PROJET ARP

May 8, 2023

Groupe

- Lin Cécile 22200215
- Gensicke Ben-Bruno 22205565 (Erasmus)
- Multari Matteo 22205573 (Erasmus)
- Venturini Agnese 22205594 (Erasmus)

1 Alternative Risk Premia - Projet

L'objectif de ce projet est de tester si, contrairement aux prédictions du CAPM, le risque spécifique (ou idiosyncrasique) est rémunéré à l'équilibre.

La mise en évidence d'une éventuelle rémunération du risque spécifique sera réalisée en utilisant deux approches différentes : - la régression Fama-MacBeth (1973) des taux de rentabilités espérés sur les valeurs de et de risque spécifique de 25 portefeuilles constitués à partir des quintiles de et de risque spécifique ; - le calcul des taux de rentabilités espérés des 25 portefeuilles sur l'année

2 Import des packages nécessaire

[1]:
```
!pip install statsmodels
import math
import numpy as np, pandas as pd
import statsmodels.api as sm
from math import sqrt
import matplotlib.pyplot as plt
```

```
Requirement already satisfied: statsmodels in c:\users\lince\anaconda3\lib\site-
packages (0.12.2)
Requirement already satisfied: numpy>=1.15 in c:\users\lince\anaconda3\lib\site-
packages (from statsmodels) (1.20.1)
Requirement already satisfied: scipy>=1.1 in c:\users\lince\anaconda3\lib\site-
packages (from statsmodels) (1.6.2)
Requirement already satisfied: pandas>=0.21 in
c:\users\lince\anaconda3\lib\site-packages (from statsmodels) (1.2.4)
Requirement already satisfied: patsy>=0.5 in c:\users\lince\anaconda3\lib\site-
packages (from statsmodels) (0.5.1)
```

```
Requirement already satisfied: python-dateutil>=2.7.3 in
c:\users\lince\anaconda3\lib\site-packages (from pandas>=0.21->statsmodels)
(2.8.1)
Requirement already satisfied: pytz>=2017.3 in
c:\users\lince\anaconda3\lib\site-packages (from pandas>=0.21->statsmodels)
(2021.1)
Requirement already satisfied: six in c:\users\lince\anaconda3\lib\site-packages
(from patsy>=0.5->statsmodels) (1.15.0)
```

3 Importation des données CRSP

[2]:
```
path = "" # les données se trouvent dans le même fichier
ret = pd.read_csv(path + "CRSP all shares code 10 and 11 monthly 01-2000 to␣
 ↪12-2022.csv")
#ret
```

4 Nettoyage des données

Les données proviennent de la base CRSP.

- Seules les rentabilités des actions ordinaires doivent être utilisés pour l'étude. Les actions ordinaires sont celles dont la valeur du champ SHRCD (share code) est égale à 10 ou 11.

La structure des données est la suivante : - PERMNO : identifiant CRSP d'une action ; - date : format aaaa-mm-jj ; - SHRCD : code du type du titre (10 ou 11 pour les actions ordinaires) ; - RET : taux de rentabilité mensuel de l'action ; - vwretd : taux de rentabilité de l'indice pondéré par les capitalisations, calculé à partir de toutes les actions ordinaires recensées dans la base CRSP. Ce portefeuille est assimilable au portefeuille de marché.

Pour réaliser la régression Fama McBeth, nous avons besoin de: - une même période commune à chaque entreprise - ici on choisira la période la plus importante, i.e. entre 31/01/2000 et 30/12/2022 - pour cela, nous devons manipuler les dates du tableau - retirer toutes les entreprises n'étant pas coté sur cette période choisie - retirer toutes les entreprises ayant des valeurs manquantes NaN irremplaçable - nous avons decidé de remplacer les valeurs NaN lorsque c'est possible afin de conserver le maximum d'entreprise possible: - si un taux de rentabilité mensuel est manquant entre 2 dates nous approximerons le taux par la moyenne des taux mensuel du mois précédent et du mois suivant - s'il n'est pas possible de faire cette moyenne, alors nous retirerons l'entreprise de l'étude - pour cela, nous allons devoir manipuler la colonne des dates - transformer la date au format yyyy-mm-dd en yyyymmdd - ajouter une colonne de période pour chaque année

[3]:
```
#remplacer une date au format yyyy-mm-dd en yyyymmdd

from datetime import datetime

# Définir une fonction pour transformer chaque date en yyyymmdd
def transform_date(date):
    return int(datetime.strptime(date, '%Y-%m-%d').strftime('%Y%m%d'))
```

```
# Appliquer la fonction à chaque élément de la colonne "date"
ret['date'] = ret['date'].apply(transform_date)

# Trouve la première année d'observation
minYear = int(ret['date'].min()/10000)

# Ajoute une colonne period
# la première période est l'année 2000 donc chaque date de la forme 2000XXXX␣
 ↪doit avoir comme période 1
# l'année 2001 donc chaque date de la forme 2001XXXX doit avoir comme période 2
# etc.
ret['period'] = (ret['date']/10000).astype(int) - minYear + 1

# Afficher le tableau résultant
ret
```

[3]:

	PERMNO	date	SHRCD	RET	vwretd	period
0	10001	20000131	11.0	-0.044118	-0.039622	1
1	10001	20000229	11.0	0.015385	0.031760	1
2	10001	20000331	11.0	-0.015758	0.053499	1
3	10001	20000428	11.0	0.011719	-0.059519	1
4	10001	20000531	11.0	-0.023166	-0.038863	1
...	...	...	...	...	...	...
1233787	93436	20220831	11.0	-0.072489	-0.036240	23
1233788	93436	20220930	11.0	-0.037589	-0.091324	23
1233789	93436	20221031	11.0	-0.142168	0.077403	23
1233790	93436	20221130	11.0	-0.144326	0.052365	23
1233791	93436	20221230	11.0	-0.367334	-0.057116	23

