Modules

Ovo su python knjižnice potrebne za naše istraživanje. Neke od bitnijih su: pandas: pruža strukturiranje podataka matplotlib: prikaz podataka i grafovi numpy: numeričke operacije nad podatcima sklear: sadrži razne algoritme strojnog učenja

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from collections import defaultdict
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from IPython.display import display
from sklearn.metrics import mean_squared_error
```

Preparing Data

U našem istraživanju koristili smo podatke o cijenama dionica. Odabrali smo oko 1000 dionica koje su danas aktivne te smo uzeli podatke od 2000 do 2021 godine. Podaci nisu reprezentativni za tržište jer smo odabrali dionice koje su danas postojeće, a one su u zadnjih 20 godina sigurno narasle u cijeni. Zbog toga ćemo naše testiranje provesti na zadnjih 7 godina kako bi izbjegli pristrane rezultate. Returns_df je data frame koji prikazuje dnevne povrate za svaku dionicu i njega ćemo najviše koristiti za računanje.

In [ ]:
```
# Load data
prices_df = pd.read_csv('data\Russell3000_prices_clean.csv')
prices_df['Date'] = pd.to_datetime(prices_df['Date'], dayfirst=True)
prices_df.sort_values(by='Date', inplace=True)

# Select stocks to use; exclude the first column (date)
all_stocks = prices_df.columns[1:]
training_stocks = all_stocks[:200]
other_stocks = all_stocks[200:]

# Calculate returns
returns_df = prices_df[all_stocks].pct_change()
returns_df = returns_df.dropna().reset_index(drop=True)
returns_df.insert(0, 'Date', prices_df['Date'][:-1])

# Convert to numpy array (for easier calculations)
returns_matrix = returns_df.iloc[:, 1:].to_numpy()
```

Defining Parameters

U ovom projektu želimo na temelju prošlih podataka doći do zaključaka o budućnosti. Način na koji ćemo mi provesti naše testiranje je taj da ćemo imati prozore koji će gledati prošle podatke o povratima dionica, na temelju tih prošlih podataka ćemo predvidjeti volatilnost budućeg prozora. Računanje volatilnosti nad prošlim prozorom je različito za svaku metodu koju koristimo u našem radu. U ovom radu smo testirali 12 različitih metoda.

Objašnjenje varijabli: windows: veličine prozora koji će gledati prošle povrate, nad tim prošlim povratima će se računati predviđena volatilnost forecast_horizon: veličina budućeg prozora za kojeg predviđamo volatilnost start_date: datum od kojeg testiramo naše metode end_date: datum do kojeg testiramo naše metode.

In [ ]:
```
global windows, foreacast_horizon, test_start, test_end

windows = [21, 63, 126, 252] # month, quarter, half-year, year
forecast_horizon = 21 # trading days
test_start_date = pd.to_datetime('2015-01-01')
test_end_date = pd.to_datetime('2021-07-15')

messages = {"test_start": [], "test_end": []}

# Check if test_start_date exists in the DataFrame, if not, take the next date that
if test_start_date not in returns_df['Date'].values:
    test_start_date = returns_df['Date'][returns_df['Date'] > test_start_date].bfil
    messages["test_start"] = messages["test_start"] or ['Start date not found, usin

# Check if test_end_date exists in the DataFrame, if not, take the previous date th
if test_end_date not in returns_df['Date'].values:
    test_end_date = returns_df['Date'][returns_df['Date'] < test_end_date].ffill().
    messages["test_end"] = messages["test_end"] or ['End date not found, using clos

test_start = returns_df.loc[returns_df['Date'] == test_start_date].index[0]
test_end = returns_df.loc[returns_df['Date'] == test_end_date].index[0]

if test_start < max(windows):
    test_start = max(windows)
    messages["test_start"] = messages["test_start"] or ['Start date too early, not

if test_end > len(returns_df):
    test_end = len(returns_df)
    messages["test_end"] = messages["test_end"] or ['End date too late, not enough

if (test_end - test_start) % forecast_horizon != 0:
    test_end = test_end - (test_end - test_start) % forecast_horizon
    messages["test_end"] = messages["test_end"] or ['End date not a multiple of for

if messages["test_start"]:
    print('Adjusted tart date: {}. Reason: {}'.format(returns_df['Date'][test_start
if messages["test_end"]:
    print('Adjusted end date: {}. Reason: {}'.format(returns_df['Date'][test_end],
```

