prétraitement des données pour l'analyse avec FinBERT

```
#Importation des librairies
import os
import re
import pandas as pd
import spacy

def remove_commas(phrase: str) -> str:

    #Supprime les virgules précédant une conjonction contrastive
    et nettoie les espaces/virgules en début de phrase.

    phrase = re.sub(
        r',\s*(?=(?:but|although|while|though)\b)',
        '',
        phrase,
        flags=re.IGNORECASE
    )
    return phrase.lstrip(', ').strip()

def sentiment_focus(phrase: str) -> str:

    #Applique la méthode Sentiment Focus.

    low = phrase.lower()
    if 'but' in low:
        idx = low.find('but')
        return phrase[idx + len('but'):].lstrip(' ,;:').strip()

    for conj in ('although', 'though'):
        if conj in low:
            if low.startswith(conj):
                pos = phrase.find(',')
                if pos != -1:
                    return phrase[pos+1:].lstrip(' ,').strip()
            return phrase[:low.find(conj)].strip()

    if 'while' in low:
        if low.startswith('while'):
            pos = phrase.find(',')
            if pos != -1:
                return phrase[pos+1:].lstrip(' ,').strip()
        return phrase[:low.find('while')].strip()

    return phrase.strip()

def process_text(speech: str, nlp) -> list[str]:

    #Segmente un texte en phrases, supprime les virgules,
    applique la méthode Sentiment Focus et renvoie la liste des phrases traitées.
```

```
    doc = nlp(speech)
    result = []
    for sent in doc.sents:
        s = sent.text.strip()
        s_clean = remove_commas(s)
        s_sf    = sentiment_focus(s_clean)
        result.append(s_sf)
    return result

def main():
    # --- Chemins des fichiers ---
    home = os.path.expanduser("~")
    input_csv  = os.path.join(home, "desktop", "code2", "processed_date_data6ok.csv")
    output_csv = os.path.join(home, "desktop", "code2", "preprocessed.csv")

    # Chargement des données
    df = pd.read_csv(input_csv)
    # Vérifie que j'ai bien la colonne 'speech' et la colonne 'date'
    if 'speech' not in df.columns or 'correct_date' not in df.columns:
        raise KeyError("Le CSV d'entrée doit contenir les colonnes 'speech' et 'date'.")

    # --- Initialisation spaCy ---
    nlp = spacy.load("en_core_web_sm")

    # --- Traitement phrase par phrase ---
    all_sentences = []
    # itère sur chaque ligne, récupère speech et date
    for _, row in df.dropna(subset=['speech']).iterrows():
        speech = row['speech']
        date   = row['correct_date']
        for ph in process_text(speech, nlp):
            all_sentences.append({
                "correct_date": date,
                "processed_sentence": ph
            })

    # --- Export CSV de sortie ---
    out_df = pd.DataFrame(all_sentences)
    out_df.to_csv(output_csv, index=False, encoding='utf-8')
    print(f"{len(out_df)} phrases sauvées dans : {output_csv}")

if __name__ == "__main__":
    main()
```

