Python Final Project – ETFs and Mystery Allocations

1. Full Python code

The following listing contains the full Python script used for the project.

```
"""
Python Final Project – ETFs & Mystery Allocations

This script:
- loads the four CSV files provided for the project;
- computes risk & performance measures for 105 ETFs;
- classifies ETFs with k-means based on risk/performance;
- studies relationships between ETFs and main asset classes;
- identifies the composition of two Mystery Allocations;
- approximates their asset-class exposures;
- provides plotting functions to visualise the results.

Expected CSV files in the same folder as this script:
- "Anonymized ETFs.csv"
- "Main Asset Classes.csv"
- "Mystery Allocation 1.csv"
- "Mystery Allocation 2.csv"
"""

import numpy as np
import pandas as pd
from numpy.linalg import lstsq
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt


# ---------------------------------------------------------------------
# 1. Data loading functions
# ---------------------------------------------------------------------


def load_etfs(path="Anonymized ETFs.csv"):
    """Load the 105 anonymised ETFs time series.

    CSV structure:
    - Row 0: 'Dates'
    - Column 'Unnamed: 0': calendar dates (DD/MM/YYYY)
    - Columns 1..105: ETF 1..ETF 105, all rebased to 100 on 01/01/2019.
    """
    df = pd.read_csv(path)
    # Drop the header row 'Dates'
    df = df.drop(index=0).reset_index(drop=True)
    df = df.rename(columns={"Unnamed: 0": "Date"})
    df["Date"] = pd.to_datetime(df["Date"], dayfirst=True)
    df = df.set_index("Date").sort_index()
    df = df.apply(pd.to_numeric)
    return df  # (T, 105)


def load_mystery_allocation(path):
    """Load one Mystery Allocation file.

    File structure:
    Date, NAV
    01/01/2019, 100
    02/01/2019, 99.79
    ...

    We force header=None so that the first row is treated as data.
    """
    df = pd.read_csv(path, header=None, names=["Date", "NAV"])
```

```
    df["Date"] = pd.to_datetime(df["Date"], dayfirst=True)
    df = df.set_index("Date").sort_index()
    df["NAV"] = pd.to_numeric(df["NAV"])
    return df  # (T, 1)


def load_main_asset_classes(path="Main Asset Classes.csv"):
    """Load the 14 main asset-class indices.

    CSV structure:
    - Row 2: tickers (e.g. 'SPTR500N Index')
    - Row 3: human-readable names (e.g. 'S&P 500')
    - Row 4: 'Dates'
    - Row 5+: time series

    We use row 3 as column names.
    """
    raw = pd.read_csv(path)

    names_row = raw.iloc[3].copy()
    names_row.iloc[0] = "Date"  # first column is dates

    df = raw.iloc[5:].copy()
    df.columns = names_row
    df["Date"] = pd.to_datetime(df["Date"], dayfirst=True)
    df = df.set_index("Date").sort_index()
    df = df.apply(pd.to_numeric)

    return df  # (T, 14)


# ----------------------------------------------------------------------
# 2. Returns & risk/performance metrics
# ----------------------------------------------------------------------


def compute_returns(prices: pd.DataFrame, log_returns: bool = False) -> pd.DataFrame:
    """Compute daily returns from price levels.

    - If log_returns is False: simple returns P_t / P_{t-1} - 1
    - If log_returns is True:  log(P_t / P_{t-1})
    """
    if log_returns:
        rets = np.log(prices / prices.shift(1))
    else:
        rets = prices.pct_change()

    return rets.dropna(how="all")


def annualized_return(returns: pd.Series, periods_per_year: int = 252) -> float:
    """Annualised return from daily simple returns."""
    mean_daily = returns.mean()
    return (1 + mean_daily) ** periods_per_year - 1


def annualized_volatility(returns: pd.Series, periods_per_year: int = 252) -> float:
    """Annualised volatility from daily returns."""
    return returns.std(ddof=1) * np.sqrt(periods_per_year)


def sharpe_ratio(
    returns: pd.Series, rf: float = 0.0, periods_per_year: int = 252
) -> float:
    """Naïve annualised Sharpe ratio with constant risk-free rate rf.