```
del test_start_date, test_end_date, messages
```

```
Adjusted tart date: 2015-01-02 00:00:00. Reason: Start date not found, using closest
next date instead.
Adjusted end date: 2021-06-29 00:00:00. Reason: End date not a multiple of forecast
horizon.
```

Naive Risk Parity

Nakon što smo predvidjeli buduće volatilnosti za svaku dionicu koristimo metodu naive risk parity kako bi odredili težine dionica i konstruirali portfelj. Naive risk parity je strategija porfelja koja se temelji na ravnoteži i nastoji smanjiti izloženost prema riziku. Težine su dodijeljene dionicama u skladu s njihovom obrnuto proporcionalnom volatilnošću, odnosno dionice s većom volatilnosti imaju manju težinu dok dionice s malim volatilnostima imaju veće težine u porfelju.

Funkcija get_portfolio_weights prima riječnik u kojem se nalaze volatilnosti za svaku dionicu u našem portfelju, a vraća nam polje težina dionica odnosno udjele koliko koje dionice ima u našem porfelju npr(dionica_1 0.2, dionica_2 0.6, dionica_3 0.2).

In [ ]:
```
def get_portfolio_weights(volatility):
    inv_volatilities = np.array([1/volatility[stock] if volatility[stock] > 0 else
    total = np.sum(inv_volatilities)
    return inv_volatilities / total if total > 0 else np.zeros(len(all_stocks))
```

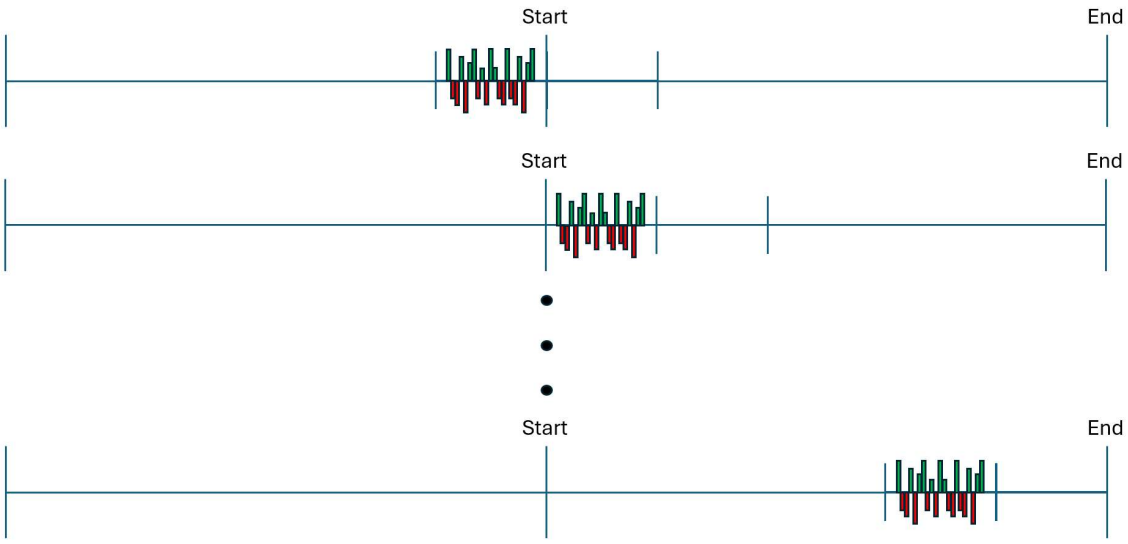
Objašnjenje varijable methods: methods je riječnik koji će za svaku metodu imati: riječnik predviđenih volatilnosti za svaku dionicu, vrijednosti portfelja koji kreira ta metoda

Objašnjenje funkcije gather_portfolio_metrics: input: ime metode koju koristimo + veličina windowa, trenutak u kojem se trenutno nalazimo, riječnik volatility koji sadrži volatilnosti svih dionica u prošlom windowu ovu metodu ćemo koristiti kako bi dodavali riječnike predviđene volatilnosti i vrijednosti porftelja u varijablu methods

Primjer implementacije: -Veličina windowa i forecast_windowa je 21, start je postavljen na 1.1.2011, a end na 1.1.2021, znači računamo performanse metoda u zadnjih 10 godina. -Nalazimo se na trenutku start i nad dataframe-om returns_df gledamo 21 (window) podatak u nazad -nad tim prozorom računamo volatilnost (način računanja različit za svaku dionicu) i pretpostavljamo da je u sljedećih 21 dan (forecast_window) takva volatilnost -To ćemo napraviti za svaku dionicu i paralelno izračunate volatilnosti dodavati u riječnik volatility (sadrži predviđene volatilnosti za dionice). -Nakon izračuna volatilnosti zovemo funkciju gather_portfolio_metrics sa inputom: metodom + 21 (window), start, riječnik volatility koji smo maloprije izračunali. -U toj funkciji ćemo -dodati riječnik volatility u riječnik methods->metoda->predicted volatility, -pozvati funkciju get_portfolio_weights(input: volatility), dobit ćemo weights, odnosno udjele dionica u našem porfelju. -Kada imamo udjele dionica u porfelju možemo vidjeti kako naš portfelj performa u budućem prozoru(forecast_window). -To ćemo napraviti tako da weights skalarno pomnožimo sa matricom stvarnih povrata. To

nam daje polje returns koje u sebi sadrži dnevne povrate našeg porfelja, koliko posto je naš portfelj narastao/pao svaki dan -polje returns dodajemo u methods->metoda->returns Nakon izvršavanja funkcije u riječniku methods za jednu metodu imati predviđene volatilnosti za svaku dionicu i ostvarene povrate portfelja.

Ovaj primjer implementacije predstavlja prvi dio slike ispod. Prije sljedećeg računanja pomičemo se za 21 (forecast_window) te ponavljamo implemetaciju na sljedećem prozoru. Tako ćemo iterirati po svim podatcima između start i end. Nakon te iteracije ćemo za jednu metodu i jedan window imati predviđene volatilnosti i polje returns koje su sadržane u methods[metoda]. To možemo napraviti za različite windowe, i za različite metode.



In [ ]:
```
methods = defaultdict(lambda: defaultdict(list))

def gather_portfolio_metrics(method, i, volatility):
    # Record predicted volatility for each stock
    for stock in all_stocks:
        methods[method][stock + " predicted volatility"].extend([volatility[stock]]

    # Calculate portfolio weights based on volatility
    weights = get_portfolio_weights(volatility)

    # Calculate portfolio returns for each day in the window
    returns = np.dot(returns_matrix[i:i+forecast_horizon], weights)