#second script

```
#Importation des librairies
import os
import re
import pandas as pd
```

```
import spacy
from spacy.lang.en.stop_words import STOP_WORDS

# Mots de négation à conserver
NEGATION_WORDS = {"not", "no", "never", "none", "nobody", "nothing", "neither", "nowhere",
"nor", "hardly", "barely", "scarcely", "less", "without", "cannot"}

#Liste de mots à filtrer en plus des stop-words
CUSTOM_STOP_WORDS = {
   'ecb', 'basis', 'cet', 'utc', 'gmt',
   'governing', 'council', 'let', 'say',
   'draghi', 'lagarde', 'trichet', 'mr',
   'ms', 'papademos', 'junker', 'euro',
   'area', 'january', 'february', 'march',
   'april', 'june', 'july', 'august',
   'september', 'october', 'november',
   'december', 'meeting', 'ii', 'iii', 'iv'
}

# stop-words ajustés
ADJUSTED_STOP_WORDS = STOP_WORDS.difference(NEGATION_WORDS)

def clean_phrase(phrase: str, nlp) -> str:
   text = phrase.lower()
   text = re.sub(r'[\d]', ' ', text)
   text = re.sub(r'[^\w\s]', ' ', text)
   doc = nlp(text)
   tokens = []
   for tok in doc:
      if (
         not tok.is_space
         and tok.text not in ADJUSTED_STOP_WORDS
         and tok.text not in CUSTOM_STOP_WORDS
         and len(tok.text) > 1
      ):
         tokens.append(tok.lemma_)
   return " ".join(tokens)

def main():
   home = os.path.expanduser("~")
   input_csv  = os.path.join(home, "desktop", "code2", "preprocessed.csv")
   output_csv = os.path.join(home, "desktop", "code2", "preprocessed_cleanedok.csv")

   df = pd.read_csv(input_csv)
   if 'processed_sentence' not in df.columns:
      raise KeyError("La colonne 'processed_sentence' est introuvable.")

   # Supprimer la première phrase par date
   if 'correct_date' not in df.columns:
      raise KeyError("La colonne 'correct_date' est introuvable.")
   df = df.groupby('correct_date', group_keys=False).apply(lambda x: x.iloc[1:])
```

```
    nlp = spacy.load("en_core_web_sm", disable=["parser", "ner"])

    # nettoyage de chaque phrase
    cleaned = []
    for sent in df['processed_sentence'].fillna(""):
        cleaned.append(clean_phrase(sent, nlp))

    df['cleaned_sentence'] = cleaned
    df.to_csv(output_csv, index=False, encoding='utf-8')
    print(f" {len(df)} phrases nettoyées enregistrées dans {output_csv}")

if __name__ == "__main__":
    main()
```

Analyse de sentiment avec le lexique

```
#Importation des librairies
import pandas as pd
from pathlib import Path

#Garde dans ngrams_list uniquement les n-grammes présents dans lex_set, et parmi ceux qui se recoupent, ne conserve que les plus longs.

def filter_and_longest(ngrams_list, lex_set):

    found = [ng for ng in ngrams_list if ng in lex_set]
