Financial_Data_Module_5_Lesson_3

April 7, 2023

0.1 FINANCIAL DATA

MODULE 5 | LESSON 3

---

1 ESTIMATING THE MARKET-IMPLIED PROBABILITY OF DEFAULT

Reading Time	25 minutes
Prior Knowledge	Probability of default, Basic Python, DataFrames
Keywords	`.apply()`, Lambda functions, `numpy.arange()`, Symbol library

So far in this module, we have been gathering all the inputs we need for the market-implied probability of default (PD). Now that we have calculated the risk-adjusted discount rate in the last lesson, we can solve for the PD using `SymPy`. We will also refresh our understanding of some essential Python, e.g., control-flow tools, data types, as well as the append and apply methods.

Note: The code that was introduced in the previous lessons 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 sympy import solve, symbols
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"
```

The cell below using a web-scraping library `selenium` to collect the data we need. However, there is a dataset that has been pre-loaded onto your virtual machine. If you'd like to download the latest data, change the value of `scrape_new_data` to `True`. Otherwise, you will use the pre-loaded data.

[3]:
```
scrape_new_data = False

if scrape_new_data:
    # 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")
```

```
else:
    bond_prices_df = pd.read_csv("bond-prices.csv")

bond_prices_df
```

[3]:

```
                       Issuer Name      Symbol Callable Sub-Product Type  Coupon  \
0                        HESS CORP      HES.GH      Yes   Corporate Bond   6.000
1                        HESS CORP      HES.GI      Yes   Corporate Bond   5.600
2                        HESS CORP  HES4136877      Yes   Corporate Bond   3.500
3                        HESS CORP  HES4405829      Yes   Corporate Bond   4.300
4                        HESS CORP  HES4405830      Yes   Corporate Bond   5.800
5   HESS MIDSTREAM OPERATIONS LP  HES4567499      Yes   Corporate Bond   5.625
6   HESS MIDSTREAM OPERATIONS LP  HES4927355      Yes   Corporate Bond   5.625
7   HESS MIDSTREAM OPERATIONS LP  HES5233164      Yes   Corporate Bond   4.250
8   HESS MIDSTREAM OPERATIONS LP  HES5392919      Yes   Corporate Bond   5.500
9     HESS MIDSTREAM PARTNERS LP  HES4918686      Yes   Corporate Bond   5.125

     Maturity Moody's®   S&P    Price  Yield
0  01/15/2040      Ba1  BBB-  107.168  5.365
1  02/15/2041      Ba1  BBB-  102.172  5.413
2  07/15/2024      Ba1  BBB-  100.089  3.451
3  04/01/2027      Ba1  BBB-   99.580  4.397
4  04/01/2047      Ba1  BBB-  104.191  5.485
5  02/15/2026      NaN   BB+  104.500  4.431
6  02/15/2026      NaN   BB+  100.625  5.227
7  02/15/2030      Ba2   BB+   90.970  5.718
8  10/15/2030      Ba2   BB+   98.430  5.738
9  06/15/2028      Ba2   BB+   96.766  5.769
```

[4]:

```
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'] = pd.Series(yearstillmaturity).round()` #␣
↪Better for Annual Payments
    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)
    # `df = df.loc[df['Maturity'] >= 0]`
    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  \
2   HESS CORP  HES4136877      Yes    Corporate Bond     3.5     465.0
3   HESS CORP  HES4405829      Yes    Corporate Bond     4.3    1455.0

  Moody's®   S&P    Price  Yield  Maturity Years
2      Ba1  BBB-  100.089  3.451             1.5
3      Ba1  BBB-   99.580  4.397             4.0
```

[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

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

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

[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

[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

[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
```

[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:  2.5474216821522098
market variance:  90.51190914101628
bond fund beta (using covariance/variance):  0.028144602255415502
```

[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:  0.4956027297587276
Pearson correlation between fund and market:  0.5402742740247123
bond fund beta (using correlation):  0.0281446022554155
```

[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]: 0.0281446022554155

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

[16]: 0.002162621687473028

[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

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

[18]: 0.0054029620687187935

1.1 1. SymPy: Solving the Probability of Default

Given the semi-annual coupon payment frequency for the bonds we are analyzing, we can feed the annual risk-adjusted discount rate into the `bonds_probability_of_default()` function below and it will convert these annual rates into semi-annual rates.

Our last step before running the `bonds_probability_of_default()` function is to define the principal payment, the recovery rate, and the symbol for probability of default (p) that the `solve()` function from the Python library `SymPy` will use to calculate the probability of default by equating future expected cash flows with the current price of the corporate bond when discounted at the risk-adjusted rate.

Notice below that the numpy library method `np.append()` is different from the pandas library method `df1.append(df2)`.

