Financial_Data_Module_5_Lesson_4

April 7, 2023

0.1 FINANCIAL DATA

MODULE 5 | LESSON 4

---

1 PLOTTING AND THE TRANSITION MATRIX

Reading Time	30 minutes
Prior Knowledge	Ratings
Keywords	Transition matrix, Loss Given Default (LGD), Matplotlib library

---

In this lesson, we compare the probabilities of default (PDs) that we estimated in the last lesson to ratings-implied PDs. We can do this because we will download the ratings agency transition matrix, which includes PD. We will also note that the PD increases with maturity by graphing, which means we will practice our graphing skills.

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 matplotlib.pyplot as plt
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
```

```python
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]:
```python
# 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]:
```python
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):
```

```python
        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  HES5392919      Yes   Corporate Bond   5.500
6  HESS MIDSTREAM OPERATIONS LP  HES4567499      Yes   Corporate Bond   5.625
7  HESS MIDSTREAM OPERATIONS LP  HES4927355      Yes   Corporate Bond   5.625
8  HESS MIDSTREAM OPERATIONS LP  HES5233164      Yes   Corporate Bond   4.250
9    HESS MIDSTREAM PARTNERS LP  HES4918686      Yes   Corporate Bond   5.125

     Maturity Moody's®   S&P    Price  Yield
0  01/15/2040     Baa3  BBB-  103.276  5.702
1  02/15/2041     Baa3  BBB-   98.664  5.717
2  07/15/2024     Baa3  BBB-   98.772  4.139
3  04/01/2027     Baa3  BBB-   99.512  4.414
4  04/01/2047     Baa3  BBB-  101.269  5.702
5  10/15/2030      Ba2   BB+   89.594  7.188
6  02/15/2026       WR   BB+  104.500  4.431
7  02/15/2026      NaN   BB+   95.520  7.051
8  02/15/2030      Ba2   BB+   84.818  6.843
9  06/15/2028      Ba2   BB+   90.268  7.164
```

[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'] = 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     Baa3  BBB-  98.772  4.139             1.5
3     Baa3  BBB-  99.512  4.414             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
```

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

[16]: 0.002162621687473028

[17]:
```python
# 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

[19]:
```
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
```

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

[21]:
```
# This calculation may take some time if there are many coupon payments
bond_df_result["Probability of Default %"] = 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)
```

[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  Probability of Default %
2     Baa3  BBB-  98.772  4.139             1.5                      6.71
```

1.1 1. Credit Ratings

As you recall from the Financial Markets course, credit ratings are used for bonds issued by corporations and government entities as well as for asset-backed securities (ABS) and mortgage-backed securities (MBS). The three major global credit rating agencies are Moody's Investors Service, Standard & Poor's, and Fitch Ratings. Each provides credit quality ratings for issuers as well as specific issues. These are ordinal ratings focusing on the probability of default.

The credit rating agencies consider the expected loss given default (LGD) by means of **notching**, which is an adjustment to the issuer rating to reflect the priority of claim for specific debt issues of that issuer and to reflect any subordination. The issuer rating is typically for senior unsecured debt. The rating on subordinated debt is then adjusted, or "notched" by lowering it one or two levels - for instance, from A+ down to A or further down to A-. This inclusion of loss given default in addition to the probability of default explains why they are called "credit ratings" and not just "default ratings."

The rating agencies report transition matrices based on their historical data. A transition matrix shows the probability that a rating changes (or stays the same) in one year's time. (The probability that the rating changes to default is the probability of default.)

We can verify the accuracy of the market-implied default probabilities with these rating agencies' transition matrices. Using the code below, we can obtain the Standard & Poor's Average One-Year Transition Rates For Global Corporates using historical data from 1981-2019 to verify the market-implied default probabilities calculated earlier.

