Apéndice

Algoritmos en Python

En este apartado se pueden consultar -*clasificados por capítulo*- los códigos python desarrollados para el análisis, la obtención de resultados y la presentación gráfica del estudio de impacto de AirBnb en los mercados de acciones de hospitalidad y la finca raíz en Cartagena de Indias, Colombia.

A continuación se muestran los códigos python del capítulo 2, correspondiente a la metodología de estudio de eventos.

C3.1 - Figura 4 - Gráfico 6 acciones de hotelería.

```
# Define the ticker list
import pandas as pd
import matplotlib.pyplot as plt
import os
import contextlib

# Disable
def blockPrint():
  sys.stdout = open(os.devnull, 'w')
# Restore
def enablePrint():
  sys.stdout = sys.__stdout__

tickers_list = ['^GSPC', 'HLT','MAR','H', 'BKNG', 'HST', 'DRH']
# Import pandas
data = pd.DataFrame(columns=tickers_list)
# Fetch the data
import yfinance as yf
from yahoofinancials import YahooFinancials as YF

# blockPrint()
for ticker in tickers_list:
  with open(os.devnull, "w") as f, contextlib.redirect_stdout(f):
    data[ticker] = yf.download(ticker, '2013-12-12', '2019-09-30')['Adj Close']
# enablePrint()

# Plot all the close prices
((data.pct_change()+1).cumprod()).plot(figsize=(10, 7))
# Show the legend
L=plt.legend()

i = 0
for ticker in tickers_list:
  co_name = YF(ticker).get_stock_quote_type_data()[ticker]['shortName']
  L.get_texts()[i].set_text(ticker+" > "+co_name)
```

```
  i+=1

# Define the label for the title of the figure
figuresNum = 4
plt.title("Figura %i - Precio Diario Cierre 6 Acciones Hospitality"%figuresNum)
# Define the labels for x-axis and y-axis
plt.ylabel('precio')
plt.xlabel('t')
# Plot the grid lines
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```

C3.2 - Figura 5 - Momentos para un estudio de eventos.

```
%%itikz
\documentclass{beamer}
\newcommand{\ImageWidth}{15cm}
\usepackage{tikz}
\usepackage{upgreek}
\usetikzlibrary{decorations.pathreplacing,positioning, arrows.meta}
\geometry{paperheight=3.3cm}
\beamertemplatenavigationsymbolsempty

\begin{document}
\begin{tikzpicture}
% draw horizontal line
\draw[thick, -Triangle] (0,0) -- (\ImageWidth,0) node[font=\scriptsize,below left=3pt and -8pt]{years};

% draw vertical lines
\foreach \x in {1, 4, 6, 9}
\draw (\x cm,3pt) -- (\x cm,-3pt);

\foreach \x/\descr in {1/T_{0},4/T_{1},5/0,6/T_{2},9/T_{3}}
\node[font=\scriptsize, text height=1.75ex,
text depth=.5ex] at (\x,-.3) {$\descr$};

% colored bar up
\foreach \x/\perccol in
{1/100,1.5/75,2/25}
\draw[lightgray!\perccol!red, line width=4pt]
(\x,.5) -- +(1,0);
\draw[-Triangle, dashed, red] (3,.5) --  +(1,0);

% colored bar down
\foreach \x/\perccol in
{4/100,4.3/75,4.7/50}
\draw[lightgray!\perccol!green, line width=4pt]
(\x,-.7) -- +(0.5,0);
\draw[-Triangle, dashed, green] (5,-.7) --  +(1,0);

% colored bar down
\foreach \x/\perccol in
{6/100,6.5/75,7/25}
\draw[lightgray!\perccol!orange, line width=4pt]
(\x,.5) -- +(1,0);
```

```
\draw[-Triangle, dashed, orange] (8,.5) --  +(1,0);

% braces
\draw [thick ,decorate,decoration={brace,amplitude=5pt}] (1,0.8)  -- +(3,0)
  node [black,midway,above=4pt, font=\scriptsize] {Ventana de Estimaci\'{o}n};
\draw [thick,decorate,decoration={brace,amplitude=5pt}] (6,-1) -- +(-2,0)
  node [black,midway,font=\scriptsize, below=5pt] {Ventana del Evento};
\draw [color=white] (6,0) -- +(-2,0)
  node [black,midway,font=\scriptsize, above=3pt]{$\uptau$};
\draw [color=white] (6,0) -- +(-2,0)
  node [black,midway,font=\scriptsize, above=15pt]{AirBnb $\uparrow$};
\draw [thick,decorate,decoration={brace,amplitude=5pt}] (6,0.8) -- +(3,0)
  node [black,midway,font=\scriptsize, above=5pt] {Ventana Post-Evento};

\end{tikzpicture}
\end{document}
```

C3.3 - Algoritmo de predicción.

```
def runPrediction(df_train, df_predict):
  # clone df_predict same index
  df_abnor_ret = df_predict.copy()[[]]
  df_abnor_ret.drop(df_abnor_ret.head(1).index, inplace=True)
  sp_500 = '^GSPC'
  for col in df_train.columns:
    if(col != sp_500):
      # joining the closing prices of the two datasets
      daily_prices = pd.concat([df_train[col], df_train[sp_500]], axis=1)
      daily_prices.columns = [col, sp_500]
      daily_returns = daily_prices.pct_change(1).dropna(axis=0)

      daily_prices_predict = pd.concat([df_predict[col], df_predict[sp_500]], axis=1)
      daily_returns_predict = daily_prices_predict.pct_change(1).dropna(axis=0)

      clean_daily_returns = daily_returns.dropna(axis=0)  # drop first missing row
      # split dependent and independent variable
      X = daily_returns[sp_500]
      y = daily_returns[col]
      slope, intercept, r_value, p_value, std_err = stats.linregress(X, y)
      df_abnor_ret[col] =  daily_returns_predict[col] - intercept - slope*daily_returns_predict[sp_500]
      df_abnor_ret[col] = df_abnor_ret[col].rolling(3).sum() + df_abnor_ret[col].shift(-1) +
df_abnor_ret[col].shift(-2)

  df_abnor_ret = df_abnor_ret.dropna()
  return df_abnor_ret
```

C3.4 - Figura 6 - Ventana móvil para modelo CAPM de retornos anormales.