    # Record portfolio returns
    methods[method]["returns"].extend(returns)
```

In [ ]:
```
def reset_portfolio_metrics(method):
    for stock in all_stocks:
        methods[method][stock + " predicted volatility"] = []
    methods[method]["returns"] = []
```

U nastavku ćemo provesti testiranja metoda. Metode prate gore objašnjenu implementaciju. Svaka metoda ima drukčiji način na koji predviđa buduću volatilnost. To je ostvareno drugačijim inputima i outputima funkcjia s kojima računamo. Nakon predviđanja volatilnosti idu u naive risk parity metodu gdje dobijamo udjele dionica i računamo vrijednosti porftelja. Ukoliko je forecast_window 21 dan to bi značilo da ćemo svaki mjesec ponovo predviđati volatilnosti i računati nove težine u našim porfteljima.

Perfect Prediction Method

Perfect prediction metoda će nam također služiti kao benchmark metoda. Metoda funkcionira suprotno od naše ideje da iz prošlih podataka računamo i predviđamo buduću volatilnost, kod ove metode će buduća volatilnost biti točno ona volatilnost koja će biti u budućem prozoru. Ova metoda služi nam da znamo kolika je minimalna greška kod predviđanja buduće volatilnosti.

In [ ]:
```
method = "PerfectPrediction" + "+" + str(forecast_horizon)
reset_portfolio_metrics(method)

for i in range(test_start, test_end, forecast_horizon):
    # Trying to predict the volatility of the next forecast_horizon days
    volatility = {stock: returns_df[stock][i: i + forecast_horizon].std() for stock

    gather_portfolio_metrics(method, i, volatility)
```

Sample Method

Sample metoda jedna je od jednostavnijih u našem projektu. Funkcionira tako da se na prošlom prozoru izračuna volatilnost pomoću .std() funkcije koja računa standardnu devijaciju i pretpostavi da je buduća jednaka tome. Sample metodu smo testirali na prošlim prozorima od 1, 3, 6 i 12 mjeseci.

In [ ]:
```
for window in windows:
    method = "Sample" + "-" + str(window)
    reset_portfolio_metrics(method)
    for i in range(test_start, test_end, forecast_horizon):
        #
        volatility = {stock: returns_df[stock][i - window: i].std() for stock in al

        gather_portfolio_metrics(method, i, volatility)
```

Equal Weight Method

Equal weight metoda služit će nam kao benchmark. Ova metoda je poznata po tome što je jako trivijalna ali jako teška za outperformanje. Equal weight portfolio je vrsta portfelja gdje su svim dionicama dodijeljene jednake težine. Strategija tog portfelja je prodaja dionica koje su narasle i kupovina dionica koje su pale. Dugoročno ovaj portfelj ostvaruje jako visoke

povrate jer u rastućem tržištu manjim dionicama daje jednak značaj i tako ostvaruje visoke povrate.

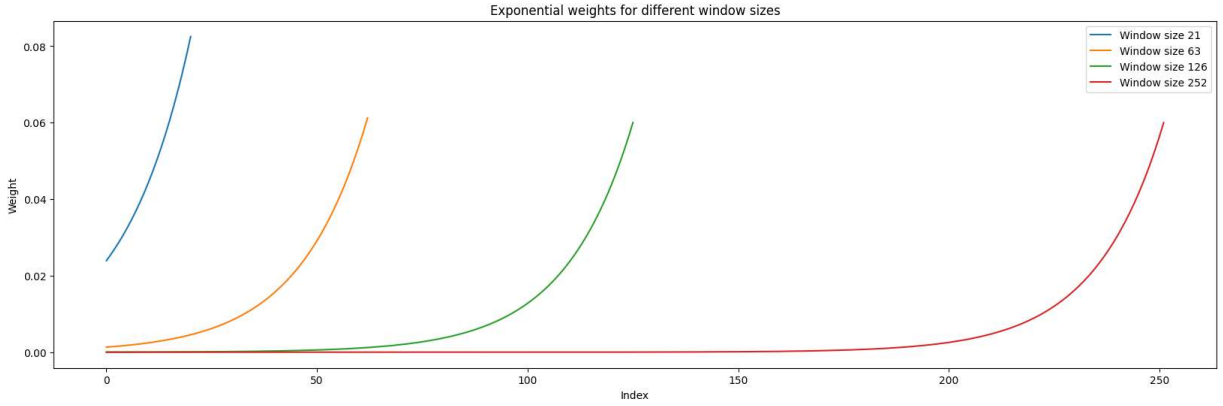
In [ ]:
```
method = "EqualWeight"
reset_portfolio_metrics(method)

for i in range(test_start, test_end, forecast_horizon):
    # We assume equal volatility across all stocks, simplifying our model to equal
    volatility = {stock: 1 for stock in all_stocks}

    gather_portfolio_metrics(method, i, volatility)
```

Exponential Weight Method

Exponential weight metoda je zapravo modificirana sample metoda. Kod sample metode buduću volatilnost računamo tako da svaki dan u prošlom prozoru gledamo jednako bitno, a kod exponential weighted metode dane koji su bliži nama gledamo eksponencijalno više, npr. povrati od jučer i prekjučer su nam bitniji nego povrati prije mjesec dana kad računamo volatilnost. Na donjem grafu možemo vidjeti koliko koji dan uzimamo u obzir, na windowu od 21 dan 21. dan ima oko 8% utjecaja u računanju volatilnosti dok 1. dan ima oko 2%. Ideja exponential weighted metode je ta da kasniji podatci o povratima bolje opisuju buduću volatilnost.

Objašnjavanje funkcija: get_exponential_weights: prima veličinu prozora i vraća težine utjecaja dana u tom prozoru get_weighted_volatility: prima težine utjecaja dana u prozoru i polje povrata, vraća volatilnost tog prozora prilagođenu težinama dana

In [ ]:
```
def get_exponential_weights(size, const=0.94):
    weights = [const**i for i in range(size, 0, -1)]
    total = sum(weights)
    return [weight / total for weight in weights]

plt.figure(figsize=(20, 6))
for window in windows:
    weights = get_exponential_weights(window)
    plt.plot(weights, label=f'Window size {window}')
plt.title('Exponential weights for different window sizes')
plt.xlabel('Index')
plt.ylabel('Weight')
plt.legend()
plt.show()
```



In [ ]:
```
def get_weighted_volatility(weights, returns):
    variance = sum(weights * returns**2)
    return variance**0.5