1.2 2. Plotting

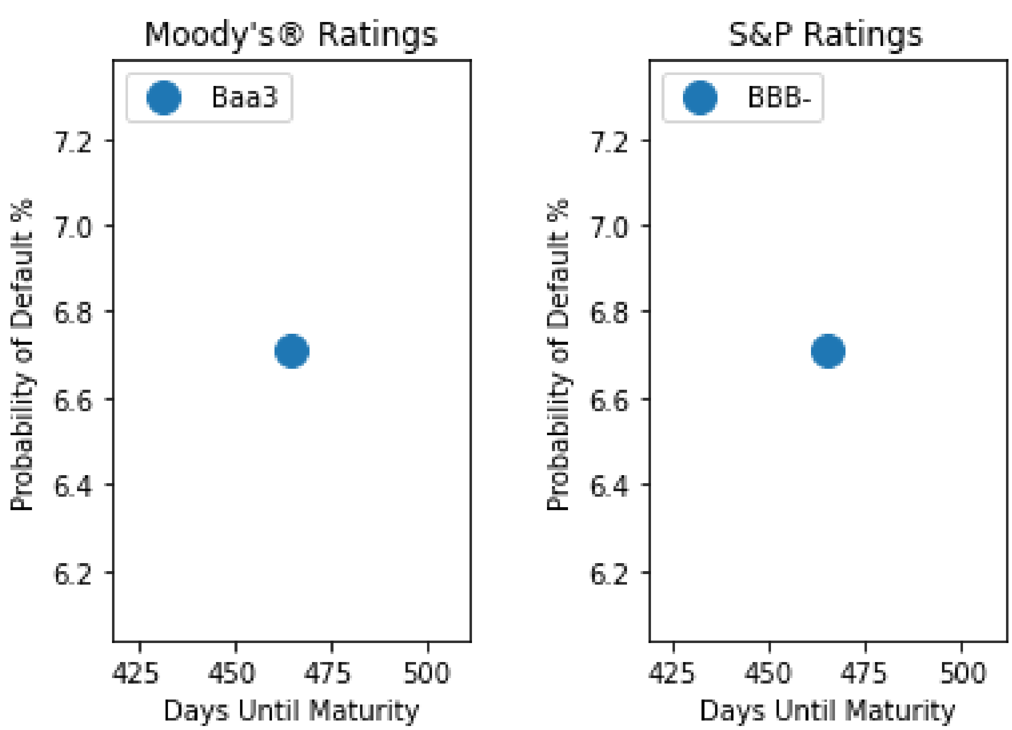
To get ready for the graphing below, please make sure you do all the required readings for this lesson and lesson 3.

[22]:
```python
def prob_default_term_structure(df):
    fig, (ax1, ax2) = plt.subplots(1, 2, clear=True)
    fig.subplots_adjust(wspace=0.5)
    Mgroups = df.groupby("Moody's®")
    ax1.clear()
    ax1.margins(0.5)
    ax1.set_xlabel("Days Until Maturity")
    ax1.set_ylabel("Probability of Default %")
    ax1.set_title("Moody's® Ratings")
    for name, group in Mgroups:
        ax1.plot(
            group["Maturity"],
            group["Probability of Default %"],
            marker="o",
            linestyle="",
            ms=12,
            label=name,
        )
    ax1.legend(numpoints=1, loc="upper left")
```

```python
    SPgroups = df.groupby("S&P")
    ax2.clear()
    ax2.margins(0.5)
    ax2.set_xlabel("Days Until Maturity")
    ax2.set_ylabel("Probability of Default %")
    ax2.set_title("S&P Ratings")

    for name, group in SPgroups:
        ax2.plot(
            group["Maturity"],
            group["Probability of Default %"],
            marker="o",
            linestyle="",
            ms=12,
            label=name,
        )
    ax2.legend(numpoints=1, loc="upper left")
    plt
```

[23]:
```python
prob_default_term_structure(bond_df_result)
```



1.3 3. Downloading the Transition Matrix

[24]:
```python
tgt_website = r"https://www.spglobal.com/ratings/en/research/articles/
↪200429-default-transition-and-recovery-2019-annual-global-corporate-default-and-rating-tran
```