```
df_px = pd.read_csv("https://drive.google.com/uc?export=download&id=1zqDUgqYN5sovTe7grkNsHF0PA7e8hLTM",
parse_dates=True, index_col='Fecha')

final_date = df_px.index[-1]
final_idx = df_px.index.get_loc(final_date)
start_date = df_px.index[0]
start_idx = df_px.index.get_loc(start_date)
```

```
end_date = start_date + timedelta(days=365)
end_idx = df_px.index.get_loc(end_date)

df_capm_diffs = df_px.copy()[[]]

# 5 trading days intervals + (-2, 2) trading days abnormal returns accumulation
while( (end_idx + 8) <= final_idx ):
  gap = - start_idx + end_idx
  pgress = 100*start_idx/(final_idx-gap)
  strPg = 'Progress %.2f%%'%pgress
  sys.stdout.write("\r"+strPg)
  df_train_range = df_px.iloc[ start_idx:end_idx ]
  df_predict_range = df_px.iloc[ end_idx-3:end_idx+7 ]
  #print(strPg,df_train_range.index[0]," - ",df_train_range.index[-1])
  # predict to estimate market deviation
  df_capm_diffs = df_capm_diffs.append(runPrediction(df_train_range, df_predict_range), sort=False)
  start_idx += 5
  end_idx += 5

sys.stdout.write(" ... end")
df_capm_diffs = df_capm_diffs.dropna()
```

C3.5 - Scraping de inmuebles en venta en Cartagena.

A continuación se muestra como se realizó web scraping del portal www.fincaraiz.com.co con la ayuda de la clase BeautifulSoup de la librería bs4.

```
from bs4 import BeautifulSoup
import json
import pandas as pd
import requests
import re
import time

def aptRowFincaRaiz(url):
  response= requests.get(url)
  soup = BeautifulSoup(response.content,"html.parser")

  pattern = re.compile(".*var sfAdvert.*")
  scripts = soup.findAll('script', text=pattern)

  aptInfoFull = scripts[0].text
  start = 'var sfAdvert = '
  end = '};'

  dictStr = (aptInfoFull.split(start))[1].split(end)[0]+"}"
  aptInfoRow = json.loads(dictStr)
  return aptInfoRow

aptsCtgDf = None
keepScrapping = True
pageCount = 1
apCount = 7500
```

```
while(keepScrapping):
  urlFincaRaiz = ("https://www.fincaraiz.com.co/apartamentos/venta/cartagena/?ad=30|"+str(pageCount)+
                  "||||1||8|||58|5800003|||||||||||||||||1|||1||griddate%20desc||||-1||")
  print(urlFincaRaiz)
  response= requests.get(urlFincaRaiz)
  soup = BeautifulSoup(response.content,"html.parser")

  listings = soup.findAll('img', {"src" : "/app_theme/images/none.gif"})

  for l in listings:
    onStr = l.get('onclick')
    if(onStr!=None):
      url = "https://www.fincaraiz.com.co"+(onStr.split("javascript:location.href='"))[1].split("'")[0]
      print(str(apCount)+" > "+url)
      apCount += 1
      row = aptRowFincaRaiz(url)
      if(aptsCtgDf is None):
        aptsCtgDf = pd.DataFrame([row])
      else:
        aptsCtgDf = aptsCtgDf.append([row])

  nextPage = soup.findAll('a', {"title" : "Ir a la pagina Siguiente", "class" : "link-pag "})
  print(nextPage)
  if(len(nextPage)==0):
    keepScrapping = False

  pageCount += 1

print(aptsCtgDf)
```

C6.3 - Extracción de los alojamientos de AirBnb.

Dado que el sitio web solo muestra los resultados en lotes de 300 y cuando detecta muchas consultas de un mismo origen puede bloquear la dirección ip, se requirió de una técnica más avanzada de scraping llamada web crawling - throttling [https://docs.scrapy.org/en/latest/intro/overview.html] que para evitar ser catalogado como un robot, programáticamente lanza varios hilos de ejecución simulando el comportamiento de usuarios habituales del sitio web.

El siguiente código es una implementación basada en la librería Scrapy de python y en el código propuesto en el repositorio GITHUB: https://github.com/kailu3/airbnb-scraper.

```
import json
import collections
import re
import numpy as np
import logging
import sys
import pprint
import time
import scrapy
from scrapy_splash import SplashRequest
from scrapy.exceptions import CloseSpider
from airbnb_scraper.items import AirbnbScraperItem
```

```
# ********************************************************************************************
# Important: Run -> docker run -p 8050:8050 scrapinghub/splash in background before crawling *
# ********************************************************************************************
# ********************************************************************************************
# Run crawler with -> scrapy crawl airbnb -o 21to25.json -a price_lb='' -a price_ub=''        *
# ********************************************************************************************

class AirbnbSpider(scrapy.Spider):
    name = 'airbnb'
    allowed_domains = ['www.airbnb.com']

    '''
    You don't have to override __init__ each time and can simply use self.parameter (See
https://bit.ly/2Wxbkd9),
    but I find this way much more readable.
    '''
    def __init__(self, city='',price_lb='', price_ub='', *args,**kwargs):
        super(AirbnbSpider, self).__init__(*args, **kwargs)
        self.city = city
        self.price_lb = price_lb
        self.price_ub = price_ub

    def start_requests(self):
        '''Sends a scrapy request to the designated url price range

