```python
import yfinance as yf
import numpy as np
import pandas as pd
from scipy.stats import wasserstein_distance
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from tqdm import tqdm
import seaborn as sns


print("Getting predicted prices")
data = pd.read_csv("Results/Price_predictions.csv",
parse_dates=['Date'])
data.set_index('Date', inplace=True)
data.loc[:, "LogReturn"] = np.log(data["Predicted"] /
data["Predicted"].shift(1))
data.dropna()
```

```
Getting predicted prices
```

```
            Actual_Price   Predicted  LogReturn
Date
2016-06-07     22.663265   22.855927   0.002966
2016-06-08     22.642670   22.945700   0.003920
2016-06-09     22.805153   22.977365   0.001379
2016-06-10     22.617485   23.076778   0.004317
2016-06-13     22.276501   23.024837  -0.002253
...                  ...         ...        ...
2025-03-31    222.130005  219.570820  -0.011448
2025-04-01    223.190002  220.406420   0.003798
2025-04-02    223.889999  221.536680   0.005115
2025-04-03    203.190002  222.429760   0.004023
2025-04-04    188.380005  212.056080  -0.047761

[2221 rows x 3 columns]
```

```python
from scipy.stats import wasserstein_distance
from sklearn.cluster import KMeans
import numpy as np
from tqdm import tqdm

# === Parameters ===
window_size = 126
n_clusters = 3
returns = data["LogReturn"].values
windows = [returns[i:i+window_size] for i in range(len(returns) -
window_size)]
```

```
# === Compute Wasserstein distance matrix (with trend augmentation)
===
n = len(windows)
dist_matrix = np.zeros((n, n))

for i in tqdm(range(n), desc="Computing Wasserstein distances"):
    for j in range(i, n):
        if len(windows[i]) == 0 or len(windows[j]) == 0:
            dist_matrix[i, j] = np.inf
        else:
            try:
                base_dist = wasserstein_distance(windows[i],
windows[j])
                trend_diff = np.abs(np.sum(windows[i]) -
np.sum(windows[j])) * 0.3
                dist = base_dist + trend_diff
                dist_matrix[i, j] = dist if not np.isnan(dist) else
np.inf
            except:
                dist_matrix[i, j] = np.inf
        dist_matrix[j, i] = dist_matrix[i, j]

max_finite = np.max(dist_matrix[np.isfinite(dist_matrix)],
initial=1.0)
dist_matrix[~np.isfinite(dist_matrix)] = 10 * max_finite
dist_matrix = dist_matrix / np.max(dist_matrix)

# === KMeans Clustering ===
kmeans = KMeans(n_clusters=n_clusters,
random_state=42).fit(dist_matrix)
regimes = kmeans.labels_

# === Label Clusters Based on Return and Volatility ===
cluster_stats = []
for i in range(n_clusters):
    cluster_returns = [windows[j] for j in range(len(windows)) if
regimes[j] == i]
    cumulative_return = np.mean([np.sum(r) for r in cluster_returns])
    mean_vol = np.mean([np.std(r) for r in cluster_returns])
    cluster_stats.append((cumulative_return, mean_vol))

sorted_clusters = np.argsort([stat[0] for stat in cluster_stats])[::-
1]

regime_labels = np.array(["Sideways"] * len(regimes))
regime_labels[regimes == sorted_clusters[0]] = "Bull"
regime_labels[regimes == sorted_clusters[-1]] = "Bear"

# === Post-processing: Re-label based on thresholds ===
```

```python
for i in range(len(regime_labels)):
    total_return = np.sum(windows[i])
    if total_return > 0.15:
        regime_labels[i] = "Bull"
    elif total_return < -0.15:
        regime_labels[i] = "Bear"
    elif -0.05 <= total_return <= 0.05:
        regime_labels[i] = "Sideways"

# === Output ===
# regime_labels -> array of "Bull", "Bear", "Sideways"
# windows -> list of return windows used
# dist_matrix -> normalized Wasserstein distance matrix
```

```
Computing Wasserstein distances: 100%|██████████| 2096/2096
[00:59<00:00, 35.49it/s]
```

```python
aapl = yf.download('AAPL')
aapl
```

```
YF.download() has changed argument auto_adjust default to True

[*********************100%***********************]  1 of 1 completed

Price            Close        High         Low        Open     Volume
Ticker            AAPL        AAPL        AAPL        AAPL       AAPL
Date
1980-12-12    0.098726    0.099155    0.098726    0.098726  469033600
1980-12-15    0.093575    0.094005    0.093575    0.094005  175884800
1980-12-16    0.086707    0.087136    0.086707    0.087136  105728000
1980-12-17    0.088853    0.089282    0.088853    0.088853   86441600
1980-12-18    0.091429    0.091858    0.091429    0.091429   73449600
...                ...         ...         ...         ...        ...
