Reto ETS Factory

Clasificación de Momentos Para Inversión en Valor o Crecimiento

---

Nombre: Ramón Vargas

fecha: 19-08-2024

1. Resumen

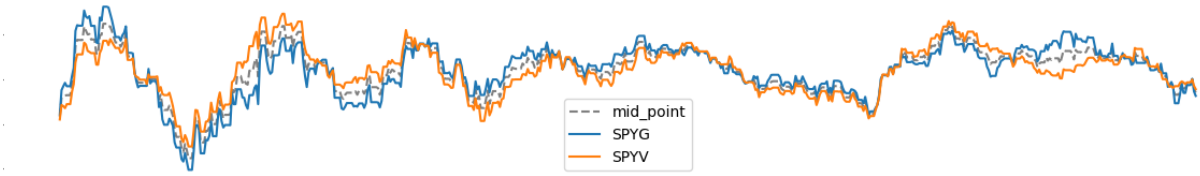
El trabajo realizado ha sido encaminado a producir una señal que ayude a determinar según una serie de variables si invertir en activos considerados de crecimiento o de valor. La señal se ha obtenido mediante el entrenamiento de varios modelos de clasificación de los cuales uno ha sido seleccionado por su precisión y mayor fiabilidad.

2. Metodología

Obtención de Datos de Entrenamiento

Los datos de entrenamiento han sido obtenidos de fuentes como Financial Modelling Prep o Yahoo Finance. El periodo de tiempo abarca de 10 años, de 2014 hasta 2024

1) **Variable a predecir**. Esta variable es binaria, 0 para value y 1 para growth. Se obtiene separando los retornos de dos ETFs SPYV (valor) y SPYG (crecimiento) sobre el mismo índice utilizando como medida el punto medio del precio de ambos activos. Con esto conseguimos saber cual de los dos tiene mejores rendimientos en todo momento.



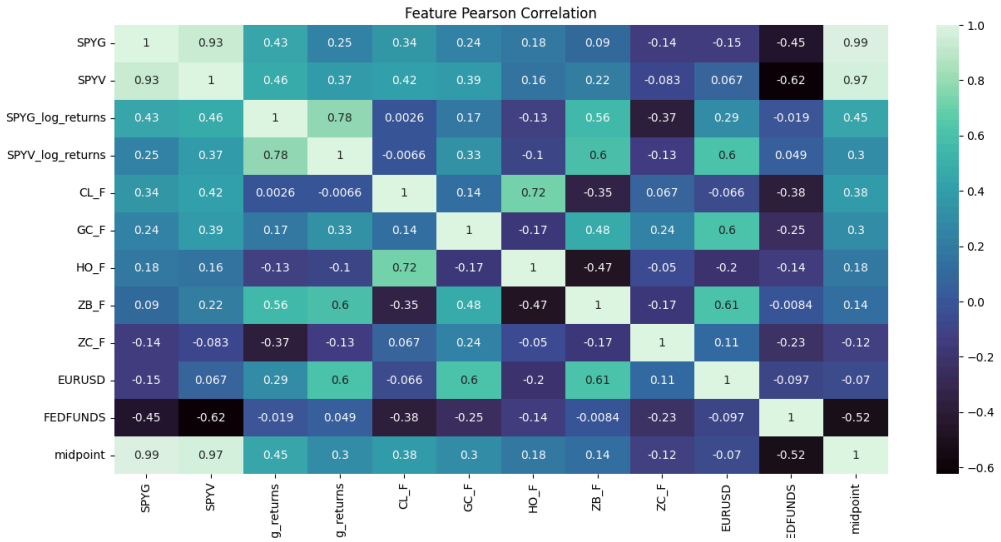
2) Variables Predictivas. A continuación se enumeran las series de datos utilizadas como variables para la predicción.
    1. Precios Históricos de Futuros: oro, petróleo crudo, bonos del tesoro de EE.UU, maíz, o aceite de calefacción.
    2. Precios del par EURUSD
    3. Reservas de la FED

Estudio y Preprocesamiento de Datos

En este punto el objetivo es limpiar y formatear los datos para asegurar que el modelo ingiere datos coherentes.

1. Primero se crea un un rango de fechas común para evitar que se descuadren los distintos activos; ya que por ejemplo los precios de futuros no son continuos.

2. Tras esto, separamos las variables y quitamos aquellas que puedan causar fuga o sesgo al modelo. Lógicamente serán aquellas que hemos utilizado para obtener la variable binaria como: retornos de SPYG y SPYV, y el punto medio.



.

3. Por último se pasa a separar los datos entre datos de entrenamiento y datos de prueba, 80% y 20% del total respectivamente.

4. Como existe un **desbalance entre las clases 0 y 1**, incluimos una mezcla de los datos de entrenamiento para mejorar la precisión, aunque trabajamos con series temporales el hecho de que el problema sea de clasificación nos permite hacerlo.

Entrenamiento y Selección de Modelos

En total se han entrenado tres modelos supervisados de clasificación: un modelo **KNN**, **Random Forest**, y **AdaBoost**.

A pesar de que tanto KNN como el RF tienen unos resultados de test altos, me he decantado por continuar con el modelo AdaBoost por dos razones: una que son resistentes al overfitting y otra porque los resultados de test en europa con diversos etfs muestran claramente dicho overfitting. Tanto KNN como RF obtienen peor rendimiento y cuando pierden lo hacen por mucho, sin embargo el AdaBoost consigue perder menos, obteniendo casi los mismos rendimientos. En el siguiente punto veremos un ejemplo de esto.