        Args:
        Returns:
        '''

        url = ('https://www.airbnb.com/api/v2/explore_tabs?_format=for_explore_search_web&_intents=p1'
               '&allow_override%5B%5D=&auto_ib=false&client_session_id='

'621cf853-d03e-4108-b717-c14962b6ab8b&currency=COP&experiences_per_grid=20&fetch_filters=true'
               '&guidebooks_per_grid=20&has_zero_guest_treatment=true&is_guided_search=true'
               '&is_new_cards_experiment=true&is_standard_search=true&items_per_grid=18'

'&key=d306zoyjsyarp7ifhu67rjxn52tv0t20&locale=en&luxury_pre_launch=false&metadata_only=false&'
               'query={1}'
               '&query_understanding_enabled=true&refinement_paths%5B%5D=%2Fhomes&s_tag=QLb9RB7g'

'&search_type=FILTER_CHANGE&selected_tab_id=home_tab&show_groupings=true&supports_for_you_v3=true'
               '&timezone_offset=-240&version=1.5.6'
               '&price_min={0}')
        new_url = url.format(self.price_lb, self.city)

        logging.log(logging.WARNING, '>>>>>>>>>>>>>>>> PRICE UP: '+self.price_lb)

        yield scrapy.Request(url=new_url, callback=self.parse_id, dont_filter=True)


    def parse_id(self, response):
        '''Parses all the URLs/ids/available fields from the initial json object and stores into dictionary

        Args:
            response: Json object from explore_tabs
        Returns:
```

```
        '''
        # Fetch and Write the response data
        data = json.loads(response.body)

        #logging.log(logging.DEBUG, ('data >>>>>>>>> ',
data.get('explore_tabs')[0].get('sections')[1].get('listings')))
        # Return a list of all homes
        homes = data.get('explore_tabs')[0].get('sections')[0].get('listings')

        if homes is None:
            try:
                homes = data.get('explore_tabs')[0].get('sections')[1].get('listings')
            except IndexError:
                try:
                    homes = data.get('explore_tabs')[0].get('sections')[2].get('listings')
                except:
                    raise CloseSpider("No homes available in the city and price parameters")

        logging.log(logging.WARNING, '>>>>>>>>>>>>>>>>>> Homes: '+str(len(homes)))
        base_url = 'https://www.airbnb.com/rooms/'
        data_dict = collections.defaultdict(dict) # Create Dictionary to put all currently available fields
in

        for home in homes:
            room_id = str(home.get('listing').get('id'))
            url = base_url + str(home.get('listing').get('id'))
            data_dict[room_id]['url'] = url
            data_dict[room_id]['price'] = home.get('pricing_quote').get('rate').get('amount')
            data_dict[room_id]['bathrooms'] = home.get('listing').get('bathrooms')
            data_dict[room_id]['bedrooms'] = home.get('listing').get('bedrooms')
            data_dict[room_id]['host_languages'] = home.get('listing').get('host_languages')
            data_dict[room_id]['is_business_travel_ready'] =
home.get('listing').get('is_business_travel_ready')
            data_dict[room_id]['is_fully_refundable'] = home.get('listing').get('is_fully_refundable')
            data_dict[room_id]['is_new_listing'] = home.get('listing').get('is_new_listing')
            data_dict[room_id]['is_superhost'] = home.get('listing').get('is_superhost')
            data_dict[room_id]['lat'] = home.get('listing').get('lat')
            data_dict[room_id]['lng'] = home.get('listing').get('lng')
            data_dict[room_id]['localized_city'] = home.get('listing').get('localized_city')
            data_dict[room_id]['localized_neighborhood'] =
home.get('listing').get('localized_neighborhood')
            data_dict[room_id]['listing_name'] = home.get('listing').get('name')
            data_dict[room_id]['person_capacity'] = home.get('listing').get('person_capacity')
            data_dict[room_id]['picture_count'] = home.get('listing').get('picture_count')
            data_dict[room_id]['reviews_count'] = home.get('listing').get('reviews_count')
            data_dict[room_id]['room_type_category'] = home.get('listing').get('room_type_category')
            data_dict[room_id]['star_rating'] = home.get('listing').get('star_rating')
            data_dict[room_id]['host_id'] = home.get('listing').get('user').get('id')
            data_dict[room_id]['avg_rating'] = home.get('listing').get('avg_rating')
            data_dict[room_id]['can_instant_book'] = home.get('pricing_quote').get('can_instant_book')
            data_dict[room_id]['monthly_price_factor'] =
home.get('pricing_quote').get('monthly_price_factor')
            data_dict[room_id]['currency'] = home.get('pricing_quote').get('rate').get('currency')
            data_dict[room_id]['amt_w_service'] =
home.get('pricing_quote').get('rate_with_service_fee').get('amount')
            data_dict[room_id]['rate_type'] = home.get('pricing_quote').get('rate_type')
```

```
            data_dict[room_id]['weekly_price_factor'] =
home.get('pricing_quote').get('weekly_price_factor')


        # Iterate through dictionary of URLs in the single page to send a SplashRequest for each
        for room_id in data_dict:
            yield SplashRequest(url=base_url+room_id, callback=self.parse_details,
                                meta=data_dict.get(room_id),
                                endpoint="render.html",
                                args={'wait': '0.5'})

