Financial_Data_Module_5_Lesson_2

April 7, 2023

0.1 FINANCIAL DATA

MODULE 5 | LESSON 2

---

1 THE RISK-ADJUSTED DISCOUNT RATE

Reading Time	30 minutes
Prior Knowledge	CAPM (Capital Asset Pricing Model), Probability of Default, Recovery Rate, Variance, Standard deviation, Correlation
Keywords	Market Risk Premium, Expected Risk Premium, Risk-free rate, DataReader

---

In the last lesson, we went into detail about the calculation of market-implied probability of default. We also took the first steps in the multi-step process of gathering the inputs to this calculation. We downloaded data from a bond database, cleaned the data, and transformed the "maturity date" into "time to maturity." Then, we used a mask to filter the bonds to fit our desired parameters.

In this lesson, we will estimate the risk-adjusted discount rate. This is the discount rate that sets the bond's expected cash flows equal to the price of the bond. In order to do this, we need to estimate the risk-free interest rate, the expected risk premium for the bond, the market risk premium, and the beta of the bond.

Note: The code that was introduced in the previous lesson is below, followed by the new code and text for this lesson.

[1]:
```
import time

import numpy as np
import pandas as pd
import yfinance as yfin

yfin.pdr_override()

from datetime import date
```

```
from datetime import datetime as dt
from datetime import timedelta

from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select, WebDriverWait
from webdriver_manager.chrome import ChromeDriverManager
```

[2]:
```
# Required
company_ticker = "HES"  # or try: 'F', 'KHC', 'DVN'

# Optional
company_name = "Hess"  # or try: 'Ford Motor', 'Kraft Heinz Co', 'Devon Energy'

# Optional Input Choices:
# ALL, Annual, Anytime, Bi-Monthly, Monthly, N/A, None,
# Pays At Maturity, Quarterly, Semi-Annual, Variable
coupon_frequency = "Semi-Annual"
```