for window in windows:
    method = "ExponentialWeight" + "-" + str(window)
    reset_portfolio_metrics(method)

    weights = get_exponential_weights(window)
    for i in range(test_start, test_end, forecast_horizon):
        #
        volatility = {stock: get_weighted_volatility(weights, returns_df[stock][i -

        gather_portfolio_metrics(method, i, volatility)
```

Least Squares Linear Regression Method

Least Squares Linear Regression je metoda koja se temelji na minimizaciji sume kvadrata predviđene vrijednosti i prave vrijednosti, u našem slučaju predviđamo volatilnost. Cilj je optimalno prilagoditi model tako da imamo što bolje rezultate. Sljedeće 4 metode se sve zasnivaju na linearnoj regresiji, razlikuju se po ulaznim podacima i načinu prilagodbe.

Train_start je trenutak kada krećemo sa treniranjem naših modela, u ovo slučaju je to najraniji podatak koji imamo. Train_end je trenutak do kojeg treniramo, to je jednako trenutku kada krećemo testiranje, test_startu.ž

Funkcija get_linear_regression_model prima podatke A i B, A predstavlja ulazne podatke na temelju kojih treba predviditi B. Model treniramo sa ugrađenom funkcijom LinearReggression. 4 metode linearne regresije će imati različite A i B te će svoje modele istrenirati na ovoj funkciji i kasnije te modele koristiti u predviđanju buduće volatilnosti.

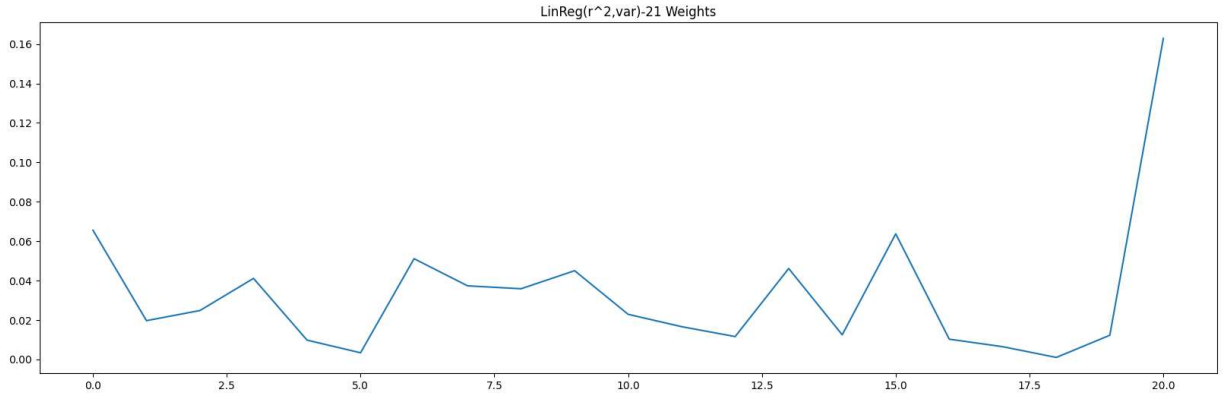
In [ ]:
```
global window, train_start, train_end
window = 21 # Set window to 21 days from now on for computational efficiency
train_start = (test_start - window) % forecast_horizon + window #
train_end = test_start # Training end is the start of the test period

A, B = [], []
def get_linear_regression_model(A, B, fit_intercept=True):
    # Fit a linear regression model with non-negative coefficients and no intercept
```

```
    model = LinearRegression(positive=True, fit_intercept=fit_intercept)
    model.fit(np.array(A), np.array(B))
    A.clear()
    B.clear()

    return model
```

Linear Regression (return^2, variance)

Sljedeća metoda je linearna regresija koja za ulazne podatke uzima kvadrirane dnevne povrate, a predviđa varijancu budućeg prozora.

In [ ]:
```
method = "LinReg(r^2,var)" + "-" + str(window)
reset_portfolio_metrics(method)

# Iterate over each window of returns
for stock in training_stocks:
    for i in range(train_start, train_end, forecast_horizon):
        # Append squared returns and variance of returns to A and B respectively
        A.append(returns_df[stock][i - window:i] ** 2)
        B.append(returns_df[stock][i:i + forecast_horizon].var())

methods[method]["model"] = get_linear_regression_model(A, B, False) # Intercept mig

plt.figure(figsize=(20, 6))
plt.plot(methods[method]["model"].coef_)
plt.title(method + " Weights")
plt.show()
```

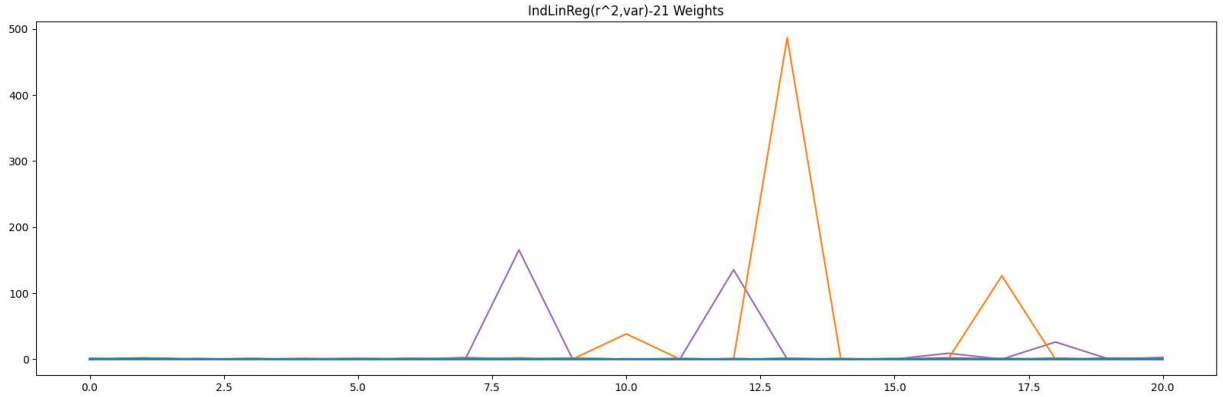


In [ ]:
```
for i in range(test_start, test_end, forecast_horizon):
    #
    volatility = {stock: methods[method]["model"].predict(returns_df[stock][i - win