        # After scraping entire listings page, check if more pages
        pagination_metadata = data.get('explore_tabs')[0].get('pagination_metadata')
        if pagination_metadata.get('has_next_page'):

            items_offset = pagination_metadata.get('items_offset')
            section_offset = pagination_metadata.get('section_offset')

            new_url =
('https://www.airbnb.com/api/v2/explore_tabs?_format=for_explore_search_web&_intents=p1'
                        '&allow_override%5B%5D=&auto_ib=false&client_session_id='
                        '621cf853-d03e-4108-b717-c14962b6ab8b&currency=COP&experiences_per_grid=20'

'&fetch_filters=true&guidebooks_per_grid=20&has_zero_guest_treatment=true&is_guided_search=true'
                        '&is_new_cards_experiment=true&is_standard_search=true&items_per_grid=18'

'&key=d306zoyjsyarp7ifhu67rjxn52tv0t20&locale=en&luxury_pre_launch=false&metadata_only=false'
                        '&query={3}'
                        '&query_understanding_enabled=true&refinement_paths%5B%5D=%2Fhomes&s_tag=QLb9RB7g'
                        '&satori_version=1.1.9&screen_height=797&screen_size=medium&screen_width=885'

'&search_type=FILTER_CHANGE&selected_tab_id=home_tab&show_groupings=true&supports_for_you_v3=true'
                        '&timezone_offset=-240&version=1.5.6'
                        '&items_offset={0}&section_offset={1}&price_min={2}')
            new_url = new_url.format(items_offset, section_offset, self.price_lb, self.city)

            # If there is a next page, update url and scrape from next page
            yield scrapy.Request(url=new_url, callback=self.parse_id)

    def parse_details(self, response):
        '''Parses details for a single listing page and stores into AirbnbScraperItem object

        Args:
            response: The response from the page (same as inspecting page source)
        Returns:
            An AirbnbScraperItem object containing the set of fields pertaining to the listing
        '''
        # New Instance
        listing = AirbnbScraperItem()

        # Fill in fields for Instance from initial scrapy call
        listing['is_superhost'] = response.meta['is_superhost']
        listing['host_id'] = str(response.meta['host_id'])
        listing['price'] = response.meta['price']
        listing['url'] = response.meta['url']
        listing['bathrooms'] = response.meta['bathrooms']
```

```
        listing['bedrooms'] = response.meta['bedrooms']
        listing['is_business_travel_ready'] = response.meta['is_business_travel_ready']
        listing['is_fully_refundable'] = response.meta['is_fully_refundable']
        listing['is_new_listing'] = response.meta['is_new_listing']
        listing['lat'] = response.meta['lat']
        listing['lng'] = response.meta['lng']
        listing['localized_city'] = response.meta['localized_city']
        listing['localized_neighborhood'] = response.meta['localized_neighborhood']
        listing['listing_name'] = response.meta['listing_name']
        listing['person_capacity'] = response.meta['person_capacity']
        listing['picture_count'] = response.meta['picture_count']
        listing['reviews_count'] = response.meta['reviews_count']
        listing['room_type_category'] = response.meta['room_type_category']
        listing['star_rating'] = response.meta['star_rating']
        listing['avg_rating'] = response.meta['avg_rating']
        listing['can_instant_book'] = response.meta['can_instant_book']
        listing['monthly_price_factor'] = response.meta['monthly_price_factor']
        listing['weekly_price_factor'] = response.meta['weekly_price_factor']
        listing['currency'] = response.meta['currency']
        listing['amt_w_service'] = response.meta['amt_w_service']
        listing['rate_type'] = response.meta['rate_type']

        # Other fields scraped from html response.text using regex (some might fail hence try/catch)
        try:
            listing['num_beds'] = int((re.search('"bed_label":"(.).*","bedroom_label"',
response.text)).group(1))
        except:
            listing['num_beds'] = 0

        try:
            listing['host_reviews'] = int((re.search(r'"badges":\[{"count":(.*?),"id":"reviews"',
                                    response.text)).group(1))
        except:
            listing['host_reviews'] = 0

        # Main six rating metrics + overall_guest_satisfication
        try:
            listing['accuracy'] = int((re.search('"accuracy_rating":(.*?),"', response.text)).group(1))
            listing['checkin'] = int((re.search('"checkin_rating":(.*?),"', response.text)).group(1))
            listing['cleanliness'] = int((re.search('"cleanliness_rating":(.*?),"',
response.text)).group(1))
            listing['communication'] = int((re.search('"communication_rating":(.*?),"',
response.text)).group(1))
            listing['value'] = int((re.search('"value_rating":(.*?),"', response.text)).group(1))
            listing['location'] = int((re.search('"location_rating":(.*?),"', response.text)).group(1))
            listing['guest_satisfication'] = int((re.search('"guest_satisfaction_overall":(.*?),"',
                                        response.text)).group(1))
        except:
            listing['accuracy'] = 0
            listing['checkin'] = 0
            listing['cleanliness'] = 0
            listing['communication'] = 0
            listing['value'] = 0
            listing['location'] = 0
            listing['guest_satisfication'] = 0
```

```
        # Extra Host Fields
        try:
            listing['response_rate'] = int((re.search('"response_rate_without_na":"(.*?)%",',
response.text)).group(1))
            listing['response_time'] = (re.search('"response_time_without_na":"(.*?)",',
response.text)).group(1)
        except:
            listing['response_rate'] = 0
            listing['response_time'] = ''

        # Finally return the object
        yield listing
```

C6.1 - Prueba Individual t-Student Simple.

```
import datetime
import matplotlib.pyplot as plt
# import the math module
import math
import numpy as np
from scipy.stats import t
# List stocks to analize
from prettytable import PrettyTable
# Sklearn
from sklearn.metrics import mean_squared_error