[3]:
```
# Selenium script
options = Options()
options.add_argument("--headless")
options.add_argument("--no-sandbox")
options.add_argument("--disable-dev-shm-usage")
driver = webdriver.Chrome(
    service=Service(ChromeDriverManager().install()), options=options
)

# store starting time
begin = time.time()

# FINRA's TRACE Bond Center
driver.get("http://finra-markets.morningstar.com/BondCenter/Results.jsp")

# click agree
WebDriverWait(driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, ".button_blue.agree"))
).click()

# click edit search
WebDriverWait(driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, "a.qs-ui-btn.blue"))
).click()
```

```
# input Issuer Name
WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.CSS_SELECTOR,␣
 ↪"input[id=firscreener-issuer]"))
)
inputElement = driver.find_element_by_id("firscreener-issuer")
inputElement.send_keys(company_name)

# input Symbol
WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.CSS_SELECTOR,␣
 ↪"input[id=firscreener-cusip]"))
)
inputElement = driver.find_element_by_id("firscreener-cusip")
inputElement.send_keys(company_ticker)

# click advanced search
WebDriverWait(driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, "a.ms-display-switcher.hide"))
).click()

# input Coupon Frequency
WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.CSS_SELECTOR,␣
 ↪"select[name=interestFrequency]"))
)
Select(
    (driver.find_elements_by_css_selector("select[name=interestFrequency]"))[0]
).select_by_visible_text(coupon_frequency)

# click show results
WebDriverWait(driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, "input.
 ↪button_blue[type=submit]"))
).click()

# wait for results
WebDriverWait(driver, 10).until(
    EC.presence_of_element_located(
        (By.CSS_SELECTOR, ".rtq-grid-row.rtq-grid-rzrow .rtq-grid-cell-ctn")
    )
)

# create DataFrame from scrape
frames = []
for page in range(1, 11):
    bonds = []
```

```
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located(
            (By.CSS_SELECTOR, (f"a.qs-pageutil-btn[value='{str(page)}']"))
        )
    )  # wait for page marker to be on expected page
    time.sleep(2)

    headers = [
        title.text
        for title in driver.find_elements_by_css_selector(
            ".rtq-grid-row.rtq-grid-rzrow .rtq-grid-cell-ctn"
        )[1:]
    ]

    tablerows = driver.find_elements_by_css_selector(
        "div.rtq-grid-bd > div.rtq-grid-row"
    )
    for tablerow in tablerows:
        tablerowdata = tablerow.find_elements_by_css_selector("div.
 ↪rtq-grid-cell")
        bond = [item.text for item in tablerowdata[1:]]
        bonds.append(bond)

        # Convert to DataFrame
        df = pd.DataFrame(bonds, columns=headers)

    frames.append(df)

    try:
        driver.find_element_by_css_selector("a.qs-pageutil-next").click()
    except:  # noqa E722
        break

bond_prices_df = pd.concat(frames)

# store end time
end = time.time()

# total time taken
print(f"Total runtime of the program is {end - begin} seconds")

bond_prices_df
```

```
/tmp/ipykernel_502/2651632054.py:30: DeprecationWarning: find_element_by_*
commands are deprecated. Please use find_element() instead
  inputElement = driver.find_element_by_id("firscreener-issuer")
/tmp/ipykernel_502/2651632054.py:37: DeprecationWarning: find_element_by_*
```

```
commands are deprecated. Please use find_element() instead
  inputElement = driver.find_element_by_id("firscreener-cusip")
/tmp/ipykernel_502/2651632054.py:50: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  (driver.find_elements_by_css_selector("select[name=interestFrequency]"))[0]
/tmp/ipykernel_502/2651632054.py:78: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  for title in driver.find_elements_by_css_selector(
/tmp/ipykernel_502/2651632054.py:83: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  tablerows = driver.find_elements_by_css_selector(
/tmp/ipykernel_502/2651632054.py:87: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  tablerowdata = tablerow.find_elements_by_css_selector("div.rtq-grid-cell")
/tmp/ipykernel_502/2651632054.py:87: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  tablerowdata = tablerow.find_elements_by_css_selector("div.rtq-grid-cell")
/tmp/ipykernel_502/2651632054.py:87: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  tablerowdata = tablerow.find_elements_by_css_selector("div.rtq-grid-cell")
/tmp/ipykernel_502/2651632054.py:87: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  tablerowdata = tablerow.find_elements_by_css_selector("div.rtq-grid-cell")
/tmp/ipykernel_502/2651632054.py:87: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  tablerowdata = tablerow.find_elements_by_css_selector("div.rtq-grid-cell")
/tmp/ipykernel_502/2651632054.py:87: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  tablerowdata = tablerow.find_elements_by_css_selector("div.rtq-grid-cell")
/tmp/ipykernel_502/2651632054.py:87: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  tablerowdata = tablerow.find_elements_by_css_selector("div.rtq-grid-cell")
/tmp/ipykernel_502/2651632054.py:87: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  tablerowdata = tablerow.find_elements_by_css_selector("div.rtq-grid-cell")
/tmp/ipykernel_502/2651632054.py:87: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
```

```
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  tablerowdata = tablerow.find_elements_by_css_selector("div.rtq-grid-cell")
/tmp/ipykernel_502/2651632054.py:87: DeprecationWarning:
find_elements_by_css_selector is deprecated. Please use
find_elements(by=By.CSS_SELECTOR, value=css_selector) instead
  tablerowdata = tablerow.find_elements_by_css_selector("div.rtq-grid-cell")

Total runtime of the program is 15.431337118148804 seconds

/tmp/ipykernel_502/2651632054.py:97: DeprecationWarning:
find_element_by_css_selector is deprecated. Please use
find_element(by=By.CSS_SELECTOR, value=css_selector) instead
  driver.find_element_by_css_selector("a.qs-pageutil-next").click()
```

[3]:

	Issuer Name	Symbol	Callable	Sub-Product Type	Coupon	Maturity	Moody's®	S&P	Price	Yield
0	HESS CORP	HES.GH	Yes	Corporate Bond	6.000	01/15/2040	Baa3	BBB-	103.168	5.703
1	HESS CORP	HES.GI	Yes	Corporate Bond	5.600	02/15/2041	Baa3	BBB-	98.762	5.711
2	HESS CORP	HES4136877	Yes	Corporate Bond	3.500	07/15/2024	Baa3	BBB-	98.100	5.070
3	HESS CORP	HES4405829	Yes	Corporate Bond	4.300	04/01/2027	Baa3	BBB-	99.055	4.562
4	HESS CORP	HES4405830	Yes	Corporate Bond	5.800	04/01/2047	Baa3	BBB-	100.271	5.779
5	HESS MIDSTREAM OPERATIONS LP	HES5392919	Yes	Corporate Bond	5.500	10/15/2030	Ba2	BB+	95.088	6.332
6	HESS MIDSTREAM OPERATIONS LP	HES4567499	Yes	Corporate Bond	5.625	02/15/2026	WR	BB+	104.500	4.431
7	HESS MIDSTREAM OPERATIONS LP	HES4927355	Yes	Corporate Bond	5.625	02/15/2026		BB+	98.250	6.303
8	HESS MIDSTREAM OPERATIONS LP	HES5233164	Yes	Corporate Bond	4.250	02/15/2030	Ba2	BB+	90.015	6.052
9	HESS MIDSTREAM PARTNERS LP	HES4918686	Yes	Corporate Bond	5.125	06/15/2028	Ba2	BB+	96.143	6.001

[4]:
```python
def bond_dataframe_filter(df):
    # Drop bonds with missing yields and missing credit ratings
    df["Yield"].replace("", np.nan, inplace=True)
    df["Moody's®"].replace({"WR": np.nan, "": np.nan}, inplace=True)
    df["S&P"].replace({"NR": np.nan, "": np.nan}, inplace=True)
    df = df.dropna(subset=["Yield"])
    df = df.dropna(subset=["Moody's®"])
    df = df.dropna(subset=["S&P"])
```

```
    # Create Maturity Years column that aligns with Semi-Annual Payments from␣
↪corporate bonds
    df["Yield"] = df["Yield"].astype(float)
    df["Coupon"] = df["Coupon"].astype(float)
    df["Price"] = df["Price"].astype(float)
    now = dt.strptime(date.today().strftime("%m/%d/%Y"), "%m/%d/%Y")
    df["Maturity"] = pd.to_datetime(df["Maturity"]).dt.strftime("%m/%d/%Y")
    daystillmaturity = []
    yearstillmaturity = []
    for maturity in df["Maturity"]:
        daystillmaturity.append((dt.strptime(maturity, "%m/%d/%Y") - now).days)
        yearstillmaturity.append((dt.strptime(maturity, "%m/%d/%Y") - now).days␣
↪/ 360)
    df = df.reset_index(drop=True)
    df["Maturity"] = pd.Series(daystillmaturity)
    df["Maturity Years"] = (
        round(pd.Series(yearstillmaturity) / 0.5) * 0.5
    )  # Better for Semi-Annual Payments

    # Target bonds with short-term maturities
    df["Maturity"] = df["Maturity"].astype(float)
    years_mask = (df["Maturity Years"] > 0) & (df["Maturity Years"] <= 5)
    df = df.loc[years_mask]
    return df
```

[5]:
```
bond_df_result = bond_dataframe_filter(bond_prices_df)
bond_df_result
```

[5]:

	Issuer Name	Symbol	Callable	Sub-Product Type	Coupon	Maturity	Moody's®	S&P	Price	Yield	Maturity Years
2	HESS CORP	HES4136877	Yes	Corporate Bond	3.5	465.0	Baa3	BBB-	98.100	5.070	1.5
3	HESS CORP	HES4405829	Yes	Corporate Bond	4.3	1455.0	Baa3	BBB-	99.055	4.562	4.0

1.1 1. Discounting Expected Cashflows

To calculate the probability of default using current corporate bond prices, we will use bond valuation techniques. The valuation of corporate bonds is similar to that of any risky asset; it is dependent on the present value of future expected cash flows, discounted at a risk-adjusted rate (similar to Discounted Cash Flow analysis).

$$\text{Bond price} = \frac{\text{ECF}_1}{1+d} + \frac{\text{ECF}_2}{(1+d)^2} + \frac{\text{ECF}_3}{(1+d)^3}$$

ECF = Expected Cash Flow d = Discount Rate

Corporate bond valuation also accounts for the probability of the bond defaulting and not paying back the principal in full.

We now need to estimate the expected cash flows and the risk-adjusted discount rate.