3. Resultados

Prueba en Estados Unidos

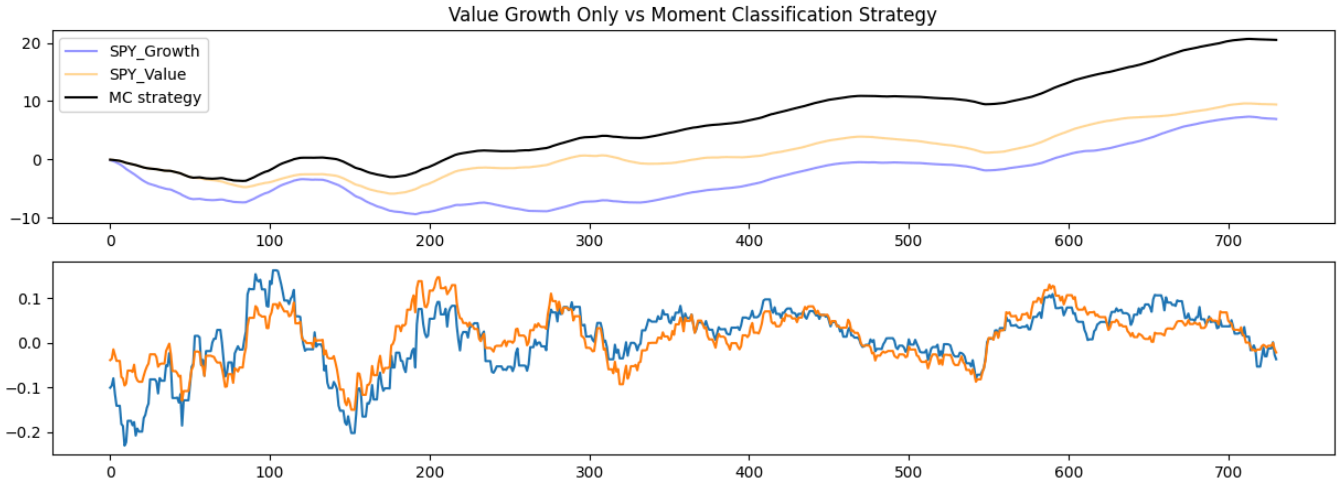
A continuación se muestran los resultados de prueba con datos nuevos en Estados Unidos:

Model	test_score
KNN	0.93%
RandomForest	0.93%
AdaBoostClassifier	0.79%

Si comparamos los retornos acumulados del SPYG y SPYV con los retornos que acumulamos si utilizamos el modelo como señal para pasar de uno a otro veremos un rendimiento claramente superior. Durante el periodo de prueba SPYV tiene un retorno acumulado de 9.43% y SPYG de 6.94% frente a un 20.50% del modelo.

USA Test

Strategies	cumulative returns
SPYV ETF only	9.43%
SPYG ETF only	6.94%
Value/Growth Momment Classification Strategy	20.51%



Backtesting

Para ir más allá de un test de precisión, he optado por desarrollar un pequeño backtest para ver el comportamiento de cada modelo en Europa. Para ello veremos a continuación una comparativa de los tres modelos sobre una serie de ETFs europeos. El backtest consiste en evaluar el rendimiento de la siguiente estrategia: entrar o salir de ETFs value según la señal del modelo cada 32 días. He tratado de simplificar la lógica de la estrategía para evaluar solo el modelo.

KNN

ETF Symbol	ROI	Strategy Returns
LEER.DE	12.16%	52.97%
EUN1.DE	51.05%	64.85%
IESE.SW	11.96%	10.55%
MTDA.DE	35.88%	11.24%
AMEQ.DE	74.88%	53.41%
CU9.PA	84.68%	0.95%
UBE.AX	-2.22%	-19.21%
ESIH.L	20.29%	7.38%
RIEG.L	26.02%	4.27%
CE9.PA	1.56%	44.98%
EUMD.L	40.51%	-32.85%
EUN0.DE	85.13%	44.89%
EL43.DE	86.19%	68.87%
EXSC.F	46.79%	38.62%
LFOD.DE	19.58%	11.84%
5HEU.F	-1.11%	-17.97%
ASRR.DE	5.03%	-7.67%
LRET.DE	40.13%	83.94%
EMSV.DE	37.80%	21.21%
EHEF.L	47.52%	115.88%
IEER.L	-3.68%	-3.67%
MVLD.AS	31.00%	-7.51%
JREE.SW	14.17%	9.11%
EL42.DE	92.24%	72.50%
SELD.DE	8.07%	-7.64%
ETLN.DE	72.18%	-18.28%
EVAL.L	33.92%	37.72%
ESIT.L	28.46%	2.14%
FREQ.L	20.80%	30.86%
EVOE.PA	66.31%	19.38%
EMSV.F	37.39%	39.62%
INGS.SW	7.19%	-10.98%
EL4E.DE	118.04%	36.15%
JREE.DE	71.61%	24.61%
ESGE.L	31.57%	-205.41%
X011.DE	50.96%	37.44%
EXSD.F	42.29%	50.78%
ASWA.DE	-9.12%	-10.44%
C6E.PA	95.03%	-9.90%
IMV.L	91.07%	28.83%
HXX.TO	95.62%	-17.98%
HSEP.L	34.60%	-22.55%
ESDU.L	38.54%	38.72%
ENGY.L	135.80%	147.19%
TEGB.L	29.25%	27.07%
LMDA.DE	77.87%	36.85%
OIL.PA	47.54%	119.38%
EEXU.DE	78.98%	-20.20%
TELE.L	1.32%	114.86%
SC00.DE	95.60%	29.57%
D5BK.DE	24.42%	85.47%
FIEE	88.07%	55.24%
SC0T.DE	104.79%	61.13%
DX2X.DE	107.49%	58.61%
FLXD.DE	46.10%	35.70%
performance :		1422.51%