# [1] Simple t-Student test
def runARbyStock(df_train, df_predict, stock):
  # clone df_predict same index
  df_abnor_ret = df_predict.copy()[[]]
  df_abnor_ret.drop(df_abnor_ret.head(1).index, inplace=True)
  sp_500 = '^GSPC'
  # joining the closing prices of the two datasets
  daily_prices = pd.concat([df_train[stock], df_train[sp_500]], axis=1)
  daily_prices.columns = [stock, sp_500]
  daily_returns = daily_prices.pct_change(1).dropna(axis=0)

  daily_prices_predict = pd.concat([df_predict[stock], df_predict[sp_500]], axis=1)
  daily_returns_predict = daily_prices_predict.pct_change(1).dropna(axis=0)

  clean_daily_returns = daily_returns.dropna(axis=0)  # drop first missing row
  # split dependent and independent variable
  X = daily_returns[sp_500]
  y = daily_returns[stock]
  slope, intercept, r_value, p_value, std_err = stats.linregress(X, y)
  df_AR_ret = daily_returns.append(daily_returns_predict, sort=False)

  df_AR_ret[stock] =  df_AR_ret[stock] - intercept - slope*df_AR_ret[sp_500]

  return df_AR_ret[stock]

t1 = PrettyTable(['1a) Prueba t: Ho: CAR_i=0', 't_CAR', 'p-valor', 'Evento', 'Resultado (alfa=0.05)'])
t2 = PrettyTable(['1b) Prueba t: Ho: CAR_i=0', 't_CAR', 'p-valor', 'Evento', 'Resultado (alfa=0.05)'])

CARS_evt1 = []
```

```
CARS_evt2 = []

keys = dict_min_2_events.keys()
alfa = 0.05

# test 1st event
for i in keys:
  start_est_idx = df_px.index.get_loc(dict_min_2_events.get(i)[0] - timedelta(days=365), method ='ffill') -
3
  start_evt_idx = df_px.index.get_loc(dict_min_2_events.get(i)[0]) - 3
  end_est_idx = df_px.index.get_loc(dict_min_2_events.get(i)[0]) - 2
  end_evt_idx = df_px.index.get_loc(dict_min_2_events.get(i)[0]) + 3

  df_train_range = df_px.iloc[ start_est_idx:end_est_idx ]
  df_predict_range = df_px.iloc[ start_evt_idx:end_evt_idx ]

  # abnormal returns estimation and event window
  df_test = runARbyStock(df_train_range, df_predict_range, i)
  df = df_test.iloc[0:252]

  df = df*df

  # standard deviation abnormal returns estimation window
  S_AR = math.sqrt(sum(df)/(df.size-2))

  # event window
  L2 = 5
  # cumulative standard deviation
  S_CAR = math.sqrt(L2*S_AR*S_AR)
  # Cumulative abnormal returns
  CAR_i = df_test.iloc[-5:-1].sum() + df_test.tail(1).sum()
  # t statistic
  t_CAR = CAR_i/S_CAR

  pval = t.sf(np.abs(t_CAR), df.size-1)*2
  t1.add_row([i, t_CAR, pval, dict_min_2_events.get(i)[0].strftime("%Y-%m-%d"),'se rechaza Ho' if pval <
alfa else 'no se puede rechazar Ho'])
  # save CAR of stock i
  CARS_evt1.append(CAR_i)

# test 2nd event
for i in keys:
  start_est_idx = df_px.index.get_loc(dict_min_2_events.get(i)[1] - timedelta(days=365), method ='ffill') -
3
  start_evt_idx = df_px.index.get_loc(dict_min_2_events.get(i)[1]) - 3
  end_est_idx = df_px.index.get_loc(dict_min_2_events.get(i)[1]) - 2
  end_evt_idx = df_px.index.get_loc(dict_min_2_events.get(i)[1]) + 3

  df_train_range = df_px.iloc[ start_est_idx:end_est_idx ]
  df_predict_range = df_px.iloc[ start_evt_idx:end_evt_idx ]

  # abnormal returns estimation and event window
  df_test = runARbyStock(df_train_range, df_predict_range, i)
  df = df_test.iloc[0:252]

  df = df*df
```

```
  # standard deviation abnormal returns estimation window
  S_AR = math.sqrt(sum(df)/(df.size-2))

  # event window
  L2 = 5
  # cumulative standard deviation
  S_CAR = math.sqrt(L2*S_AR*S_AR)
  # Cumulative abnormal returns
  CAR_i = df_test.iloc[-5:-1].sum() + df_test.tail(1).sum()
  # t statistic
  t_CAR = CAR_i/S_CAR

  pval = t.sf(np.abs(t_CAR), df.size-1)*2
  t2.add_row([i, t_CAR, pval, dict_min_2_events.get(i)[1].strftime("%Y-%m-%d"), 'se rechaza Ho' if pval <
alfa else 'no se puede rechazar Ho'])
  # save CAR of stock i
  CARS_evt2.append(CAR_i)

print(t1)
```

C6.2 - Prueba conjunta t-Student de las acciones de hospitalidad.

```
# [2] Cross-Sectional Test
N = len(CARS_evt1)

S_CAAR_1 = math.sqrt(np.sum(np.square(CARS_evt1 - np.mean(CARS_evt1))))/(N-1))
S_CAAR_2 = math.sqrt(np.sum(np.square(CARS_evt2 - np.mean(CARS_evt2))))/(N-1))

t_CAAR_1 = math.sqrt(N)*np.mean(CARS_evt1)/S_CAAR_1
t_CAAR_2 = math.sqrt(N)*np.mean(CARS_evt2)/S_CAAR_2


pval_1 = t.sf(np.abs(t_CAAR_1), N-1)*2
pval_2 = t.sf(np.abs(t_CAAR_2), N-1)*2

t3 = PrettyTable(['2) Ho: CAAR=0', 't_CAAR', 'p-valor', 'Resultado (alfa=0.05)'])
t3.add_row(["Q1 2018", t_CAAR_1, pval_1, 'se rechaza Ho' if pval_1 < alfa else 'no se puede rechazar Ho'])
t3.add_row(["Q3 2018", t_CAAR_2, pval_2, 'se rechaza Ho' if pval_2 < alfa else 'no se puede rechazar Ho'])
print(t3)
```

C6.3 - Prueba de serie de tiempo con desviación estándar.