    gather_portfolio_metrics(method, i, volatility)
```

Individual Linear Regression (return^2, variance)

Sljedeća metoda je individualna linearna regresija koja za ulazne podatke uzima kvadrirane dnevne povrate, a predviđa varijancu budućeg prozora. Razlika kod ove metode od prošle je što se ovdje za svaku dionicu posebno računaju optimalne težine prošlog prozora za predviđanje volatilnosti.

In [ ]:
```
method = "IndLinReg(r^2,var)" + "-" + str(window)
reset_portfolio_metrics(method)

# Store individual models for each stock
for stock in all_stocks:
    for i in range(train_start, train_end, forecast_horizon):
        # Append squared returns and variance of returns to A and B respectively
        A.append(returns_df[stock][i - window:i] ** 2)
        B.append(returns_df[stock][i:i + forecast_horizon].var())

    methods[method][stock + "model"] = get_linear_regression_model(A, B, False)

plt.figure(figsize=(20, 6))
for stock in all_stocks:
    plt.plot(methods[method][stock + "model"].coef_)
plt.title(method + " Weights")
plt.show()
```



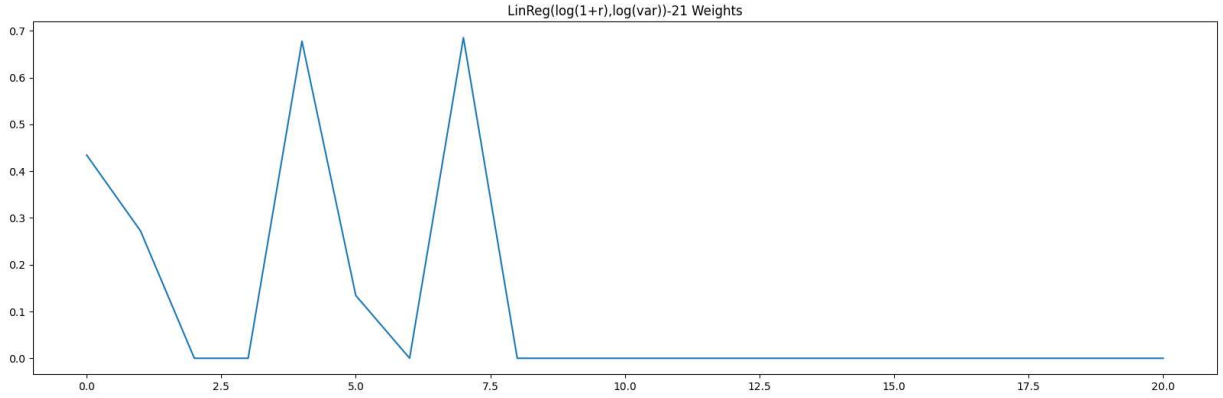
In [ ]:
```
for i in range(test_start, test_end, forecast_horizon):
    #
    volatility = {stock: methods[method][stock + "model"].predict(returns_df[stock]

    gather_portfolio_metrics(method, i, volatility)
```

Linear Regression (log(1+return), log(variance))

Sljedeća metoda je linearna regresija koja za ulazne podatke uzima logaritam od (1 + dnevni povrati), a predviđa logaritam varijance budućeg prozora.

In [ ]:
```
method = "LinReg(log(1+r),log(var))" + "-" + str(window)
reset_portfolio_metrics(method)

# Iterate over each window of returns
for stock in training_stocks:
    for i in range(train_start, train_end, forecast_horizon):
        variance = returns_df[stock][i:i + forecast_horizon].var()
        if variance > 0: # If variance is 0, we cannot take the log
            A.append(np.log(1 + returns_df[stock][i - window:i]))
            B.append(np.log(variance))

methods[method]["model"] = get_linear_regression_model(A, B)
```

```
plt.figure(figsize=(20, 6))
plt.plot(methods[method]["model"].coef_)
plt.title(method + " Weights")
plt.show()
```

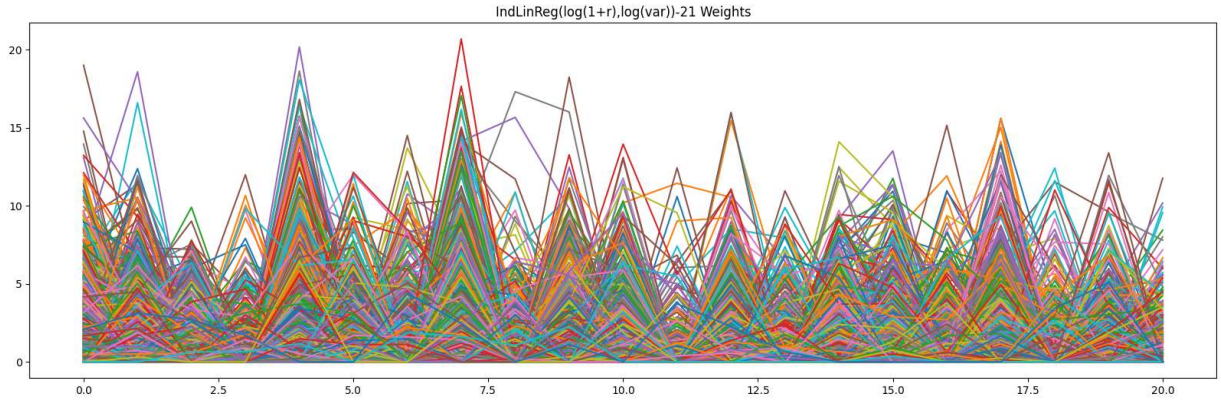


In [ ]:
```
for i in range(test_start, test_end, forecast_horizon):
    #
    volatility = {stock: np.exp(methods[method]["model"].predict(np.log(1+returns_d