    kept  = []
    for ng in sorted(found, key=lambda x: -len(x.split())):
        if not any(ng in other for other in kept):
            kept.append(ng)
    return kept

#Applique la formule de Picault et Renault (2017) pour calculer les probabilités

def calc_probs(ngram_counts, lex_df, prefix):

    # on récupère les colonnes du lexique pour la catégorie prefix (mp_* ou ec_*)
    cols = [c for c in lex_df.columns if c.startswith(prefix + '_')]
    sub = lex_df.loc[ngram_counts.index, cols].fillna(0)
    # on pondère chaque ligne par count (occurrence)
    weighted = sub.multiply(ngram_counts, axis=0)
    # numérateur par inclinaison i
    numerators = weighted.sum(axis=0)
    # dénominateur : somme tous i
    denom = numerators.sum()
    if denom > 0:
        return (numerators / denom).to_dict()
    else:
        # Si pas de n-grammes trouvés : probabilités nulles
        return {c: 0.0 for c in cols}

def indice_MP(probs):
    # I_s^{MP} = P_s^{MP,hawkish} – P_s^{MP,dovish}
    return probs.get('mp_rest', 0.0) - probs.get('mp_acco', 0.0)

def indice_EC(probs):
    # I_s^{EC} = P_s^{EC,positive} – P_s^{EC,negative}
    return probs.get('ec_posi', 0.0) - probs.get('ec_nega', 0.0)

if __name__ == '__main__':
    #chemins
    base_dir    = Path.home() / 'Desktop' / 'code2'
    ngrams_path = base_dir / 'processed_ngramsok.csv'
    lex_path    = base_dir / 'export_lexicon.csv'
    output_path = base_dir / 'sentiment_indices_final.csv'
```

```
# chargement
df_ngrams = pd.read_csv(ngrams_path, encoding='utf-8-sig')
lex       = pd.read_csv(lex_path, sep=';', encoding='utf-8-sig')

# agrégation : compter chaque (date, ngram)
df_counts = (
   df_ngrams
   .groupby(['date', 'ngram'])
   .size()
   .reset_index(name='count')
)

# préparation du lexique
lex = lex.rename(columns={'keyword':'ngram', 'ngram':'ngram_len'})
lex = lex.set_index('ngram')
lex_set = set(lex.index)

# boucle de calcul des indices
results = []
for date, grp in df_counts.groupby('date'):
   # serie : index = ngram, valeur = count
   counts = grp.set_index('ngram')['count']

   #filtrage des n-grammes
   kept = filter_and_longest(counts.index.tolist(), lex_set)
   doc_ngram_counts = counts.loc[kept]

   #pondération
   probs_mp = calc_probs(doc_ngram_counts, lex, 'mp')
   probs_ec = calc_probs(doc_ngram_counts, lex, 'ec')

   #calcul des indices finaux
   results.append({
      'date': date,
      'I_MP': indice_MP(probs_mp),
      'I_EC': indice_EC(probs_ec),
      'n_ngrams_utilisés': len(doc_ngram_counts)
   })

#sauvegarde
pd.DataFrame(results).to_csv(output_path, index=False, encoding='utf-8-sig')
print("Analysis complete →", output_path)
```