AdaBoostClassifier

ETF Symbol	ROI	Strategy Returns
LEER.DE	12.16%	-3.20%
EUN1.DE	51.05%	48.71%
IESE.SW	11.96%	4.12%
MTDA.DE	35.88%	16.54%
AMEQ.DE	74.88%	15.50%
CU9.PA	84.68%	-6.86%
UBE.AX	-2.22%	26.00%
ESIH.L	20.29%	-15.21%
RIEG.L	26.02%	22.60%
CE9.PA	1.56%	-2.06%
EUMD.L	40.51%	13.56%
EUN0.DE	85.13%	17.13%
EL43.DE	86.19%	33.83%
EXSC.F	46.79%	11.22%
LFOD.DE	19.58%	60.68%
5HEU.F	-1.11%	19.38%
ASRR.DE	5.03%	20.81%
LRET.DE	40.13%	31.19%
EMSV.DE	37.80%	29.98%
EHEF.L	47.52%	34.29%
IEER.L	-3.68%	-3.67%
MVLD.AS	31.00%	-3.71%
JREE.SW	14.17%	5.10%
EL42.DE	92.24%	18.92%
SELD.DE	8.07%	-7.64%
ETLN.DE	72.18%	-1.38%
EVAL.L	33.92%	118.84%
ESIT.L	28.46%	72.52%
FREQ.L	20.80%	-6.75%
EVOE.PA	66.31%	20.68%
EMSV.F	37.39%	-6.90%
INGS.SW	7.19%	32.04%
EL4E.DE	118.04%	10.41%
JREE.DE	71.61%	32.58%
ESGE.L	31.57%	31.29%
X011.DE	50.96%	63.08%
EXSD.F	42.29%	79.16%
ASWA.DE	-9.12%	14.16%
C6E.PA	95.03%	-7.94%
IMV.L	91.07%	7.33%
HXX.TO	95.62%	16.70%
HSEP.L	34.60%	-7.13%
ESDU.L	38.54%	20.59%
TEGB.L	29.25%	-23.29%
LMDA.DE	77.87%	30.98%
OIL.PA	47.54%	30.76%
EEXU.DE	78.98%	12.85%
TELE.L	1.32%	105.93%
SC00.DE	95.60%	9.71%
D5BK.DE	24.42%	113.98%
FIEE	88.07%	25.36%
SC0T.DE	104.79%	23.99%
DX2X.DE	107.49%	34.94%
FLXD.DE	46.10%	51.74%
performance :		1293.42%

4. Conclusiones y Mejoras

Como conclusión, vemos como el modelo es capaz de clasificar ambos momentos mediante el precio de activos o variables macro importantes. A continuación se exponen los coeficientes del modelo.

Feature	Weight
HO_F	0.085
GC_F	0.095
EURUSD	0.1
ZC_F	0.105
CL_F	0.115
ZB_F	0.13
FEDFUNDS	0.37

Vemos como el peso de todas las variables predictivas es significativo, siendo los fondos de la FED lo que más pesa. Parece que la utilización de estas variables tiene capacidad predictiva y un estudio y aplicación de estas mejoraría el rendimiento de la señal.

Como posibles mejoras cabría un estudio más detallado de las variables predictivas, la utilización de noticias para análisis de sentimiento, la introducción de modelos más complejos (HMM, LSTM…), o introducir medidas de riesgo/volatilidad.

Por último, gracias a ETS Factory por organizar este tipo de iniciativas.

5. Links

Repositorio Github

Cuaderno Versión Web

6. Anexos

VALUE vs GROWTH DETECTION

This notebook contains all the code that will produce a model/signal to predict the optimum shift between value or growth investing.

IMPORTS

In [ ]:
```
import os import json import
pickle  import  dotenv  from
tabulate import tabulate from
datetime import datetime



import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

LOAD DATA

In [ ]:
```
root_dir = os.path.dirname(os.getcwd())
DATA_DIR = os.path.join(root_dir, "data")
dotenv_file = os.path.join(root_dir, '.env')

#Loading all etfs data
etfs_dict = {}
for f in os.listdir(os.path.join(DATA_DIR, "etf_data")):
    if f.startswith("SPY"):
        etfs_dict[f.strip(".csv")] = pd.read_csv(os.path.join(DATA_DIR, "etf_dat

with open (os.path.join(DATA_DIR, "etf_data", "us_etfs_list.pkl"), "rb") as f:
    etfs_list : dict = pickle.load(f)

#Load Commodities Futures Prices: .csv
commodities_dict = {}
commodities_files = os.listdir(os.path.join(DATA_DIR, "commodities"))
for f in commodities_files:
    commodities_dict[f.strip(".csv")] = pd.read_csv(os.path.join(DATA_DIR, "comm

#Load Currencies Prices: .csv
currencies_dict = {}
currencies_files = os.listdir(os.path.join(DATA_DIR, "currencies"))
for f in currencies_files:
    currencies_dict[f.strip(".csv")] = pd.read_csv(os.path.join(DATA_DIR, "curre

#FEDFUNDS and BALANCE: .pkl
macro_files = os.listdir(os.path.join(DATA_DIR, "macro_data"))
with open(os.path.join(DATA_DIR, "macro_data", "macro_data.pkl"), "rb") as f:
    macro_dict = pickle.load(f)
```

EDA & Feature Eng

Create Dataframes

In [ ]:
```
start, end = "2014-05-01", "2024-05-01"
daterange = pd.date_range(start=start, end=end, freq="d").strftime("%Y-%m-%d")
period = 30

etfs = [i[:-4] for i in os.listdir(os.path.join(DATA_DIR, "etf_data")) if i.star
etfs_df = pd.concat(
    [pd.read_csv(os.path.join(DATA_DIR, "etf_data",f"{i}.csv"),index_col="Date")
    axis=1,
) etfs_df.columns = etfs etfs_df["SPYG_log_returns"] = np.log(etfs_df["SPYG"] /
etfs_df["SPYG"].shift(per etfs_df["SPYV_log_returns"] = np.log(etfs_df["SPYV"] /
etfs_df["SPYV"].shift(per   etfs_df   =   etfs_df[(etfs_df.index   >=   start)   &
(etfs_df.index <= end)] #-------------------------------------------------------
----------------------      currencies      =      [i[:-4]      for      i      in
os.listdir(os.path.join(DATA_DIR, "currencies"))] curr_df = pd.concat(


    [pd.read_csv(os.path.join(DATA_DIR, "currencies",f"{i}.csv"), index_col="Dat
    axis=1,
) curr_df.columns = currencies curr_df = np.log(curr_df/curr_df.shift(period))
curr_df = curr_df[(curr_df.index >= start) & (curr_df.index <= end)] # #---------
------------------------------------------------------------------- macro_ind =
[i[:-4] for i in os.listdir(os.path.join(DATA_DIR, "macro_data")) if macro_df =
pd.read_csv(os.path.join(DATA_DIR,"macro_data", "FEDFUNDS.csv"),index macro_df =
np.log(macro_df/macro_df.shift(1)) macro_df.columns = macro_ind # #--------------
------------------------------------------------------------- comm = [i[:-4]
for i in os.listdir(os.path.join(DATA_DIR, "commodities")) if i.e commodities_df
= pd.concat(