    gather_portfolio_metrics(method, i, volatility)
```

Individual Linear Regression (log(1+return), log(variance))

Sljedeća metoda je linearna regresija koja za ulazne podatke uzima logaritam od (1 + dnevni povrati), a predviđa logaritam varijance budućeg prozora. Isto kao i u prethodnom paru linearnih regresija, ova metoda nauči težine prošlog prozora za svaku dionicu posebno. Na grafu ispod možemo vidjeti prikazane težine svakog dana prošlog prozora.

In [ ]:
```
method = "IndLinReg(log(1+r),log(var))" + "-" + str(window)
reset_portfolio_metrics(method)

# Iterate over each window of returns
for stock in all_stocks:
    for i in range(train_start, train_end, forecast_horizon):
        variance = returns_df[stock][i:i + forecast_horizon].var()
        if variance > 0: # If variance is 0, we cannot take the log
            A.append(np.log(1 + returns_df[stock][i - window:i]))
            B.append(np.log(variance))

    methods[method][stock + "model"] = get_linear_regression_model(A, B)

plt.figure(figsize=(20, 6))
for stock in all_stocks:
    plt.plot(methods[method][stock + "model"].coef_)
plt.title(method + " Weights")
plt.show()
```



In [ ]:
```
for i in range(test_start, test_end, forecast_horizon):
    #
    volatility = {stock: np.exp(methods[method][stock + "model"].predict(np.log(1+r

    gather_portfolio_metrics(method, i, volatility)
```

Random Forest Method

Random forest metoda je ensemble metoda koja se sastoji od više stabala odluke, svako stablo se trenira na posebnom subsetu naših podataka, to se zove bagging. Recimo da naš random forest ima 100 stabala i glavnu odluku donosi prosječnom vrijednosti svih stabala. Tako izbjegavamo pristranost u našim rezultatima. Također radi na principu da ga treniramo sa prošlim prozorima i volatilnosti budućih prozora kako bi naučio predvidviđati. U nastavku iammo 4 metode random foresta koje se razlikuju u obliku ulaznih podataka i podataka koje treba predvidjeti.

In [ ]:
```
def get_random_forest_model(A, B):
    # Fit a random forest
    model = RandomForestRegressor()
    model.fit(np.array(A), np.array(B))
    A.clear()
    B.clear()

    return model
```

Random Forest (return, log(variance))

Sljedeća metoda random foresta za ulazne podatke uzima prošli prozor povrata i logaritmiranu buduću varijancu.

In [ ]:
```
method = "RandFor(r,log(var))" + "-" + str(window)
reset_portfolio_metrics(method)

# Iterate over each window of returns
for stock in training_stocks:
    for i in range(train_start, train_end, forecast_horizon):
        variance = returns_df[stock][i:i + forecast_horizon].var()
        if variance > 0: # If variance is 0, we cannot take the log
            A.append(returns_df[stock][i - window:i])
```

```
            B.append(np.log(variance))

methods[method]["model"] = get_random_forest_model(A, B)

for i in range(test_start, test_end, forecast_horizon):
    #
    volatility = {stock: np.exp(methods[method]["model"].predict(returns_df[stock][

    gather_portfolio_metrics(method, i, volatility)
```

Random Forest (return^2, log(variance))

Sljedeća metoda random foresta za ulazne podatke uzima kvadrate povrata u prošlom prozoru i logaritmiranu buduću varijancu.

In [ ]:
```
method = "RandFor(r^2,log(var))" + "-" + str(window)
reset_portfolio_metrics(method)

# Iterate over each window of returns
for stock in training_stocks:
    for i in range(train_start, train_end, forecast_horizon):
        variance = returns_df[stock][i:i + forecast_horizon].var()
        if variance > 0: # If variance is 0, we cannot take the log
            A.append(returns_df[stock][i - window:i]**2)
            B.append(np.log(variance))

methods[method]["model"] = get_random_forest_model(A, B)

for i in range(test_start, test_end, forecast_horizon):
    #
    volatility = {stock: np.exp(methods[method]["model"].predict(returns_df[stock][

    gather_portfolio_metrics(method, i, volatility)
```

Random Forest (|return|, log(variance))

Sljedeća metoda random foresta za ulazne podatke uzima apsolutne povrate u prošlom prozoru i logaritmiranu buduću varijancu.

In [ ]:
```
method = "RandFor(|r|,log(var))" + "-" + str(window)
reset_portfolio_metrics(method)

# Iterate over each window of returns
for stock in training_stocks:
    for i in range(train_start, train_end, forecast_horizon):
        variance = returns_df[stock][i:i + forecast_horizon].var()
        if variance > 0: # If variance is 0, we cannot take the log
            A.append(abs(returns_df[stock][i - window:i]))
            B.append(np.log(variance))

methods[method]["model"] = get_random_forest_model(A, B)

for i in range(test_start, test_end, forecast_horizon):
```

```
    #
    volatility = {stock: np.exp(methods[method]["model"].predict(abs(returns_df[sto

    gather_portfolio_metrics(method, i, volatility)
```

Random Forest (log(1+return), log(variance))

Sljedeća metoda random foresta za ulazne podatke uzima prošli prozor logaritmiranih (povrati + 1) i logaritmiranu buduću varijancu.

In [ ]:
```
method = "RandFor(log(1+r),log(var))" + "-" + str(window)
reset_portfolio_metrics(method)