[1233792 rows x 6 columns]

4.0.1 Trouvons la première et la dernière date de cotation de chaque entreprise

Rappel: on souhaite que les entreprises soient cotées chaque mois entre le 31/01/2000 et le 30/12/2022

[4]:
```
start_date = 20000131
end_date = 20221230

# on cherche les dates de début et de fin de cotation pour chaque entreprise
date_ranges = {}
for i in ret["PERMNO"].unique():
    entreprise_i = ret[ret["PERMNO"] == i] # on ne garde que les entreprises de␣
 ↪PERMNO = i
    date_ranges[i] = (entreprise_i["date"].min(), entreprise_i["date"].max()) #␣
 ↪(date de début, date de fin)
```

```
date_ranges = np.transpose(pd.DataFrame(date_ranges)) # afin d'avoir un tableau
date_ranges.reset_index(inplace = True)

date_ranges.columns = ['PERMNO','début','fin']
date_ranges

# on ne garde que les entreprises cotés dès l'année 2000 à 2022
entreprise_2000_2022 = date_ranges['PERMNO'][(date_ranges['début'] ==␣
 ↪start_date) & (date_ranges['fin'] == end_date)]

ret = ret[ret['PERMNO'].isin(entreprise_2000_2022)]
ret = ret.reset_index(drop = True)
ret
```

[4]:

	PERMNO	date	SHRCD	RET	vwretd	period
0	10026	20000131	11.0	-0.067073	-0.039622	1
1	10026	20000229	11.0	-0.035948	0.031760	1
2	10026	20000331	11.0	0.077966	0.053499	1
3	10026	20000428	11.0	-0.198113	-0.059519	1
4	10026	20000531	11.0	-0.066667	-0.038863	1
...	...	...	...	...	...	...
383083	92807	20220831	11.0	-0.027919	-0.036240	23
383084	92807	20220930	11.0	0.000000	-0.091324	23
383085	92807	20221031	11.0	0.060526	0.077403	23
383086	92807	20221130	11.0	0.022332	0.052365	23
383087	92807	20221230	11.0	-0.036675	-0.057116	23

[383088 rows x 6 columns]

4.0.2 Remplaçons les NaN, si possible

Rappel: Seules les rentabilités des actions ordinaires doivent être utilisés pour l'étude. Les actions ordinaires sont celles dont la valeur du champ SHRCD (share code) est égale à 10 ou 11.

- pour une entreprise, certaines lignes ont des valeurs NaN dans la colonne SHRCD, néanmoins on peut retrouver ces valeurs si une des lignes concernant cette même entreprise à sa valeur SHRCD connue

[5]:
```
# remplace les NaN par les bonnes valeurs de SHRCD (sur le premier fichier␣
 ↪excel)
# avec ce fichier excel, rien ne sera remplacer puisque toutes les valeurs ont␣
 ↪déjà été ajouté

list_nan = ret[ret['SHRCD'].isna()].index #list d'indice des SHRCD = NaN

# vérifions si le code manquant d'une ligne peut être trouver ou non
for i in list_nan:
```

```
    if ret.loc[i,'PERMNO'] == ret.loc[i-1,'PERMNO']:
        ret.loc[i,'SHRCD'] = ret.loc[i-1,'SHRCD']
    else:
        if ret.loc[i,'PERMNO'] == ret.loc[i+1,'PERMNO']:
            ret.loc[i,'SHRCD'] = ret.loc[i+1,'SHRCD']

#ret
```

4.0.3 GARDER LES SHRCD = 10 ou 11

- une fois que les données manquant mais remplaçable sont completé nous pouvons trier les actions non ordinaires

[6]:
```
#Sélectionne les lignes avec des valeurs différentes de 10 ou 11 dans la colonne␣
↪SHRCD
index = ret.loc[~ret['SHRCD'].isin([10, 11])].index

#Supprimer les lignes correspondantes du tableau
ret.drop(index, inplace=True)

#ret
```

4.0.4 Correction de la colonne des taux de rentabilité RET

La base CRSP indique les valeurs manquantes par (-99, -88, -77, -66, B, C). Remplaçons ces valeurs par NaN.

[7]:
```
# on corrige les valeurs manquant en NAN
NADict = {-99:np.nan, -88:np.nan, -77:np.nan, -66:np.nan, 'B':np.nan, 'C':np.nan}
ret = ret.replace({'RET': NADict})
ret['RET'] = ret['RET'].astype(float)

#ret
```

[8]:
```
# on regarde s'il y a des données manquantes dans la colonne RET
ret_nan = ret[ret.isna().any(axis=1)]

print("On a bien des valeurs NaN dans la colonne aux lignes suivantes:")
ret_nan
```

```
On a bien des valeurs NaN dans la colonne aux lignes suivantes:
```

[8]:

	PERMNO	date	SHRCD	RET	vwretd	period
423	10028	20120430	11.0	NaN	-0.006840	13
424	10028	20120531	11.0	NaN	-0.065641	13
425	10028	20120629	11.0	NaN	0.038181	13
426	10028	20120731	11.0	NaN	0.010306	13
427	10028	20120831	11.0	NaN	0.026336	13

```
...        ...       ...    ...  ...       ...     ...
377620   87725  20040528   11.0  NaN  0.014068       5
377621   87725  20040630   11.0  NaN  0.021611       5
377622   87725  20040730   11.0  NaN -0.037698       5
377623   87725  20040831   11.0  NaN  0.002703       5
377624   87725  20040930   11.0  NaN  0.020556       5

[12012 rows x 6 columns]
```

Rappel: - nous avons decidé de remplacer les valeurs NaN lorsque c'est possible afin de conserver le maximum d'entreprise possible: - si un taux de rentabilité mensuel est manquant entre 2 dates nous approximerons le taux par la moyenne des taux mensuel du mois précédent et du mois suivant - s'il n'est pas possible de faire cette moyenne, alors nous retirerons l'entreprise de l'étude

[9]:
```
#liste des lignes à supprimer
sup = []

# moyenne du taux précédent et suivant afin d'approximer une valeur NaN si␣
 ↪possible
for i in ret_nan.index:
    if (ret.loc[i,'PERMNO'] != ret.loc[i-1,'PERMNO']) or (ret.loc[i,'PERMNO'] !=␣
 ↪ret.loc[i+1,'PERMNO']):
        sup.append(i)
    else:
        if pd.isna(ret.loc[i-1,'RET']) or pd.isna(ret.loc[i+1,'RET']):
            sup.append(i)
        else:
            ret.loc[i,'RET'] = (ret.loc[i-1,'RET'] + ret.loc[i-1,'RET'])/2

ret = ret.drop(sup,axis=0)
ret = ret.reset_index(drop = True)
#ret
```

[10]:
```
# on verifie qu'il ne manque plus aucune donnée dans le tableau
ret.isna().any()
```

[10]:
```
PERMNO    False
date      False
SHRCD     False
RET       False
vwretd    False
period    False
dtype: bool
```

4.0.5 Vérification

[11]:
```
# verifions que chaque entreprise ait bien toutes les cotations mensuels␣
 ↪necessaire
print('Il y a', ret['PERMNO'].nunique(), 'entreprises\nIl faut donc' ,␣
 ↪ret['PERMNO'].nunique(), 'taux mensuels chaque mois\n')
print(ret.groupby(['date'])['PERMNO'].count())
print('\nOn voit que certaines entreprises ne sont plus cotées sur tous les mois␣
 ↪de la période choisie')
print("Par exemple le 31/01/2000 nous n'avons que 1334 entreprises cotées")
print("Et le 29/02/2000 nous avons 1337 entreprises cotées")
print()
print('Nous devrions avoir', 12*ret['period'].nunique()*ret['PERMNO'].nunique(),␣
 ↪'lignes dans le tableau soit', 12*23, "mois de donnée pour chaque entreprise")
print("Or ce n'est pas le cas, nous avons", len(ret), "lignes")
```

```
Il y a 1388 entreprises
Il faut donc 1388 taux mensuels chaque mois

date
20000131    1334
20000229    1337
20000331    1336
20000428    1336
20000531    1336
            ...
20220831    1384
20220930    1385
20221031    1386
20221130    1387
20221230    1383
Name: PERMNO, Length: 276, dtype: int64

On voit que certaines entreprises ne sont plus cotées sur tous les mois de la
période choisie
Par exemple le 31/01/2000 nous n'avons que 1334 entreprises cotées
Et le 29/02/2000 nous avons 1337 entreprises cotées

Nous devrions avoir 383088 lignes dans le tableau soit 276 mois de donnée pour
chaque entreprise
Or ce n'est pas le cas, nous avons 371088 lignes
```

[12]:
```
# nb de données mensuel par entreprise
print('On a le nombre de donnée mensuel suivant par entreprise:\n')
print(ret.groupby(['PERMNO'])['date'].count())

# supprimer les entreprises avec des données manquante
```

```
manque_data = (ret.groupby(['PERMNO'])['date'].count() != 12*ret['period'].
 ↪nunique())
sup = manque_data[manque_data == True].index

ret = ret[~(ret['PERMNO'].isin(sup))]

# nb de données mensuel par entreprise
print("Après suppression des entreprises n'ayant pas le bon nombre de donnée, on␣
 ↪a :\n")
ret
```

```
On a le nombre de donnée mensuel suivant par entreprise:

PERMNO
10026    276
10028    269
10032    276
10044    276
10066     49
        ...
91556    276
91855    276
92583    276
92655    276
92807    276
Name: date, Length: 1388, dtype: int64
Après suppression des entreprises n'ayant pas le bon nombre de donnée, on a :
```

[12]:

	PERMNO	date	SHRCD	RET	vwretd	period
0	10026	20000131	11.0	-0.067073	-0.039622	1
1	10026	20000229	11.0	-0.035948	0.031760	1
2	10026	20000331	11.0	0.077966	0.053499	1
3	10026	20000428	11.0	-0.198113	-0.059519	1
4	10026	20000531	11.0	-0.066667	-0.038863	1
...	...	...	...	...	...	...
371083	92807	20220831	11.0	-0.027919	-0.036240	23
371084	92807	20220930	11.0	0.000000	-0.091324	23
371085	92807	20221031	11.0	0.060526	0.077403	23
371086	92807	20221130	11.0	0.022332	0.052365	23
371087	92807	20221230	11.0	-0.036675	-0.057116	23

[345276 rows x 6 columns]

[13]:
```
print('Il y a', ret['PERMNO'].nunique(), 'entreprises\nIl faut donc' ,␣
 ↪ret['PERMNO'].nunique(), 'taux mensuel chaque mois\n')
print(ret.groupby(['date'])['PERMNO'].count())
```

```
print('\nNous devrions avoir', 12*ret['period'].nunique()*ret['PERMNO'].
 ↪nunique(), "lignes dans le tableau ce qui est le cas,\nOn peut le verifier:␣
 ↪len(ret) == 12*ret['period'].nunique()*ret['PERMNO'].nunique() renvoie",␣
 ↪len(ret) == 12*23*ret['PERMNO'].nunique())
print("Avec ret['PERMNO'].nunique() le nombre d'entreprise \nEt len(ret) la␣
 ↪longueur du tableau")
#ret
```

```
Il y a 1251 entreprises
Il faut donc 1251 taux mensuel chaque mois

date
20000131    1251
20000229    1251
20000331    1251
20000428    1251
20000531    1251
            ...
20220831    1251
20220930    1251
20221031    1251
20221130    1251
20221230    1251
Name: PERMNO, Length: 276, dtype: int64

Nous devrions avoir 345276 lignes dans le tableau ce qui est le cas,
On peut le verifier: len(ret) ==
12*ret['period'].nunique()*ret['PERMNO'].nunique() renvoie True
Avec ret['PERMNO'].nunique() le nombre d'entreprise
Et len(ret) la longueur du tableau
```

5 Importation du Fama-French factors dataset

Les chiffres donnés sont des pourcentages et doivent donc tous être divisés par 100. Seuls les champs Mkt-RF et RF sont pertinents pour l'étude.

[14]:
```
FF = pd.read_csv(path + "F-F_Research_Data_Factors.CSV.txt", encoding =␣
 ↪'latin-1', sep=',', decimal='.')
FF[['Mkt-RF','SMB','HML','RF']] = FF[['Mkt-RF','SMB','HML','RF']]/100
FF
```

[14]:

	YearMonth	Mkt-RF	SMB	HML	RF
0	192607	0.0296	-0.0256	-0.0243	0.0022
1	192608	0.0264	-0.0117	0.0382	0.0025
2	192609	0.0036	-0.0140	0.0013	0.0023
3	192610	-0.0324	-0.0009	0.0070	0.0032
4	192611	0.0253	-0.0010	-0.0051	0.0031

```
...          ...     ...     ...     ...     ...
1215       2018 -0.0695 -0.0321 -0.0973  0.0183
1216       2019  0.2828 -0.0611 -0.1034  0.0215
1217       2020  0.2366  0.1318 -0.4656  0.0045
1218       2021  0.2356 -0.0389  0.2553  0.0004
1219       2022 -0.2160 -0.0682  0.2580  0.0143

[1220 rows x 5 columns]
```

5.0.1 Ajout d'une colonne dans la table ret pour jointure

Afin de pouvoir joindre les deux tables de données nous avons besoin d'une colonne commune.

[15]:
```
ret['YearMonth'] = (ret['date']/100).astype(int)

# remet les indices du tableau à 0,1,2, etc.
ret.reset_index(inplace = True)
ret.drop('index', axis=1, inplace = True)

#ret
```

6 Régressions Fama-MacBeth

En notant n l'année pour laquelle est réalisée la régression Fama-McBeth, la procédure à utiliser est la suivante :

1. Années n 6 à n 4 (36 mois) :
   - estimation du de chacune des actions à l'aide du modèle de marché et formation de 5 portefeuilles correspondant aux 5 quintiles de des actions.
2. Années n 3 à n 1 (36 mois) :
   - ré-estimation du de chacune des actions et calcul du risque spécifique de chaque action à partir du bêta estimé sur cette période ;
   - constitution de 25 portefeuille, à savoir 5 portefeuilles formés sur la base du et, à l'intérieur de chacun de ces 5 portefeuilles, constitution de 5 portefeuilles sur la base du risque spécifique. Le d'un portefeuille sera calculé comme la moyenne équipondérée des des titres entrant dans sa composition. Le risque spécifique d'un portefeuille sera calculé comme la moyenne équipondérée des risques spécifiques des titres entrant dans sa composition.
3. Pour chacun des 12 mois de l'année n, le vecteur des 25 taux de rentabilité des 25 portefeuilles précédemment constitués est régressé contre la matrice formée d'une constante, du vecteur des 25 et du vecteur des 25 valeurs de risque spécifique. Vous reporterez au niveau du résultat final la moyenne des moyennes obtenues sur chacune des années de test.

[16]:
```
formPerL = 3 # periode de formation = 3 ans
estPerL = 3  # periode d'estimation = 3 ans
```

```
testL = 1 #  periode de test = 1 an

nTests = ret['period'].max() - (formPerL + estPerL + testL) + 1
#nTests
```

6.0.1 Fonction de régression

[17]:
```
def computeBeta(data, yvar, xvar):
    Ri = data[yvar]
    Rm = data[xvar]
    Rm['intercept'] = 1
    output = sm.OLS(Ri, Rm).fit()
    return output.params
```

6.0.2 Calcul des bêtas sur la période de formation des portefeuilles et formation des portefeuilles

[18]:
```
def formPeriod(formPerBounds):

#on selectionne les données dont la valeur de la colonne période est comprise␣
 ↪entre formPerBounds[0] et formPerBounds[1]
    formPeriod = ret[ret['period'].between(formPerBounds[0], formPerBounds[1],␣
 ↪'both')]

    # groupe les données par PERMNO car je veux calculer le bêta de chacune des␣
 ↪actions
    # données qui figurent dans formPeriod
    by_PERMNO = formPeriod.groupby('PERMNO')

    # estimation faite d'abord pour la premiere action puis la deuxieme, etc.
    # calcul des betas action par action
    formBetas = pd.DataFrame(by_PERMNO.apply(computeBeta,'RET', ['vwretd']))

    # print(formBetas)

    # on supprime l'intercept qu'on n'utilisera pas                          ␣
 ↪
    del formBetas['intercept']
    formBetas.columns = ['beta_form']

    # formation de 5 portefeuilles en fonction de leur beta
    formPFBetas = pd.DataFrame(pd.qcut(formBetas['beta_form'], q=5,␣
 ↪labels=False))

    formPFBetas.columns = ['formPF']
    formPFBetas = pd.merge(formPFBetas, formBetas, on='PERMNO')
```

```
    # print(formPFBetas)

    return formPFBetas # retourne le tableau
```