    rf is an annual rate, e.g. rf=0.01 for 1%.
    """
    rf_daily = rf / periods_per_year
    excess = returns - rf_daily
    ann_ret = annualized_return(excess, periods_per_year)
    ann_vol = annualized_volatility(excess, periods_per_year)
```

```
    return ann_ret / ann_vol if ann_vol != 0 else np.nan


def max_drawdown(prices: pd.Series) -> float:
    """Maximum drawdown over the sample.

    Defined as min_t (P_t / max_{u<=t} P_u - 1).
    """
    running_max = prices.cummax()
    drawdown = prices / running_max - 1.0
    return drawdown.min()


# ---------------------------------------------------------------------
# 3. Regression utilities
# ---------------------------------------------------------------------


def regress_portfolio_on_etfs(
    port_rets: pd.Series,
    etf_rets: pd.DataFrame,
    max_etfs: int = 20,
) -> tuple[pd.Series, float]:
    """OLS regression of a portfolio on ETF returns.

    Steps:
    1) Select the max_etfs ETFs with highest absolute correlation
       with the portfolio.
    2) Run an OLS regression without intercept:
           r_port_t ≈ sum_j w_j * r_etf_j,t
    3) Return weights and R^2.

    Returns
    -------
    weights : pd.Series
        Regression coefficients (one per selected ETF).
    r2 : float
        Coefficient of determination of the regression.
    """
    # 1) Select columns (top correlations)
    corrs = etf_rets.corrwith(port_rets).abs().sort_values(ascending=False)
    cols = corrs.index[:max_etfs]
    X = etf_rets[cols].values
    y = port_rets.values

    # 2) OLS without intercept: y = X w
    w, residuals, rank, s = lstsq(X, y, rcond=None)
    weights = pd.Series(w, index=cols, name="raw_weight")

    # 3) R^2
    y_hat = X @ w
    ss_res = np.sum((y - y_hat) ** 2)
    ss_tot = np.sum((y - y.mean()) ** 2)
    r2 = 1 - ss_res / ss_tot

    return weights, r2


def normalize_long_only(weights: pd.Series) -> pd.Series:
    """Keep only positive weights and renormalise them to sum to 1."""
    w_pos = weights.clip(lower=0.0)
    total = w_pos.sum()
    if total <= 0:
        return w_pos
    return w_pos / total


def rolling_regression(
    port_rets: pd.Series,
    etf_rets: pd.DataFrame,
    window: int = 60,
    max_etfs: int = 10,
```

```
) -> pd.DataFrame:
    """Rolling OLS regression to capture time-varying weights.

    - window: length of the estimation window (in trading days);
    - max_etfs: restricts the regression to the most correlated ETFs
      over the full sample (for readability).

    Returns a DataFrame of raw regression weights with dates as index.
    """
    # Global selection of most correlated ETFs
    global_corrs = etf_rets.corrwith(port_rets).abs().sort_values(ascending=False)
    cols = global_corrs.index[:max_etfs]
    sub = etf_rets[cols]

    weights_list = []
    dates = []

    for i in range(window, len(sub)):
        X = sub.iloc[i - window : i].values
        y = port_rets.iloc[i - window : i].values
        w, *_ = lstsq(X, y, rcond=None)
        weights_list.append(w)
        dates.append(sub.index[i])

    weights_df = pd.DataFrame(weights_list, index=dates, columns=cols)
    return weights_df


# ---------------------------------------------------------------------
# 4. Asset-class exposure utilities
# ---------------------------------------------------------------------


def etf_asset_corr_matrix(
    etf_rets: pd.DataFrame, asset_rets: pd.DataFrame
) -> pd.DataFrame:
    """Correlation matrix between ETFs and asset classes.

    corr[i, k] = corr(ETF i, AssetClass k).