[25]:
```python
def get_transition_matrix(tgt_website):
    df_list = pd.read_html(tgt_website)
    matrix_result_df = df_list[22]

    return matrix_result_df


scrape_new_data = False
if scrape_new_data:
    transition_matrix_df = get_transition_matrix(tgt_website)
else:
    transition_matrix_df = pd.read_csv("transition-matrix.csv")
```

[26]:
```python
sp_clean_result_df = pd.DataFrame(transition_matrix_df.iloc[:34, :19].
↪dropna(axis=0))
sp_clean_result_df
```

[26]:

	From/to	AAA	AA+	AA	AA-	A+	A	A-	BBB+	BBB \
0	AAA	87.03	5.89	2.51	0.69	0.16	0.24	0.13	0.00	0.05
2	AA+	2.31	78.94	10.91	3.54	0.71	0.33	0.19	0.05	0.09
4	AA	0.42	1.31	80.76	8.53	2.72	1.15	0.36	0.39	0.13
6	AA-	0.04	0.11	3.77	78.80	9.68	2.19	0.60	0.25	0.15
8	A+	0.00	0.06	0.44	4.44	78.38	8.73	2.15	0.61	0.34
10	A	0.03	0.04	0.22	0.41	5.32	78.88	6.74	2.38	0.86
12	A-	0.04	0.01	0.06	0.15	0.42	6.49	78.12	7.23	1.98
14	BBB+	0.00	0.01	0.05	0.06	0.20	0.74	7.13	75.83	7.98
16	BBB	0.01	0.01	0.04	0.03	0.10	0.31	1.00	7.73	76.00
18	BBB-	0.01	0.01	0.02	0.04	0.06	0.14	0.25	1.17	9.31
20	BB+	0.04	0.00	0.00	0.03	0.03	0.08	0.08	0.41	1.59
22	BB	0.00	0.00	0.03	0.01	0.00	0.06	0.05	0.16	0.47
24	BB-	0.00	0.00	0.00	0.01	0.01	0.01	0.05	0.09	0.23
26	B+	0.00	0.01	0.00	0.03	0.00	0.03	0.06	0.04	0.05
28	B	0.00	0.00	0.01	0.01	0.00	0.03	0.04	0.02	0.05
30	B-	0.00	0.00	0.00	0.00	0.02	0.03	0.00	0.06	0.05
32	CCC/C	0.00	0.00	0.00	0.00	0.03	0.00	0.08	0.05	0.08

	BBB-	BB+	BB	BB-	B+	B	B-	CCC	D
0	0.00	0.03	0.05	0.03	0.00	0.03	0.00	0.05	0.00
2	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.08	0.05	0.03	0.02	0.02	0.00	0.02	0.05	0.02
6	0.07	0.03	0.00	0.00	0.03	0.08	0.00	0.00	0.03
8	0.09	0.06	0.09	0.01	0.07	0.03	0.00	0.00	0.05

10	0.27	0.10	0.10	0.06	0.08	0.02	0.00	0.01	0.05
12	0.57	0.13	0.13	0.11	0.10	0.02	0.01	0.03	0.06
14	1.56	0.36	0.29	0.13	0.15	0.10	0.02	0.06	0.10
16	6.11	1.34	0.58	0.27	0.22	0.11	0.03	0.05	0.16
18	72.40	5.47	2.08	0.83	0.36	0.22	0.16	0.21	0.25
20	11.33	65.29	7.42	2.61	0.95	0.53	0.24	0.36	0.31
22	2.00	9.44	65.41	8.46	2.22	1.02	0.31	0.52	0.51
24	0.35	1.69	9.57	63.71	8.42	3.04	0.81	0.66	0.91
26	0.10	0.31	1.42	8.17	62.91	9.20	2.51	1.71	1.98
28	0.03	0.11	0.23	1.09	7.38	62.00	9.32	3.85	3.20
30	0.10	0.08	0.13	0.46	2.18	10.06	54.63	11.70	6.49
32	0.05	0.03	0.16	0.40	0.98	2.57	9.41	43.64	27.08

The above is the Standard & Poor's 2019 transition matrix. It shows the probabilities of a particular rating transitioning to another over the course of the following year. An A-rated issuer has a 78.88% probability of remaining at that level, a 0.03% probability of moving up to AAA; a 0.22% probability of moving up to AA; an 0.86% probability of moving down to BBB; 0.10% down to BB; 0.02% to B, 0.01% to CCC, CC, or C; and 0.05% to D, where it is in default.