1.2 2. Estimating Expected Cash Flows

The first step in valuing the bond is to find the expected cash flow at each period. This is done by adding the product of the default payout and the probability of default (p) with the product of the promised payment (coupon payments and repayment of principal) and the probability of not defaulting ($1-p$), which is also referred to as the probability of survival.

$\text{ECF}_1 = p$ (Default Payout) $+ (1-p)$ (Coupon Payment) $\text{ECF}_2 = (1-p)$ (p (Default Payout) $+$ $(1-p)$ (Coupon Payment)) $\text{ECF}_3 = (1-p)^2$ (p (Default Payout) $+ (1-p)$ (Coupon Payment $+$ Principal))

p = Probability of Default Default Payout = Principal $\times$ Recovery Rate

If the bond defaults, the default payout is the product of recovery rate and the principal. In the following example, the principal will be at par value for the bond (e.g. \$100). The recovery rate is the percentage of the loss recovered from a bond in default. The recovery rate varies by industry, the degree of seniority in the capital structure, the amount of leverage in the capital structure in total, and whether a particular security is secured or otherwise collateralized. We assume a 40% recovery rate for the corporate bonds in the following example, which is a common baseline assumption in practice.

The code in the follow-up Lesson 3 will show how the expected cash flow is calculated at each period. We then use the `solve()` function from the Python library `SymPy` to calculate the probability of default that will equate future expected cash flows with the current price of the corporate bond when discounted at the risk-adjusted rate.

1.3 3. The Market Risk Premium and the Expected Risk Premium

After the expected cash flows are calculated, they are discounted back to period 0 at a risk-adjusted discount rate (d) to calculate the bond's price. A risk-adjusted discount rate is the rate obtained by combining an expected risk premium with the risk-free rate during the calculation of the present value of a risky investment.

Risk-adjusted Discount Rate = Risk-free Interest Rate + Expected Risk Premium

We use the (risk-adjusted) discount rate in order to account for the liquidity, maturity, and tax considerations that cause corporate bonds to have an observed spread over the yield on a risk-free bond like the bonds issued by the government in a stable economy. (We grouped all of these factors together under the term "credit spread" in the Financial Markets course.) The minimum required return expected for a bond investor is equal to the sum of the following, which accounts for this spread between corporate bonds and risk-free bonds:

- **Default Risk Premium** – Compensates investors for the business' likelihood of default.
- **Liquidity Premium** – Compensates investors for investing in less liquid securities such as bonds. Government bonds typically are more liquid than corporate bonds. Government bonds are available in greater supply than even the most liquid corporates and have demand

from a wider set of institutional investors. In addition, government bonds can be used more readily as collateral in repo transactions and for centrally cleared derivatives.

- **Maturity Premium** – Compensates investors for the risk associated with bonds that mature many years into the future, which inherently carry more risk.
- **Taxation Premium** – Compensates investors for the taxable income that bonds generate. Interest income on U.S. corporate bonds is taxable by both the federal and state governments. Government debt, however, is exempt from taxes at the state level.
- **Projected Inflation** – Accounts for the devaluation of currency over time.
- **Risk-free Rate** – Refers to the rate of return an investor can expect on a riskless security (such as a T-bill).

We begin our calculation of the risk-adjusted discount rate by first turning our attention to estimating the expected risk premium.

The expected risk premium is obtained by subtracting the risk-free rate of return from the market rate of return and then multiplying the result by the beta that adjusts based on the magnitude of the investment risk involved. By carefully selecting a proxy short-term corporate bond's beta to the overall market, we can calculate an expected risk premium that will result in a risk-adjusted discount rate that incorporates liquidity, maturity, and tax considerations to produce a more accurate probability of default when using the bond valuation technique.

Expected Risk Premium = (Market Rate of Return - Risk-free Rate of Return) * Beta

To calculate the expected risk premium, we must first calculate the market rate of return. We can use the capital asset pricing model (CAPM) to determine the market rate of return.

$$r_{m} = r_{f} + \beta \cdot \text{MRP}$$

r_{m} = Market Rate of Return r_{f} = Risk-free Rate β = Beta MRP = Market Risk Premium

CAPM is an equilibrium model that takes the risk-free rate, the stock market's beta, and the market risk premium as inputs. Let's now determine the value for each of these inputs.