    [pd.read_csv(os.path.join(DATA_DIR, "commodities", f"{i}.csv"),index_col="Da
    axis=1,
)    commodities_df    =    np.log(commodities_df/commodities_df.shift(period))
commodities_df = commodities_df[(commodities_df.index >= start) & (commodities_d
commodities_df.columns = comm # #-----------------------------------------------
----------------------------    #Set    common    indexes    all_data    =
pd.DataFrame(index=daterange) all_data = all_data.merge(etfs_df, left_index=True,
right_index=True,   how="left"   all_data   =   all_data.merge(commodities_df,
left_index=True,  right_index=True,  how  all_data  =  all_data.merge(curr_df,
left_index=True, right_index=True, how="left" all_data = all_data.merge(macro_df,
left_index=True,  right_index=True,  how="left  all_data.index.name  =  "Date"
all_data.fillna(method="ffill", inplace=True)



all_data["midpoint"] = (all_data["SPYG"] + all_data["SPYV"]) / 2
```

In [ ]:
```
all_data.head()
```

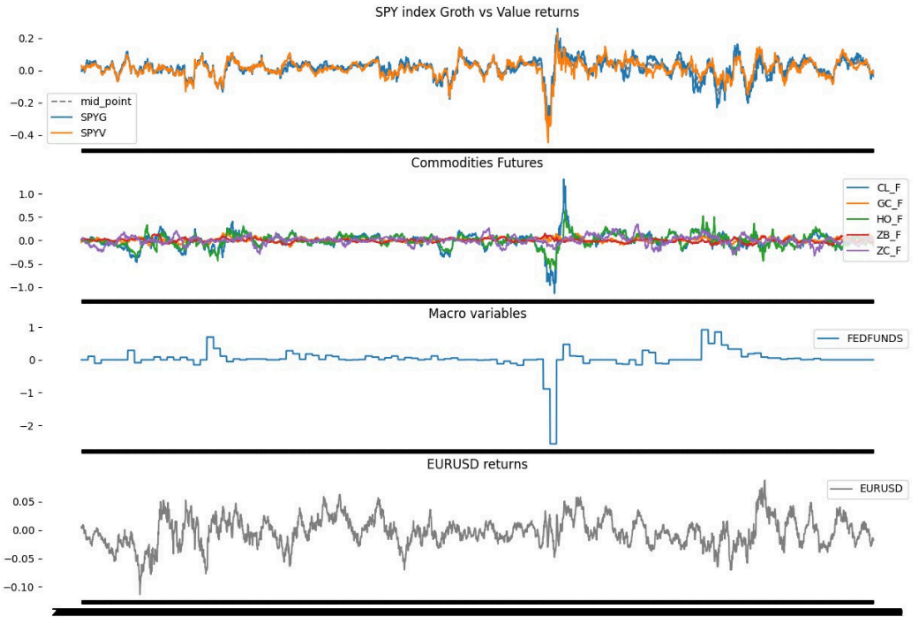
Out[ ]:

Date	SPYG	SPYV	SPYG_log_returns	SPYV_log_returns	CL_F	GC_F
2014-05-01	19.034834	18.636133	-0.002614	0.029767	-0.009510	-0.044435
2014-05-02	19.045816	18.571642	-0.004788	0.020409	0.003313	-0.021193
2014-05-03	19.045816	18.571642	-0.004788	0.020409	0.003313	-0.021193
2014-05-04	19.045816	18.571642	-0.004788	0.020409	0.003313	-0.021193
2014-05-05	19.098553	18.552103	0.002880	0.020000	0.000201	-0.020417

Visualize Data

In [ ]:

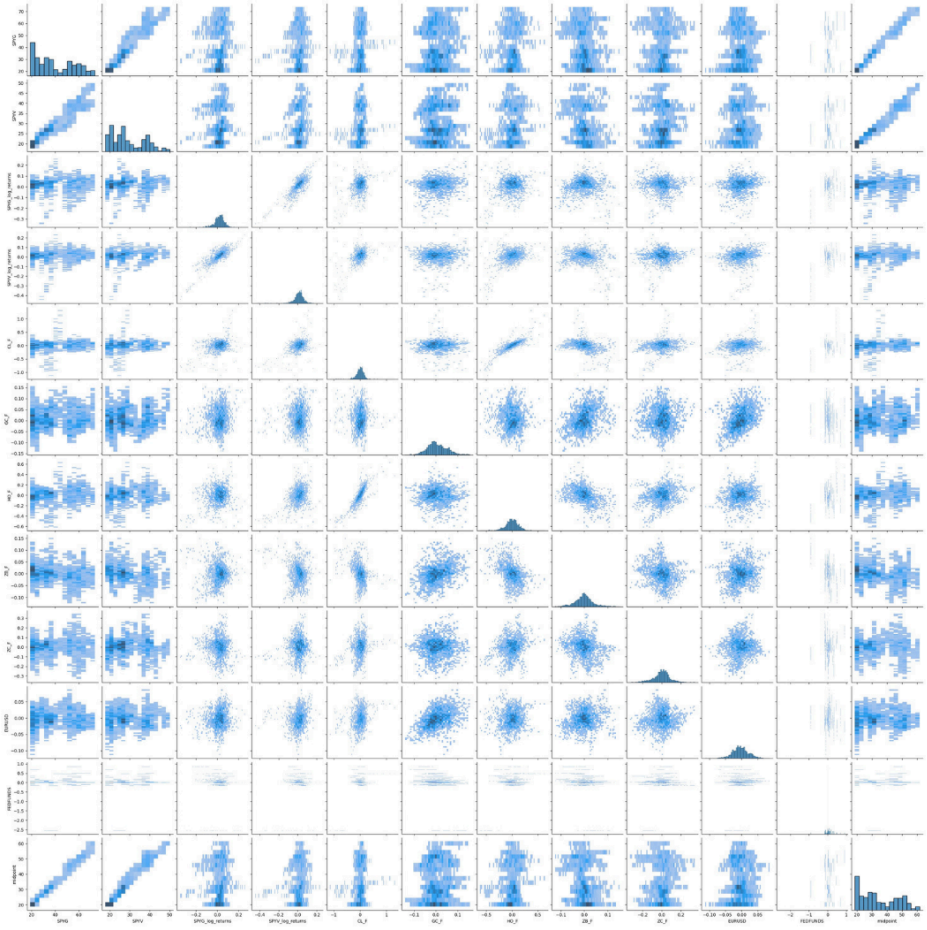
```
fig, axs = plt.subplots(nrows=4, figsize=(15, 10), sharex=True)
start,     finish     =     0,     len(all_data)     all_data     =
all_data.iloc[start:finish]         #ETFs         SPYG         =
all_data.SPYG_log_returns    SPYV    =    all_data.SPYV_log_returns
SPY_midpoint  =  (SPYG  +  SPYV)  /  2  axs[0].plot(SPY_midpoint,
color="grey",                                  linestyle="dashed")
axs[0].plot(all_data["SPYG_log_returns"])
axs[0].plot(all_data["SPYV_log_returns"])
axs[0].legend(["mid_point",           "SPYG",           "SPYV"])
axs[0].set_title("SPY index Groth vs Value returns")