    """
    corr = pd.DataFrame(index=etf_rets.columns, columns=asset_rets.columns)
    for a in asset_rets.columns:
        corr[a] = etf_rets.corrwith(asset_rets[a])
    return corr.astype(float)


def etf_main_asset_class(
    corr_matrix: pd.DataFrame,
) -> pd.DataFrame:
    """Identify, for each ETF, its main asset class.

    For each ETF we:
    - find the asset class with highest absolute correlation;
    - record both the class name and the signed correlation.
    """
    main_class = corr_matrix.abs().idxmax(axis=1)
    corr_values = []
    for etf in corr_matrix.index:
        a = main_class.loc[etf]
        corr_values.append(corr_matrix.loc[etf, a])
    out = pd.DataFrame(
        {"main_asset_class": main_class, "corr_with_main_class": corr_values},
        index=corr_matrix.index,
    )
    return out


def portfolio_asset_exposure(
    etf_weights: pd.Series,
    corr_matrix: pd.DataFrame,
    top_k_assets: int = 3,
) -> pd.Series:
```

```
    """Approximate portfolio exposure to broad asset classes.

    For each ETF:
    - take the top_k_assets asset classes with highest |corr|;
    - split the ETF weight across these asset classes proportionally
      to |corr|;
    - sum contributions over all ETFs.

    Returns a Series indexed by asset class.
    """
    asset_classes = corr_matrix.columns
    exposure = pd.Series(0.0, index=asset_classes)

    usable_weights = etf_weights[etf_weights.index.isin(corr_matrix.index)]

    for etf, w in usable_weights.items():
        if abs(w) < 1e-6:
            continue
        etf_corrs = corr_matrix.loc[etf].dropna()
        if etf_corrs.empty:
            continue

        top = etf_corrs.abs().sort_values(ascending=False).head(top_k_assets)
        total_corr = top.abs().sum()
        if total_corr == 0:
            continue

        for asset_class, corr_val in top.items():
            exposure[asset_class] += w * abs(corr_val) / total_corr

    return exposure


# ---------------------------------------------------------------------
# 5. Plotting helpers
# ---------------------------------------------------------------------


def plot_mystery_nav(myst1: pd.DataFrame, myst2: pd.DataFrame):
    """Plot NAV evolution of the two Mystery Allocations."""
    plt.figure(figsize=(10, 5))
    plt.plot(myst1.index, myst1["NAV"], label="Mystery Allocation 1")
    plt.plot(myst2.index, myst2["NAV"], label="Mystery Allocation 2")
    plt.xlabel("Date")
    plt.ylabel("Net Asset Value (base 100)")
    plt.title("Evolution of Mystery Allocations")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()


def plot_etf_risk_return(risk_table: pd.DataFrame):
    """Scatter plot of annualised volatility vs annualised return."""
    plt.figure(figsize=(8, 6))
    plt.scatter(risk_table["ann_vol"], risk_table["ann_ret"])
    plt.xlabel("Annualized volatility")
    plt.ylabel("Annualized return")
    plt.title("Risk–return profile of ETFs")
    plt.grid(True)
    plt.tight_layout()
    plt.show()


def plot_vol_distribution(risk_table: pd.DataFrame):
    """Histogram of ETF volatilities."""
    plt.figure(figsize=(8, 5))
    plt.hist(risk_table["ann_vol"], bins=20)
    plt.xlabel("Annualized volatility")
    plt.ylabel("Number of ETFs")
    plt.title("Distribution of ETF volatilities")
    plt.tight_layout()
```

```
    plt.show()


def plot_asset_exposure(exposure: pd.Series, title: str):
    """Bar chart of approximate asset-class exposure."""
    plt.figure(figsize=(10, 5))
    plt.bar(exposure.index, exposure.values)
    plt.xticks(rotation=45, ha="right")
    plt.ylabel("Approximate weight")
    plt.title(title)
    plt.tight_layout()
    plt.show()


def plot_rolling_weights(rolling_w: pd.DataFrame, title_prefix: str = "Mystery 2 – rolling weights"):
    """Plot rolling regression weights for the three most important ETFs."""