6.0.3 Calcul des bêtas sur la seconde période d'estimation et calcul des risques spécifiques de chaque titre à partir du bêta estimé sur cette période

on sait que

$$var(r_i) = (\beta_i)^2 * var(r_m) + (error_i)^2$$

donc

$$(error_i)^2 = var(r_i) - (\beta_i)^2 * var(r_m)$$

avec risque specifique = $error_i$

[19]:
```
def estPeriod_risk(EstPerBounds, formPeriodBetas):

    estPeriodData = ret[ret['period'].between(estPerBounds[0], estPerBounds[1],␣
↪'both')]

    # relie les titres à leur portefeuille respectif dans la periode d'estimation
    estPeriodData = pd.merge(estPeriodData, formPeriodBetas, how='left',␣
↪on='PERMNO')

    # on re-estime le beta de chaque action sur cette nouvelle période
    # je groupe mes données par PERMNO et j'estime pour chacune des titres sur␣
↪la
    # période d'estimation des bêtas
    by_PERMNO = estPeriodData.groupby('PERMNO')
    estBetas = pd.DataFrame(by_PERMNO.apply(computeBeta,'RET', ['vwretd']))

    # on supprime l'intercept
    # on renomme la colonne pour bien differencier les betas calculés sur la␣
↪periode de formation
    # et les betas calculés sur la periode d'estimation
    del estBetas['intercept']
    estBetas.columns = ['beta_est']

    # calculons le risque idiosyncratiques de chaque titre
    # on a le taux de rentabilité du marché = vwred

    var_ri = estPeriodData.groupby(['PERMNO'])['RET'].var()
    var_rm = estPeriodData.groupby(['PERMNO'])['vwretd'].var()
    error_i2 = var_ri - estBetas['beta_est']**2 * var_rm
    error_i = error_i2.apply(math.sqrt)
    error_i.name = 'idiosyncratic_risk'
```

```
    #print(error_i)

    estBetas = pd.merge(estBetas, error_i, how='left', on='PERMNO')

    #print(estBetas)

    return estBetas #retourne le tableau
```

6.0.4 Rolling

La procédure est itérée sur les années n + 1, n + 2, . . . jusqu'à la dernière année disponible dans la base