#Commodities
for i in comm:
    axs[1].plot(all_data[i])
axs[1].legend(comm)
axs[1].set_title("Commodities Futures")

#Macro
axs[2].plot(all_data["FEDFUNDS"])
axs[2].legend(macro_df.columns)
axs[2].set_title("Macro variables")

#currencies
axs[3].plot(all_data["EURUSD"], color="grey")
axs[3].legend(curr_df.columns);
axs[3].set_title("EURUSD returns")


for i in range(0,4):
    axs[i].set(frame_on=False)                      extent                      =
    axs[i].get_window_extent().transformed(fig.dpi_scale_trans.inverted
    plt.savefig(os.path.join(root_dir,"charts",f'{axs[i].get_title()}.png'), bbo
```



Statistical Measures

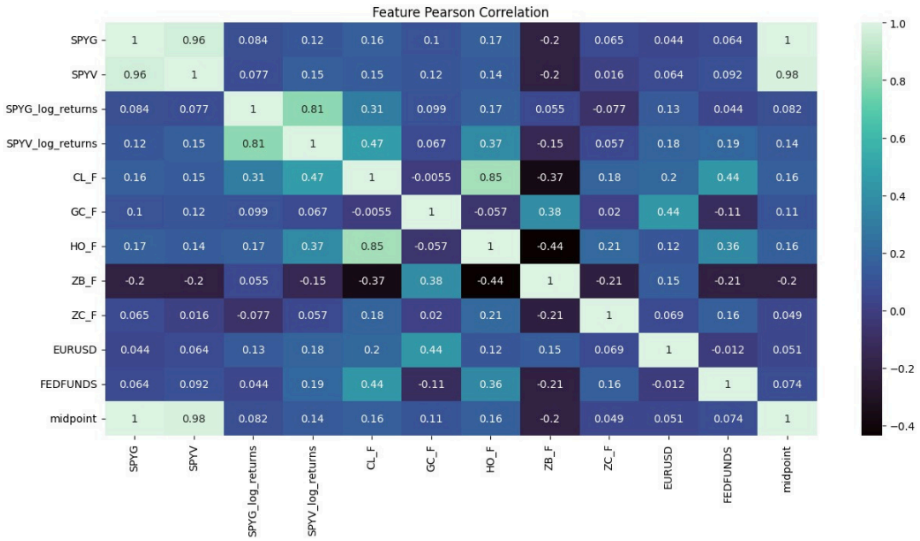
In [ ]:
```
sns.pairplot(all_data, kind="hist")
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x1f10d754bc0>
```



Correlation

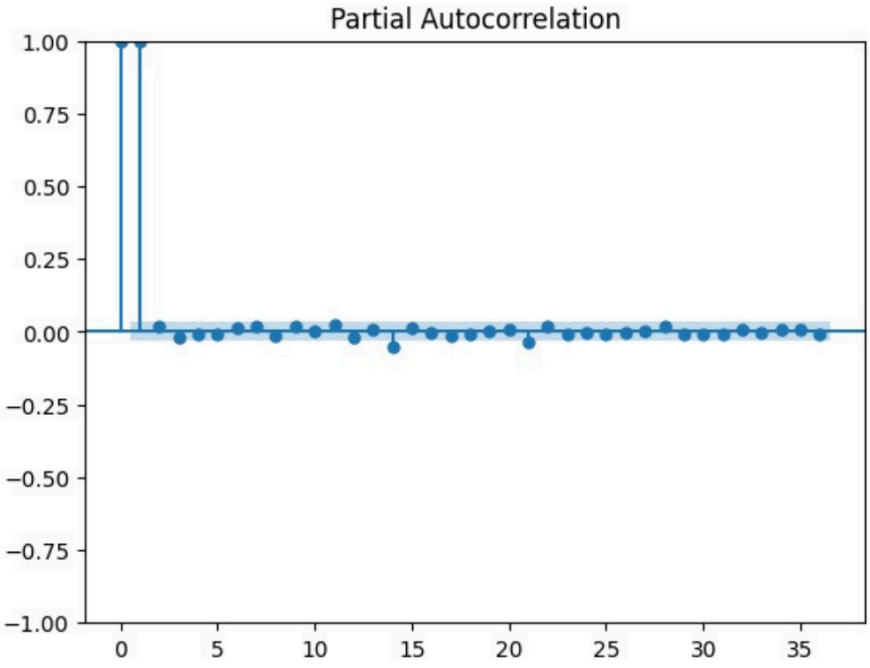
In [ ]:
```
fig = plt.figure(figsize=(15, 7))
sns.heatmap(all_data.corr(method="pearson"),cmap="mako",annot=True)
plt.title("Feature Pearson Correlation")
plt.savefig(os.path.join(root_dir,"charts",f'Feature_Correlation.png'))
```

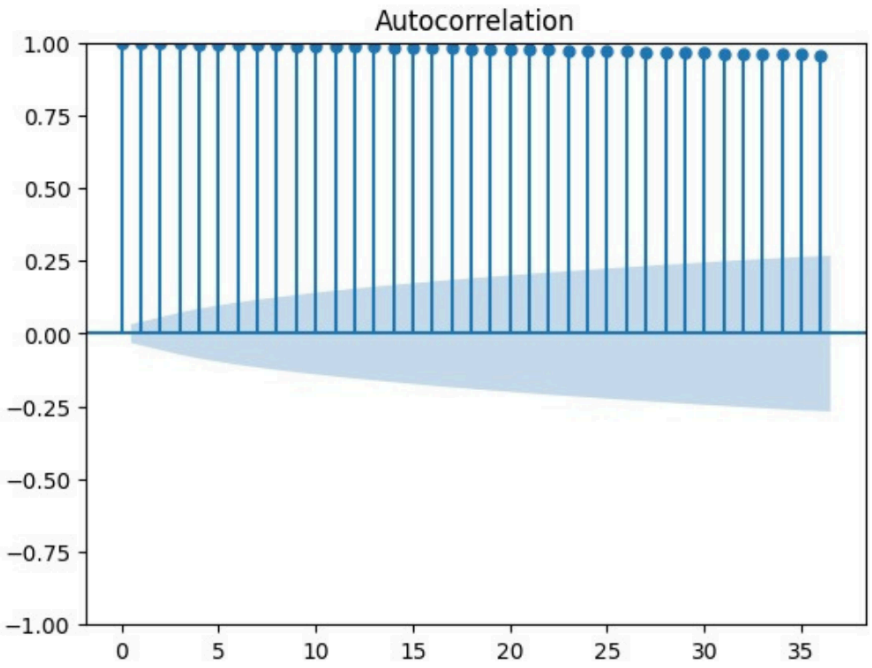


Time Series Analysis

Autocorrelation

In [ ]:
```
from statsmodels.tsa.stattools import pacf, acf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plot_pacf(all_data["midpoint"].values);
plt.savefig(os.path.join(root_dir,"charts",f'PACF.png'))
plot_acf(all_data["midpoint"].values);
```





Clasification of Growth/Value Moments