```
# [3] Time-Series Standard Deviation or Crude Dependence Test (Abbr.: CDA T)
def runSaarCaarTimeSeries(df_train, df_predict):
  # clone df_predict same index
  df_train_abnor_ret = df_train.copy()[[]]
  df_train_abnor_ret.drop(df_train_abnor_ret.head(1).index, inplace=True)
  df_predict_abnor_ret = df_predict.copy()[[]]
  df_predict_abnor_ret.drop(df_predict_abnor_ret.head(1).index, inplace=True)

  sp_500 = '^GSPC'
  for col in df_train.columns:
    if(col != sp_500):
      # joining the closing prices of the two datasets
```

```
      daily_prices = pd.concat([df_train[col], df_train[sp_500]], axis=1)
      daily_prices.columns = [col, sp_500]
      daily_returns = daily_prices.pct_change(1).dropna(axis=0)

      daily_prices_predict = pd.concat([df_predict[col], df_predict[sp_500]], axis=1)
      daily_returns_predict = daily_prices_predict.pct_change(1).dropna(axis=0)

      clean_daily_returns = daily_returns.dropna(axis=0)  # drop first missing row
      # split dependent and independent variable
      X = daily_returns[sp_500]
      y = daily_returns[col]
      slope, intercept, r_value, p_value, std_err = stats.linregress(X, y)
      df_train_abnor_ret[col] =  daily_returns[col] - intercept - slope*daily_returns[sp_500]
      df_predict_abnor_ret[col] =  daily_returns_predict[col] - intercept -
slope*daily_returns_predict[sp_500]
      df_train_abnor_ret = df_train_abnor_ret.dropna()
      df_predict_abnor_ret = df_predict_abnor_ret.dropna()
      #df_abnor_ret[col] = df_abnor_ret[col].rolling(3).sum() + df_abnor_ret[col].shift(-1) +
df_abnor_ret[col].shift(-2)

  df_train_abnor_ret['AAR'] = df_train_abnor_ret.mean(axis=1)
  df_predict_abnor_ret['AAR'] = df_predict_abnor_ret.mean(axis=1)

  AAR_MEAN = np.mean(df_train_abnor_ret['AAR'])
  S_AAR = np.sqrt( np.sum( np.square(df_train_abnor_ret['AAR'] - AAR_MEAN)
)/(len(df_train_abnor_ret['AAR'])-2) )
  CAAR = np.sum(df_predict_abnor_ret['AAR'])

  return S_AAR, CAAR

t4 = PrettyTable(['3a) Prueba t, serie de tiempo:', 'Ho: CAAR=0, t_CAAR', 'p-valor', 'Resultado
(alfa=0.05)'])
t5 = PrettyTable(['3b) Prueba t, serie de tiempo:', 'Ho: CAAR=0, t_CAAR', 'p-valor', 'Resultado
(alfa=0.05)'])

N = len(dict_min_2_events)
alfa = 0.05

CARS_evt1 = []
CARS_evt2 = []

evt1 = dict_min_2_events.get('HLT')[0]
evt2 = dict_min_2_events.get('MAR')[1]

st_est_idx_1 = df_px.index.get_loc(evt1 - timedelta(days=365), method ='ffill') - 3
st_est_idx_2 = df_px.index.get_loc(evt2 - timedelta(days=365), method ='ffill') - 3
st_evt_idx_1, st_evt_idx_2 = df_px.index.get_loc(evt1) - 3, df_px.index.get_loc(evt2) - 3
ed_est_idx_1, ed_est_idx_2 = df_px.index.get_loc(evt1) - 2, df_px.index.get_loc(evt2) - 2
ed_evt_idx_1, ed_evt_idx_2 = df_px.index.get_loc(evt1) + 3, df_px.index.get_loc(evt2) + 3

df_train_range = df_px.iloc[ st_est_idx_1:ed_est_idx_1 ]
df_predict_range = df_px.iloc[ st_evt_idx_1:ed_evt_idx_1 ]
# abnormal returns estimation and event window for time series standard deviation test
SAAR_1, CAAR_1 = runSaarCaarTimeSeries(df_train_range, df_predict_range)

df_train_range = df_px.iloc[ st_est_idx_2:ed_est_idx_2 ]
```

```
df_predict_range = df_px.iloc[ st_evt_idx_2:ed_evt_idx_2 ]
# abnormal returns estimation and event window for time series standard deviation test
SAAR_2, CAAR_2 = runSaarCaarTimeSeries(df_train_range, df_predict_range)

T2_T1 = ed_evt_idx_1 - ed_est_idx_1
t_CAAR_1 = CAAR_1/(SAAR_1*np.sqrt(T2_T1))
t_CAAR_2 = CAAR_2/(SAAR_2*np.sqrt(T2_T1))

pval_1 = t.sf(np.abs(t_CAAR_1), N-1)*2
pval_2 = t.sf(np.abs(t_CAAR_2), N-1)*2

# [3] Time-Series Standard Deviation or Crude Dependence Test (Abbr.: CDA T)
# estimation window = [-257, -3]
# event window =  [-2, 2]
print("Ventana Evento 1: Irrupción AirBnb en", evt1.strftime("%Y-%m-%d"))
t4.add_row([evt1, t_CAAR_1, pval_1, 'se rechaza Ho' if pval_1 < alfa else 'no se puede rechazar Ho'])
print(t4)
print("")
print("Ventana Evento 2: Irrupción AirBnb en", evt2.strftime("%Y-%m-%d"))
t5.add_row([evt2, t_CAAR_2, pval_2, 'se rechaza Ho' if pval_2 < alfa else 'no se puede rechazar Ho'])
print(t5)
```

C6.4 - Prueba residual estandarizada (Patell *Z*).