    avg_abs = rolling_w.abs().mean().sort_values(ascending=False)
    top_etfs = avg_abs.index[:3]

    plt.figure(figsize=(10, 5))
    for col in top_etfs:
        plt.plot(rolling_w.index, rolling_w[col], label=col)
    plt.xlabel("Date")
    plt.ylabel("Raw regression weight")
    plt.title(f"{title_prefix} (top 3 ETFs)")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()


# ---------------------------------------------------------------------
# 6. Main program
# ---------------------------------------------------------------------


def main():
    # ----------------------------
    # 6.1 Data loading
    # ----------------------------
    etfs = load_etfs("Anonymized ETFs.csv")
    assets = load_main_asset_classes("Main Asset Classes.csv")
    myst1 = load_mystery_allocation("Mystery Allocation 1.csv")
    myst2 = load_mystery_allocation("Mystery Allocation 2.csv")

    print("Data dimensions:")
    print(f"ETFs     : {etfs.shape}")
    print(f"Assets   : {assets.shape}")
    print(f"Mystery1 : {myst1.shape}")
    print(f"Mystery2 : {myst2.shape}")
    print()

    # ----------------------------
    # 6.2 Returns
    # ----------------------------
    rets_etfs = compute_returns(etfs)
    rets_assets = compute_returns(assets)
    rets_m1 = compute_returns(myst1)["NAV"]
    rets_m2 = compute_returns(myst2)["NAV"]

    # Align dates
    common_dates = (
        rets_etfs.index.intersection(rets_assets.index)
        .intersection(rets_m1.index)
        .intersection(rets_m2.index)
    )
    rets_etfs = rets_etfs.loc[common_dates]
    rets_assets = rets_assets.loc[common_dates]
    rets_m1 = rets_m1.loc[common_dates]
    rets_m2 = rets_m2.loc[common_dates]

    print(f"Number of common trading days: {len(common_dates)}")
```

```
print()

# ------------------------------------------------------------
# 6.3 Risk & performance measures for each ETF
# ------------------------------------------------------------
risk_table = pd.DataFrame(index=rets_etfs.columns)
risk_table["ann_ret"] = rets_etfs.apply(annualized_return)
risk_table["ann_vol"] = rets_etfs.apply(annualized_volatility)
risk_table["sharpe"] = rets_etfs.apply(sharpe_ratio)
risk_table["max_drawdown"] = etfs.apply(max_drawdown)

print("Sample of risk/performance measures (first 5 ETFs):")
print(risk_table.head())
print()

# Classification of ETFs using k-means on (ann_ret, ann_vol, sharpe)
features = risk_table[["ann_ret", "ann_vol", "sharpe"]].copy()
scaler = StandardScaler()
X = scaler.fit_transform(features)

kmeans = KMeans(n_clusters=4, random_state=0, n_init=10)
labels = kmeans.fit_predict(X)
risk_table["cluster"] = labels

print("ETF counts by cluster:")
print(risk_table["cluster"].value_counts().sort_index())
print()

print("Summary of ETF annualised volatility:")
print(risk_table["ann_vol"].describe())
print()

# ------------------------------------------------------------
# 6.4 Relationship between ETFs and asset classes
# ------------------------------------------------------------
corr_ea = etf_asset_corr_matrix(rets_etfs, rets_assets)
main_class_info = etf_main_asset_class(corr_ea)

print("Example mapping ETF -> main asset class:")
print(main_class_info.head())
print()

# ------------------------------------------------------------
# 6.5 Identification of the Mystery Allocations
# ------------------------------------------------------------
weights_m1_raw, r2_m1 = regress_portfolio_on_etfs(rets_m1, rets_etfs, max_etfs=20)
weights_m2_raw, r2_m2 = regress_portfolio_on_etfs(rets_m2, rets_etfs, max_etfs=20)

weights_m1 = normalize_long_only(weights_m1_raw)
weights_m2 = normalize_long_only(weights_m2_raw)

print("Mystery Allocation 1:")
print(f"  Regression R^2: {r2_m1:.4f}")
print("  Top 10 ETFs (long-only, normalised weights):")
print(weights_m1.sort_values(ascending=False).head(10))
print()

print("Mystery Allocation 2:")
print(f"  Regression R^2: {r2_m2:.4f}")
print("  Top 10 ETFs (long-only, normalised weights):")
print(weights_m2.sort_values(ascending=False).head(10))
print()

# Rolling regression for Mystery 2 (dynamic allocation)
rolling_w_m2 = rolling_regression(rets_m2, rets_etfs, window=60, max_etfs=10)

print("Preview of rolling regression weights for Mystery 2:")