```
# seperate both investing styles using the midpoint:
```

In [ ]:
```
# 1 for growth 0 for value
from imblearn.over_sampling import SMOTE



oversampler = SMOTE()

all_data["moment"] = (SPYG > SPY_midpoint).astype(int)
all_data.dropna(inplace=True)
not_features = ["SPYG", "SPYV", "moment", "midpoint", "SPYG_log_returns", "SPYV_
features = [i for i in all_data.columns if i not in not_features]
X, y = all_data[features], all_data["moment"]


growth, value = y.value_counts()
```

In [ ]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle

#Models
models = {
    "KNN":GridSearchCV(estimator=KNeighborsClassifier(algorithm="auto"), param_g
    "RandomForest": RandomForestClassifier(criterion="gini",random_state=42),
    "AdaBoostClassifier": AdaBoostClassifier(n_estimators=200)
}
```

```
results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    results.append([name, f"{format(model.score(X_test, y_test),'.2f')}%"])
    print(classification_report(y_test, predictions))

with open(os.path.join(root_dir, "Research", "Model_Scores.txt"), 'w+') as f:
    print(tabulate(results, headers=["model", "test_score"], tablefmt="github"),
```

```
              precision    recall f1-score    support

           0       0.89      0.89      0.89       279
           1       0.93      0.93      0.93       452

    accuracy                           0.92       731
   macro avg       0.91      0.91      0.91       731
weighted avg       0.92      0.92      0.92       731

              precision    recall f1-score    support

           0       0.92      0.87      0.89       279
           1       0.92      0.95      0.94       452

    accuracy                           0.92       731
   macro avg       0.92      0.91      0.91       731
weighted avg       0.92      0.92      0.92       731

              precision    recall f1-score    support

           0       0.77      0.61      0.68       279
           1       0.79      0.89      0.84       452

    accuracy                           0.78       731
   macro avg       0.78      0.75      0.76       731
weighted avg       0.78      0.78      0.78       731
```

In [ ]:
```
# Convertir los resultados a un DataFrame de pandas
df = pd.DataFrame(results, columns=["Model", "test_score"])


table_str = tabulate(results, headers=["Model", "test_score"], tablefmt="grid")

fig, ax = plt.subplots(figsize=(6, 2))

ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=df.values, colLabels=df.columns, cellLoc='center', loc

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

plt.savefig("Model_Scores.png", bbox_inches='tight', dpi=300)
plt.show()
```

Model	test_score
KNN	0.93%
RandomForest	0.93%
AdaBoostClassifier	0.79%

Testing Predictions

In [ ]:
```
# Calculate cumulative returns for independent assets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle

predictions = models["KNN"].predict(X_test)

SPYG_cret = SPYG[y_test.index].cumsum()
SPYV_cret = SPYV[y_test.index].cumsum()
STRAT_cret = []
for i in range(len(SPYG_cret)):
if predictions[i] == 1:
        STRAT_cret.append(SPYG[y_test.index][i])
    if predictions[i] == 0:
        STRAT_cret.append(SPYV[y_test.index][i])

STRAT_cret = np.cumsum(STRAT_cret)
results = [
    ["SPYV ETF only", f"{format(SPYV_cret.values[-1],'.2f')}%"],
    ["SPYG ETF only", f"{format(SPYG_cret.values[-1],'.2f')}%"],
    ["Value/Growth Momment Classification Strategy", f"{format(STRAT_cret[-1],'.
]

headers = ["Strategies","cumulative returns"]

print(tabulate(results, headers, tablefmt="github"))
with open(os.path.join(root_dir, "Research", "SPY_Testing.txt"),'w+') as f:
    print(tabulate(results, headers, tablefmt="github"), file=f)

df = pd.DataFrame(results, columns=["Strategies","cumulative returns"])
# Generar la tabla en formato tabulate
table_str = tabulate(results, headers=["Strategies","cumulative returns"], table

fig, ax = plt.subplots(figsize=(10, 2))
fig.suptitle("USA Test",fontsize=14)

ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=df.values, colLabels=df.columns, cellLoc='center', loc

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

plt.savefig(os.path.join(root_dir, "charts","USA_test_performance.png"), bbox_in
plt.show()
```