# Iterate over each window of returns
for stock in training_stocks:
    for i in range(train_start, train_end, forecast_horizon):
        variance = returns_df[stock][i:i + forecast_horizon].var()
        if variance > 0: # If variance is 0, we cannot take the log
            A.append(np.log(1 + returns_df[stock][i - window:i]))
            B.append(np.log(variance))

methods[method]["model"] = get_random_forest_model(A, B)

for i in range(test_start, test_end, forecast_horizon):
    #
    volatility = {stock: np.exp(methods[method]["model"].predict(np.log(1 + returns

    gather_portfolio_metrics(method, i, volatility)
```

Portfolios' Performances

U nastavku kreiramo za svaku metodu njezin portfolio i računamo standardnu devijaciju, prosječni godišnji povrat te sharpe ratio.

In [ ]:
```
portfolios_df = {
    "Portfolio": [method for method in methods],
    "Std": [np.std(portfolio["returns"]) * 252**0.5 for portfolio in methods.values
    "Return": [(1 + np.average(portfolio["returns"]))**252 - 1 for portfolio in met
    "Sharpe": [((1 + np.average(portfolio["returns"]))**252 - 1) / (np.std(portfoli
}

display(pd.DataFrame(portfolios_df).sort_values('Sharpe', ascending=False).reset_in
```

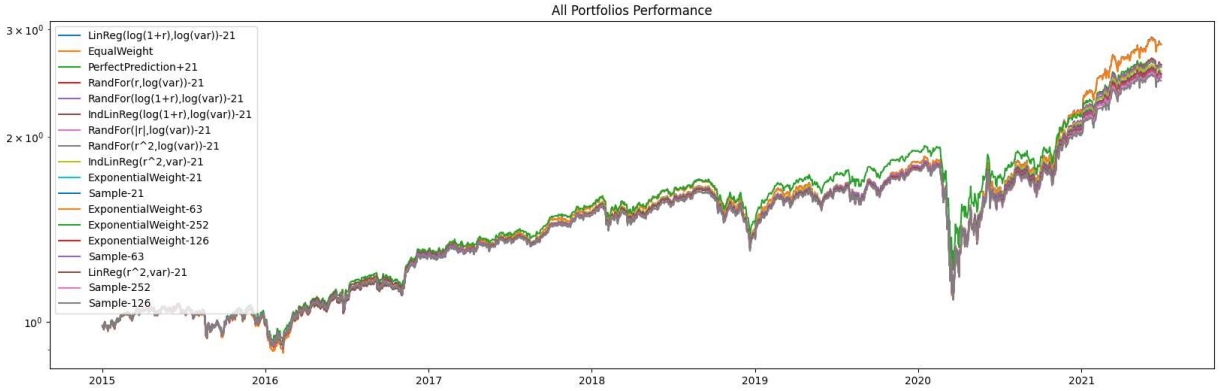
	Portfolio	Std	Return	Sharpe
0	PerfectPrediction+21	0.188517	0.178922	0.949102
1	LinReg(log(1+r),log(var))-21	0.215529	0.199336	0.924869
2	EqualWeight	0.215308	0.198808	0.923362
3	RandFor(r,log(var))-21	0.203240	0.182165	0.896307
4	RandFor(log(1+r),log(var))-21	0.203385	0.181995	0.894829
5	IndLinReg(r^2,var)-21	0.197610	0.176687	0.894121
6	RandFor(|r|,log(var))-21	0.202194	0.180509	0.892755
7	RandFor(r^2,log(var))-21	0.202161	0.180306	0.891892
8	ExponentialWeight-21	0.197818	0.175251	0.885921
9	Sample-21	0.198320	0.175103	0.882935
10	ExponentialWeight-63	0.198568	0.174741	0.880007
11	ExponentialWeight-252	0.198626	0.174586	0.878970
12	ExponentialWeight-126	0.198625	0.174579	0.878938
13	IndLinReg(log(1+r),log(var))-21	0.209673	0.183437	0.874870
14	LinReg(r^2,var)-21	0.198377	0.172419	0.869146
15	Sample-63	0.199813	0.173604	0.868832
16	Sample-252	0.201409	0.172843	0.858170
17	Sample-126	0.200192	0.170751	0.852939

	Portfolio	Std	Return	Sharpe
0	PerfectPrediction+21	0.188517	0.178922	0.949102
1	IndLinReg(r^2,var)-21	0.197610	0.176687	0.894121
2	ExponentialWeight-21	0.197818	0.175251	0.885921
3	Sample-21	0.198320	0.175103	0.882935
4	LinReg(r^2,var)-21	0.198377	0.172419	0.869146
5	ExponentialWeight-63	0.198568	0.174741	0.880007
6	ExponentialWeight-126	0.198625	0.174579	0.878938
7	ExponentialWeight-252	0.198626	0.174586	0.878970
8	Sample-63	0.199813	0.173604	0.868832
9	Sample-126	0.200192	0.170751	0.852939
10	Sample-252	0.201409	0.172843	0.858170
11	RandFor(r^2,log(var))-21	0.202161	0.180306	0.891892
12	RandFor(|r|,log(var))-21	0.202194	0.180509	0.892755
13	RandFor(r,log(var))-21	0.203240	0.182165	0.896307
14	RandFor(log(1+r),log(var))-21	0.203385	0.181995	0.894829
15	IndLinReg(log(1+r),log(var))-21	0.209673	0.183437	0.874870
16	EqualWeight	0.215308	0.198808	0.923362
17	LinReg(log(1+r),log(var))-21	0.215529	0.199336	0.924869

Portfolio Value (initial investment: 1$)

In [ ]:
```
# Iterate over each portfolio in methods
for portfolio in methods.values():
    # Initialize the value list with a starting value of 1
    portfolio["value"] = [1]