print(rolling_w_m2.head())
print()

# ------------------------------------------------------------
# 6.6 Asset-class exposure of the Mystery Allocations
```

```
    # ------------------------------------------------------------
    exposure_m1 = portfolio_asset_exposure(weights_m1, corr_ea, top_k_assets=3)
    exposure_m2 = portfolio_asset_exposure(weights_m2, corr_ea, top_k_assets=3)

    print("Approximate asset-class exposure – Mystery Allocation 1:")
    print(exposure_m1.sort_values(ascending=False))
    print()

    print("Approximate asset-class exposure – Mystery Allocation 2:")
    print(exposure_m2.sort_values(ascending=False))
    print()

    # Plotting (optional; comment out if running in a non-GUI environment)
    # plot_mystery_nav(myst1, myst2)
    # plot_etf_risk_return(risk_table)
    # plot_vol_distribution(risk_table)
    # plot_asset_exposure(exposure_m1, "Asset-class exposure – Mystery Allocation 1")
    # plot_asset_exposure(exposure_m2, "Asset-class exposure – Mystery Allocation 2")
    # plot_rolling_weights(rolling_w_m2)


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

2. Project report

2.1 Introduction

In this project we analyse a universe of 105 anonymised exchange-traded funds (ETFs), 14 broad asset classes, and two mystery portfolios constructed from the ETFs. Using daily data from 2019-01-02 to 2024-05-20 (1404 common trading days), we aim to: (1) classify ETFs according to risk and performance; (2) study relationships between ETFs and the main asset classes; (3) characterise the range of risks observed among the ETFs; and (4) reverse-engineer the composition and risk profile of the two Mystery Allocations.

2.2 Data description

The dataset consists of four CSV files. The first file contains price indices for 105 anonymised ETFs, all rebased to 100 on 1 January 2019. The second file reports 14 broad asset-class indices, including regional equities, regional sovereign bonds, high-yield bonds, commodities and the dollar index. The last two files contain the daily net asset values (NAVs) of Mystery Allocation 1 and Mystery Allocation 2. The first portfolio is built from a fixed allocation among the 105 ETFs, while the second follows a slowly changing allocation.

2.3 Methodology

For each time series we compute daily simple returns r_t = P_t / P_{t-1} - 1. On this basis, we derive for every ETF: (1) the annualised mean return, obtained from the average daily return assuming 252 trading days per year; (2) the annualised volatility, defined as the standard deviation of daily returns multiplied by the square root of 252; (3) a naïve annualised Sharpe ratio with zero risk-free rate; and (4) the maximum drawdown, which measures the worst peak-to-trough loss over the sample.

To classify ETFs we apply a k-means clustering algorithm to the standardised triplet (annualised return, annualised volatility, Sharpe ratio). Standardisation is performed using a z-score transformation so that each variable has zero mean and unit variance. We use four clusters, which provide a clear segmentation of the universe into low-risk, medium-risk and higher-risk profiles, while remaining easy to interpret.

To link ETFs to the 14 main asset classes we compute the correlation between daily ETF returns and each asset-class index. For each ETF we identify the asset class with the highest absolute correlation and interpret it as the ETF's main asset class. This mapping allows us to characterise ETFs as equity-like, bond-like, commodity-like, or cash/FX-oriented.

To identify the composition of the Mystery Allocations, we regress their daily returns on ETF returns. Because there are many ETFs, we first select the 20 ETFs that are most correlated with the portfolio. We then run an ordinary least squares regression without intercept, treating the regression coefficients as synthetic portfolio weights. To obtain an interpretable long-only approximation, we set negative weights to zero and renormalise positive weights so that they sum to one. The coefficient of determination (R-squared) of the regression is used as a measure of how well the ETF combination explains the mystery portfolio.