[19]:
```python
def bonds_probability_of_default(
    coupon, maturity_years, bond_price, principal_payment,␣
 ↪risk_adjusted_discount_rate
):
    price = bond_price
    prob_default_exp = 0

    #     `times = np.arange(1, maturity_years+1)` # For Annual Cashflows
    #     annual_coupon = coupon # For Annual Cashflows
    times = np.arange(0.5, (maturity_years - 0.5) + 1, 0.5)  # For Semi-Annual␣
 ↪Cashflows
    semi_annual_coupon = coupon / 2  # For Semi-Annual Cashflows

    # Calculation of Expected Cash Flow
    cashflows = np.array([])
    for i in times[:-1]:
        #         cashflows = np.append(cashflows, annual_coupon) # For Annual␣
 ↪Cashflows
        #     cashflows = np.append(cashflows,␣
 ↪annual_coupon+principal_payment)#  For Annual Cashflows
        cashflows = np.append(
            cashflows, semi_annual_coupon
        )  # For Semi-Annual Cashflows
    cashflows = np.append(
        cashflows, semi_annual_coupon + principal_payment
    )  # For Semi-Annual Cashflows
```

```
    for i in range(len(times)):
        #         This code block is used if there is only one payment remaining
        if len(times) == 1:
            prob_default_exp += (
                cashflows[i] * (1 - P) + cashflows[i] * recovery_rate * P
            ) / np.power((1 + risk_adjusted_discount_rate), times[i])
        #         This code block is used if there are multiple payments␣
↪remaining
        else:
            #             For Annual Cashflows
            #             if times[i] == 1:
            #                 prob_default_exp += ((cashflows[i]*(1-P) +␣
↪principal_payment*recovery_rate*P) / \
            #                                     np.power((1 +␣
↪risk_adjusted_discount_rate), times[i]))
            #             For Semi-Annual Cashflows
            if times[i] == 0.5:
                prob_default_exp += (
                    cashflows[i] * (1 - P) + principal_payment * recovery_rate␣
↪* P
                ) / np.power((1 + risk_adjusted_discount_rate), times[i])
            #             Used for either Annual or Semi-Annual Cashflows
            else:
                prob_default_exp += (
                    np.power((1 - P), times[i - 1])
                    * (cashflows[i] * (1 - P) + principal_payment *␣
↪recovery_rate * P)
                ) / np.power((1 + risk_adjusted_discount_rate), times[i])

    prob_default_exp = prob_default_exp - price
    implied_prob_default = solve(prob_default_exp, P)
    implied_prob_default = round(float(implied_prob_default[0]) * 100, 2)

    if implied_prob_default < 0:
        return 0.0
    else:
        return implied_prob_default
```

Understand all of the code above: how the function works, how the methods work (here and in general). Also, see the required reading section "A.7.5 Solving Equations Symbolically" and "A.7.6. Symbolic Plotting" from the following website, which shows you how powerful and easy to use the `SymPy` library is.

[20]:
```
# Variables defined for bonds_probability_of_default function
principal_payment = 100
recovery_rate = 0.40
P = symbols("P")
```

We are now ready to run the `bonds_probability_of_default()` function to calculate the market-implied probability of default for the chosen corporate bonds.

[21]:
```
bond_df_result.head(1)
```

[21]:
```
  Issuer Name      Symbol Callable Sub-Product Type  Coupon  Maturity  \
2   HESS CORP  HES4136877      Yes   Corporate Bond     3.5     465.0

  Moody's®  S&P    Price  Yield  Maturity Years
2      Ba1  BBB-  100.089  3.451             1.5
```

[22]:
```
# This calculation may take some time if there are many coupon payments
bond_df_result.head(1).apply(
    lambda row: bonds_probability_of_default(
        row["Coupon"],
        row["Maturity Years"],
        row["Price"],
        principal_payment,
        risk_adjusted_discount_rate,
    ),
    axis=1,
)

bond_df_result.head(1)
```

[22]:
```
  Issuer Name      Symbol Callable Sub-Product Type  Coupon  Maturity  \
2   HESS CORP  HES4136877      Yes   Corporate Bond     3.5     465.0

  Moody's®  S&P    Price  Yield  Maturity Years
2      Ba1  BBB-  100.089  3.451             1.5
```

1.2 2. Relevant and Essential Python: Control Flow, Data Types, and Apply

Read sections 4.7 and 4.8 of this website.

Read section 5.6 of this website.

Read this website.

These resources will help you understand the dense line of code above.

1.3 3. Conclusion

In this lesson, we used the inputs we found in the previous lessons to finally estimate the probability of default for the bond issuer of our choice. We had to provide an assumed recovery rate, and then use a solver (the `solve()` method) embedded in the `bonds_probability_of_default()` function, to find the probability of default implied by the price of the relevant bonds.

In the next lesson, we look at the ratings for these bonds provided by the ratings agencies. We will download the transition matrices (which include the probability of default for each rating) and compare the probability of default (PD) that we calculated in this lesson to the PD associated with the ratings agencies' ratings.

References

- "More Control Flow Tools." Python.org. https://docs.python.org/3/tutorial/controlflow.html
- Sargent, Thomas J., and John Stachurski. "Python Programming for Finance and Economics." QuantEcon.org, https://python-programming.quantecon.org/python_essentials.html#id7
- "Pandas.DataFrame.apply." Pandas.pydata.org. https://pandas.pydata.org/docs/reference/api/pandas.Dat
- The code and related documentation used in this lesson is adapted from: **Hugh Donnelly, CFA***AlphaWave Data* **March 2021** under the following MIT License:

Note: The above MIT license notice is copied here to comply with its requirements but it does **not** apply to the content in these lesson notes.