Government securities are assumed to be risk-free, at least from a credit standpoint. With this assumption, the appropriate rate to use in the market rate of return calculation is the government security having approximately the same duration as the asset being valued and sufficient liquidity so that the yield does not have an embedded liquidity risk premium. Equities are assumed to have a long duration, so a long-term government bond yield is an appropriate proxy for the risk-free rate.

In this step, the yield on the 10-year U.S. Treasury note will be used as the risk-free rate. We can scrape the current yield on the 10-year U.S. Treasury note from Yahoo Finance using the code below.

[6]:
```
# Ten-Year Risk-free Rate
timespan = 100
current_date = date.today()
past_date = current_date - timedelta(days=timespan)
ten_year_risk_free_rate_df = yfin.download("^TNX", past_date, current_date)
ten_year_risk_free_rate = (
    ten_year_risk_free_rate_df.iloc[len(ten_year_risk_free_rate_df) - 1, 4]
```

```
) / 100
ten_year_risk_free_rate
```

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

[6]:
```
0.032880001068115235
```

The market risk premium should be the expected return on the market index less the expected return (or yield) on the long-term government bond. For our purposes, we use the annual market risk premium provided by Aswath Damodaran, a professor at the Stern School of Business at New York University.

[7]:
```
# Market Risk Premium
market_risk_premium = 0.0472
```

According to asset pricing theory, beta represents the type of risk, systematic risk, that cannot be diversified away. By definition, the market itself has a beta of 1. As a result, beta will be equal to 1 when calculating the market rate of return.

[8]:
```
# Market Equity Beta
stock_market_beta = 1
```

We now have all the inputs required to calculate the market rate of return.

[9]:
```
# Market Rate of Return
market_rate_of_return = ten_year_risk_free_rate + (
    stock_market_beta * market_risk_premium
)
market_rate_of_return
```

[9]:
```
0.08008000106811523
```

Now that we have calculated the market rate of return, we can determine the expected risk premium by subtracting the risk-free rate from the market rate of return and multiplying the result by the beta for the bond.

Expected Risk Premium = (Market Rate of Return - Risk-free Rate of Return) * Beta

In this step, we will use a one-year risk-free rate so that the expected risk premium matches the duration we want for the risk-adjusted discount rate. We accomplish this by taking the yield on the very liquid 10-year U.S. Treasury note and raising it to the power of 1/10 in order to convert the yield to a one-year equivalent.

Note that `2**3` is Pythonic for 2^3, or 2 raised to the power of 3 (= 8). By the same token, `8**(1/3)` is the cube root of 8 (=2).

[10]:
```
# One-Year Risk-free Rate
one_year_risk_free_rate = (1 + ten_year_risk_free_rate) ** (1 / 10) - 1
one_year_risk_free_rate
```

[10]: 0.0032403403812457654

The final component needed to calculate the expected risk premium is the corporate bond's beta. Knowing that differences in liquidity within the universe of corporate bonds are great, we use the Vanguard Short-Term Corporate Bond Index Fund ETF Shares (VCSH) as a proxy for short-term maturity bonds. The beta from this index will enable us to embed some of the liquidity and maturity risk into the risk-adjusted discount rate that will be used to calculate the probability of default for the corporate bonds we will be analyzing. This should allow for better isolation of the credit risk associated with the chosen bonds.

A bond's beta is the sensitivity of that bond's return to the return of the market index. It is a measure of undiversifiable, systematic risk. As you see below, it can be calculated in (at least) two ways.

[11]:
```
# Vanguard Short-Term Corporate Bond Index Fund ETF Shares
bond_fund_ticker = "VCSH"
```

[12]:
```
# Download data for the bond fund and the market
market_data = yfin.download("SPY", past_date, current_date)  # the market
fund_data = yfin.download("VCSH", past_date, current_date)  # the bond fund
```

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

Calculate the beta of the bond fund (with respect to the S&P):

[13]:
```
# Approach #1 - Covariance/Variance Method:

# Calculate the covariance between the fund and the market -- this is the␣
 ↪numerator in the Beta calculation
fund_market_cov = fund_data["Adj Close"].cov(market_data["Adj Close"])
print("covariance between fund and market: ", fund_market_cov)

# Calculate market (S&P) variance -- this is the denominator in the Beta␣
 ↪calculation
market_var = market_data["Adj Close"].var()
print("market variance: ", market_var)

# Calculate Beta
bond_fund_beta_cv = fund_market_cov / market_var
print("bond fund beta (using covariance/variance): ", bond_fund_beta_cv)
```

```
covariance between fund and market:  2.547426578299434
market variance:  90.51190914101628
bond fund beta (using covariance/variance):  0.02814465634937143
```