[20]:
```
gammas = pd.DataFrame()
renta_mean = pd.DataFrame()
renta_monthyear_mean = pd.DataFrame()

for iter in range(1, nTests + 1): # +1 due to the way Python handles loops:␣
 ↪final value is last value minus 1

    print("\n ****** Iteration: " + str(iter) + "/" + str(nTests) + " ******\n")

    # 3.1.1

    # estimation du beta de chacune des actions et formation de 5 portefeuilles␣
 ↪correspondant aux 5 quintiles des betas des actions
    formPerBounds = (iter, iter + formPerL - 1)
    print("formation period: " + str(formPerBounds))
    formPeriodBetas = formPeriod(formPerBounds)


  ␣
 ↪##############################################################################
    ␣
 ↪##############################################################################
        ␣
 ↪##############################################################################

    # 3.1.2

    # ré-estimation du beta de chacune des actions
    # calcul du risque spécifique de chaque action à partir du bêta estimé sur␣
 ↪cette période

    estPerBounds = (formPerBounds[1] + 1, formPerBounds[1] + estPerL)
    print("estimation period: " + str(estPerBounds))
    estPeriodBetas = estPeriod_risk(estPerBounds, formPeriodBetas) # tableau des␣
 ↪beta estimé + risque specifique
```

```
   # joiture des betas estimés, du risque specifique et des portefeuilles␣
↪respectifs
   combinedBetas = pd.merge(formPeriodBetas, estPeriodBetas, on='PERMNO')

   # nous avons besoin des PERMNO comme une colonne
   combinedBetas.reset_index(inplace=True)

   # calcul du beta des 5 portefeuilles equi-pondérés
   by_PF =  combinedBetas.groupby('formPF')
   PFBeta = pd.DataFrame(by_PF['beta_est'].mean())
   PFBeta.columns = ['beta_PF']

   # jointure des titres de chaque portefeuille au beta de leur portefeuille␣
↪respectif
   mapping = pd.merge(combinedBetas, PFBeta, on='formPF')

   # à partir de chaque portefeuille, formation de 5 sous-portefeuille basé sur␣
↪le risque spécifique
   mapping['formPFrisk'] = mapping.groupby('formPF')['idiosyncratic_risk'].
↪apply(lambda x: pd.qcut(x, q=5, labels=False))

   # calcul du beta des 25 portefeuilles equi-pondéré
   by_PF_risk =  mapping.groupby(['formPF','formPFrisk'])
   PFBeta_risk = pd.DataFrame(by_PF_risk['beta_est'].mean())
   PFBeta_risk.columns = ['beta_PF_risk']

   # jointure des titres de chacun des 25 portefeuille au beta de leur␣
↪portefeuille respectif
   mapping = pd.merge(mapping, PFBeta_risk, on=['formPF','formPFrisk'])

   # calcul du risque spécifique des 25 portefeuille
   # calculé comme la moyenne équipondérée des risques spécifiques des titres␣
↪entrant dans sa composition
   PFvolatility_risk = pd.DataFrame(by_PF_risk['idiosyncratic_risk'].mean())
   PFvolatility_risk.columns = ['PF_idiosyncratic_risk']

   # jointure des titres au risque spécifique de leur portefeuille respectif
   mapping = pd.merge(mapping, PFvolatility_risk, on=['formPF','formPFrisk'])


  ␣
↪#############################################################################
    ␣
↪#############################################################################
       ␣
↪#############################################################################
```

```
   # 3.1.3
   # Pour chacun des 12 mois de l'année n, le vecteur des 25 taux de␣
↪rentabilité des 25 portefeuilles précédemment
   # constitués est régressé contre la matrice formée
       # d'une constante
       # du vecteur des 25 betas
       # du vecteur des 25 valeurs de risque spécifique.
   # Vous reporterez au niveau du résultat final la moyenne des moyennes␣
↪obtenues sur chacune des années de test

   # période de test
   testPerBounds = (estPerBounds[1] + 1, estPerBounds[1] + testL)
   print("test period: " + str(testPerBounds))
   testPeriod = ret[ret['period'].between(testPerBounds[0], testPerBounds[1],␣
↪'both')]

   # jointure pour avoir à chaque mois les données du portefeuille correspondant
   testPeriod = pd.merge(testPeriod, mapping, on='PERMNO')

   # jointure du tableau testPeriod avec le tableau FF
   testPeriod = pd.merge(testPeriod, FF, on='YearMonth')

   #print(testPeriod)

   # je groupe par mois et années et par 'sous-portefeuille'
   by_month_PF = testPeriod.groupby(['YearMonth','formPF','formPFrisk'])

   # calcul de la moyenne des taux de rentabilité des actions
   PFRet = pd.
↪DataFrame(by_month_PF[['beta_est','idiosyncratic_risk','RET','RF']].mean())

   # nous avons le beta de chaque portfeuille ainsi que le risque spécifique de␣
↪chaque portefeuille
   # de même RF est le meme pour chaque année-mois
   # néanmoins en recalculant chaque valeur en même temps que le taux de␣
↪rentabilité des portefeuilles
   # nous n'aurons pas à joindre chaque élément au bon titre
   PFRet.reset_index(inplace=True)

   # calcul du taux en excès
   PFRet['XRET'] = PFRet['RET'] - PFRet['RF']

   #print(PFRet)

   # Régression par année-mois de chaque taux en excès
```

```
    by_month = PFRet.groupby('YearMonth')

   # matrice formée
        # d'une constante (ajouté par la fonction computeBeta)
        # du vecteur des 25 betas
        # du vecteur des 25 valeurs de risque spécifique.
    gammasTemp = pd.DataFrame(by_month.apply(computeBeta,'XRET',␣
 ↪['beta_est','idiosyncratic_risk']))
    gammasTemp.columns =␣
 ↪['gamma_systematic_risk','gamma_idiosyncratic_risk','intercept']

    gammas = gammas.append(gammasTemp)

    # Pour chaque année n, calcul du taux de rentabilité mensuel moyen
    # des 25 portefeuilles sur les 12 mois de l'année.
    # = chaque mois, faire la moyenne des 25 taux de renta de portefeuille

    by_monthRet = PFRet.groupby(['formPF','formPFrisk'])['RET'].mean()

    renta_cal = pd.DataFrame({'Moyenne des 25 taux de rentabilité mensuel de␣
 ↪portefeuille': by_monthRet.values}, index=by_monthRet.index)
    #print(renta_cal)
    renta_monthyear_mean = renta_monthyear_mean.append(renta_cal)
    #PFRet


    ␣
 ↪#############################################################################
     ␣
 ↪#############################################################################
         ␣
 ↪#############################################################################

print("\nfinished\n")

# calcul de la moyenne des moyennes
renta_monthyear_meanRet = renta_monthyear_mean.
 ↪groupby(['formPF','formPFrisk'])['Moyenne des 25 taux de rentabilité mensuel␣
 ↪de portefeuille'].mean()

renta_month_mean = pd.DataFrame({'Taux de rentabilité mensuel moyen de chaque␣
 ↪portefeuille sur la période': renta_monthyear_meanRet.values},␣
 ↪index=renta_monthyear_meanRet.index)
#renta_month_mean
```

```
****** Iteration: 1/17 ******
```

```
formation period: (1, 3)
estimation period: (4, 6)
test period: (7, 7)

 ****** Iteration: 2/17 ******

formation period: (2, 4)
estimation period: (5, 7)
test period: (8, 8)

 ****** Iteration: 3/17 ******

formation period: (3, 5)
estimation period: (6, 8)
test period: (9, 9)

 ****** Iteration: 4/17 ******

formation period: (4, 6)
estimation period: (7, 9)
test period: (10, 10)

 ****** Iteration: 5/17 ******

formation period: (5, 7)
estimation period: (8, 10)
test period: (11, 11)

 ****** Iteration: 6/17 ******

formation period: (6, 8)
estimation period: (9, 11)
test period: (12, 12)

 ****** Iteration: 7/17 ******

formation period: (7, 9)
estimation period: (10, 12)
test period: (13, 13)

 ****** Iteration: 8/17 ******

formation period: (8, 10)
estimation period: (11, 13)
test period: (14, 14)

 ****** Iteration: 9/17 ******
```

```
formation period: (9, 11)
estimation period: (12, 14)
test period: (15, 15)

 ****** Iteration: 10/17 ******

formation period: (10, 12)
estimation period: (13, 15)
test period: (16, 16)

 ****** Iteration: 11/17 ******

formation period: (11, 13)
estimation period: (14, 16)
test period: (17, 17)

 ****** Iteration: 12/17 ******

formation period: (12, 14)
estimation period: (15, 17)
test period: (18, 18)

 ****** Iteration: 13/17 ******

formation period: (13, 15)
estimation period: (16, 18)
test period: (19, 19)

 ****** Iteration: 14/17 ******

formation period: (14, 16)
estimation period: (17, 19)
test period: (20, 20)

 ****** Iteration: 15/17 ******

formation period: (15, 17)
estimation period: (18, 20)
test period: (21, 21)

 ****** Iteration: 16/17 ******

formation period: (16, 18)
estimation period: (19, 21)
test period: (22, 22)

 ****** Iteration: 17/17 ******
```

```
formation period: (17, 19)
estimation period: (20, 22)
test period: (23, 23)

finished
```

[21]:
```
# changeons les indices pour une meilleure lecture
change_indice = {0: 'q1_faible', 1: 'q2', 2: 'q3',3: 'q4',4:'q5_élevé'}

index = renta_month_mean.index.map(lambda x: (change_indice[x[0]],␣
 ↪change_indice[x[1]]))

renta_month_mean.index = pd.MultiIndex.from_tuples(index, names=['formPF',␣
 ↪'formPFrisk'])
renta_month_mean.index.names = ['beta','risque spécifique']

tableau_2entree = renta_month_mean.unstack()
tableau_2entree
```