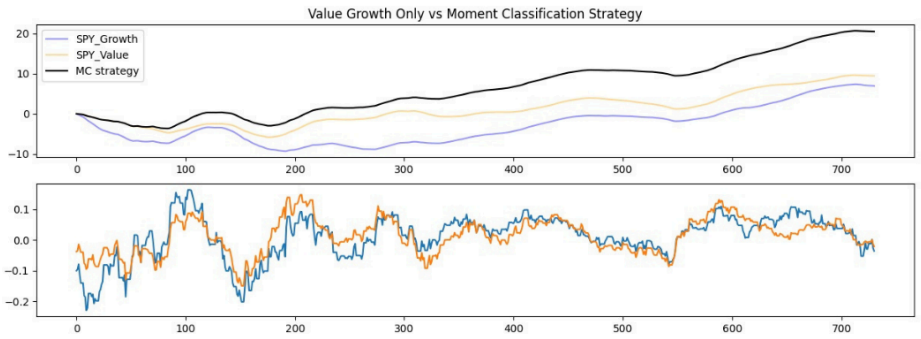
Strategies	cumulative returns
SPYV ETF only	9.43%
SPYG ETF only	6.94%
Value/Growth Momment Classification Strategy	20.51%

USA Test

Strategies	cumulative returns
SPYV ETF only	9.43%
SPYG ETF only	6.94%
Value/Growth Momment Classification Strategy	20.51%

visualize results

In [ ]:
```
fig, ax = plt.subplots(nrows=2, figsize=(15,5))

ax[0].plot(SPYG_cret.values,                color="blue",                alpha=0.4)
ax[0].plot(SPYV_cret.values,  color="orange",  alpha=0.4)  ax[0].plot(STRAT_cret,
color="black")  tittle  =  "Value  Growth  Only  vs  Moment  Classification  Strategy"
ax[0].set_title(tittle)  ax[0].legend(["SPY_Growth",  "SPY_Value",  "MC  strategy"])
ax[1].plot(SPYG[y_test.index].values)        ax[1].plot(SPYV[y_test.index].values)
plt.savefig(os.path.join(root_dir,  "charts",  f"{tittle.replace("  ",  "_")}.png"))
```



Backtesting

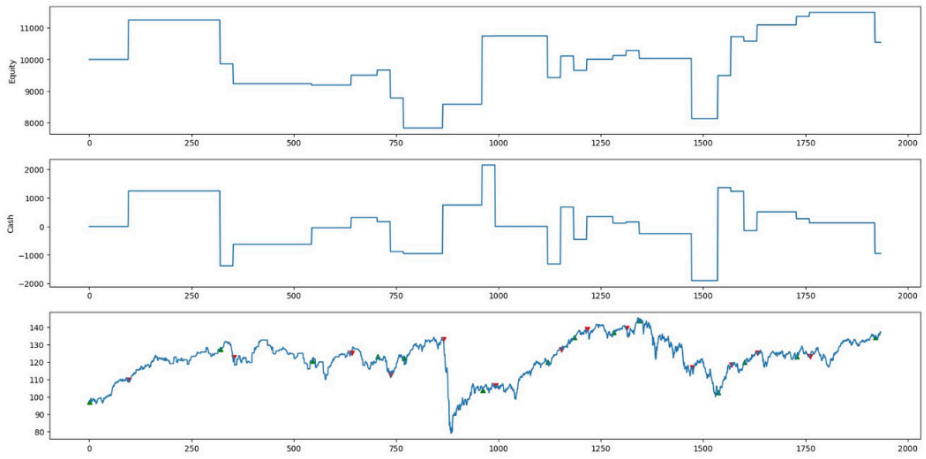
Backtesting with Long/Short Value etf in Europe