```
# [4] Patell or Standardized Residual Test (Abbr.: Patell Z)
def runPatellZbyStock(df_train, df_predict):
  # clone df_predict same index
  df_train_abnor_ret = df_train.copy()[[]]
  df_train_abnor_ret.drop(df_train_abnor_ret.head(1).index, inplace=True)
  df_predict_abnor_ret = df_predict.copy()[[]]
  df_predict_abnor_ret.drop(df_predict_abnor_ret.head(1).index, inplace=True)

  CSAR_i_evt = []

  sp_500 = '^GSPC'
  for col in df_train.columns:
    if(col != sp_500):
      # joining the closing prices of the two datasets
      daily_prices = pd.concat([df_train[col], df_train[sp_500]], axis=1)
      daily_prices.columns = [col, sp_500]
      daily_returns = daily_prices.pct_change(1).dropna(axis=0)

      daily_prices_predict = pd.concat([df_predict[col], df_predict[sp_500]], axis=1)
      daily_returns_predict = daily_prices_predict.pct_change(1).dropna(axis=0)

      clean_daily_returns = daily_returns.dropna(axis=0)  # drop first missing row
      # split dependent and independent variable
      X = daily_returns[sp_500]
      y = daily_returns[col]
      slope, intercept, r_value, p_value, std_err = stats.linregress(X, y)

      df_train_abnor_ret[col] =  daily_returns[col] - intercept - slope*daily_returns[sp_500]
      df_predict_abnor_ret[col] =  daily_returns_predict[col] - intercept -
slope*daily_returns_predict[sp_500]
      df_train_abnor_ret = df_train_abnor_ret.dropna()
```

```
      df_predict_abnor_ret = df_predict_abnor_ret.dropna()

      M_i = df_train_abnor_ret[col].size

      S_AR_i =  np.sqrt( np.square(df_train_abnor_ret[col]).sum() / (M_i - 2) )

      R_m_mean = daily_returns[sp_500].mean()

      df_predict_abnor_ret['S_AR_i_t'] = S_AR_i * np.sqrt(1 + 1/M_i +
np.square(daily_returns_predict[sp_500]-R_m_mean) / ((daily_returns[sp_500] - R_m_mean)**2).sum())

      CSAR = (df_predict_abnor_ret[col]/df_predict_abnor_ret['S_AR_i_t']).sum()

      CSAR_i_evt.append( CSAR )

  return CSAR_i_evt

t1 = PrettyTable(['4a) Prueba Patell: Ho: CAAR=0', 'Z_Patell', 'p-valor', 'Resultado (alfa=0.05)'])
t2 = PrettyTable(['4b) Prueba Patell: Ho: CAAR=0', 'Z_Patell', 'p-valor', 'Resultado (alfa=0.05)'])

evt1 = dict_min_2_events.get('HLT')[0]
evt2 = dict_min_2_events.get('MAR')[1]
alfa = 0.05
N = len(dict_min_2_events)

st_est_idx_1 = df_px.index.get_loc(evt1 - timedelta(days=365), method ='ffill') - 3
st_est_idx_2 = df_px.index.get_loc(evt2 - timedelta(days=365), method ='ffill') - 3
st_evt_idx_1, st_evt_idx_2 = df_px.index.get_loc(evt1) - 3, df_px.index.get_loc(evt2) - 3
ed_est_idx_1, ed_est_idx_2 = df_px.index.get_loc(evt1) - 2, df_px.index.get_loc(evt2) - 2
ed_evt_idx_1, ed_evt_idx_2 = df_px.index.get_loc(evt1) + 3, df_px.index.get_loc(evt2) + 3

L2 = ed_evt_idx_1 - ed_est_idx_1 - 1
M_i = ed_est_idx_1 - st_est_idx_1 - 1
S_CSAR_i_1 = np.sqrt(L2 * (M_i-2)/(M_i-4))

L2 = ed_evt_idx_2 - ed_est_idx_2 - 1
M_i = ed_est_idx_2 - st_est_idx_2 - 1
S_CSAR_i_2 = np.sqrt(L2 * (M_i-2)/(M_i-4))

df_train_range = df_px.iloc[ st_est_idx_1:ed_est_idx_1 ]
df_predict_range = df_px.iloc[ st_evt_idx_1:ed_evt_idx_1 ]
# 1st event test
CSAR_i_evt1 = runPatellZbyStock(df_train_range, df_predict_range)

df_train_range = df_px.iloc[ st_est_idx_2:ed_est_idx_2 ]
df_predict_range = df_px.iloc[ st_evt_idx_2:ed_evt_idx_2 ]
# 2nd event test
CSAR_i_evt2 = runPatellZbyStock(df_train_range, df_predict_range)

Z_Patell_1 = np.sum(CSAR_i_evt1)/(S_CSAR_i_1*np.sqrt(N))
Z_Patell_2 = np.sum(CSAR_i_evt2)/(S_CSAR_i_2*np.sqrt(N))

import scipy.stats as st

p_val_1 =  1 - st.norm.cdf(abs(Z_Patell_1))
p_val_2 = 1 - st.norm.cdf(abs(Z_Patell_2))
```

```
print("Ventana 1: Crecimiento AirBnb Q1 2018")
t1.add_row([evt1.strftime("%Y-%m-%d"), Z_Patell_1, p_val_1, 'se rechaza Ho' if p_val_1 < alfa else 'no se
puede rechazar Ho'])
print(t1)
print("")
print("Ventana 2: Crecimiento AirBnb Q3 2018")
t2.add_row([evt2.strftime("%Y-%m-%d"), Z_Patell_2, p_val_2, 'se rechaza Ho' if p_val_2 < alfa else 'no se
puede rechazar Ho'])
print(t2)
```

C6.5 - Prueba jerárquica de Corrado (Rank *Z*).