2025-03-31  222.130005  225.619995  216.229996  217.009995   65299300
2025-04-01  223.190002  223.679993  218.899994  219.809998   36412700
2025-04-02  223.889999  225.190002  221.020004  221.320007   35905900
2025-04-03  203.190002  207.490005  201.250000  205.539993  103419000
2025-04-04  188.380005  199.880005  187.339996  193.889999  125569000

[11169 rows x 5 columns]
```

```python
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# === Extract values ===
dates = data.index.to_numpy()
price = data["LSTM_Predicted"].values
log_returns = data["LogReturn"].values

# === Create subplots ===
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(16, 10),
```

```python
gridspec_kw={'height_ratios': [2, 1]})

# === Regime colors ===
colors = {'Bull': 'green', 'Bear': 'red', 'Sideways': 'gray'}

# === Plot 1: Price with regime shading ===
ax1.plot(dates, price, color='black', lw=1)

for i, label in enumerate(regime_labels):
    start = dates[i]
    end = dates[i + window_size]
    ax1.axvspan(start, end, color=colors[label], alpha=0.1)

ax1.set_title(f"{data.index.name} with {window_size}-Day Regimes",
fontsize=14)
ax1.set_ylabel("Price", fontsize=12)

# === Plot 2: Log returns with regime-colored points ===
for i, label in enumerate(regime_labels):
    start = i + window_size
    end = start + 1 if i < len(regime_labels) - 1 else len(data)
    ax2.scatter(dates[start:end], log_returns[start:end],
                color=colors[label], s=5, alpha=0.5, label=label if i
== 0 else "")

ax2.set_title("Daily Log Returns by Regime", fontsize=14)
ax2.set_ylabel("Log Returns", fontsize=12)

# === Format x-axis ===
for ax in [ax1, ax2]:
    ax.xaxis.set_major_locator(mdates.YearLocator(5))
    ax.xaxis.set_major_formatter(mdates.DateFormatter("%Y"))
    ax.grid(True, linestyle='--', alpha=0.5)

# === Legend and Save ===
ax2.legend()
plt.tight_layout()
plt.savefig(f'regimes_{window_size}days.png', dpi=300,
bbox_inches='tight')
plt.show()

----------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
File ~\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-
packages\Python312\site-packages\pandas\core\indexes\base.py:3805, in
Index.get_loc(self, key)
   3804 try:
```

```
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas\\_libs\\hashtable_class_helper.pxi:7081, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas\\_libs\\hashtable_class_helper.pxi:7089, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'LSTM_Predicted'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call
last)
Cell In[8], line 6
      4 # === Extract values ===
      5 dates = data.index.to_numpy()
----> 6 price = data["LSTM_Predicted"].values
      7 log_returns = data["LogReturn"].values
      9 # === Create subplots ===

File ~\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-
packages\Python312\site-packages\pandas\core\frame.py:4102, in
DataFrame.__getitem__(self, key)
   4100 if self.columns.nlevels > 1:
   4101     return self._getitem_multilevel(key)
-> 4102 indexer = self.columns.get_loc(key)
   4103 if is_integer(indexer):
   4104     indexer = [indexer]

File ~\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-
packages\Python312\site-packages\pandas\core\indexes\base.py:3812, in
Index.get_loc(self, key)
   3807     if isinstance(casted_key, slice) or (
   3808         isinstance(casted_key, abc.Iterable)
   3809         and any(isinstance(x, slice) for x in casted_key)
   3810     ):
   3811         raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
   3813 except TypeError:
   3814     # If we have a listlike key, _check_indexing_error will
raise
   3815     #  InvalidIndexError. Otherwise we fall through and re-
```

```
raise
   3816     #  the TypeError.
   3817     self._check_indexing_error(key)

KeyError: 'LSTM_Predicted'
```

```
# === Assign real label to first 126 days based on similarity to
clusters ===
# Compute metrics for first 126 days
initial_returns = returns[:window_size]
initial_return_sum = np.sum(initial_returns)
initial_vol = np.std(initial_returns)

# Compute similarity with each cluster's return/vol stats
similarity = []
for cum_ret, vol in cluster_stats:
    dist = np.sqrt((initial_return_sum - cum_ret)**2 + (initial_vol -
vol)**2)
    similarity.append(dist)

# Pick the closest matching cluster
closest_cluster_idx = np.argmin(similarity)

# Get the actual label of that cluster (Bull/Bear/Sideways)
sorted_clusters = np.argsort([stat[0] for stat in cluster_stats])[::-
1]
regime_ordered_labels = np.array(["Sideways"] * n_clusters)
regime_ordered_labels[sorted_clusters[0]] = "Bull"
regime_ordered_labels[sorted_clusters[-1]] = "Bear"
# The rest remain Sideways
initial_label = regime_ordered_labels[closest_cluster_idx]

# === Now assign full regime labels ===
regime_full = np.array([initial_label] * window_size)  # Assign first
126
regime_full = np.append(regime_full, regime_labels)


data["Regime"] = regime_full
data[["Pre", "Regime"]].to_csv("Results/Regimes.csv")


import pandas as pd

# Load the CSV (date is index)
Regimes_csv = pd.read_csv("Results/Regimes.csv", index_col=0)

# Ensure 'Regime' column exists
if "Regime" not in data.columns:
    raise ValueError("Column 'Regime' not found in ABC.csv")
```

```python
# Replace any unmapped/undefined values with 'Sideways'
Regimes_csv["Regime"] = Regimes_csv["Regime"].fillna("Sideways")
Regimes_csv["Regime"] = Regimes_csv["Regime"].replace("Undefined",
"Sideways")

# Map regime to integer and safely convert to int
regime_mapping = {"Sideways": 0, "Bull": 1, "Bear": -1}
Regimes_csv["RegimeCode"] = Regimes_csv["Regime"].map(regime_mapping)

# Now safely convert to int (no NaNs remain)
Regimes_csv["RegimeCode"] = Regimes_csv["RegimeCode"].astype(int)

# Save to file
Regimes_csv[["Prediction",
"RegimeCode"]].to_csv("Results/Regime_for_Ensemble.csv", index=True)

print("□ Saved Regime_for_Ensemble.csv with integer codes.")
```

```
□ Saved Regime_for_Ensemble.csv with integer codes.
```