[21]:
```
                  Taux de rentabilité mensuel moyen de chaque portefeuille sur
la période  \
risque spécifique
q1_faible
beta
q1_faible                                                          0.008317
q2                                                                 0.008768
q3                                                                 0.009377
q4                                                                 0.010551
q5_élevé                                                           0.011078


risque spécifique        q2        q3        q4  q5_élevé
beta
q1_faible          0.010016  0.009058  0.011593  0.021690
q2                 0.009973  0.010101  0.008356  0.013489
q3                 0.009063  0.009589  0.010652  0.012725
q4                 0.009473  0.010425  0.012830  0.016321
q5_élevé           0.010992  0.012202  0.015217  0.015172
```

[22]:
```
# Création d'une copie du tableau
renta_month_mean_bis = renta_month_mean.copy()

fig, ax = plt.subplots(figsize=(11,8))

# Couleur
q1_beta_faible_color = ['blue']
```

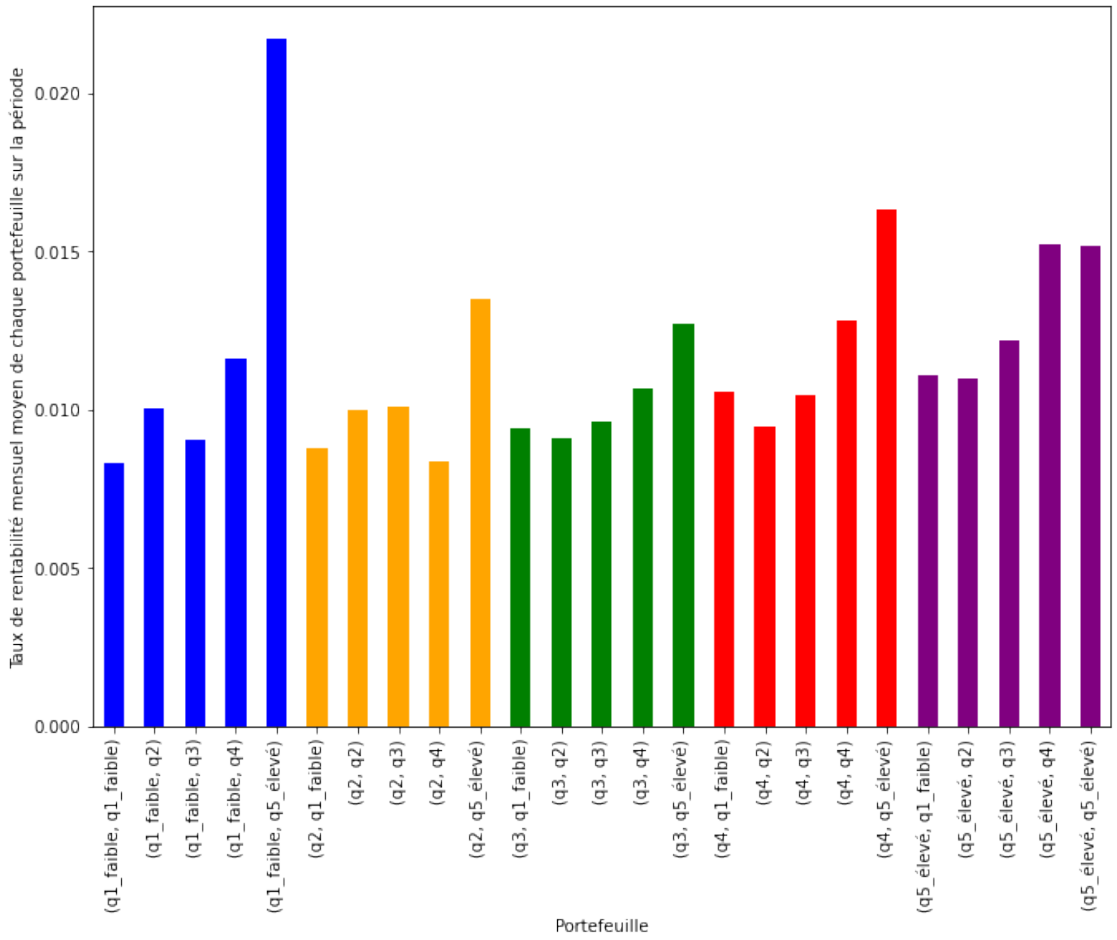
```
q2_color = ['orange']
q3_color = ['green']
q4_color = ['red']
q5_beta_eleve_color = ['purple']

bar_colors = q1_beta_faible_color*5 + q2_color*5 + q3_color*5 + q4_color*5 +␣
 ↪q5_beta_eleve_color*5

# Trace un graphique
renta_month_mean_bis.plot(kind='bar', y='Taux de rentabilité mensuel moyen de␣
 ↪chaque portefeuille sur la période', ax=ax, color=bar_colors, legend=None)

# Légende
ax.set_ylabel('Taux de rentabilité mensuel moyen de chaque portefeuille sur la␣
 ↪période')
ax.set_xlabel('Portefeuille')

# Affiche le graphique
plt.show()
```