Analyse de sentiment avec FinBERT

 # 1er script
#Importation des librairies

```
import os
import pandas as pd
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

#Charge et retourne le tokenizer et le modèle pour classification.

def load_model(model_name: str):

    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model     = AutoModelForSequenceClassification.from_pretrained(model_name)
    model.eval()
    return tokenizer, model

  # Pour une phrase, renvoie :
     - best_label  : label à probabilité maximale (positif, négatif, neutre)
     - probs_dict  : dict {label: probability}

def predict_label_and_probs(text: str, tokenizer, model):
    inputs = tokenizer(
        text,
        return_tensors="pt",
        truncation=True,
        max_length=512
    )
    with torch.no_grad():
        logits = model(**inputs).logits.squeeze(0)
        probs  = torch.softmax(logits, dim=0).cpu().tolist()

    labels = [model.config.id2label[i] for i in range(len(probs))]
    probs_dict = dict(zip(labels, probs))
    best_label = labels[int(torch.argmax(torch.tensor(probs)).item())]
    return best_label, probs_dict

def main():
    # chemins
    home       = os.path.expanduser("~")
    code2_dir   = os.path.join(home, "Desktop", "code2")
    input_csv   = os.path.join(code2_dir, "preprocessed_cleanedok.csv")
    output_csv  = os.path.join(code2_dir, "preprocessedESG_SENT_scored.csv")


    df = pd.read_csv(input_csv)
    if 'cleaned_sentence' not in df.columns:
        raise KeyError("Le CSV d'entrée doit contenir la colonne 'cleaned_sentence'.")
```

```
    # On conserve la colonne 'correct_date'
    has_date = 'correct_date' in df.columns

    #chargement des modèles FinBERT
    esg_tokenizer,  esg_model  = load_model("yiyanghkust/finbert-esg")
    sent_tokenizer, sent_model = load_model("ZiweiChen/FinBERT-FOMC")

    records = []
    for _, row in df.iterrows():
        text = row['cleaned_sentence']
        if not isinstance(text, str) or not text.strip():
            continue

        # classification ESG
        esg_label,  esg_probs  = predict_label_and_probs(text, esg_tokenizer,  esg_model)
        # classification sentiment
        sent_label, sent_probs = predict_label_and_probs(text, sent_tokenizer, sent_model)
        rec = {
            "cleaned_sentence": text,
            "esg_label":        esg_label,
            "sent_label":       sent_label,
        }
        # ajoute les probabilités détaillées
        rec.update({f"esg_{lbl}":  p for lbl, p in esg_probs.items()})
        rec.update({f"sent_{lbl}": p for lbl, p in sent_probs.items()})

        if has_date:
            rec["correct_date"] = row["correct_date"]

        records.append(rec)

    #écriture du CSV de sortie
    out_df = pd.DataFrame(records)
    # si date était présente, remet-la en première colonne
    if has_date:
        cols = ["correct_date"] + [c for c in out_df.columns if c != "correct_date"]
        out_df = out_df[cols]

    out_df.to_csv(output_csv, index=False, encoding="utf-8")
    print(f"{len(out_df)} phrases traitées, résultats enregistrés dans :\n  {output_csv}")

if __name__ == "__main__":
main()
```

```
#2e script : calcul des indices de sentiment
#Importation des librairies
import os
import pandas as pd
import numpy as np

def main():
    #chemins
    base_dir     = os.path.expanduser("~/Desktop/Code2")
    input_csv    = os.path.join(base_dir, "preprocessedESG_SENT_scored.csv")
    output_csv   = os.path.join(base_dir, "ESG_scores_simplemean.csv")

    #Chargement des données
    df = pd.read_csv(input_csv, parse_dates=["correct_date"])
    # Remplacement des valeurs manquantes
    df["esg_label"]  = df["esg_label"].fillna("None")
    df["sent_label"] = df["sent_label"].fillna("Neutral")

    #Filtrage sur les dimensions ESG et sentiments (sans 'none' et neutre)
    dims      = ["Environmental", "Social", "Governance"]
    sentiments = ["Positive", "Negative"]
    df_filt = df[df["esg_label"].isin(dims) & df["sent_label"].isin(sentiments)]

    #Comptage des phrases positives et négatives par date et dimension ESG
    counts = (
        df_filt
        .groupby(["correct_date", "esg_label", "sent_label"])
        .size()
        .unstack(fill_value=0)
    )

    #Construction du DataFrame des scores normalisés
    #    I_{D,s} = (N_pos - N_neg) / (N_pos + N_neg)
    dates = counts.index.get_level_values(0).unique().sort_values()
    scores = pd.DataFrame(index=dates)
    scores.index.name = "correct_date"

    for dim in dims:
        # Extraire les comptes positifs et négatifs pour la dimension
        pos = counts.loc[(slice(None), dim), "Positive"].droplevel(1)
        neg = counts.loc[(slice(None), dim), "Negative"].droplevel(1)

        # Calcul de l'indice
        idx = (pos - neg) / (pos + neg)
        scores[f"I_{dim[0]}"] = idx


  #Remplacement des éventuels NaN par 0 sur l'ensemble des scores
    scores = scores.fillna(0)

    #Calcul de l'indice ESG global (moyenne des trois composantes)
    scores['I_ESG'] = scores[['I_E', 'I_S', 'I_G']].mean(axis=1)
```

```
    #Sauvegarde du résultat
    scores.to_csv(output_csv, index=True)
    print(f" Indices ESG (incluant I_ESG) sauvegardés dans {output_csv}")

if __name__ == "__main__":
    main()
```