    # Calculate the cumulative value of the portfolio
    for ret in portfolio["returns"]:
        portfolio["value"].append((ret + 1) * portfolio["value"][-1])

# Get the dates for the test period
dates = returns_df.iloc[test_start:test_end, 0]

# Sort the methods by the final value of the portfolio, in descending order
sorted_methods = sorted(methods.items(), key=lambda method: method[1]["value"][-1],

# Plot the performance of all portfolios
plt.figure(figsize=(20, 6))
for method, portfolio in sorted_methods:
```

```
    plt.semilogy(dates, portfolio["value"][1:], label=method)
plt.title("All Portfolios Performance")
plt.legend()
plt.show()
```



Mean Squared Error

In [ ]:
```
def calculate_mse(actual, predicted, stocks=all_stocks):
    mse = []
    for stock in stocks:
        mse.append(mean_squared_error(actual[stock + " predicted volatility"], pred
    return np.mean(mse)
```

Methods' Performace

In [ ]:
```
methods_df = {
    "Method": [method for method in methods if method != "EqualWeight"],
    "MSE All Stocks": [calculate_mse(methods["PerfectPrediction+" + str(forecast_ho
    "MSE Trained Stocks": [calculate_mse(methods["PerfectPrediction+" + str(forecas
    "MSE Other Stocks": [calculate_mse(methods["PerfectPrediction+" + str(forecast_
}

display(pd.DataFrame(methods_df).sort_values('MSE All Stocks', ascending=True).rese
```

	Method	MSE All Stocks	MSE Trained Stocks	MSE Other Stocks
0	PerfectPrediction+21	0.000000	0.000000	0.000000
1	RandFor(r^2,log(var))-21	0.000286	0.000086	0.000328
2	RandFor(|r|,log(var))-21	0.000287	0.000086	0.000328
3	RandFor(r,log(var))-21	0.000298	0.000089	0.000341
4	RandFor(log(1+r),log(var))-21	0.000298	0.000089	0.000342
5	ExponentialWeight-252	0.000302	0.000116	0.000340
6	ExponentialWeight-126	0.000302	0.000116	0.000340
7	ExponentialWeight-63	0.000303	0.000117	0.000342
8	LinReg(r^2,var)-21	0.000319	0.000123	0.000359
9	ExponentialWeight-21	0.000323	0.000122	0.000365
10	Sample-252	0.000323	0.000134	0.000362
11	Sample-63	0.000324	0.000138	0.000363
12	Sample-126	0.000327	0.000139	0.000366
13	Sample-21	0.000347	0.000138	0.000390
14	IndLinReg(log(1+r),log(var))-21	0.000362	0.000129	0.000411
15	LinReg(log(1+r),log(var))-21	0.000400	0.000126	0.000456
16	IndLinReg(r^2,var)-21	0.003277	0.000133	0.003931

In [ ]:
```
display(pd.DataFrame(methods_df).sort_values('MSE Trained Stocks', ascending=True).
display(pd.DataFrame(methods_df).sort_values('MSE Other Stocks', ascending=True).re
```

	Method	MSE All Stocks	MSE Trained Stocks	MSE Other Stocks
0	PerfectPrediction+21	0.000000	0.000000	0.000000
1	RandFor(r^2,log(var))-21	0.000286	0.000086	0.000328
2	RandFor(|r|,log(var))-21	0.000287	0.000086	0.000328
3	RandFor(r,log(var))-21	0.000298	0.000089	0.000341
4	RandFor(log(1+r),log(var))-21	0.000298	0.000089	0.000342
5	ExponentialWeight-252	0.000302	0.000116	0.000340
6	ExponentialWeight-126	0.000302	0.000116	0.000340
7	ExponentialWeight-63	0.000303	0.000117	0.000342
8	ExponentialWeight-21	0.000323	0.000122	0.000365
9	LinReg(r^2,var)-21	0.000319	0.000123	0.000359
10	LinReg(log(1+r),log(var))-21	0.000400	0.000126	0.000456
11	IndLinReg(log(1+r),log(var))-21	0.000362	0.000129	0.000411
12	IndLinReg(r^2,var)-21	0.003277	0.000133	0.003931
13	Sample-252	0.000323	0.000134	0.000362
14	Sample-63	0.000324	0.000138	0.000363
15	Sample-21	0.000347	0.000138	0.000390
16	Sample-126	0.000327	0.000139	0.000366

	Method	MSE All Stocks	MSE Trained Stocks	MSE Other Stocks
0	PerfectPrediction+21	0.000000	0.000000	0.000000
1	RandFor(r^2,log(var))-21	0.000286	0.000086	0.000328
2	RandFor(|r|,log(var))-21	0.000287	0.000086	0.000328
3	ExponentialWeight-252	0.000302	0.000116	0.000340
4	ExponentialWeight-126	0.000302	0.000116	0.000340
5	RandFor(r,log(var))-21	0.000298	0.000089	0.000341
6	RandFor(log(1+r),log(var))-21	0.000298	0.000089	0.000342
7	ExponentialWeight-63	0.000303	0.000117	0.000342
8	LinReg(r^2,var)-21	0.000319	0.000123	0.000359
9	Sample-252	0.000323	0.000134	0.000362
10	Sample-63	0.000324	0.000138	0.000363
11	ExponentialWeight-21	0.000323	0.000122	0.000365
12	Sample-126	0.000327	0.000139	0.000366
13	Sample-21	0.000347	0.000138	0.000390
14	IndLinReg(log(1+r),log(var))-21	0.000362	0.000129	0.000411
15	LinReg(log(1+r),log(var))-21	0.000400	0.000126	0.000456
16	IndLinReg(r^2,var)-21	0.003277	0.000133	0.003931

In [ ]:
```
# del window, i, volatility, method, train_start, train_end, test_start, test_end,
```