Using the Selenium script mentioned earlier to retrieve the Standard & Poor's credit ratings, we can use the corporate bond's credit rating to determine the probability of a particular rating transitioning to D (default) during the next year according to the Standard & Poor's 2019 transition matrix.

[27]:
```python
# Will scrape the default probability for each rating

sp_rating_list = [
    "AAA",
    "AA+",
    "AA",
    "AA-",
    "A+",
    "A",
    "A-",
    "BBB+",
    "BBB",
    "BBB-",
    "BB+",
    "BB",
    "BB-",
    "B+",
    "B",
    "B-",
]

ccc_list = ["CCC+", "CCC", "CCC-", "CC+", "CC", "CC-", "C+", "C", "C-"]

sp_rating = None
```

```python
for i in sp_rating_list:
    if bond_df_result["S&P"].iloc[0] == i:
        sp_rating = bond_df_result["S&P"].iloc[0]

if sp_rating is None:
    for i in ccc_list:
        if bond_df_result["S&P"].iloc[0] == i:
            sp_rating = "CCC/C"

sp_transition_dp = 0

for i in range(33):
    if transition_matrix_df.loc[i][0] == sp_rating:
        sp_transition_dp += float(sp_clean_result_df.loc[i][18])

sp_transition_dp
```

[27]: 0.25

It appears that the market-implied probability of default we calculated for the nearest maturity corporate bond is close to the probability of default as determined from the historical data in the Standard & Poor's 2019 transition matrix.

[28]:
```python
# Compare the nearest maturity Market-implied probability of default with
# the historical probability of default in the Standard & Poor's 2019␣
 ↪transition matrix
print(
    "Market-implied probability of default = %s"
    % (bond_df_result["Probability of Default %"].iloc[0])
    + "%"
)
print("Standard & Poor's probability of default = %s" % (sp_transition_dp) +␣
 ↪"%")
```

```
Market-implied probability of default = 6.71%
Standard & Poor's probability of default = 0.25%
```

1.4 4. Conclusion

In the example above, the bond valuation techniques using a risk-adjusted discount rate do a reasonably good job of estimating the market-implied default probabilities. We calculated the expected cash flow at each period 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 $(1-p)$. One reason for any differences between historical and market-implied default probabilities is that historical default probabilities do not include the default risk premium associated with uncertainty over the timing of possible default loss.

The model used here is very sensitive to the discount and recovery rates selected. For simplicity, we assume a flat government bond yield curve, but it could be upward or downward sloping. A more sophisticated and realistic model of the discount rates would require that they be calculated sequentially by a process known as "bootstrapping." We also assume in this example that the recovery rate is 40%, but another possibility is to change the assumed recovery rate to either 30% or 60% of exposure. Another simplifying assumption is that recovery is instantaneous. In practice, lengthy time delays can occur between the event of default and eventual recovery of cash. Notice that we assume that the recovery rate applies to interest as well as principal.

Also, we assume that default occurs only on coupon payment dates and that default will not occur on date 0, the current date. Although we assumed the annual default probability is the same each year, this does not need to be the case.

Even with the assumptions made in this analysis, the market-implied probability of default model built here does a fairly good job at identifying risk of corporate defaults and may suffice for simply rank ordering firms by credit worthiness.

References

- Vanderplas, Jake. "04.00-Introduction-To-Matplotlib.ipynb." *GitHub*, https://github.com/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/04.00-Introduction-To-Matplotlib.ipynb
- Sargent, Thomas, and John Stachurski. "10. Matplotlib." *Python Programming for Economics and Finance.* https://python-programming.quantecon.org/matplotlib.html.
- Coleman, Chase, et al. "GroupBy." *QuantEcon DataScience.* https://datascience.quantecon.org/pandas/groupby.html
- 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.