Scripts R : Analyse de régression

Équation 8

```
#importation des librairies
library(readr)
library(readxl)
library(dplyr)
library(lubridate)

#chemins
setwd("~/Desktop/code2")
rm(list=ls())

#chargement des données
sentiment_data <- read_csv("sentiment_indices_final.csv") %>%
  mutate(date = as.Date(date))  # conversion au format Date

euro_data <- read_excel("eurostoxx50.xlsx") %>%
  mutate(Date = as.Date(Date, format = "%d/%m/%y")) %>%
  arrange(Date) %>%
  select(Date, Return) %>%
  mutate(
    R_tm1 = lag(Return), #Rendement à t-1
    R_tp1 = lead(Return), #Rendement à t+1
    R_tp2 = lead(Return, 2) #Rendement à t+2
  )

gdp_dates <- read_excel("gdp_cleaned.xlsx") %>%
  mutate(Date_GDP = as.Date(Date_GDP))

hicp_dates <- read_excel("HICP.xlsx") %>%
  mutate(Date_HICP = as.Date(Date_HICP))

  #Création des variables de contrôles
  sentiment_data <- sentiment_data %>%
  mutate(
    gdp_event = ifelse(
      date %in% gdp_dates$Date_GDP |
        (date - 1) %in% gdp_dates$Date_GDP |
        (date + 1) %in% gdp_dates$Date_GDP,
      1, 0
    ),
```

```
   hicp_event = ifelse(
     date %in% hicp_dates$Date_HICP |
       (date - 1) %in% hicp_dates$Date_HICP |
       (date + 1) %in% hicp_dates$Date_HICP,
     1, 0
   )
 )
#Jointure des dates
  conference_data <- sentiment_data %>%
  inner_join(euro_data, by = c("date" = "Date")) %>%
  rename(R_t = Return) %>%
  filter(!is.na(R_tm1) & !is.na(R_t) & !is.na(R_tp1) & !is.na(R_tp2))

#Vérifications
cat("Conférences avec date de marché trouvée :", nrow(conference_data), "\n")
print(head(conference_data))

dates_non_match <- anti_join(sentiment_data, euro_data, by = c("date" = "Date")) %>% pull(date)
cat("Conférences sans donnée de marché :", length(dates_non_match), "→",
paste(head(dates_non_match), collapse = ", "), "\n")

 #Régressions
model_d0 <- lm(R_t ~ R_tm1 + I_MP + I_EC + gdp_event + hicp_event, data = conference_data)

model_d1 <- lm(R_tp1 ~ R_t + I_MP + I_EC + gdp_event + hicp_event, data = conference_data)

model_d2 <- lm(R_tp2 ~ R_tp1 + I_MP + I_EC + gdp_event + hicp_event, data = conference_data)


#Résultats
cat("\nRégression pour d = 0 (jour de la conférence) :\n")
print(summary(model_d0))

cat("\nRégression pour d = 1 (lendemain de la conférence) :\n")
print(summary(model_d1))

cat("\nRégression pour d = 2 (deux jours après la conférence) :\n")
print(summary(model_d2))


#Export des résultats avec stargazer
library(stargazer)

stargazer(model_d0, model_d1, model_d2,
        type = "html",
        out = "resultats.html",
        title = "Résultats des régressions (d = 0, d = 1 et d = 2)",
        align = TRUE,
        digits = 4,
        no.space = TRUE)
```

Équation 9

```
#Importation des librairies
library(readr)    # pour read_csv
library(readxl)   # pour read_excel
library(dplyr)    # manipulation de données

setwd("~/Desktop/code2")
rm(list=ls())

#chargement des données
sentiment_data <- read_csv("sentiment_indices_final.csv") %>%
  mutate(date = as.Date(date))  # format Date

vstoxx_data <- read_excel("vstoxx.xlsx") %>%
  mutate(Date = as.Date(Date)) %>%
  arrange(Date) %>%
  select(Date, Variation) %>%
  mutate(
    V_tm1 = lag(Variation),
    V_tp1 = lead(Variation)
  )

gdp_dates <- read_excel("gdp_cleaned.xlsx") %>%
  mutate(Date_GDP = as.Date(Date_GDP))

hicp_dates <- read_excel("HICP.xlsx") %>%
  mutate(Date_HICP = as.Date(Date_HICP))

#Création des variables de contrôle
sentiment_data <- sentiment_data %>%
  mutate(
    gdp_event = ifelse(
      date %in% gdp_dates$Date_GDP |
        (date - 1) %in% gdp_dates$Date_GDP |
        (date + 1) %in% gdp_dates$Date_GDP,
      1, 0
    ),
    hicp_event = ifelse(
      date %in% hicp_dates$Date_HICP |
        (date - 1) %in% hicp_dates$Date_HICP |
        (date + 1) %in% hicp_dates$Date_HICP,
      1, 0
    )
  )

# 2. Jointure des dates
conference_data <- sentiment_data %>%
  inner_join(vstoxx_data, by = c("date" = "Date")) %>%
  rename(V_t = Variation) %>%
  filter(!is.na(V_tm1) & !is.na(V_t) & !is.na(V_tp1))
```