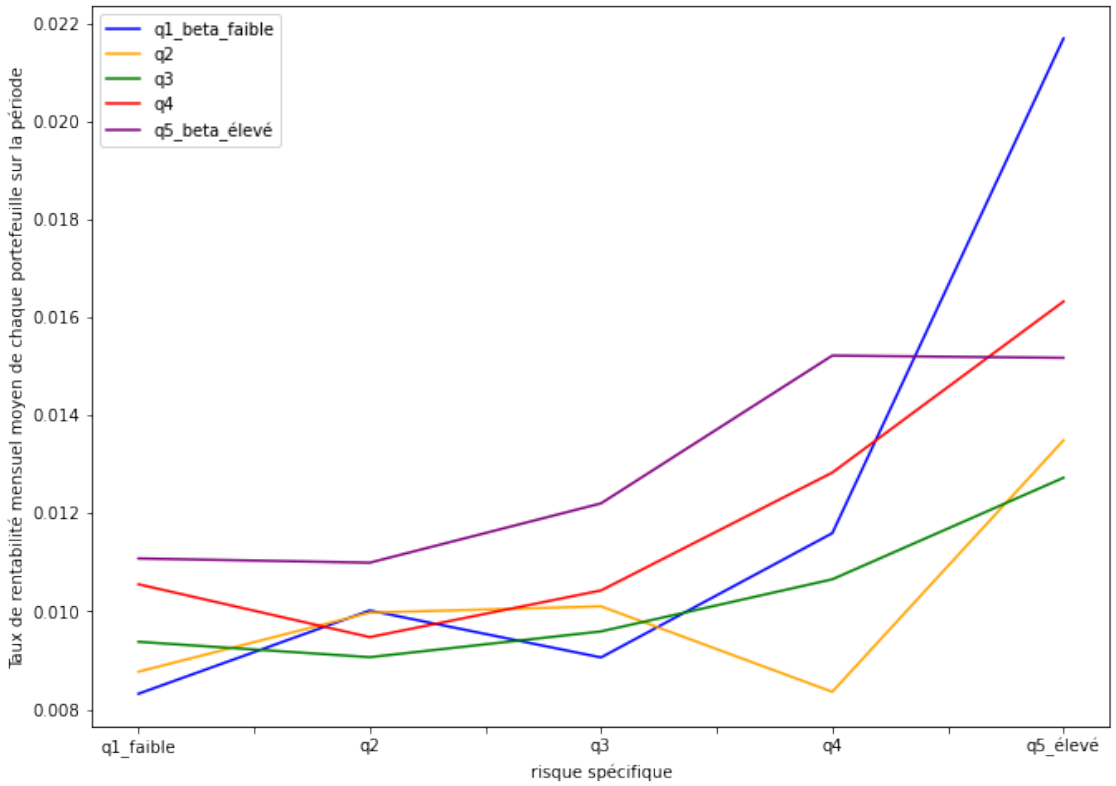
[23]:
```
# Création d'une copie du tableau transposer pour inverser les axes
tableau_2entree_bis = tableau_2entree.T


# Tracer une courbe
fig, ax = plt.subplots(figsize=(11, 8))
graph_colors = q1_beta_faible_color + q2_color + q3_color + q4_color +␣
 ↪q5_beta_eleve_color
tableau_2entree_bis.plot(ax=ax,color=graph_colors, use_index=True)

# Légende
ax.legend(['q1_beta_faible', 'q2', 'q3', 'q4', 'q5_beta_élevé'])
ax.set_xlabel('risque spécifique')
ax.set_ylabel('Taux de rentabilité mensuel moyen de chaque portefeuille sur la␣
 ↪période')
ax.set_xticklabels(['','q1_faible', '','q2', '','q3','', 'q4', '','q5_élevé',''])
#plt.xticks(rotation=45)

#ax.set_title('Graphe des taux en fonction du risque spécifique et de beta')

# Affiche le graphique
plt.show()
```



6.1 T-test on gammas

Le test de student est un test de significativité, nous cherchons à savoir si les estimateurs sont ou non significativement différent de 0.

$$\begin{cases} H_0 : \text{les estimateurs ne sont pas significativement différent de } 0 \\ H_a : \text{les estimateurs sont significativement différent de } 0 \end{cases}$$

Statistique de test :

$$Z = \sqrt{n} * \frac{\bar{X} - 0}{\sigma}$$

avec - $X =$ estimateur - $\sigma =$ écart type des gammas - $n =$ nombre d'observation, ici de test

et - niveau de confiance = 5%

on a la règle suivant: - si $t - stat > 1,96$ alors on rejete H_0 - si $t - stat < 1,96$ alors on ne rejete pas H_0

[24]:
```
print("Avec l'ensemble des régressions nous avons obtenu:")
gammas
```

Avec l'ensemble des régressions nous avons obtenu:

[24]:

YearMonth	gamma_systematic_risk	gamma_idiosyncratic_risk	intercept
200601	0.019060	0.508073	-0.003327
200602	-0.003192	0.115621	-0.003213
200603	-0.004803	0.424609	0.008080
200604	0.004292	0.022445	0.000822
200605	-0.043816	0.086464	0.006865
...	...	...	...
202208	-0.017389	0.261168	-0.031956
202209	-0.050605	-0.118781	-0.029280
202210	0.056642	-0.189543	0.060197
202211	0.018201	-0.360096	0.063069
202212	-0.037170	-0.059195	-0.003922

[204 rows x 3 columns]

[25]:
```
print('nous obtenons les statistiques suivante:')
t_stat_systematic_risk = gammas['gamma_systematic_risk'].mean() * math.
 ↪sqrt(gammas['gamma_systematic_risk'].count()) /␣
 ↪gammas['gamma_systematic_risk'].std()
t_stat_idiosyncratic_risk = gammas['gamma_idiosyncratic_risk'].mean() * math.
 ↪sqrt(gammas['gamma_idiosyncratic_risk'].count()) /␣
 ↪gammas['gamma_idiosyncratic_risk'].std()


tab_gamma = pd.DataFrame({'gamma_systematic_risk':
 ↪[gammas['gamma_systematic_risk'].
 ↪mean(),t_stat_systematic_risk],'gamma_idiosyncratic_risk':
 ↪[gammas['gamma_idiosyncratic_risk'].mean(),t_stat_idiosyncratic_risk]})
tab_gamma.index = ['Moyenne','t-stat']
tab_gamma
```

nous obtenons les statistiques suivante:

[25]:

	gamma_systematic_risk	gamma_idiosyncratic_risk
Moyenne	0.001545	0.059290
t-stat	0.427134	2.453488