```
# [5] Corrado Rank Test (Abbr.: Rank Z)
def runCorradoRankTest(df_train, df_predict):
  # clone df_predict same index
  df_train_abnor_ret = df_train.copy()[[]]
  df_train_abnor_ret.drop(df_train_abnor_ret.head(1).index, inplace=True)
  df_predict_abnor_ret = df_predict.copy()[[]]
  df_predict_abnor_ret.drop(df_predict_abnor_ret.head(1).index, inplace=True)

  df_K_i_t_bar = df_train.append(df_predict).copy()[[]]

  sp_500 = '^GSPC'
  M_i = L_i = 0
  # compute abnormal returns ranks K_t_bar
  for col in df_train.columns:
    if(col != sp_500):
      # joining the closing prices of the two datasets
      daily_prices = pd.concat([df_train[col], df_train[sp_500]], axis=1)
      daily_prices.columns = [col, sp_500]
      daily_returns = daily_prices.pct_change(1).dropna(axis=0)

      daily_prices_predict = pd.concat([df_predict[col], df_predict[sp_500]], axis=1)
      daily_returns_predict = daily_prices_predict.pct_change(1).dropna(axis=0)

      clean_daily_returns = daily_returns.dropna(axis=0)  # drop first missing row
      # split dependent and independent variable
      X = daily_returns[sp_500]
      y = daily_returns[col]
      slope, intercept, r_value, p_value, std_err = stats.linregress(X, y)

      df_train_abnor_ret[col] =  daily_returns[col] - intercept - slope*daily_returns[sp_500]
      df_predict_abnor_ret[col] =  daily_returns_predict[col] - intercept -
slope*daily_returns_predict[sp_500]
      df_train_abnor_ret = df_train_abnor_ret.dropna()
      df_predict_abnor_ret = df_predict_abnor_ret.dropna()

      M_i = df_train_abnor_ret[col].size
      L_i = df_predict_abnor_ret[col].size

      df_K_i_t_bar[col] = (df_train_abnor_ret[col]).append(df_predict_abnor_ret[col])

      df_K_i_t_bar[col] = df_K_i_t_bar[col].rank()/(1+M_i+L_i)
```

```
  # ranks
  df_K_i_t_bar = df_K_i_t_bar.dropna()
  # K^_i_t
  df_K_i_t_bar['K_i_t_bar'] = df_K_i_t_bar.mean(axis = 1, skipna = True)
  # S2_K_var
  S2_K_var = (np.square(df_K_i_t_bar['K_i_t_bar']-0.5)).sum()/(M_i+L_i)
  # K_T1_T2_var
  K_T1_T2_var = (df_K_i_t_bar[M_i+1:M_i+L_i+1]['K_i_t_bar']).sum()/L_i
  # t_rank
  t_rank = (K_T1_T2_var-0.5)*np.sqrt(L_i/S2_K_var)

  return t_rank

t1 = PrettyTable(['5a) Prueba Corrado: Ho: CAAR=0', 't_rank', 'p-valor', 'Resultado (alfa=0.05)'])
t2 = PrettyTable(['5b) Prueba Corrado: Ho: CAAR=0', 't_rank', 'p-valor', 'Resultado (alfa=0.05)'])

evt1 = dict_min_2_events.get('HLT')[0]
evt2 = dict_min_2_events.get('MAR')[1]
alfa = 0.05
N = len(dict_min_2_events)

st_est_idx_1 = df_px.index.get_loc(evt1 - timedelta(days=365), method ='ffill') - 3
st_est_idx_2 = df_px.index.get_loc(evt2 - timedelta(days=365), method ='ffill') - 3
st_evt_idx_1, st_evt_idx_2 = df_px.index.get_loc(evt1) - 3, df_px.index.get_loc(evt2) - 3
ed_est_idx_1, ed_est_idx_2 = df_px.index.get_loc(evt1) - 2, df_px.index.get_loc(evt2) - 2
ed_evt_idx_1, ed_evt_idx_2 = df_px.index.get_loc(evt1) + 3, df_px.index.get_loc(evt2) + 3

df_train_range = df_px.iloc[ st_est_idx_1:ed_est_idx_1 ]
df_predict_range = df_px.iloc[ st_evt_idx_1:ed_evt_idx_1 ]
# 1st event test
t_RANK_1 = runCorradoRankTest(df_train_range, df_predict_range)

df_train_range = df_px.iloc[ st_est_idx_2:ed_est_idx_2 ]
df_predict_range = df_px.iloc[ st_evt_idx_2:ed_evt_idx_2 ]
# 2nd event test
t_RANK_2 = runCorradoRankTest(df_train_range, df_predict_range)

p_val_1 = t.sf(np.abs(t_RANK_1), N-1)*2
p_val_2 = t.sf(np.abs(t_RANK_2), N-1)*2
print("Ventana 1: Crecimiento AirBnb Q1 2018")
t1.add_row([evt1.strftime("%Y-%m-%d"), t_RANK_1, p_val_1, 'se rechaza Ho' if p_val_1 < alfa else 'no se
puede rechazar Ho'])
print(t1)
print("")
print("Ventana 2: Crecimiento AirBnb Q3 2018")
t2.add_row([evt2.strftime("%Y-%m-%d"), t_RANK_2, p_val_2, 'se rechaza Ho' if p_val_2 < alfa else 'no se
puede rechazar Ho'])
print(t2)
```