#Régressions

```
model_d0 <- lm(V_t ~ V_tm1 + I_MP + I_EC + gdp_event + hicp_event, data = conference_data)

model_d1 <- lm(V_tp1 ~ V_t + I_MP + I_EC + gdp_event + hicp_event, data = conference_data)
```

#Résultats

```
cat("\nRégression VSTOXX pour d = 0 (jour de la conférence) :\n")
print(summary(model_d0))

cat("\nRégression VSTOXX pour d = 1 (lendemain de la conférence) :\n")
print(summary(model_d1))
```

Équation 10

```
#Importation des librairies
library(readr)
library(readxl)
library(dplyr)
library(stargazer)

setwd("~/Desktop/code2")
rm(list = ls())

#Chargement des données
sentiment_data <- read_csv("sentiment_indices_final.csv") %>%
  mutate(date = as.Date(date))

euro_data <- read_excel("eurostoxx50.xlsx") %>%
  mutate(Date = as.Date(Date)) %>%
  arrange(Date) %>%
  select(Date, Return)

ff_data <- read_excel("FFok.xlsx") %>% #Données fama-french
  mutate(Date = as.Date(Date)) %>%
  arrange(Date) %>%
  filter(!is.na(Mkt))

#Construction autour des conférences
conference_data <- sentiment_data %>%
  inner_join(euro_data, by = c("date" = "Date")) %>%
  rename(R_t = Return) %>%
  inner_join(ff_data, by = c("date" = "Date")) %>%
  mutate(
    R_t_excess = R_t - rf
  )

#Vérifications
cat("Nombre de conférences utilisées :", nrow(conference_data), "\n")
dates_non_match <- anti_join(sentiment_data, euro_data, by = c("date" = "Date")) %>% pull(date)
cat("Conférences sans données de marché :", length(dates_non_match), "→",
paste(head(dates_non_match), collapse = ", "), "\n")

#Régressions
model_d0_FF <- lm(R_t_excess ~ I_MP + I_EC + Mkt + SMB + HML, data = conference_data)

#Résultats
cat("\nRégression Fama-French (d = 0) :\n")
print(summary(model_d0_FF))
```

Équation 11

```
#Importation des librairies
library(readr)
library(readxl)
library(dplyr)
library(stargazer)

#Chargement des données
setwd("~/Desktop/code2")
rm(list = ls())

sentiment_data <- read_csv("sentiment_indices_final.csv") %>%
  mutate(date = as.Date(date))

euro_data <- read_excel("eurostoxx50.xlsx") %>%
  mutate(Date = as.Date(Date)) %>%
  arrange(Date) %>%
  select(Date, Return)

ff_data <- read_excel("FFok.xlsx") %>% #Données fama-french
  mutate(Date = as.Date(Date)) %>%
  arrange(Date) %>%
  filter(!is.na(Mkt))

# Construction autour des dates de conférences
conference_data <- sentiment_data %>%
  inner_join(euro_data, by = c("date" = "Date")) %>%
  rename(R_t = Return) %>%
  inner_join(ff_data, by = c("date" = "Date")) %>%
  mutate(
    R_t_excess = R_t - rf
  )

#Vérifications
cat("Nombre de conférences utilisées :", nrow(conference_data), "\n")
dates_non_match <- anti_join(sentiment_data, euro_data, by = c("date" = "Date")) %>% pull(date)
cat("Conférences sans données de marché :", length(dates_non_match), "→",
paste(head(dates_non_match), collapse = ", "), "\n")

#Régressions
model_d0_FF <- lm(R_t_excess ~ I_MP + I_EC + Mkt + SMB + HML, data = conference_data)

#Résultats
cat("\nRégression Fama-French (d = 0) :\n")
print(summary(model_d0_FF))
```