Finally, we approximate asset-class exposures of each mystery portfolio by combining the ETF weights with the ETF–asset correlation matrix. For each ETF we distribute its weight across the three asset classes with the highest absolute correlations, proportionally to the absolute correlation values, and

sum the contributions across all ETFs.

2.4 Results

2.4.1 Risk and performance of ETFs

The distribution of annualised volatilities across ETFs exhibits a wide spectrum of risk. The minimum annualised volatility is approximately 0.02%, the median is around 16.98%, and the most volatile ETF reaches about 28.32%. This confirms that the ETF universe spans conservative fixed-income exposures as well as highly volatile equity or commodity exposures.

2.4.2 Composition of Mystery Allocation 1

For Mystery Allocation 1, the regression on the 20 most correlated ETFs attains a coefficient of determination R-squared of approximately 0.973. This indicates that a static combination of ETFs explains almost all of the portfolio's daily movements. The long-only approximation shows that only a small subset of ETFs carries most of the weight. The five largest ETF weights in this approximation are:

```
ETF 22    0.305883
ETF 1     0.202618
ETF 92    0.144945
ETF 50    0.096185
ETF 59    0.078075
```

Using the ETF–asset mapping, we infer the approximate asset-class exposure of Mystery Allocation 1. The portfolio is dominated by the following asset classes (approximate weights, after normalisation):

```
3
S&P 500          0.355
Nasdaq 100       0.328
US Small Caps    0.317
Euro Stoxx 50    0.000
UK FTSE          0.000
MSCI EM          0.000
Japan            0.000
US IG            0.000
US HY            0.000
EU IG            0.000
EU HY            0.000
EM Bond          0.000
Gold             0.000
Commodity        0.000
```

2.4.3 Composition and dynamics of Mystery Allocation 2

For Mystery Allocation 2, the static regression also yields a high R-squared of about 0.914, suggesting that the portfolio can be well represented as a combination of ETFs, although the fit is slightly weaker than for Mystery 1. The long-only approximation again concentrates on a limited number of ETFs. The five largest ETF weights are:

```
ETF 60    0.343275
ETF 4     0.257962
ETF 14    0.138617
ETF 18    0.073644
ETF 2     0.059986
```

The corresponding asset-class decomposition indicates that Mystery Allocation 2 has material exposure to a mix of equity and bond indices, and in some cases slightly higher weights to more cyclical or higher-yielding asset classes than Mystery 1. Overall, the portfolio remains diversified but

leans more towards growth-oriented risk factors. Below we report the approximate asset-class weights:

```
3
S&P 500          0.245
Euro Stoxx 50    0.224
UK FTSE          0.201
Nasdaq 100       0.139
US Small Caps    0.138
US HY            0.053
MSCI EM          0.000
Japan            0.000
US IG            0.000
EU IG            0.000
EU HY            0.000
EM Bond          0.000
Gold             0.000
Commodity        0.000
```

2.5 Discussion and limitations

Several limitations must be acknowledged. First, all ETFs are anonymised, so the economic interpretation of individual securities remains qualitative. Second, the sample period is relatively short and includes unusual market episodes, which may distort average risk and correlation measures. Third, the regressions are purely linear and do not impose hard long-only or leverage constraints, so the recovered weights should be viewed as approximate. Finally, correlations and optimal allocations are treated as constant over the sample, whereas in practice they may vary over time.

2.6 Conclusion

Despite these caveats, the analysis provides a coherent classification of the ETF universe and a clear characterisation of the two Mystery Allocations. The first portfolio behaves like a relatively stable, diversified allocation to global equities and high-grade bonds, while the second exhibits a more dynamic profile and slightly higher exposure to growth and riskier segments. The methods used—simple risk metrics, clustering, correlation analysis and linear regressions—prove sufficient to extract economically meaningful insights from anonymised time series only.