In [ ]:
```
valu_etf  =  pd.read_csv(os.path.join(DATA_DIR,  "etf_data",  "VALU.DE.csv")).set_in
valu_etf["returns"]  =  np.log(valu_etf["Close"]/valu_etf["Close"].shift(1))  #peri
valu_etf.dropna(inplace=True)

X_eu = X[X.index.isin(valu_etf.index)]
europe_predictions = models["AdaBoostClassifier"].predict(X_eu)
X_eu["predictions"] = europe_predictions
valu_etf = valu_etf[valu_etf.index.isin(X_eu.index)]
#making predictions:

buy_idx = np.where(europe_predictions == 0)
sell_idx = np.where(europe_predictions == 1)

class my_backtester:
```

```
def __init__(self) -> None:
    pass

def run(self, cash, data, buy_and_hold_days, print_results:bool=0):
    if data is None or data.empty:
        return None, None
    self.buy_and_hold_days = buy_and_hold_days self.equity
    = cash self.cash = cash self.data = data self.n =
    len(data) self.last_entry = 0 self.current_signal =
    None   self.position   =   {"short":   0,   "long":   0}
    self.history = {"equity":[], "cash": [], "entries":[]}
    i = 0 while i < self.n:




        self.current_signal = self.data.predictions[i]
        Q = 1 * self.equity
        if self.current_signal == 1 and self.position["short"] == 0 and i %
            if self.position["long"] != 0:
                value = ((self.data.Close[i] - self.last_entry) / self.last_
                    * (self.position["long"] * self.last_entry)
                self.equity += value
                self.cash = self.equity
                self.position["long"] = 0
            self.position["short"] = int((Q) / self.data.Close[i])
            self.last_entry = self.data.Close[i]
            self.cash -= Q
            i+=1
            self.history["entries"].append((self.current_signal, i))
            self.history["equity"].append(self.equity)
            self.history["cash"].append(self.cash)
            continue
        if self.current_signal == 0 and self.position["long"] == 0 and i % s
            if self.position["short"] != 0:
                value = ((self.last_entry - self.data.Close[i]) / self.data.
                    * (self.position["short"] * self.last_entry)
                self.equity += value
                self.cash = self.equity
                self.position["short"] = 0
            self.position["long"] = int((Q) / self.data.Close[i])
            self.last_entry = self.data.Close[i]
            self.cash -= Q
            i+=1
            self.history["entries"].append((self.current_signal,i))
            self.history["equity"].append(self.equity)
            self.history["cash"].append(self.cash)
            continue
        else:
            self.history["equity"].append(self.equity)
            self.history["cash"].append(self.cash)
            i+=1

    returns = (((self.history["equity"][-1] - self.history["equity"][0]) / s
    asset_returns = ((self.data["Close"][-1] - self.data["Close"][0]) / self
    results = [
        ["Asset ROI", f"{format(asset_returns,'.2f')}%"],
        ["Strategy Returs", f"{format(returns,'.2f')}%"],
```

```
        ]
        h e a d e r s   =   [ " S t r a t e g i e s " , " r e t u r n s " ]
        i f   p r i n t _ r e s u l t s :
            print(tabulate(results, headers, tablefmt="github"))
        return returns, asset_returns

    def visualize(self):
        fig,      ax      =      plt.subplots(nrows=3,      figsize=(20,      10))
        ax[0].plot(self.history["equity"])              ax[0].set_ylabel("Equity")
        ax[1].plot(self.history["cash"])                  ax[1].set_ylabel("Cash")
        ax[2].plot(self.data.Close.values)   entries_buy   =   [i[1]   for   i   in
        self.history["entries"] if i[0] == 0] entries_sell = [i[1] for i in
        self.history["entries"]   if   i[0]   ==   1]   ax[2].scatter(entries_sell,
        self.data["Close"][entries_sell],color="red"    ax[2].scatter(entries_buy,
        self.data["Close"][entries_buy],color="green"



data = pd.concat(
    [
        valu_etf,
        X_eu["predictions"]
    ],
    axis=1,
)               data.index              =
pd.DatetimeIndex(data.index)

backtester = my_backtester()
returns = backtester.run(cash=10000, data=data, buy_and_hold_days= 32,print_resu
backtester.visualize()
```

Strategies	returns
Asset ROI	40.40%
Strategy Returs	5.44%



Backtesting for many european ETFs

```
]: def make_predictions(ticker, X) -> pd.DataFrame:
        etf = pd.read_csv(os.path.join(DATA_DIR, "testing_data", f"{ticker}.csv"
        etf["returns"] = np.log(etf["Close"]/etf["Close"].shift(1)) #period = 5
        etf.dropna(inplace=True)
        X_eu = X[X.index.isin(etf.index)]
```

```
        if len(X_eu) == 0:
             return
        europe_predictions = models["KNN"].predict(X_eu)
        X_eu["predictions"] = europe_predictions
        etf = etf[etf.index.isin(X_eu.index)]
        data = pd.concat(
            [
                etf,
                X_eu["predictions"]
            ],
            axis=1,
        )
        data.index = pd.DatetimeIndex(data.index)
        return data

with open(os.path.join(DATA_DIR, "testing_data", "testing_etfs_eu.pkl"),'rb') as
    testing_etfs = pickle.load(f)

results = []
performance = 0
for i, ticker in enumerate(testing_etfs):
    data    =    make_predictions(ticker,    X=X)    returns,    asset_returns    =
    backtester.run(cash=10000,  data=data,  buy_and_hold_  if  returns  is  None  or
    asset_returns is None or asset_returns < -10 or return
         continue
    performance += returns
    if returns and asset_returns:
        results.append(
                [
                    f"{ticker}", f"{format(asset_returns,'.2f')}%",
                    f"{format(returns,'.2f')}%"],
        )
    if i + 1 == len(testing_etfs):
        results.append(["performance : ", "", f"{format(performance,'.2f')}%"])
    else:
        continue

  with open(os.path.join(root_dir, "Research", "Moment_Classification_Results.txt"
      print(tabulate(results, headers=["ETF Symbol","ROI", "Strategy Returns"], ta
```

In [ ]:

```
df = pd.DataFrame(results, columns=["ETF Symbol","ROI", "Strategy Returns"])

table_str = tabulate(results, headers=["ETF Symbol","ROI", "Strategy Returns"],

fig, ax = plt.subplots(figsize=(6, 2))
fig.suptitle("KNN",fontsize=14, x=0.1, ha='left', y=3.5)

ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=df.values, colLabels=df.columns, cellLoc='center', loc
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

plt.savefig(os.path.join(root_dir, "charts","KNN_EU_performance.png"), bbox_inch
plt.show()
```

TELE.L	1.32%	114.86%
SC00.DE	95.60%	29.57%
D5BK.DE	24.42%	85.47%
FIEE	88.07%	55.24%
SC0T.DE	104.79%	61.13%
DX2X.DE	107.49%	58.61%
FLXD.DE	46.10%	35.70%
performance :		1422.51%

In [ ]:
```
ada = models["AdaBoostClassifier"]
features_dict = dict(zip(ada.feature_names_in_,ada.feature_importances_))

results = []
for k, v in  sorted(features_dict.items(), key=lambda item: item[1]):
    results.append([k, v])

df = pd.DataFrame(results, columns=["Feature", "Weight"])
table_str = tabulate(results, headers=["Feature", "Weight"], tablefmt="grid")

fig, ax = plt.subplots(figsize=(6, 2))

ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=df.values, colLabels=df.columns, cellLoc='center', loc
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

plt.savefig(os.path.join(root_dir, "charts","AdaFeatures.png"), bbox_inches='tig
plt.show()
print(tabulate(results))
```

Feature	Weight
HO_F	0.085
GC_F	0.095
EURUSD	0.1
ZC_F	0.105
CL_F	0.115
ZB_F	0.13
FEDFUNDS	0.37

```
-------- -----
HO_F 0.085
GC_F 0.095
EURUSD 0.1
ZC_F 0.105
CL_F 0.115
ZB_F 0.13
FEDFUNDS 0.37
-------- -----
```

In [ ]:
```
```