[14]:
```
# Approach #2 - Correlation Method:
```

```
# Calculate the standard deviation of the market by taking the square root of␣
 ↪the variance, for use in the denominator
market_stdev = market_var**0.5
print("market standard deviation: ", market_stdev)

# Calculate bond fund standard deviation, for use in the numerator
fund_stdev = fund_data["Adj Close"].std()
print("fund standard deviation: ", fund_stdev)

# Calculate Pearson correlation between bond fund and market (S&P), for use in␣
 ↪the numerator
fund_market_Pearson_corr = fund_data["Adj Close"].corr(
    market_data["Adj Close"], method="pearson"
)
print("Pearson correlation between fund and market: ", fund_market_Pearson_corr)

# Calculate Beta
fund_beta_corr = fund_stdev * fund_market_Pearson_corr / market_stdev
print("bond fund beta (using correlation): ", fund_beta_corr)
```

```
market standard deviation:  9.513774705184913
fund standard deviation:  0.49560327566190693
Pearson correlation between fund and market:  0.5402747173233677
bond fund beta (using correlation):  0.028144656349371428
```

Note that `.corr()` above can be used to calculate any of the three correlation metrics we have discussed, taking the arguments 'pearson', 'kendall', or 'spearman' (with 'pearson' as the default). We include the argument here to emphasize this fact.

[15]:
```
# Bond's Beta: use the result of either of the two above approaches,␣
 ↪bond_fund_beta_cv or fund_beta_corr
bond_beta = fund_beta_corr
bond_beta
```

[15]:
```
0.028144656349371428
```

We now have all the components required to calculate the expected risk premium.

[16]:
```
# Expected Risk Premium
expected_risk_premium = (market_rate_of_return - one_year_risk_free_rate) *␣
 ↪bond_beta
expected_risk_premium
```

[16]:
```
0.002162625844034247
```

With the expected risk premium now in hand, we revisit the (risk-adjusted) discount rate equation:

Discount Rate = Risk-free Rate + Expected Risk Premium

The final input required for the risk-adjusted discount rate is the risk-free interest rate, which we define next.

Estimating the Risk-Free Rate We will again use a one-year risk-free rate so that it matches the duration we want for the risk-adjusted discount rate, which we will use to discount expected cash flows to determine the probability of default.

[17]:
```
# One-Year Risk-free Rate (same code as above)
one_year_risk_free_rate = (1 + ten_year_risk_free_rate) ** (1 / 10) - 1
one_year_risk_free_rate
```

[17]:
```
0.0032403403812457654
```

We can now combine the risk-free interest rate and the expected risk premium to obtain the risk-adjusted discount rate.

[18]:
```
# Risk-adjusted Discount Rate
risk_adjusted_discount_rate = one_year_risk_free_rate + expected_risk_premium
risk_adjusted_discount_rate
```

[18]:
```
0.005402966225280012
```

1.4 4. Conclusion

In this lesson, we reviewed CAPM and found the risk-adjusted discount rate, which is an input to our market-implied probability of default estimation.

In order to find the risk-adjusted discount rate, we had to find the one-year risk-free rate and the expected risk premium.

We found the one-year risk-free rate by taking the tenth-root of the ten-year risk-free rate.

We found the expected risk premium by first finding the market risk premium and the beta.

We saw that beta can be calculated using correlation and standard deviations, or covariance and the market variance, due to the mathematical relationships between these variables.

In the next lesson, we can use this newly found data point to finally calculate the market-implied probability of default, with a function built for this purpose.

References

- Donnelly, Hugh. "Calculating a Company's Probability of Default with Python." AlphaWave Data. https://github.com/AlphaWaveData/Jupyter-Notebooks/blob/master/AlphaWave%20Market-Implied%20Probability%20of%20Default%20Example.ipynb

- The code and related documentation used in this lesson is adapted from: **Hugh Donnelly, CFA***AlphaWave Data* **March 2021** under the following MIT License: